MPLAB® Harmony Help - Peripheral Libraries

MPLAB Harmony Integrated Software Framework v1.11

Peripheral Libraries Help

This section provides descriptions of the peripheral libraries that are available in MPLAB Harmony.

Description

The source code for the MPLAB Harmony Peripheral Libraries (PLIBs) is distributed in the installation (`<install-dir>/framework/`peripheral). Each PLIB folder has (at least) two sub-folders: `processor` and `templates`. The files within the `processor` directory are generated mechanically during the MPLAB Harmony development process. These files define register macro translations and translate between the API function prototypes and different variants of the "inline" function implementations that are contained in the `templates` folder.

With a few exceptions PLIBs are implemented as C-language inline functions and translated by the preprocessor at build time. If almost any optimization is used and constants are passed into them, each call will normally compile away to just a few bytes of code. Prebuilt binary (`.a`) files are also provided (built at a -O3 optimization level) for times when no optimization is used or any time the compiler decides to generate a true function call instead of inlining the function. The prebuilt binary (`.a`) libraries are generated by changing the definitions of the PLIB_INLINE and PLIB_INLINE_API macros attached to the PLIB implementations from "inline extern" to just "extern" when the binaries are built. A MPLAB X IDE project is provided in the `<install-dir>/build/framework/peripheral` folder to allow a user to rebuild these binaries, if desired. Be sure to inspect the compiler settings and post-build steps, to see how this is done and to ensure the desired settings are used.

Peripheral Library Overview

This topic provides an overview of the peripheral libraries that are available in MPLAB Harmony.

Introduction

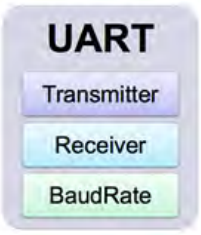
This topic provides an overview of the peripheral libraries in MPLAB Harmony.

Description

Supported PIC32 Devices and Release Type

Note: Refer to the *Release Contents* > *Peripheral Libraries* section in the MPLAB Harmony Release Notes for the list of supported PIC32 devices and their release type. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

MPLAB Harmony peripheral libraries (PLIBs) model the hardware peripheral modules available on Microchip microcontrollers by breaking each peripheral down into a set of individual features. For example, a (simplified) UART peripheral module may have three features, as shown in the following diagram.



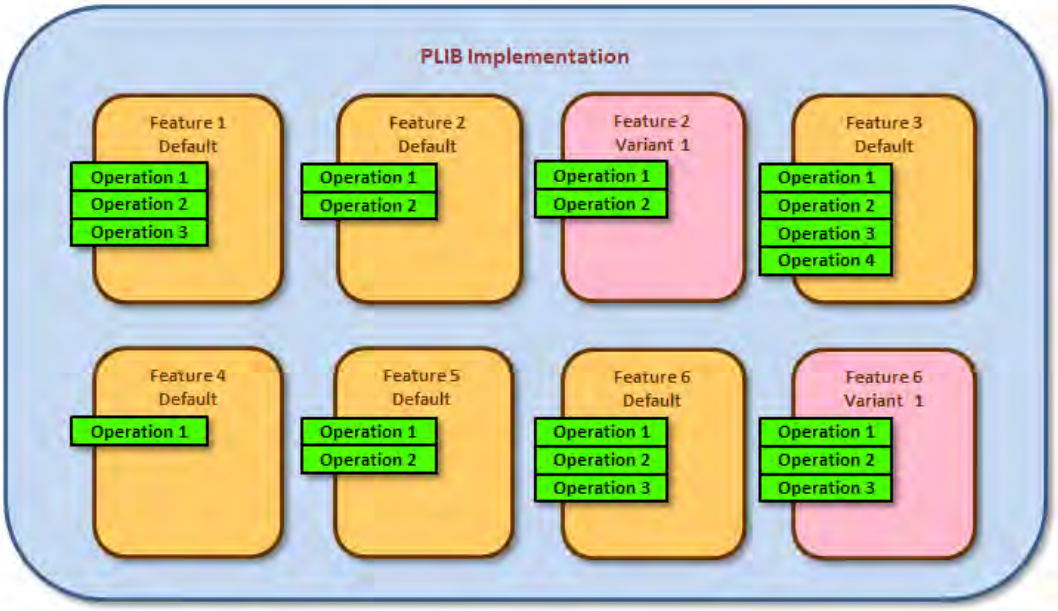
Every feature of a peripheral will have one or more primitive operations that can be performed using that feature. These operations are named in a way that identifies the module, feature, and operation and the fact that they are Peripheral Library functions, as shown in the following figure.

The simplified UART peripheral example has a "baud rate" feature. That feature may have two operations that it can perform. One operation might be a "set" function to set the current baud rate at which the UART will send and receive data and the other might be a "get" operation to find out the baud rate at which the UART is currently transmitting and receiving. Example C-language function signatures of these operations is as follows:

```
void    PLIB_USART_BaudRateSet( USART_MODULE_ID index, uint32_t clockFrequency,
                                uint32_t baudRate );

uint32_t    PLIB_USART_BaudRateGet( USART_MODULE_ID index, int32_t clockFrequency );
```

Notice that each function accepts an "index" parameter, as well as data parameters relevant to the operation itself, such as the input clock frequency and desired baud rate. The "index" parameter allows one set of PLIB functions to support any number of instances of the peripheral on a give microcontroller. Thus, each peripheral library exposes all of the features available on all instances of a given type of peripheral.

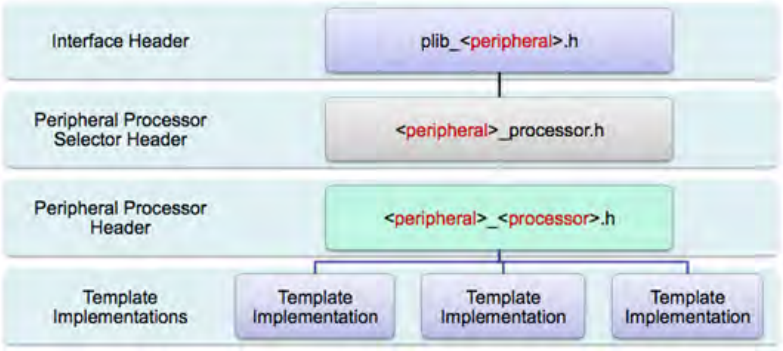
The function signature (name, parameters, and return value) for the operations supported by that feature will be the same on all microcontrollers even if the implementations of those functions are different from one microcontroller to another. These are known as "implementation variants". As illustrated in the following diagram, a PLIB implementation is made up of the complete set of default and variant implementations of the operations of every feature supported by a specific type of peripheral on a specific microcontroller.



Using this method, MPLAB Harmony Peripheral Libraries provide consistent C-language functional interfaces to access the peripherals available on all supported microcontrollers.

However, if a feature is not supported on all instances of the peripheral on all supported microcontrollers, the peripheral library communicates this to the developer in one of two ways. If a feature is not supported by any instance of the peripheral on a given microcontroller, the primitive functions related to that feature will also be unsupported and they will cause build warnings indicating this if they are called. If a feature is supported on at least one, but not on all of the available instances of a the peripheral on the microcontroller in use, the functions will be supported; however, they will provide a run-time error to facilitate debugging. Which features are supported on each instance of each microcontroller is documented in the help material for each peripheral library.

Peripheral libraries are primarily implemented as C-language "inline" functions. This allows the implementation of each function to be selected at build time, based upon the processor selection. To facilitate this, MPLAB Harmony peripheral libraries are implemented in layers of C-language header (`.h`) files, as shown in the following diagram.



For a given peripheral, "Interface Header" defines set of functions and data types (although, potentially not all of the data types, as discussed in a following section) that make up the interface to that peripheral library. To select the correct implementation for each processor, a "Peripheral Processor Selector Header" includes only the appropriate "Peripheral Processor Header" for the selected processor. This header then acts as a "mapping" header that translates the interface functions defined in the interface header to the correct implementation variant template for each function in the "Template Implementation" files.

This may seem a little confusing at first, but since all functions are implemented in header files as "inline" functions, the compiler can perform all of the mapping at build time, resulting in very small static implementations when static data is passed to the PLIB functions.

One more key concept is that, although peripheral libraries do provide a level of abstraction, they are still very low level. PLIBs combine accesses to multiple registers to implement a single operation when possible. They hiding register details from the caller and provide a consistent "functional" interface to a peripheral that hides differences in implementation variants. But, PLIBs do not manage the state of the peripheral to keep it running and they do not control access to the peripheral to prevent conflicts between different clients. Those jobs belong to the device drivers.

Fundamentally, peripheral libraries are peripheral access libraries. PLIB function calls do not block or call anything outside of the PLIB itself (with the exception of some removable debugging support). A PLIB function does not maintain any state data (outside of the data stored in hardware registers) as it may be called from within the main line of execution, from within an RTOS thread, or from within an ISR. And, PLIB functions are normally generated as "PLIB_INLINE", not as actual function calls (although optimized, prebuilt PLIB libraries are provided to support those times when the compiler does not generate the function "PLIB_INLINE"). A PLIB simply provides the ability to directly access and manipulate the features of a given peripheral using primitive operations. It is the responsibility of the calling module (usually the device driver) to store and appropriately protect any state data necessary to keep the peripheral running.

So, while an application (or any other layer) may directly access the PLIB for any particular peripheral in the system, if it does, it must be the only

module in the system that does this and it then becomes responsible for the correct operation that peripheral instance in that system. That is why PLIBs are not normally the recommended way to access peripherals. It is usually better to access a peripheral through a driver or system service that manages the state machine of the peripheral and protects the peripheral so that accesses to it from multiple clients will not interfere with the correct operation of the peripheral.

Configuring a Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Description

Peripheral library interfaces are common across all supported processors. But, their implementations are part specific and are provided in two forms that must normally be used together to avoid the possibility of build errors.

The first form of the PLIB implementation is as a set of C-language "inline" functions found in `.h` header files in the `<install-dir>/framework/peripheral` folder in the MPLAB Harmony installation. The correct implementation template files for the functions supported by each variant of a feature found on a selected processor is selected and included in the header file hierarchy when a PLIB interface header is included in a source file that uses it (either directly or by including `peripheral.h`). The second form in which the peripheral library implementations are provided is as a set of part-specific prebuilt binary library `.a` files (available in the `<install-dir>/bin/framework/peripheral` folder in the MPLAB Harmony installation.

To ensure that your MPLAB Harmony project builds correctly (regardless of the level of optimization selected), you must do two things. You must include the appropriate PLIB interface header file (either by including `peripheral.h` or the specific PLIB interface header) in any C-language file that calls a PLIB function. And, you must include the appropriate prebuilt binary library file (the one whose file name includes the appropriate part name) for the processor you selected in your MPLAB X IDE project.

Both of these steps are necessary because the compiler decides at build time if it will generate an actual function call to each PLIB function or if it will generate the function directly in line with the code that calls it. When the optimization level is high enough, the compiler will directly generate the PLIB function code inline with the calling code and no prebuilt library would be required. But, if the compiler "judges" that the function is too large to inline (or if optimizations are turned off), it will generate an actual function call. When this happens, the prebuilt library is required or the linker will generate an error when it tries to link the function call to an actual implementation. Since the compiler makes this determination for each function call it encounters, the safest choice is to always include the prebuilt binary `.a` file in your project.

Fortunately, the MPLAB Harmony installation provides prebuilt binary `.a` forms of the peripheral libraries that were built using a "-O3" level of optimization so that users of free versions of the compiler, which do not support this advanced level of optimization, can benefit from them. This level of optimization usually provides the best over-all compromise between code size and speed. However, if a different level of optimization is desired, users of the pro version of the compiler can rebuild the peripheral library `.a` file using the MPLAB X IDE project provided in the `<install-dir>/build/framework/peripheral` folder of the MPLAB Harmony installation.

To build the binary `.a` form of the peripheral libraries, the PLIBs require the project to define two macros before any file that includes any PLIB header. These macros are used to enable or disable the "inline" attribute on the peripheral library function definitions so that they can be built into a binary `.a` file that exports the PLIB API function symbols. These macros, their usage, and their default definitions are described as follows.

Macro:

PLIB_INLINE_API

Summary:

Determines if PLIB interface (API) functions are treated by the compiler as inline or extern (called) functions.

Description:

This macro is used as an attribute of every peripheral library interface (API) function. Its default definition allows the compiler to choose to either generate the function directly in line with the calling code or to generate an actual function call which must later be linked to an actual function implementation, based on the size of the code generated with the current optimization settings.

Remarks:

The default definition of this macro is:

```
#ifndef PLIB_INLINE_API
    #define PLIB_INLINE_API extern inline
#endif
```

To build a binary `.a` form of the peripheral libraries, define this macro as shown in the following example to export all PLIB API functions so that calls to them can be linked to the library generated.

```
#define PLIB_INLINE_API  extern
```

Macro:

PLIB_INLINE

Summary:

Determines if PLIB support functions are generated as static inline or extern inline functions.

Description:

This macro is used as an attribute of every peripheral library (PLIB) support function. PLIB support functions are functions that are called by the PLIB interface (API) functions, but are not themselves intended to be exported as PLIB interface functions. This macro's default definition prevents the compiler from generating superfluous error messages that would occur if the PLIB support functions were declared with a different scope (static versus extern) from the PLIB interface functions, which would occur when the PLIB headers are included in calling code.

This superfluous error message would occur because, the support functions do not need to be exported when a binary `.a` file is generated (and, thus should be declared with "static" scope). But, the compiler would detect the mismatch between the scope of the support functions when the API functions are declared with external scope when the PLIBs are directly included in calling code.

Remarks:

The default definition of this macro is:

```
#ifndef PLIB_INLINE
    #define PLIB_INLINE extern inline
#endif
```

To build a binary `.a` form of the peripheral libraries, define this macro as shown in the following example. This prevents the binary `.a` library from exporting all PLIB support functions, dramatically reducing the size of the generated library's symbol table.

```
#define PLIB_INLINE  static inline
```

Peripheral Library Porting Example

Peripheral Library Porting Example Help

Introduction

Provides an example on how to port a legacy (i.e., prior to MPLAB Harmony) USART Peripheral Library (PLIB) demonstration application to a MPLAB Harmony application using the MPLAB Harmony Configurator (MHC).

Description

This section describes the process to port the legacy UART PLIB Interrupt demonstration application (`<compiler-install-dir>/examples/plib_examples/uart/uart_interrupt`) to MPLAB Harmony.

The following assumptions are made:

- The PIC32MX795F512L device will be used; however, the process described in this section is applicable for other PIC32 devices with appropriate changes
- The Explorer 16 Development Board is the hardware used in this example
- For the v1.33 MPLAB XC32 C/C++ Compiler, the `examples` folder is not present. To view the legacy USART PLIB example, refer to v1.31 or earlier of the MPLAB XC32 C/C++ compiler.

Porting Procedure

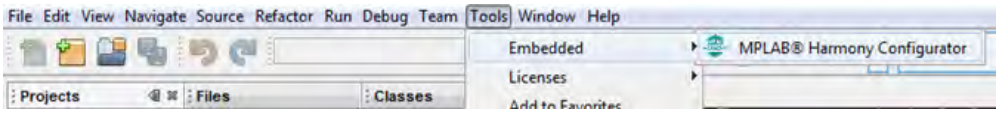
Describes the steps to set up the porting process.

Description

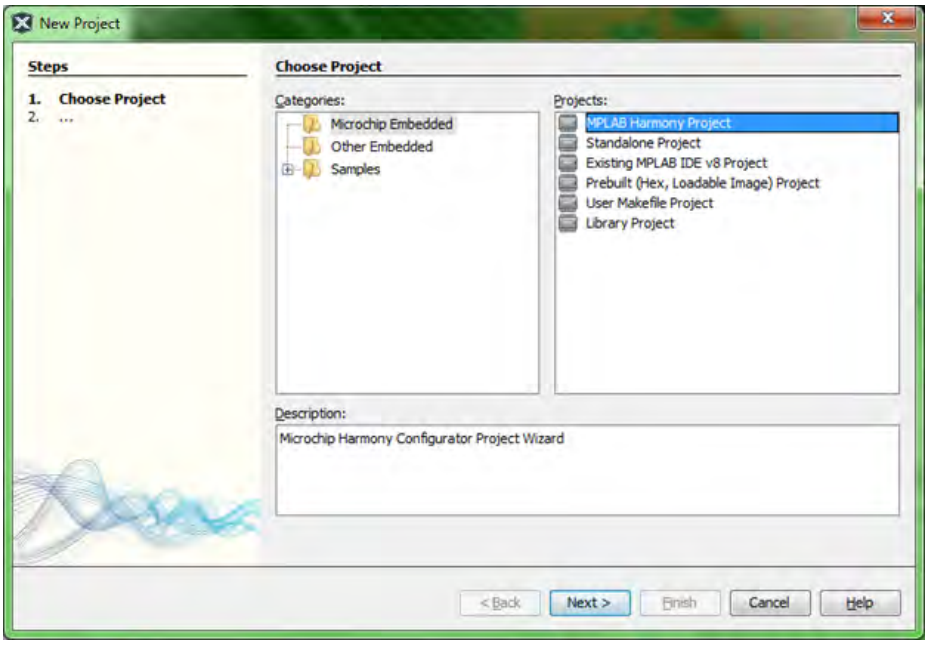
Step 1: Install the latest version of MPLAB Harmony. Throughout this porting guide, it is assumed that MPLAB Harmony is installed in its default location: `C:/microchip/harmony/<version>`. The `<version>` folder is assumed to be the root directory and all further steps will be explained relative to this root folder.

Step 2: Open MPLAB X IDE.

Step 3: Since the project will be created using the MHC, ensure that the MHC plug-in has been installed in MPLAB X IDE. You can verify the installation by selecting Tools > Embedded. If MHC is installed, you will see MPLAB Harmony Configurator is available as an option. Refer to Installing a Plug-in Module for information on installing the plug-in.

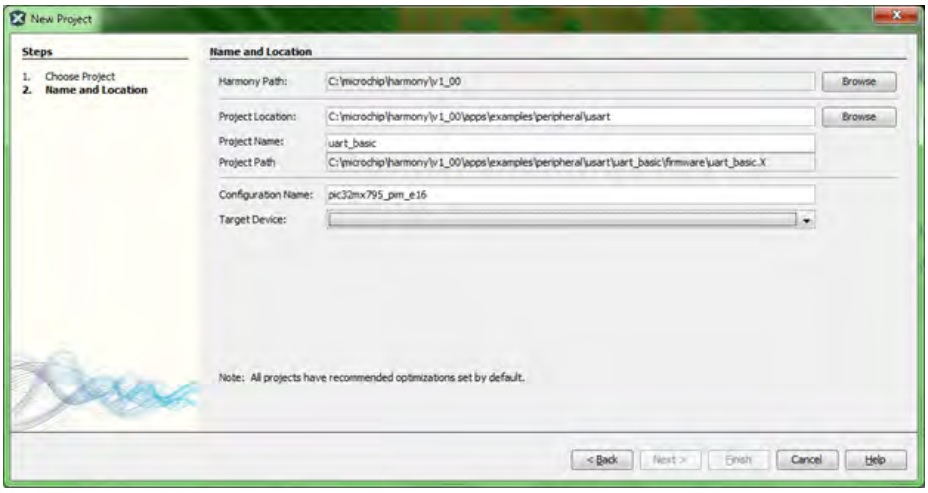


Step 4: Select *File > New Project* or click the New Project icon in MPLAB X IDE. In Categories, select **Microchip Embedded** and in Projects select **MPLAB Harmony Project** from the list of available project templates, and then click **Next** to launch the Microchip Harmony Configurator Project Wizard.

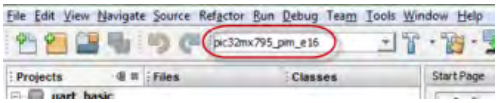


Step 5: Specify the following in the New Project dialog:

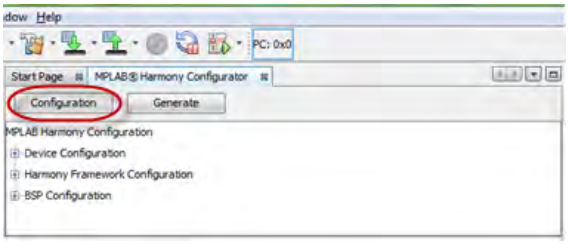
- Project Location: `<install-dir>/apps/examples/peripheral/usart`
  - Project Name: uart_basic
  - Configuration Name: pic32mx795_pim_e16
  - Target Device of PIC32MX795F512L



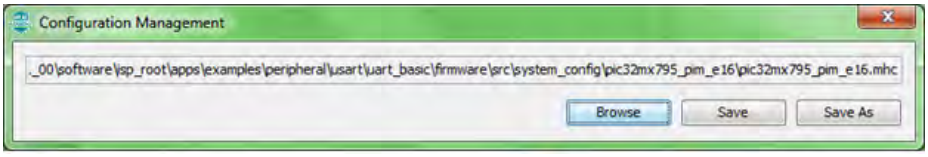
Step 6: Once the project is created, the configuration name will be set as "pic32mx795_pim_e16", as shown in the following figure.



Step 7: In the MPLAB Harmony Configurator tab, click **Configuration**.

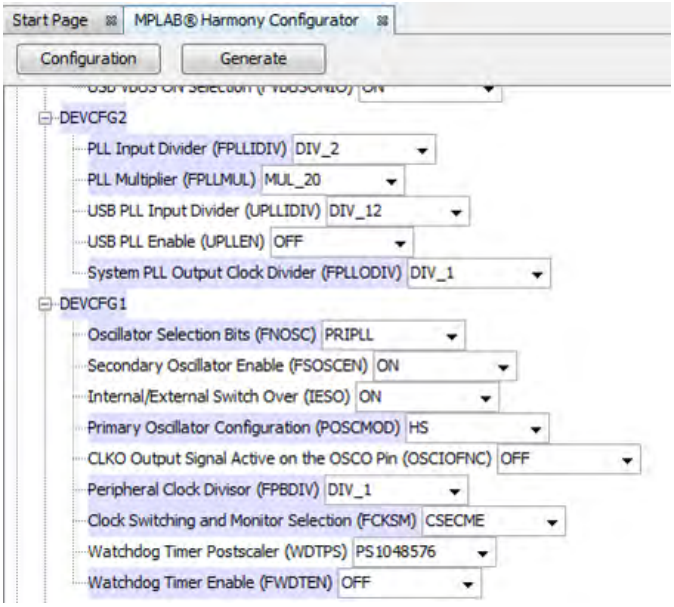


The Configuration Management dialog appears, which lists the folder path where the default configurator file (`.mhc`) will be saved. It is recommended not to modify the default path. Click **Save** to continue.

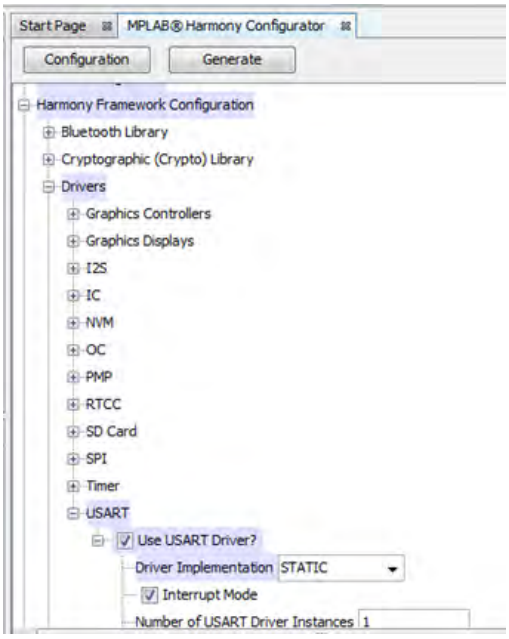


Step 8: In the MPLAB Harmony Configuration tab, configure the Device Configuration, Harmony Framework Configuration and BSP Configuration by selecting appropriate items from each drop down menu.

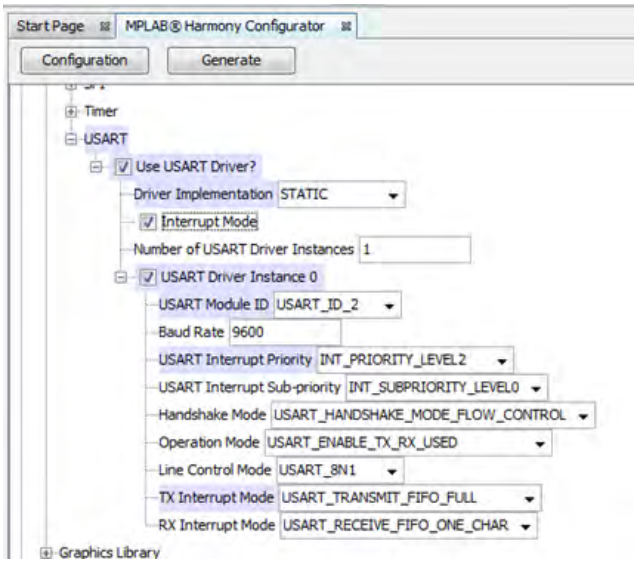
- Device Configuration: Since the controller should be running at 80 MHz with an 8 MHz external crystal, the configuration bits are set from the drop down menu, as shown in the following figure



- Harmony Framework Configuration: To enabler use of the UART Peripheral Library, the UART driver should be selected in the STATIC configuration in Interrupt mode, as shown in the following figure



- UART Configuration: Configure the UART2 with baud rate set to 9600, and interrupt enabled only for RX and Error. Also, set the other UART parameters, as shown in the following figure.

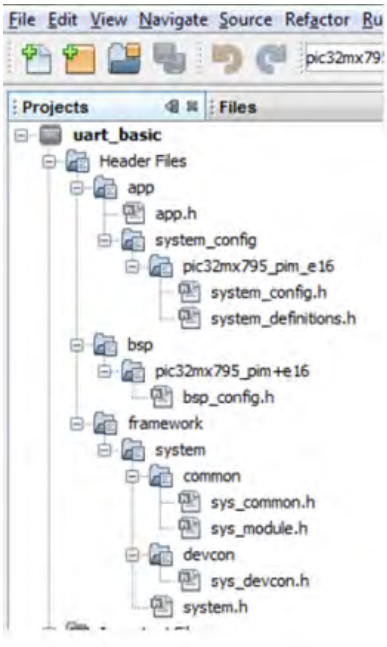


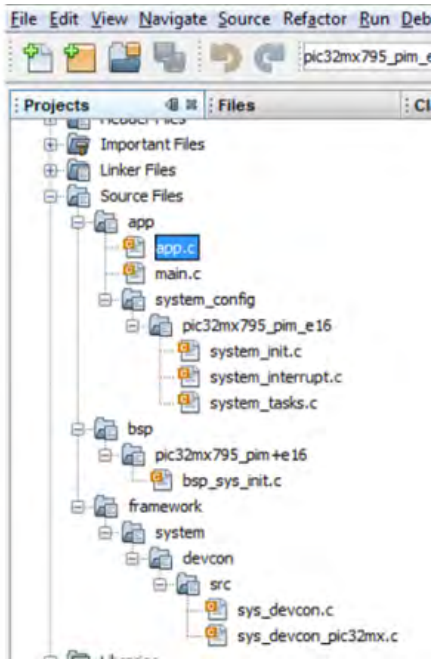
- BSP Configuration: Configure the BSP to use the PIC32MX795F512L and Explorer 16 Development Board, as shown in the following figure

Step 9: Click **Generate**. Since, the configurations were modified from the default values, MHC will ask whether or not the new configuration setting file (`.mhc`) should be saved. Click **Save** to continue.

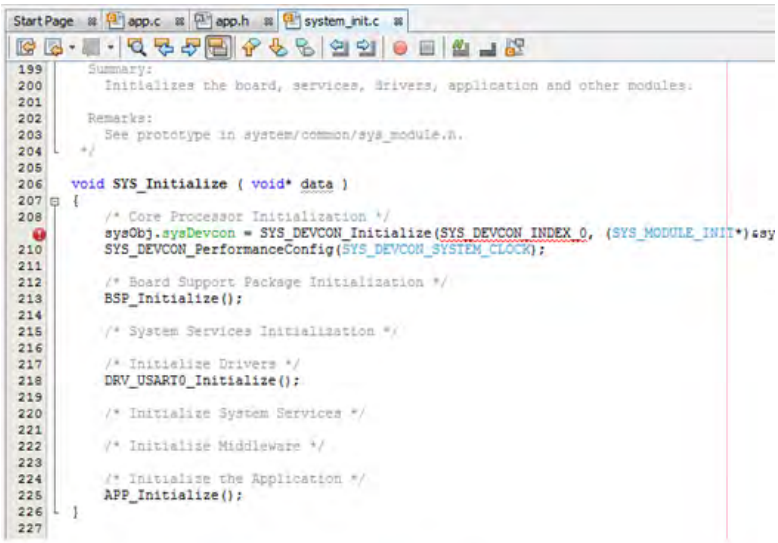
Step 10: Click **Generate**. Once the files are generated, the header, source, and library files will be added to the `uart_basic` project. The explanation of each logical folder is as follows:

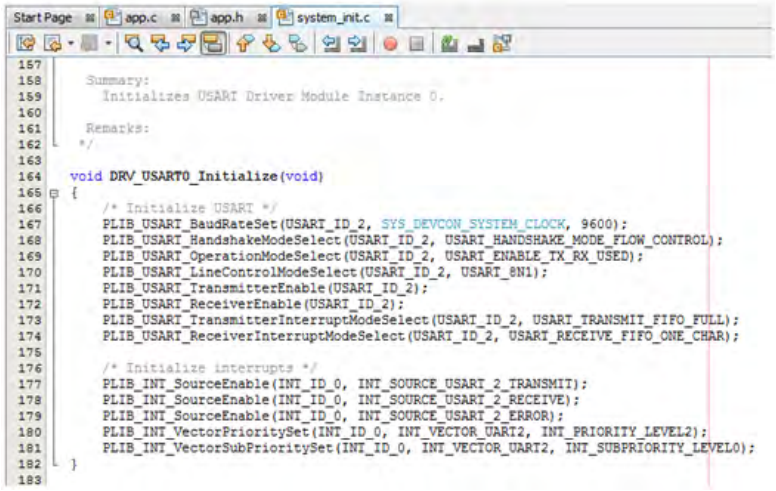
- `app` – Contains all application specific source files
- `bsp` – Contains all board specific source files
- `framework` – Contains all framework files from MPLAB Harmony
- `system_config` – Contains the system and application configuration, initialization, and Interrupt Service Routine (ISR)





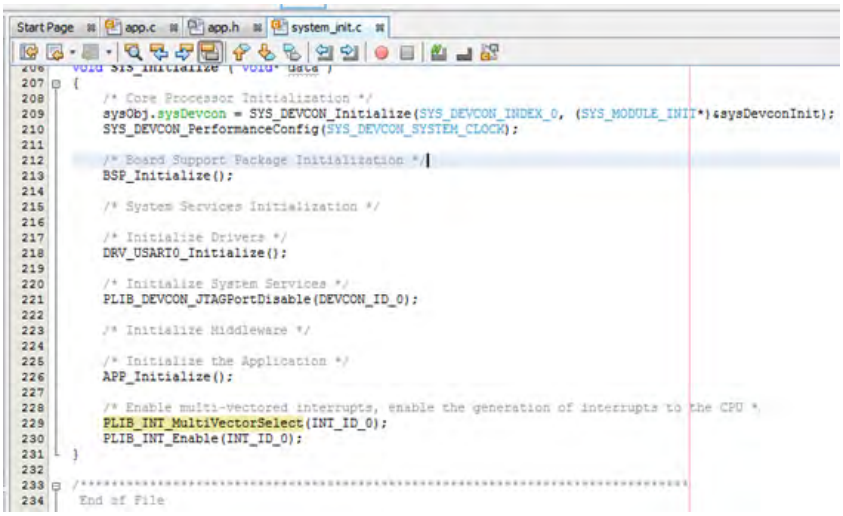
Step 11: At this point in the process, the `system_init.c` file will have functions/code to initialize the device clock and initialize the UART2. The code will be consistent to the settings previously done with MHC in **Step 8**.





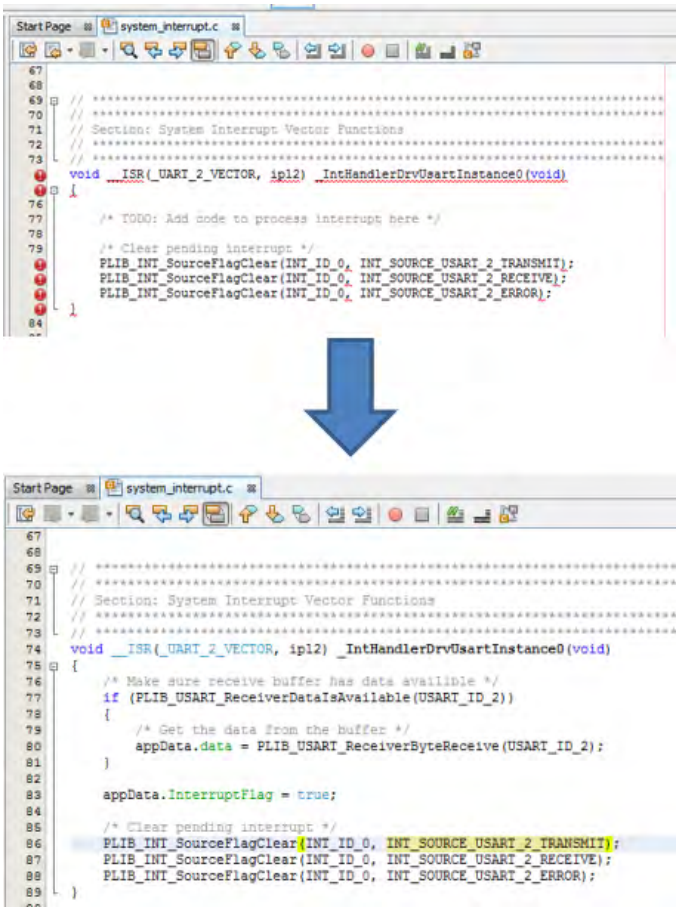
Step 12: Since the LEDs on the Explorer 16 Development Board are controlled by pins of PIC32MX795F512L device, which are shared with the JTAG function, JTAG should be disabled by manually adding the PLIB_DEVCON_JTAGPortDisable function in `system_init.c`. In addition, manually add the functions to set Multi-vector mode (PLIB_INT_MultiVectorSelect) and enable interrupt (PLIB_INT_Enable).

```
/* Initialize System Services */
PLIB_DEVCON_JTAGPortDisable(DEVCON_ID_0);

/* Enable multi-vectored interrupts, enable the generation of interrupts to the CPU */
PLIB_INT_MultiVectorSelect(INT_ID_0);
PLIB_INT_Enable(INT_ID_0);
```



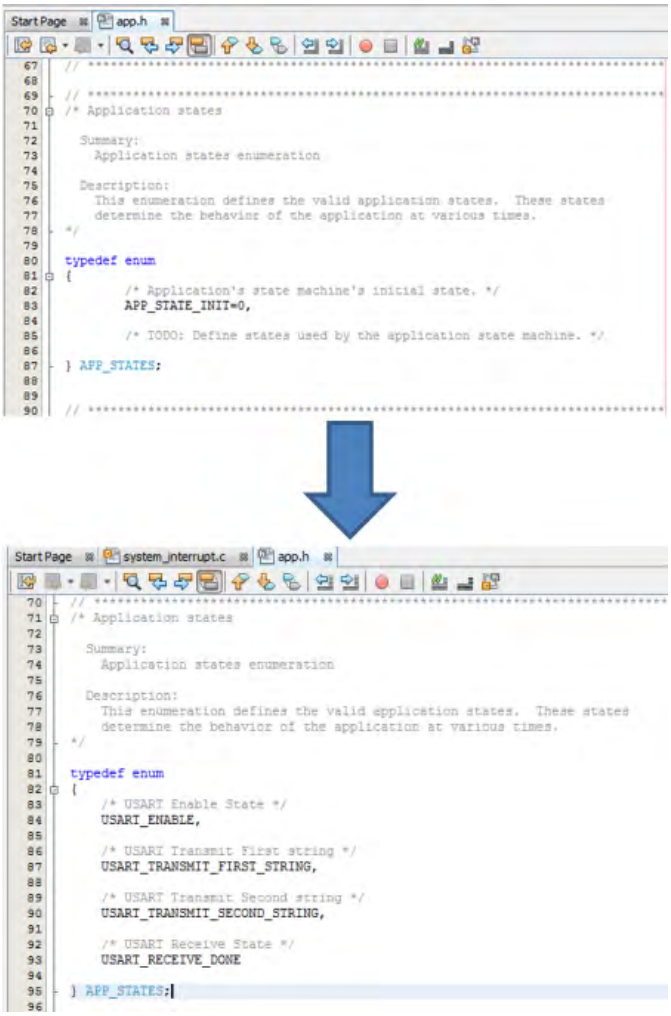
Step 13: In the ISR function of the `system_interrupt.c` file, add code to echo back the received character.

```
    /* Make sure receive buffer has data available */
    if (PLIB_USART_ReceiverDataIsAvailable(USART_ID_2))
    {
        /* Get the data from the buffer */
        appData.data = PLIB_USART_ReceiverByteReceive(USART_ID_2);
    }

    appData.InterruptFlag = true;
;
```



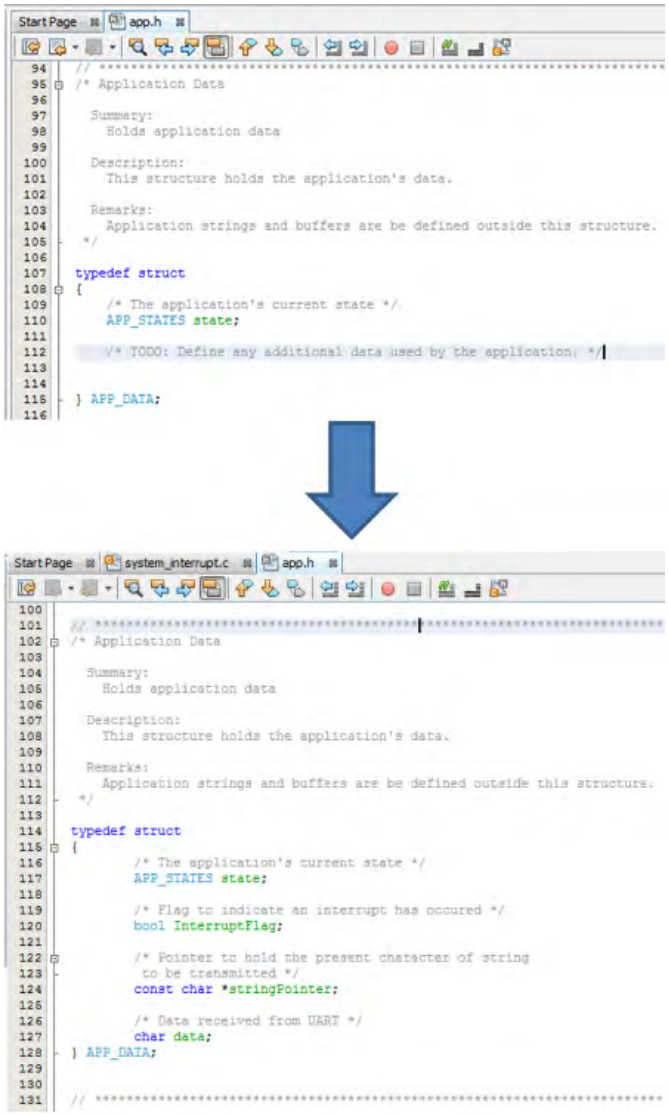
Step 14: In the file `app.h`, define the application-specific members of the APP_STATES enumeration.

```
    /* USART Enable State */
    USART_ENABLE,

    /* USART Transmit First string */
    USART_TRANSMIT_FIRST_STRING,

    /* USART Transmit Second string */
    USART_TRANSMIT_SECOND_STRING,

    /* USART Receive State */
    USART_RECEIVE_DONE
```



Step 15: For APP_DATA structure, define three variables. The first variable is of type Boolean to act as a flag for interrupt indication, the second pointer variable will hold the address of the string that needs to be transmitted to the UART, and the third variable will hold the data received from the UART.

```
const char *stringPointer;
char data;
bool InterruptFlag;
```

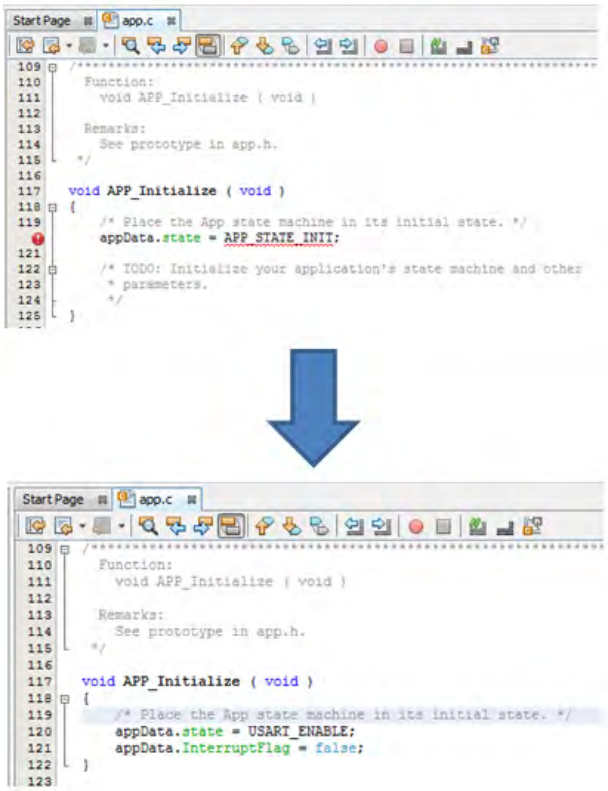


Step 16: Towards the end of the file, insert the prototype declaration for two functions: PutCharacter and WriteString. These functions will later be added to `app.c` (in **Step 20**). Also, declare the `extern APP_DATA appData`, so that the appData variable is available across the project.

```
bool PutCharacter(const char character);

bool WriteString(void);

extern APP_DATA appData;
```

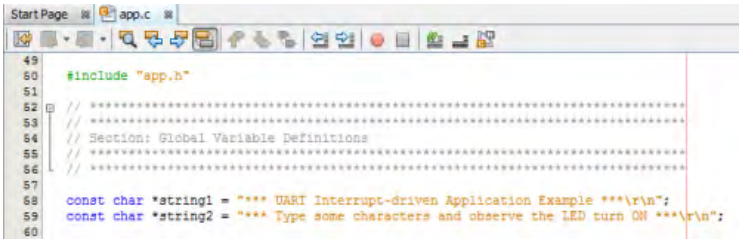
Step 17: In the APP_Initialize of the `app.c` file, add code to set the initial state of application.

```
    /* Place the App state machine in its initial state. */
    appData.state = USART_ENABLE;
    appData.InterruptFlag = false;
```



Step 18: Declare two global strings: string1 and string2.

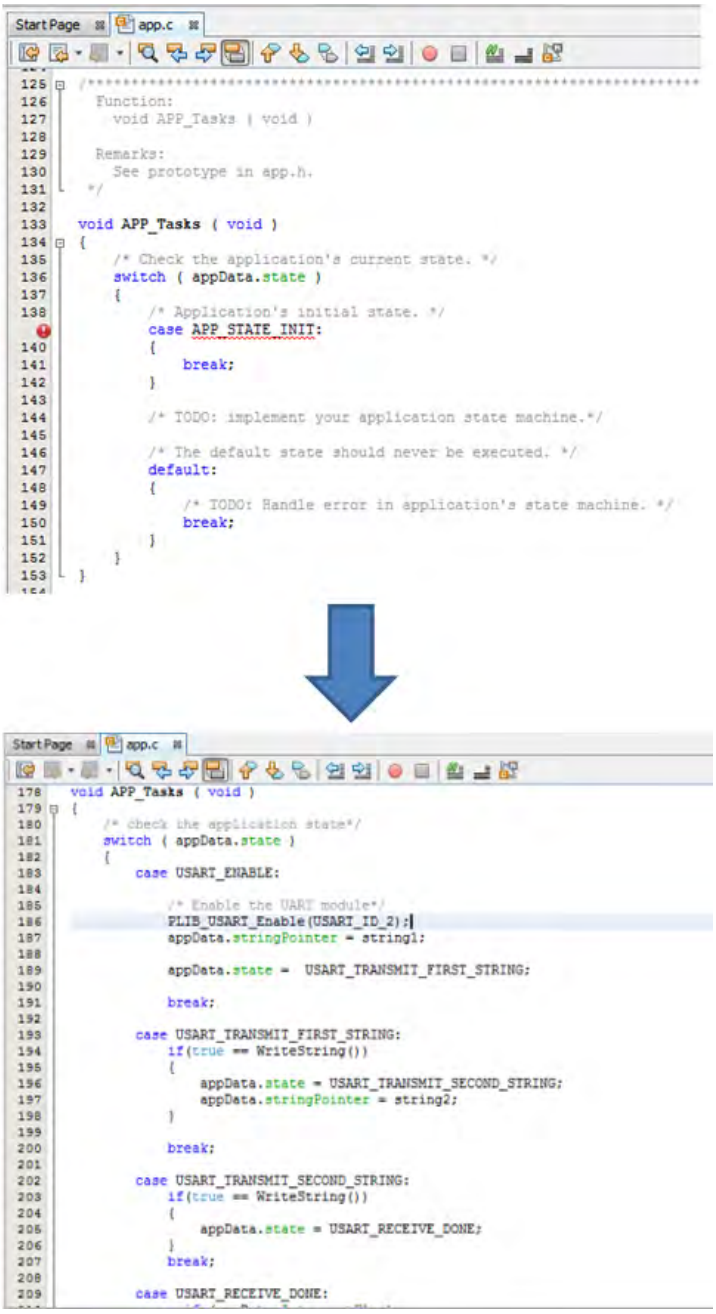
```
const char *string1 = "*** UART Interrupt-driven Application Example ***\r\n";
const char *string2 = "*** Type some characters and observe the LED turn ON ***\r\n";
```



Step 19: Next, add application-specific code to the APP_Tasks function.

```
void APP_Tasks ( void )
{
    /* check the application state*/
    switch ( appData.state )
    {
        case USART_ENABLE:

            /* Enable the UART module*/
            PLIB_USART_Enable(USART_ID_2);
            appData.stringPointer = string1;

            appData.state =  USART_TRANSMIT_FIRST_STRING;

            break;

        case USART_TRANSMIT_FIRST_STRING:
            if(true == WriteString())
            {
                appData.state = USART_TRANSMIT_SECOND_STRING;
                appData.stringPointer = string2;
            }

            break;

        case USART_TRANSMIT_SECOND_STRING:
            if(true == WriteString())
```

```
            {
                appData.state = USART_RECEIVE_DONE;
            }
            break;

        case USART_RECEIVE_DONE:
              if (appData.InterruptFlag)
              {
                  if(true == PutCharacter(appData.data))
                  {
                    BSP_LEDOn(BSP_LED_3);
                    appData.InterruptFlag = false;
                  }
              }
            break;

        default:
            SYS_DEBUG (SYS_ERROR_FATAL,"ERROR! Invalid state\r\n");
            while (1);

    }

}
```



Step 20: Next, add functions for PutCharacter and WriteString.

```
bool WriteString(void)
{
    if(*appData.stringPointer == '\0')
    {
        return true;
    }

    /* Write a character at a time, only if transmitter is empty */
    while (PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, *appData.stringPointer);

        /* Increment to address of next character */
        appData.stringPointer++;

        if(*appData.stringPointer == '\0')
        {
            return true;
```

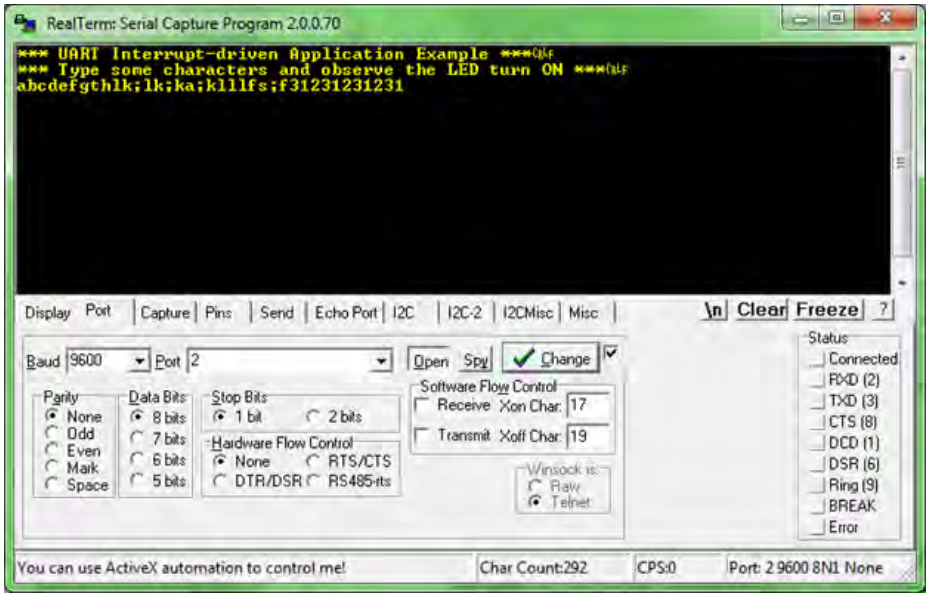
```
        }
    }
    return false;
}

bool PutCharacter(const char character)
{
    /* Check if buffer is empty for a new transmission */
    if(PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, character);
        return true;
    }
    else
        return false;
}
```

```
/*******************************************************************************
  Function:
    bool WriteString (void)

  Summary:
    Writes a string to the console
*/
bool WriteString(void)
{
    if(*appData.stringPointer == '\0')
    {
        return true;
    }

    /* Write a character at a time, only if transmitter is empty */
    while (PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, *appData.stringPointer);

        /* Increment to address of next character */
        appData.stringPointer++;

        if(*appData.stringPointer == '\0')
        {
            return true;
        }
    }
    return false;
}
```

```
/*******************************************************************
  Function:
    bool PutCharacter (const char character)

  Summary:
    Sends a character to the console
*/
bool PutCharacter(const char character)
{
    /* Check if buffer is empty for a new transmission */
    if(PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, character);
        return true;
    }
    else
        return false;
}
```

Step 21: At this point in the process, all of the files have been added to the project with the proper code. Once the project is built, it should build without any errors or warnings.

Step 22: Once the device is programmed and run, a serial cable should be connected to the Explorer 16 Development Board. The appropriate COM port is selected with a baud rate of 9600. As shown in the following RealTerm example, the terminal will echo the typed characters and the LED on the Explorer 16 Development Board will illuminate.



Notes:

1. Refer to the USART Peripheral Library section in the MPLAB Harmony help for a detailed explanation and the example code for the peripheral library functions used in the application.
2. The demonstration code is provided at the same location `<install-dir>/apps/examples/peripheral/usart`.

ADC Peripheral Library

This section describes the Analog-to-Digital Converter (ADC) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Analog-to-Digital Converter (ADC) module, which is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

An ADC is an analog peripheral that converts a continuous quantity to a discrete digital number. An ADC is a signal conversion process that periodically samples and converts a continuously varying signal - analog level into digital values. An ADC might be used to make an isolated measurement. ADCs are also used to quantize time-varying signals by turning them into a sequence of digital samples. This results in the signal being quantized in both time and value. The resolution of a converter indicates the number of discrete values it can produce over the range of analog values.

Using the Library

This topic describes the basic architecture of the ADC Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adc.h`

The interface to the ADC library is defined in the `plib_adc.h` header file, which is included by the `peripheral.h` peripheral library header file. Any C language source (`.c`) file that uses the ADC library must include `peripheral.h`.

Library File:

The ADC peripheral library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using a Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

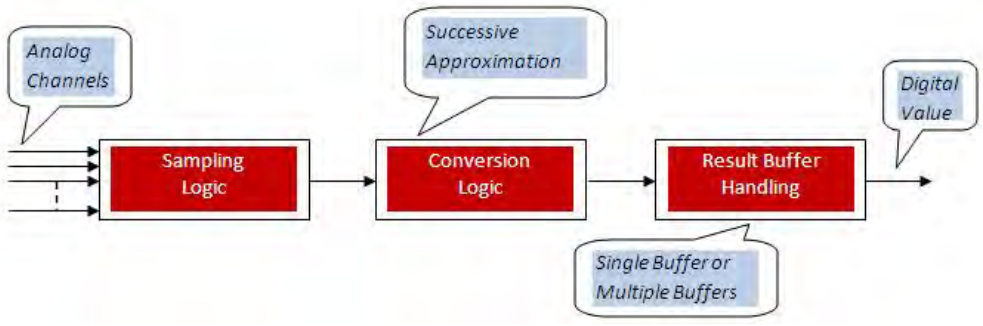
Hardware Abstraction Model

This library provides a low-level abstraction of the ADC module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The ADC module accepts an analog signal at any one instance and converts it to a corresponding 10-bit digital value. It can accommodate a number of analog inputs and separate reference inputs; the actual number available on a particular device depends on the package size.

Hardware Abstraction Block Diagram



A combination of input multiplexers can select the signal to be converted from multiple analog input pins. The entire multiplexer path includes provision for differential analog input, although the number of negative input pins is limited, and the signal difference must remain positive (i.e., unipolar).

Sampling Logic

An internal Sample and Hold (S&H) circuit acquires a sample of an input signal, and then holds that value constant during the conversion process. The purpose of the S&H circuitry is to take a snapshot of the sensor signal and hold the value. The sampled voltage is held and converted to a digital value, which strictly speaking, represents the ratio of that input voltage to a reference voltage. Configuration choices can allow connection of an external reference or use of the device power and ground (AVDD and AVSS).

Conversion Logic

The heart of the ADC is the conversion logic that converts the analog signal value into its equivalent discrete representation. Conversions can be started individually by program control, continuously free-running, or triggered by selected hardware events. A single channel may be repeatedly converted, alternate conversions may be performed on two channels, or any or all of the channels may be sequentially scanned and converted according to a user-defined bit map.

Result Handling

The resulting conversion output is 10-bit digital number in a 16-bit word.

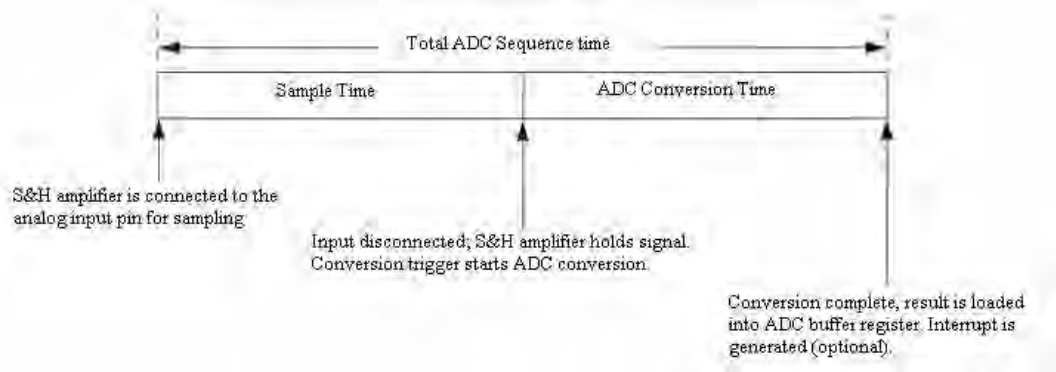
ADC Timing Details

Sample time is the time that the ADC module's S&H circuit is connected to the analog input pin. The sample time may be started and ended automatically by the ADC's hardware or under direct program control. There is a minimum sample time to ensure that the S&H circuit will provide sufficient accuracy for the analog-to-digital conversion.

Conversion time is the time required for the ADC to convert the voltage held by the S&H circuit. The conversion trigger ends the sampling time and begins an analog-to-digital conversion or a repeating sequence. The conversion trigger sources can be taken from a variety of hardware sources or can be controlled directly in software.

Once the conversion is complete, the S&H circuit can be reconnected to the input pin and a CPU interrupt may be generated. The sum of the sample time and the analog-to-digital conversion time provides the total ADC sequence time. The following figure shows the basic conversion sequence and the relationship between intervals.

ADC Sample/Convert Sequence



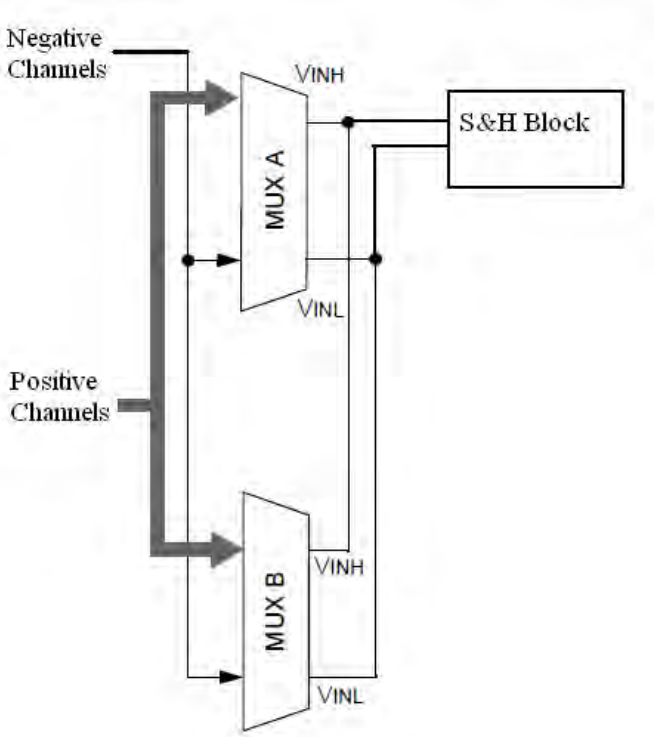
The conversion trigger sources can be taken from a variety of hardware sources, or can be controlled directly by software. One of the conversion trigger options is an auto-conversion, which uses a counter and the ADC clock to set the time between auto-conversions. The Auto-Sample mode and auto-conversion trigger can be used together to provide continuous automatic conversions without software intervention.

A sample/convert sequence that uses multiple S&H channels can be simultaneously sampled or sequentially sampled. Simultaneously sampling multiple signals ensures that the snapshot of the analog inputs occurs at precisely the same time for all inputs. Sequential sampling takes a snapshot of each analog input just before conversion starts on that input. The sampling of multiple inputs is not correlated.

Channel Multiplexers

On some devices, S&H circuits have analog multiplexers on both their non-inverting and inverting inputs to select which analog input(s) are sampled. The ADC of some devices incorporate two independent sets of input multiplexers (MUX A and MUX B), which allow users to choose the analog channels that are to be sampled. Functionally, MUX A and MUX B are very similar to each other. Both multiplexers allow any of the analog input channels to be selected for individual sampling and allow selection of a negative reference source for differential signals. In addition, MUX A can be configured for sequential analog channel scanning. By default the ADC only samples and converts the inputs selected by MUX A. There is also a possibility to alternate between two sets of inputs selected by MUX A and MUX B during successive samples.

MUX Abstraction Model



When using MUX A to select analog inputs, the ADC module has the ability to scan multiple analog channels sequentially.

Input Selection

The ADC module provides a flexible mechanism to select analog inputs for conversion:

- Fixed input selection
- Alternate input selection
- Channel scanning

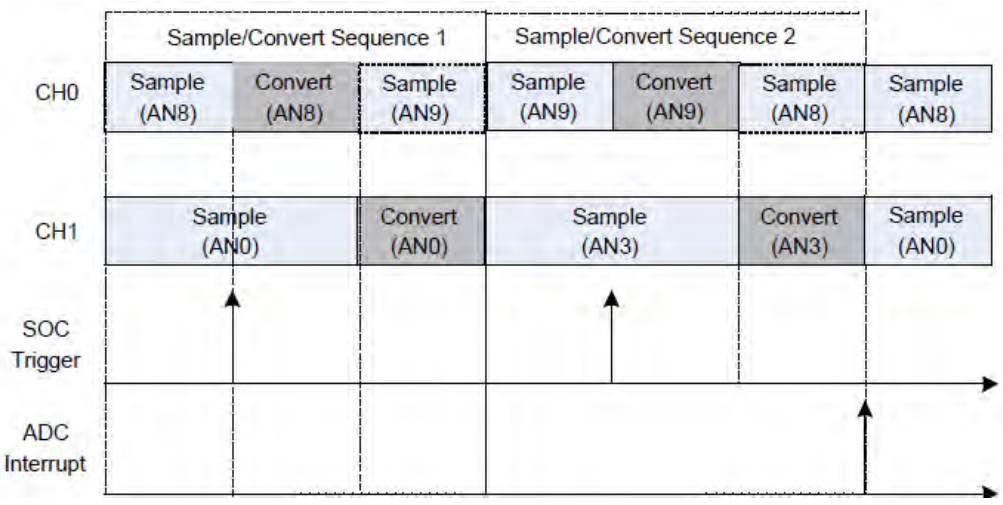
Fixed Input Selection

This is achieved through one or more of the S&H channels available in the device. The S&H channels are connected to the analog input pins through the analog multiplexer.

Alternate Sampling

In an Alternate Input Selection mode, the ADC completes one sweep using the MUX A selection, and then another sweep using the MUX B selection, and then another sweep using the MUX A selection, and so on.

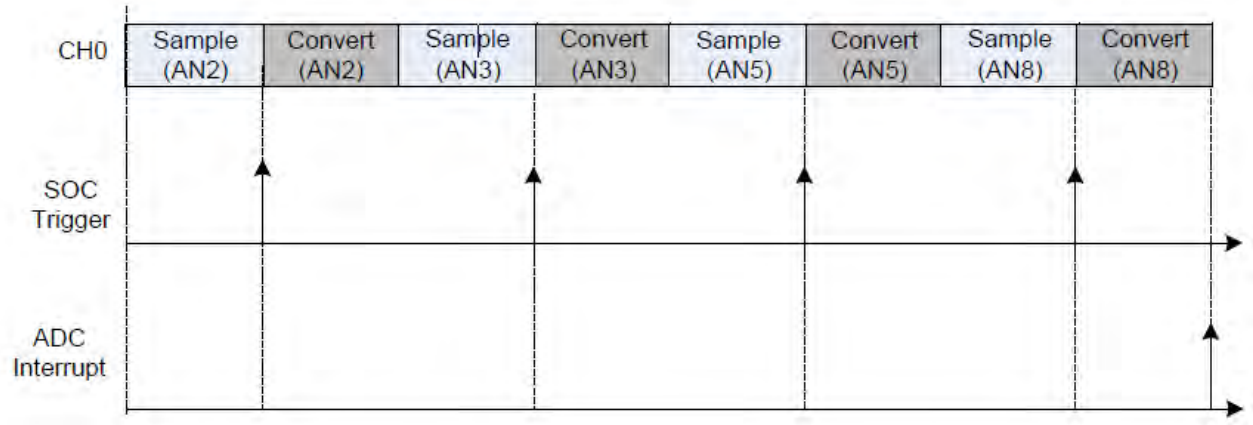
Alternate Input Selection in 2-Channel Sequential Sampling Configuration



Channel Scanning

On some devices, the ADC module supports the Channel Scan mode using S&H Channel 0 (CH0). The number of inputs scanned is software selectable. Any subset of the analog inputs from AN0 to AN31 (depending on the number of analog inputs present on a specific device) can be selected for conversion. The selected inputs are converted in ascending order. For example, if the input selection includes AN4, AN1, and AN3, the conversion sequence is AN1, AN3, and AN4.

Scan Four Analog Inputs Using CH0



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADC Peripheral Library.

Library Interface Section	Description
General Configuration	Interface routines for: • Voltage reference selection (positive/negative) • Channel group selection • Positive/negative channel selection • Add/remove channels for scan • Enabling/disabling the ADC module • Enabling/disabling the calibration • Stop in Idle enable/disable • Internal reference channel enable/disable
MUX Selection and Channel Scan	Interface routines for: • Channel 0 positive/negative input selection for MUX A/MUX B • MUX A scan enable/disable
Sample and Hold Control Logic	Interface routines for: • Sampling start/stop, status • Enabling/disabling of sample auto start • Acquisition/auto-sample time selection • Sample per interrupt selection
Conversion Control Logic	Interface routines for: • Conversion start/status • Conversion clock selection set/get • Conversion clock source selection • Conversion trigger source selection • Conversion stop sequence enable/disable
Output Configuration	Interface routines for: • Result format selection • Result buffer fill status • Result buffer mode select • Result based on the buffer index
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

How the Library Works

The following processes are involved while using an ADC module:

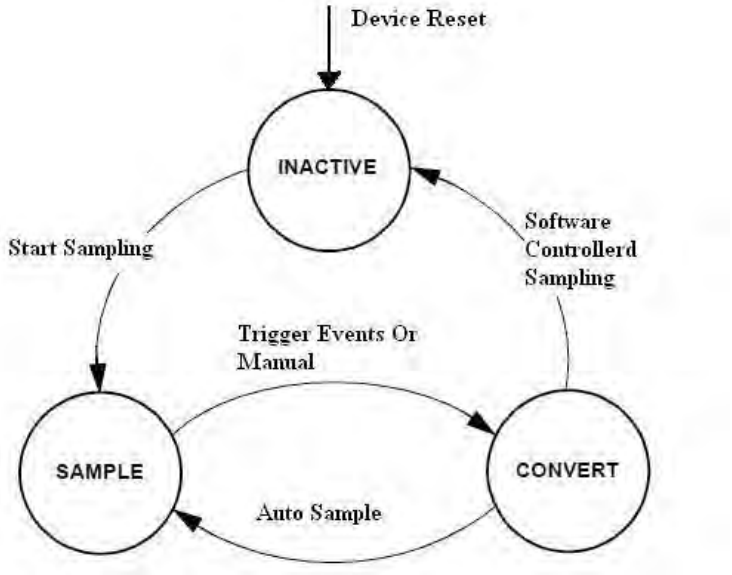
- Initialization
- Controlling the sampling process
- Controlling the conversion process
- Accessing the results buffer

General

The ADC conversion process can be thought of in terms of a finite state machine. The sample state represents the time that the input channel is connected to the S&H circuit and the signal is passed to the converter input. The convert state is transitory; the module enters this state as soon as it exits the sample state and transitions to a different state when that is done. The inactive state is the default state prior to module initialization and following a software-controlled conversion; it can be avoided in operation by using Auto-Sample mode.

Description

ADC Conversion Sequence or State Machine



Initialization

This topic provides information on the different processes needed to perform an analog-to-digital conversion.

Description

The following processes should be followed for performing an analog-to-digital conversion:

Initialize the ADC Module:

Number	Description	Functions associated
1	Selecting the voltage reference source Idle mode control	PLIB_ADC_VoltageReferenceSelect PLIB_ADC_StopInIdleEnable PLIB_ADC_StopInIdleDisable
2	Selecting the ADC conversion clock	PLIB_ADC_ConversionClockSet

3	Input channel selection Configuring MUX A and MUX B inputs, Alternating MUX A and MUX B input selections, Scanning through several inputs	**Scan Mask Selection** PLIB_ADC_InputScanMaskAdd PLIB_ADC_InputScanMaskRemove **Positive Inputs** PLIB_ADC_MuxChannel0InputPositiveSelect **Negative Inputs** PLIB_ADC_MuxChannel0InputNegativeSelect **Scan Mode Selection** PLIB_ADC_MuxAInputScanEnable PLIB_ADC_MuxAInputScanDisable
4	Enabling the ADC module	PLIB_ADC_Enable
5	Determine how sampling will occur	**Sampling Control** PLIB_ADC_SamplingModeSelect PLIB_ADC_SampleAcquisitionTimeSet
6	Selecting Manual or Auto-Sampling	PLIB_ADC_SampleAutoStartEnable PLIB_ADC_SampleAutoStartDisable PLIB_ADC_SamplingStart
7	Select conversion trigger and sampling time	PLIB_ADC_ConversionStart PLIB_ADC_ConversionClockSourceSelect PLIB_ADC_ConversionTriggerSourceSelect PLIB_ADC_ConversionStopSequenceEnable PLIB_ADC_ConversionStopSequenceDisable
8	Select how conversion results are stored in buffer	PLIB_ADC_ResultBufferModeSelect
9	Select the result format	PLIB_ADC_ResultFormatSelect
10	Select the number of readings per interrupt	PLIB_ADC_SamplesPerInterruptSelect

The ADC is configured by the following steps:

1. Select the acquisition time using PLIB_ADC_SampleAcquisitionTimeSet.
2. Select the conversion clock for ADC using PLIB_ADC_ConversionClockSet.
3. The reference for ADC can be set using the function PLIB_ADC_VoltageReferenceSelect.
4. Select the appropriate analog MUX and analog input where the analog voltage is connected.
5. Select the appropriate trigger source using PLIB_ADC_ConversionTriggerSourceSelect.
6. Configure the ADC interrupt (if required):
   - Clear the interrupt status flag
   - Select the ADC interrupt priority
   - Enable ADC interrupt
7. Turn on the ADC module using PLIB_ADC_Enable.

Example Initialization:

```
// Include all channels in scan
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable scanning of channels
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Controlling the Sampling Process

This topic describes the modes and related sampling functionality for the sampling process.

Description

The sampling process can be set up using the following two modes:

Manual Sampling

Calling PLIB_ADC_SamplingStart causes the ADC to begin sampling. One of several options as discussed in "Controlling the Conversion Process" can be used to end sampling and complete conversions. Sampling does not resume until PLIB_ADC_SamplingStart is called again.

Automatic Sampling

Setting the ADC in the Auto-Sampling mode using PLIB_ADC_SampleAutoStartEnable automatically begins sampling a channel whenever a conversion is not active on that channel. One of several options can be used to end sampling and complete conversions, as discussed in "Controlling the Conversion Process". If the simultaneous sampling is selected using PLIB_ADC_SamplingModeSelect with parameter ADC_SAMPLING_MODE_SIMULTANEOUS, sampling on a channel resumes after the conversion of all channels completes.

Other sampling related functionality:

- **Monitoring Sample Status:** PLIB_ADC_SamplingIsActive obtains the status as sampling or holding for the ADC module.
- **Aborting a Sample:** While in Manual Sampling mode, calling PLIB_ADC_SamplingStop will terminate sampling. If the conversion trigger source is selected as ADC_CONVERSION_TRIGGER_SAMP_CLEAR using PLIB_ADC_ConversionTriggerSourceSelect, this causes a conversion to start automatically. While in Auto-Sampling mode, calling PLIB_ADC_SampleAutoStartEnable does not terminate an outgoing sample/convert sequence; however, sampling will not resume after a subsequent conversion.
- **Sampling Modes:** Different sampling modes can be changed using PLIB_ADC_SamplingModeSelect with the appropriate parameter such as ADC_SAMPLING_MODE_ALTERNATE_INPUT for alternate input mode, ADC_SAMPLING_MODE_SIMULTANEOUS for simultaneous sampling mode, or ADC_SAMPLING_MODE_SEQUENTIAL for the sequential sampling mode. There is a possibility to combine the sampling modes say the Alternate input mode with either the simultaneous or the sequential sampling modes.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Controlling the Conversion Process

The conversion trigger source will terminate sampling and start a selected sequence of conversions. It is also possible to obtain the value of the conversion clock, which is obtained by calling PLIB_ADC_ConversionClockGet. 'PLIB_ADC_ConversionTriggerSourceSelect' selects the source of the conversion trigger.

Description

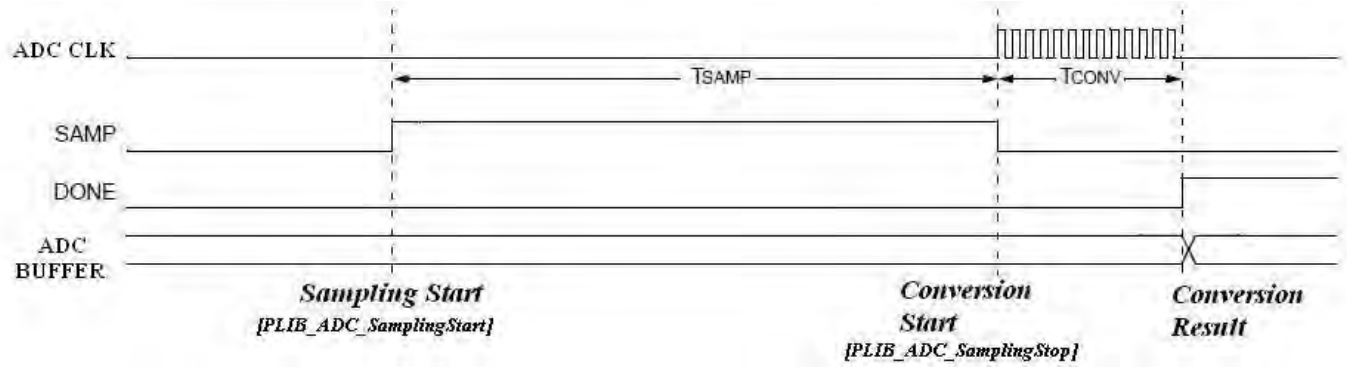
Conversion can be started in one of the following three ways:

Manual Conversion Sequence

Manual Sample Start, Manual Conversion Start

When ADC_CONVERSION_TRIGGER_SAMP_CLEAR is selected using PLIB_ADC_ConversionTriggerSourceSelect, the conversion trigger is under software control. Calls to PLIB_ADC_SamplingStop will stop the sampling and start the conversion sequence. The user must call PLIB_ADC_SamplingStart and PLIB_ADC_SamplingStop in a timed manner, to ensure adequate sampling time.

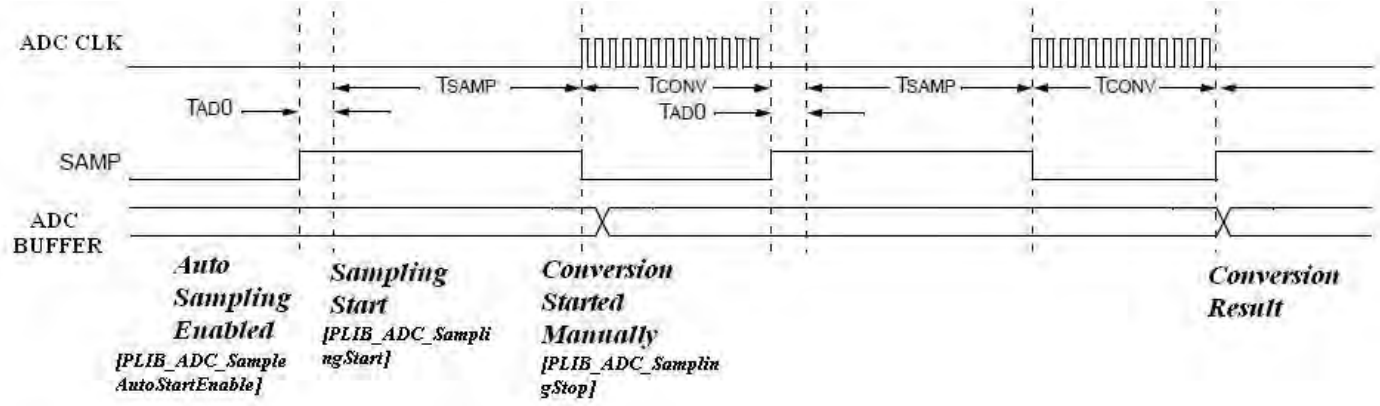
Converting One Channel, Manual Sample Start, Manual Conversion Start



Automatic Sample Start, Manual Conversion Start

Automatic sampling is initiated using PLIB_ADC_SampleAutoStartEnable, calling PLIB_ADC_SamplingStop will terminate sampling and start conversion. After the conversion, the sampling starts again automatically. The user must call PLIB_ADC_SamplingStop in a timed manner, to ensure adequate sampling time. Wait for required acquisition/auto sample time (minimum of 1 TAD), and then call PLIB_ADC_SamplingStop to start the conversion process.

Converting One Channel, Automatic Sample Start, Manual Conversion Start

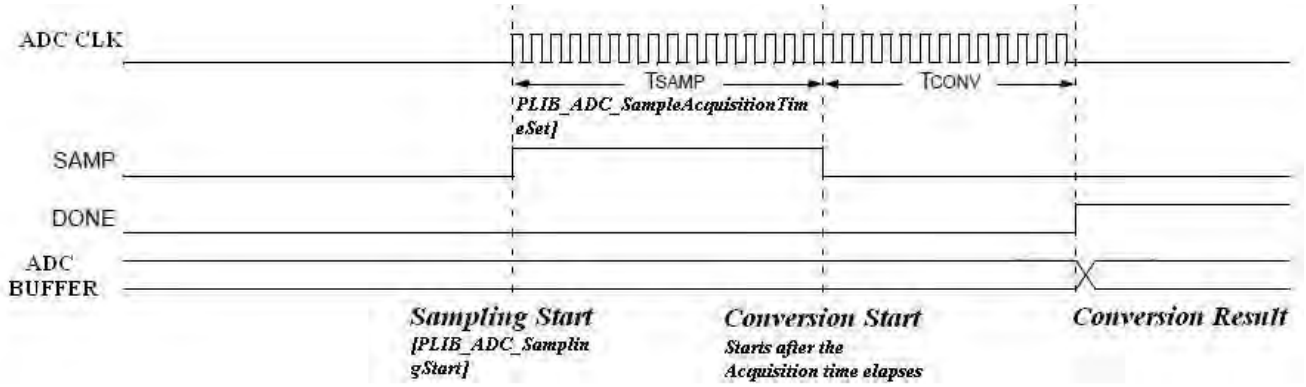


Clocked Conversion Sequence

Manual Sample Start and TAD based Conversion Start

When ADC_CONVERSION_TRIGGER_INTERNAL_COUNT is selected using PLIB_ADC_ConversionTriggerSourceSelect, the conversion trigger is under analog-to-digital clock control. PLIB_ADC_SampleAcquisitionTimeSet selects the TAD clock cycles between the start of sampling and the start of conversion. [Minimum 1 clock cycle has to be selected to ensure the sampling requirements are met]. PLIB_ADC_SamplingStart starts the sampling for the configured acquisition time, and then PLIB_ADC_SamplingStop starts the conversion process.

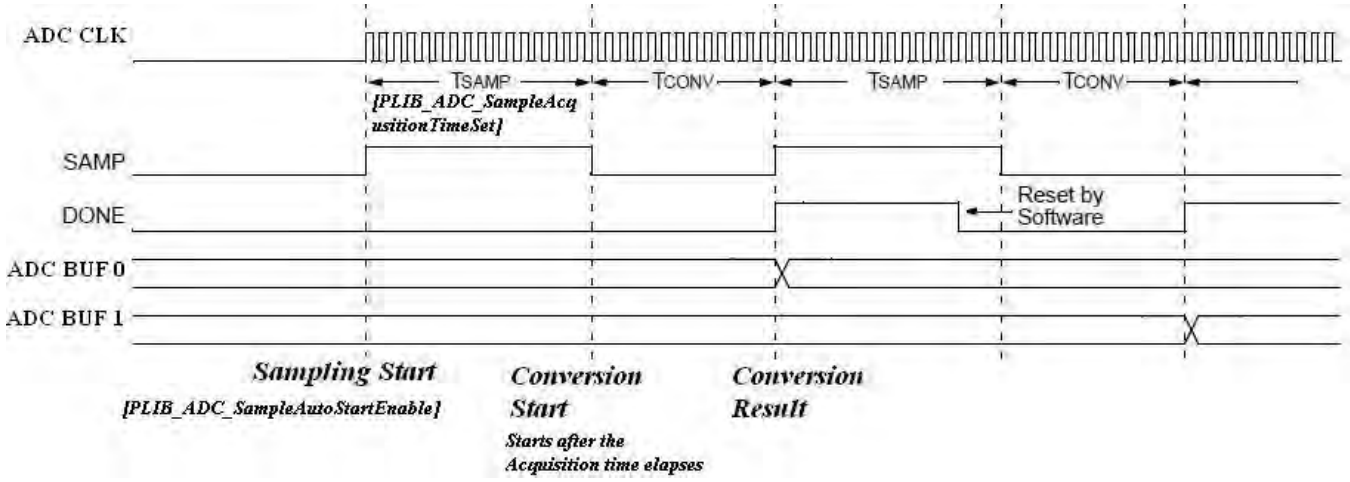
Converting One Channel, Manual Sample Start, TAD-Based Conversion Start



Auto Sample Start and TAD based Conversion Start or Free Running Sample Conversion

With the selection of ADC_CONVERSION_TRIGGER_INTERNAL_COUNT using PLIB_ADC_ConversionTriggerSourceSelect and Automatic sampling initiation using PLIB_ADC_SampleAutoStartEnable allows the ADC module to schedule sample/conversion sequences with no intervention by the user or other device resources.

Converting One Channel, Auto-Sample Start, TAD-Based Conversion Start



Event Trigger Conversion Start

It may be necessary to synchronize the end of sampling and the start of conversion with some other time event. The ADC module may use one of following as conversion trigger sequences:

- **External Pin Trigger:** When ADC_CONVERSION_TRIGGER_INT0_TRANSITION is selected using PLIB_ADC_ConversionTriggerSourceSelect
- **General Purpose Timer Compare Match:** When ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH or ADC_CONVERSION_TRIGGER_TMR5_COMPARE_MATCH is selected using PLIB_ADC_ConversionTriggerSourceSelect

Both of the event trigger conversion modes previously described can be used in combination with auto-sampling (PLIB_ADC_SampleAutoStartEnable) to cause the ADC to synchronize the sample conversion events to the trigger pulse source.

Users should note that some devices have additional conversion trigger sources as part of the enumeration ADC_CONVERSION_TRIGGER_SOURCE.

Other Operations During Conversion Process

- **Monitoring Conversion process:** The status of the conversion can be obtained using PLIB_ADC_ConversionHasCompleted
- **Generating ADC Interrupt:** PLIB_ADC_SamplesPerInterruptSelect controls the generation of the ADC interrupt. To enable the interrupt it is also essential to enable the ADC interrupt.
- **Aborting the Conversion:** Calling PLIB_ADC_Disable will abort the current conversion. The result buffer is not updated with the partially completed ADC conversion sequence.

Timing Details

TAD - The ADC module has a maximum rate at which conversions may be completed. An analog module clock, TAD, controls the conversion timing.

TSAMP - The time required to sample and hold the sampled analog signal, configured through PLIB_ADC_SampleAcquisitionTimeSet.

TCONV - The time required to convert the sampled analog signal, configured through PLIB_ADC_ConversionClockSet. The conversion clock can be verified using PLIB_ADC_ConversionClockGet.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Accessing the Result Buffers

The result buffers can be formatted to the desired format using the function PLIB_ADC_ResultFormatSelect.

Description

As the analog-to-digital conversions are completed, the ADC module writes the results of the conversions into the ADC result buffer. This buffer is a RAM array of 16 words, accessed through the Special Function Register (SFR) space.

User software may attempt to read each ADC conversion result as it is generated; however, this might consume too much CPU time. Generally, to minimize software overhead, the ADC module will fill the buffer with results, and then generate an interrupt when the buffer is filled. There are two different modes for accessing the result buffers.

Note: Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet to determine the correct mode for accessing the result buffer for your device.

Single Buffer Mode

Conversion results are automatically stored in a dedicated buffer. The module sets its interrupt flag after the conversion is complete, it also marks the conversion status as complete. After the interrupt, the conversion sequence can restart. The converted values are available through PLIB_ADC_ResultGetByIndex.

Multiple Buffer Mode

Conversion results are automatically stored in a dedicated position in the result buffer comprising an array of 16 words, allowing for multiple successive readings to be taken before software service is needed. Successive conversions are placed into sequential buffer positions. Alternatively, the buffer can be split into two arrays of 8 words for simultaneous conversion and read operations. The ADC module sets its interrupt flag after a selectable number of conversions, from 1 to 16, when the all buffer positions can be read. After the interrupt, the sequence restarts at the beginning of the buffer. When the interrupt flag is set, scan selections and the output buffer pointer return to their starting positions. The ADC result buffer is a set of 16 words, accessed through PLIB_ADC_ResultGetByIndex , where buffer index can be can be any value ranging from 0 to 15.

PLIB_ADC_SamplesPerInterruptSelect selects how many analog-to-digital conversions will take place before the CPU is interrupted.

The buffer fill mode can be selected using PLIB_ADC_ResultBufferModeSelect with the parameter of the type ADC_BUFFER_MODE.

When the conversion result buffer is split (ADC_BUFFER_MODE_TWO_8WORD_BUFFERS used as the parameter for PLIB_ADC_ResultBufferModeSelect), PLIB_ADC_ResultBufferStatusGet indicates which half of the buffer is being currently written by the ADC module.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Power-Saving Modes

This topic provides information on the power-saving modes available for use with the ADC module.

Description

Operation in Sleep Mode

Operation in Sleep mode requires that the internal RC clock is selected using PLIB_ADC_ConversionClockSourceSelect with the parameter ADC_CLOCK_SOURCE_INTERNAL_RC. If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt. On some microcontrollers, operation in Sleep mode requires that ADC_CONVERSION_CLOCK_FRC is selected using PLIB_ADC_ConversionClockSourceSelect. If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt.

Operation in Idle Mode

PLIB_ADC_StopInIdleEnable and PLIB_ADC_StopInIdleDisable determine if the ADC module stops or continues operation in Idle mode. If PLIB_ADC_StopInIdleDisable is used, the module will continue operation in Idle mode. If the ADC interrupt is enabled, the device will wake up from Idle mode on the ADC interrupt. If PLIB_ADC_StopInIdleEnable is used, the module will stop in Idle mode. If the device enters Idle mode in the middle of conversion, the conversion is aborted.

Operation in other Power-Saving modes

If the ADC module is expected to operate in a power-saving mode, configure the acquisition time and the conversion clock using the functions in accordance with the power-saving mode clock that will be used. After the power-saving mode is entered, an analog-to-digital acquisition can be started. Once the acquisition is started, the device can continue to be clocked by the same power-saving source until the conversion has completed. If desired, the device may be placed in the power-saving Idle mode during conversion.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Conversion Sequence Examples

This topic provides examples on how the sampling and conversion will occur in various configurations.

Description

Converting Single Channel, Manual Sample Start, Manual Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
```

```
// Enabling the sampling manually
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_SAMP_CLEAR);

// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Connect AN2 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);

// Manual Sample and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Converting Single Channel, Manual Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);
// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Converting Single Channel, Automatic Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
```

```
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000,20000000);

// Set the interrupt every 3 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_3SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 2 ADC values
   for(index = 0; index < 2; index++)
   {
      ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 1;
} // Repeat
```

Converting Single Channel, Automatic Sample Start, Conversion Trigger-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;

// General Purpose Timer 3 compare match ends sampling and starts conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSelect(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSelect(MY_ADC_INSTANCE, 1);

// Configure the timer to generate period match as per the acquisition requirements
TMR3  = 0x0000; // set TMR3 to time out every 125 ms
PR3   = 0x3FFF;
T3CON = 0x8010;

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
```

```
{
   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   ADCValue += PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}
```

Sampling and Converting a Single Channel Multiple Times

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 16 ADC value
   for(index = 0; index < 16; index++)
   {
      ADCValue = ADCValue  + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 4;
} // Repeat
```

Sampling and Converting all Channels

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int ADCValue, index;

// Include all channels in scan
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);
```

```
// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable scanning of channels
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 16 ADC value
   for(index = 0; index < 16; index++)
   {
      ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 4;
} // Repeat
```

Wait for Sample, Manual Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

   // Start Sampling
   PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);

   // Is Conversion Done ??
```

```
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Wait for Sample, Triggered Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Manual Sample and Conversion Clock
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
  Delay(); // Ensure, correct sampling time has elapsed

  PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INT0_TRANSITION);

  // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

  ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}
```

Manual Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Sample Time = 1TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 0);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGet(MY_ADC_INSTANCE);
}
```

Configuring the Library

The library is configured for the supported ADC modules with the multi-buffer interface when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration

	Name	Description
	PLIB_ADC_Enable	ADC module is enabled (turned ON).
	PLIB_ADC_Disable	ADC module is disabled (turned OFF).
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.
	PLIB_ADC_CalibrationEnable	Calibration is performed on the next ADC conversion.
	PLIB_ADC_CalibrationDisable	Normal ADC module operation (no calibration is performed).
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskAddExtended	Select extended ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemoveExtended	Omits extended ADC analog channel for input scan.

c) Conversion Control Logic

	Name	Description
	PLIB_ADC_ConversionStart	Starts ADC module manual conversion process.
	PLIB_ADC_ConversionHasCompleted	Provides the conversion completion status of the ADC.
	PLIB_ADC_ConversionTriggerSourceSelect	Selects the conversion trigger source.
	PLIB_ADC_ConversionStopSequenceEnable	Stop conversion sequence (when the first ADC module interrupt is generated).
	PLIB_ADC_ConversionStopSequenceDisable	Normal conversion sequence.
	PLIB_ADC_ConversionClockSet	Sets the ADC module conversion clock.
	PLIB_ADC_ConversionClockGet	Obtains the conversion clock.
	PLIB_ADC_ConversionClockSourceSelect	Selects the ADC module conversion clock source.

d) Sample and Hold Control Logic

	Name	Description
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.

f) Output Configuration

	Name	Description
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.

g) MUX Selection and Channel Scan

	Name	Description
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.

h) Feature Existence Functions

	Name	Description
	PLIB_ADC_ExistsCalibrationControl	Identifies whether the CalibrationControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionClock	Identifies whether the ConversionClock feature exists on the ADC module
	PLIB_ADC_ExistsConversionClockSource	Identifies whether the ConversionClockSource feature exists on the ADC module
	PLIB_ADC_ExistsConversionControl	Identifies whether the ConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionStatus	Identifies whether the ConversionStatus feature exists on the ADC module
	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0NegativeInput	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0PositiveInput	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanControl	Identifies whether the MuxInputScanControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelect	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferFillStatus	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferMode	Identifies whether the ResultBufferMode feature exists on the ADC module
	PLIB_ADC_ExistsResultFormat	Identifies whether the ResultFormat feature exists on the ADC module
	PLIB_ADC_ExistsResultGetByIndex	Identifies whether the ResultGetByIndex feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelectExtended	Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module

i) Data Types & Constants

	Name	Description
	ADC_MODULE_ID	Identifies the available ADC modules.
	ADC_VOLTAGE_REFERENCE	Defining the different ADC Voltage Reference by which the ADC can be configured.
	ADC_INPUTS_NEGATIVE	Defines the different ADC Negative Input Enumeration.
	ADC_SAMPLES_PER_INTERRUPT	Defining the Samples Per Interrupt Enumeration.
	ADC_CLOCK_SOURCE	Defines the ADC Clock Source Select.
	ADC_CONVERSION_TRIGGER_SOURCE	Defines the ADC Conversion Trigger Source.
	ADC_BUFFER_MODE	Defines the ADC Buffer Mode.
	ADC_RESULT_BUF_STATUS	Defines the ADC Result Buffer Status
	ADC_MUX	Defining the different ADC MUX Enumeration.
	ADC_SAMPLING_MODE	Defines the ADC Sampling Mode Select.
	ADC_INPUTS_SCAN	Defines the ADC Scan inputs.
	ADC_RESULT_FORMAT	Defines the ADC Result Format.
	ADC_INPUTS_POSITIVE	Defines the ADC inputs.
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the ADC sample register.

Description

This section describes the Application Programming Interface (API) functions of the Pipelined Analog-to-Digital Converter (ADC) Peripheral Library.
Refer to each section for a detailed description.

a) General Configuration

PLIB_ADC_Enable Function

ADC module is enabled (turned ON).

File

plib_adc.h

C

```
void PLIB_ADC_Enable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_Enable(ADC_MODULE_ID index)

PLIB_ADC_Disable Function

ADC module is disabled (turned OFF).

File

plib_adc.h

C

```
void PLIB_ADC_Disable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_Disable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_Disable(ADC_MODULE_ID index)

PLIB_ADC_StopInIdleDisable Function

Continue ADC module operation when the device is in Idle mode.

File

plib_adc.h

C

```
void PLIB_ADC_StopInIdleDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables the ADC module to continue operation when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_StopInIdleDisable(ADC_MODULE_ID index)

PLIB_ADC_StopInIdleEnable Function

Discontinue ADC module operation when device enters Idle mode.

File

plib_adc.h

C

```
void PLIB_ADC_StopInIdleEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues ADC module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_StopInIdleEnable(ADC_MODULE_ID index)

PLIB_ADC_VoltageReferenceSelect Function

Voltage reference configuration.

File

plib_adc.h

C

```
void PLIB_ADC_VoltageReferenceSelect(ADC_MODULE_ID index, ADC_VOLTAGE_REFERENCE configValue);
```

Returns

None.

Description

This function configures the ADC module voltage reference.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsVoltageReference in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_VoltageReferenceSelect(MY_ADC_INSTANCE, ADC_REFERENCE_VREFPLUS_TO_AVSS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
configValue	One of the possible values from ADC_VOLTAGE_REFERENCE

Function

void PLIB_ADC_VoltageReferenceSelect(ADC_MODULE_ID index,
ADC_VOLTAGE_REFERENCE configValue)

PLIB_ADC_CalibrationEnable Function

Calibration is performed on the next ADC conversion.

File

plib_adc.h

C

```
void PLIB_ADC_CalibrationEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables calibration to be performed on the next ADC conversion.

When the Calibration bit is enabled, inputs are disconnected and tied to AVss. This sets the inputs of the ADC to zero. Then, the user can perform a conversion. Use of the Calibration mode is not affected if the ADC line has been configured as analog or digital, nor by channel input selection. Any analog input switches are disconnected from the ADC module in this mode. The conversion result is stored by the user software and is used to compensate subsequent conversions. This can be done by adding the two's complement of the result obtained during calibration to all normal ADC conversions.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsCalibrationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_CalibrationEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_CalibrationEnable(ADC_MODULE_ID index)

PLIB_ADC_CalibrationDisable Function

Normal ADC module operation (no calibration is performed).

File

plib_adc.h

C

```
void PLIB_ADC_CalibrationDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables normal ADC module operation without any calibration.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsCalibrationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_CalibrationDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_CalibrationDisable(ADC_MODULE_ID index)

PLIB_ADC_InputScanMaskAdd Function

Select ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskAdd(ADC_MODULE_ID index, ADC_INPUTS_SCAN scanInputs);
```

Returns

None.

Description

This function selects the ADC analog channel for input scanning.

Remarks

Multiple channels can be added simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Single channel addition
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);
// Multiple channels addition
PLIB_ADC_InputScanMaskAdd(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN. Inputs are added for scanning.

Function

void PLIB_ADC_InputScanMaskAdd(ADC_MODULE_ID index,
ADC_INPUTS_SCAN scanInputs)

PLIB_ADC_InputScanMaskRemove Function

Omits ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index, ADC_INPUTS_SCAN scanInputs);
```

Returns

None.

Description

This function allows the ADC analog channel to be omitted from input scanning.

Remarks

Multiple channels can be removed simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Single channel removing
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);
// Multiple channels removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN. Inputs are removed from scanning.

Function

void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index,
ADC_INPUTS_SCAN scanInputs)

PLIB_ADC_InputScanMaskAddExtended Function

Select extended ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskAddExtended(ADC_MODULE_ID index, ADC_INPUTS_SCAN_EXTENDED scanInputs);
```

Returns

None.

Description

This function selects the extended ADC analog channel for input scanning.

Remarks

Multiple channels can be added simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelectExtended in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Single channel addition
PLIB_ADC_InputScanMaskAddExtended(ADC_ID_1, ADC_INPUT_SCAN_AN36);
// Multiple channels addition
PLIB_ADC_InputScanMaskAddExtended(ADC_ID_1, ADC_INPUT_SCAN_AN36 | ADC_INPUT_SCAN_AN39);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN_EXTENDED. Inputs are added for scanning.

Function

void PLIB_ADC_InputScanMaskAddExtended(ADC_MODULE_ID index,

ADC_INPUTS_SCAN_EXTENDED scanInputs)

PLIB_ADC_InputScanMaskRemoveExtended Function

Omits extended ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskRemoveExtended(ADC_MODULE_ID index, ADC_INPUTS_SCAN_EXTENDED scanInputs);
```

Returns

None.

Description

This function allows the extended ADC analog channel to be omitted from input scanning.

Remarks

Multiple channels can be removed simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelectExtended in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Single channel removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN36);
// Multiple channels removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN36 | ADC_INPUT_SCAN_AN39);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN_EXTENDED. Inputs are removed from scanning.

Function

void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index,

ADC_INPUTS_SCAN_EXTENDED scanInputs)

b) DMA Transactions

c) Conversion Control Logic

PLIB_ADC_ConversionStart Function

Starts ADC module manual conversion process.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStart(ADC_MODULE_ID index);
```

Returns

None.

Description

This function starts the ADC module manual conversion process.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStart(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStart(ADC_MODULE_ID index)

PLIB_ADC_ConversionHasCompleted Function

Provides the conversion completion status of the ADC.

File

plib_adc.h

C

```
bool PLIB_ADC_ConversionHasCompleted(ADC_MODULE_ID index);
```

Returns

Boolean:

- true - ADC conversion is done/completed
- false - ADC conversion is in progress or has not started

Description

This function provides the completion status of analog-to-digital conversion.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_ADC_ConversionHasCompleted(ADC_MODULE_ID index)

PLIB_ADC_ConversionTriggerSourceSelect Function

Selects the conversion trigger source.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionTriggerSourceSelect(ADC_MODULE_ID index, ADC_CONVERSION_TRIGGER_SOURCE source);
```

Returns

None.

Description

This function selects the ADC module conversion trigger source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionTriggerSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CONVERSION_TRIGGER_SOURCE

Function

void PLIB_ADC_ConversionTriggerSourceSelect(ADC_MODULE_ID index,
ADC_CONVERSION_TRIGGER_SOURCE source)

PLIB_ADC_ConversionStopSequenceEnable Function

Stop conversion sequence (when the first ADC module interrupt is generated).

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStopSequenceEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function stops conversions when the first ADC module interrupt is generated. Hardware clears the Automatic Sampling bit when the ADC interrupt is generated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStopSequenceControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStopSequenceEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStopSequenceEnable(ADC_MODULE_ID index)

PLIB_ADC_ConversionStopSequenceDisable Function

Normal conversion sequence.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStopSequenceDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables normal operation, wherein the buffer contents will be overwritten by the next conversion sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStopSequenceControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStopSequenceDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStopSequenceDisable(ADC_MODULE_ID index)

PLIB_ADC_ConversionClockSet Function

Sets the ADC module conversion clock.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionClockSet(ADC_MODULE_ID index, uint32_t baseFrequency, ADC_CONVERSION_CLOCK value);
```

Returns

None.

Description

This function sets the ADC module conversion clock prescaler.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Following functions passed the input clock to ADC as 80MHz and
// required conversion clock as 20MHz.
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseFrequency	Input clock frequency in Hertz (Hz). This is the input clock to ADC module.
value	Unsigned value of type ADC_CONVERSION_CLOCK. This is the required conversion clock of ADC module in Hz.

Function

void PLIB_ADC_ConversionClockSet(ADC_MODULE_ID index,

uint32_t baseFrequency,

ADC_CONVERSION_CLOCK value)

PLIB_ADC_ConversionClockGet Function

Obtains the conversion clock.

File

plib_adc.h

C

```
ADC_CONVERSION_CLOCK PLIB_ADC_ConversionClockGet(ADC_MODULE_ID index, uint32_t baseFrequency);
```

Returns

ADC_CONVERSION_CLOCK - ADC Conversion clock value (in Hz) of type ADC_CONVERSION_CLOCK

Description

This function obtains the conversion clock that is being used by the ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// baseFrequency is the peripheral input frequency
ADC_CONVERSION_CLOCK conversionClock; // To store the conversion clock value
conversionClock = PLIB_ADC_ConversionClockGet(MY_ADC_INSTANCE, 8000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseFrequencyHz	Input clock frequency to the ADC module in Hz

Function

ADC_CONVERSION_CLOCK PLIB_ADC_ConversionClockGet(ADC_MODULE_ID index,
uint32_t baseFrequencyHz)

PLIB_ADC_ConversionClockSourceSelect Function

Selects the ADC module conversion clock source.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionClockSourceSelect(ADC_MODULE_ID index, ADC_CLOCK_SOURCE source);
```

Returns

None.

Description

This function selects the ADC module conversion clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionClockSourceSelect(MY_ADC_INSTANCE, ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CLOCK_SOURCE

Function

void PLIB_ADC_ConversionClockSourceSelect(ADC_MODULE_ID index,
ADC_CLOCK_SOURCE source)

d) Sample and Hold Control Logic

PLIB_ADC_SampleAutoStartDisable Function

Sampling auto-start is disabled.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAutoStartDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables auto-sampling and enables manual sampling.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAutoStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SampleAutoStartDisable(ADC_MODULE_ID index)

PLIB_ADC_SampleAutoStartEnable Function

Sampling auto-start is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAutoStartEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables auto-sampling. Sampling begins immediately after the last conversion is completed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAutoStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SampleAutoStartEnable(ADC_MODULE_ID index)

PLIB_ADC_SamplesPerInterruptSelect Function

Interrupts at the completion of conversion for each nth sample.

File

plib_adc.h

C

```
void PLIB_ADC_SamplesPerInterruptSelect(ADC_MODULE_ID index, ADC_SAMPLES_PER_INTERRUPT value);
```

Returns

None.

Description

This function interrupts at the completion of conversion for each nth sample/convert sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplesPerInterruptSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Possible values from ADC_SAMPLES_PER_INTERRUPT

Function

void PLIB_ADC_SamplesPerInterruptSelect(ADC_MODULE_ID index,
ADC_SAMPLES_PER_INTERRUPT value)

PLIB_ADC_SamplingIsActive Function

Provides the ADC sampling status.

File

plib_adc.h

C

```
bool PLIB_ADC_SamplingIsActive(ADC_MODULE_ID index);
```

Returns

Boolean:

- true - ADC Sample and Hold circuit is sampling
- false - ADC Sample and Hold circuit is holding

Description

This function returns the ADC sampling status on whether the ADC is sampling or holding.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_SamplingIsActive(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_ADC_SamplingIsActive(ADC_MODULE_ID index)

PLIB_ADC_SamplingModeSelect Function

Enable the selected sampling mode.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingModeSelect(ADC_MODULE_ID index, ADC_SAMPLING_MODE mode);
```

Returns

None.

Description

This function selects the sampling mode.

Remarks

Sampling mode could be alternate input or Simultaneous or Sequential mode. Alternate input can be combined with Simultaneous or Sequential modes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingModeSelect(MY_ADC_INSTANCE, ADC_SAMPLING_MODE_ALTERNATE_INPUT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_SAMPLING_MODE

Function

void PLIB_ADC_SamplingModeSelect(ADC_MODULE_ID index, ADC_SAMPLING_MODE mode)

PLIB_ADC_SamplingStart Function

Sampling is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingStart(ADC_MODULE_ID index);
```

Returns

None.

Description

This function starts the ADC Sample and Hold circuit to sample the input channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SamplingStart(ADC_MODULE_ID index)

PLIB_ADC_SamplingStop Function

Holding is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingStop(ADC_MODULE_ID index);
```

Returns

None.

Description

This function stops the ADC Sample and Hold circuit from sampling and holds the sampled data.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
```

```
PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SamplingStop(ADC_MODULE_ID index)

PLIB_ADC_SampleAcquisitionTimeSet Function

Sets the ADC acquisition/auto-sample time in TADs.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAcquisitionTimeSet(ADC_MODULE_ID index, ADC_ACQUISITION_TIME acqTime);
```

Returns

None.

Description

This function sets the ADC acquisition/auto-sample time in TADs.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAcquisitionTime in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
acqTime	Unsigned value of type ADC_ACQUISITION_TIME

Function

void PLIB_ADC_SampleAcquisitionTimeSet(ADC_MODULE_ID index,
ADC_ACQUISITION_TIME acqTime)

e) Channel Pairs Control

f) Output Configuration

PLIB_ADC_ResultBufferModeSelect Function

Selects the result buffer mode.

File

plib_adc.h

C

```
void PLIB_ADC_ResultBufferModeSelect(ADC_MODULE_ID index, ADC_BUFFER_MODE mode);
```

Returns

None.

Description

This function selects the result buffer mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultBufferMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultBufferModeSelect(MY_ADC_INSTANCE,
                                ADC_BUFFER_MODE_TWO_8WORD_BUFFERS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_BUFFER_MODE

Function

void PLIB_ADC_ResultBufferModeSelect(ADC_MODULE_ID index,
ADC_BUFFER_MODE mode)

PLIB_ADC_ResultBufferStatusGet Function

Provides the buffer fill status.

File

plib_adc.h

C

```
ADC_RESULT_BUF_STATUS PLIB_ADC_ResultBufferStatusGet(ADC_MODULE_ID index);
```

Returns

Boolean:

- true = ADC is currently filling buffer 08-0F, user should access data in 00-07
- false = ADC is currently filling buffer 00-07, user should access data in 08-0F

Description

This function obtains the buffer fill status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultBufferFillStatus in your application to determine whether this feature is available.

Preconditions

ADC multi-buffer support is available and configured.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_RESULT_BUF_STATUS my_status;

my_status = PLIB_ADC_ResultBufferStatusGet(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

ADC_RESULT_BUF_STATUS PLIB_ADC_ResultBufferStatusGet(ADC_MODULE_ID index)

PLIB_ADC_ResultFormatSelect Function

Selects the result format.

File

plib_adc.h

C

```
void PLIB_ADC_ResultFormatSelect(ADC_MODULE_ID index, ADC_RESULT_FORMAT resultFormat);
```

Returns

None.

Description

This function selects the result format.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultFormat in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultFormatSelect(MY_ADC_INSTANCE, ADC_RESULT_FORMAT_INTEGER_16BIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
resultFormat	One of the possible values from ADC_RESULT_FORMAT

Function

void PLIB_ADC_ResultFormatSelect(ADC_MODULE_ID index,
ADC_RESULT_FORMAT resultFormat)

PLIB_ADC_ResultGetByIndex Function

Provides the ADC conversion result based on the buffer index.

File

plib_adc.h

C

```
ADC_SAMPLE PLIB_ADC_ResultGetByIndex(ADC_MODULE_ID index, uint8_t bufferIndex);
```

Returns

int16_t - ADC Conversion result at the respective bufferIndex

Description

This function provides the ADC module conversion result based on the buffer index.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultGetByIndex in your application to determine whether this feature is available.

Preconditions

ADC multi-buffer support available and configured.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_SAMPLE my_res = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 15);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bufferIndex	Value ranging from 0 to 15

Function

ADC_SAMPLE PLIB_ADC_ResultGetByIndex(ADC_MODULE_ID index,
uint8_t bufferIndex)

g) MUX Selection and Channel Scan

PLIB_ADC_MuxAInputScanDisable Function

Do not scan input selections for CH0+ of MUX A.

File

plib_adc.h

C

```
void PLIB_ADC_MuxAInputScanDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables scan input for CH0+ of MUX A.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxAInputScanDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_MuxAInputScanDisable(ADC_MODULE_ID index)

PLIB_ADC_MuxAInputScanEnable Function

Scan input selections for CH0+ of MUX A.

File

plib_adc.h

C

```
void PLIB_ADC_MuxAInputScanEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables scan input for CH0+ of MUX A.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_MuxAInputScanEnable(ADC_MODULE_ID index)

PLIB_ADC_MuxChannel0InputNegativeSelect Function

Channel 0 negative input select for multiplexer setting.

File

plib_adc.h

C

```
void PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_MODULE_ID index, ADC_MUX muxType, ADC_INPUTS_NEGATIVE
input);
```

Returns

None.

Description

This function selects the negative input for channel 0 of MUX A or MUX B.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxChannel0NegativeInput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
```

```
PLIB_ADC_MuxChannel0InputNegativeSelect(MY_ADC_INSTANCE, ADC_MUX_A, ADC_INPUT_NEGATIVE_VREF_MINUS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Function

void PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_MODULE_ID index,
ADC_MUX muxType,
ADC_INPUTS_NEGATIVE input)

PLIB_ADC_MuxChannel0InputPositiveSelect Function

Channel 0 positive input select for multiplexer setting.

File

plib_adc.h

C

```
void PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_MODULE_ID index, ADC_MUX muxType, ADC_INPUTS_POSITIVE
input);
```

Returns

None.

Description

This function selects the positive input for channel 0 of MUX A or MUX B.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxChannel0PositiveInput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxChannel0InputPositiveSelect(MY_ADC_INSTANCE, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Function

void PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_MODULE_ID index,
ADC_MUX muxType,
ADC_INPUTS_POSITIVE input)

h) Feature Existence Functions

PLIB_ADC_ExistsCalibrationControl Function

Identifies whether the CalibrationControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsCalibrationControl(ADC_MODULE_ID index);
```

Returns

- true - The CalibrationControl feature is supported on the device
- false - The CalibrationControl feature is not supported on the device

Description

This function identifies whether the CalibrationControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_CalibrationEnable
- PLIB_ADC_CalibrationDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsCalibrationControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionClock Function

Identifies whether the ConversionClock feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionClock(ADC_MODULE_ID index);
```

Returns

- true - The ConversionClock feature is supported on the device
- false - The ConversionClock feature is not supported on the device

Description

This function identifies whether the ConversionClock feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_ConversionClockSet
- PLIB_ADC_ConversionClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionClock(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionClockSource Function

Identifies whether the ConversionClockSource feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionClockSource(ADC_MODULE_ID index);
```

Returns

- true - The ConversionClockSource feature is supported on the device
- false - The ConversionClockSource feature is not supported on the device

Description

This function identifies whether the ConversionClockSource feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionClockSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionClockSource(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionControl Function

Identifies whether the ConversionControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionControl(ADC_MODULE_ID index);
```

Returns

- true - The ConversionControl feature is supported on the device
- false - The ConversionControl feature is not supported on the device

Description

This function identifies whether the ConversionControl feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionStatus Function

Identifies whether the ConversionStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionStatus(ADC_MODULE_ID index);
```

Returns

- true - The ConversionStatus feature is supported on the device
- false - The ConversionStatus feature is not supported on the device

Description

This function identifies whether the ConversionStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionStopSequenceControl Function

Identifies whether the ConversionStopSequenceControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionStopSequenceControl(ADC_MODULE_ID index);
```

Returns

- true - The ConversionStopSequenceControl feature is supported on the device
- false - The ConversionStopSequenceControl feature is not supported on the device

Description

This function identifies whether the ConversionStopSequenceControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_ConversionStopSequenceEnable
- PLIB_ADC_ConversionStopSequenceDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionStopSequenceControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionTriggerSource Function

Identifies whether the ConversionTriggerSource feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionTriggerSource(ADC_MODULE_ID index);
```

Returns

- true - The ConversionTriggerSource feature is supported on the device
- false - The ConversionTriggerSource feature is not supported on the device

Description

This function identifies whether the ConversionTriggerSource feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionTriggerSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionTriggerSource(ADC_MODULE_ID index)

PLIB_ADC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsEnableControl(ADC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_Enable
- PLIB_ADC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsEnableControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxChannel0NegativeInput Function

Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxChannel0NegativeInput(ADC_MODULE_ID index);
```

Returns

- true - The MuxChannel0NegativeInput feature is supported on the device
- false - The MuxChannel0NegativeInput feature is not supported on the device

Description

This function identifies whether the MuxChannel0NegativeInput feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_MuxChannel0InputNegativeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxChannel0NegativeInput(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxChannel0PositiveInput Function

Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxChannel0PositiveInput(ADC_MODULE_ID index);
```

Returns

- true - The MuxChannel0PositiveInput feature is supported on the device
- false - The MuxChannel0PositiveInput feature is not supported on the device

Description

This function identifies whether the MuxChannel0PositiveInput feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_MuxChannel0InputPositiveSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxChannel0PositiveInput(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanControl Function

Identifies whether the MuxInputScanControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanControl(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanControl feature is supported on the device
- false - The MuxInputScanControl feature is not supported on the device

Description

This function identifies whether the MuxInputScanControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_MuxAInputScanEnable
- PLIB_ADC_MuxAInputScanDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanSelect Function

Identifies whether the MuxInputScanSelect feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanSelect(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanSelect feature is supported on the device
- false - The MuxInputScanSelect feature is not supported on the device

Description

This function identifies whether the MuxInputScanSelect feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_InputScanMaskAdd
- PLIB_ADC_InputScanMaskRemove

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanSelect(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultBufferFillStatus Function

Identifies whether the ResultBufferFillStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultBufferFillStatus(ADC_MODULE_ID index);
```

Returns

- true - The ResultBufferFillStatus feature is supported on the device
- false - The ResultBufferFillStatus feature is not supported on the device

Description

This function identifies whether the ResultBufferFillStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultBufferStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultBufferFillStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultBufferMode Function

Identifies whether the ResultBufferMode feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultBufferMode(ADC_MODULE_ID index);
```

Returns

- true - The ResultBufferMode feature is supported on the device
- false - The ResultBufferMode feature is not supported on the device

Description

This function identifies whether the ResultBufferMode feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultBufferModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultBufferMode(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultFormat Function

Identifies whether the ResultFormat feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultFormat(ADC_MODULE_ID index);
```

Returns

- true - The ResultFormat feature is supported on the device
- false - The ResultFormat feature is not supported on the device

Description

This function identifies whether the ResultFormat feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultFormatSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultFormat(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultGetByIndex Function

Identifies whether the ResultGetByIndex feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultGetByIndex(ADC_MODULE_ID index);
```

Returns

- true - The ResultGetByIndex feature is supported on the device
- false - The ResultGetByIndex feature is not supported on the device

Description

This function identifies whether the ResultGetByIndex feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultGetByIndex

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultGetByIndex(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplesPerInterruptSelect Function

Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplesPerInterruptSelect(ADC_MODULE_ID index);
```

Returns

- true - The SamplesPerInterruptSelect feature is supported on the device
- false - The SamplesPerInterruptSelect feature is not supported on the device

Description

This function identifies whether the SamplesPerInterruptSelect feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplesPerInterruptSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplesPerInterruptSelect(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingAcquisitionTime Function

Identifies whether the SamplingAcquisitionTime feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingAcquisitionTime(ADC_MODULE_ID index);
```

Returns

- true - The SamplingAcquisitionTime feature is supported on the device
- false - The SamplingAcquisitionTime feature is not supported on the device

Description

This function identifies whether the SamplingAcquisitionTime feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SampleAcquisitionTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingAcquisitionTime(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingAutoStart Function

Identifies whether the SamplingAutoStart feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingAutoStart(ADC_MODULE_ID index);
```

Returns

- true - The SamplingAutoStart feature is supported on the device
- false - The SamplingAutoStart feature is not supported on the device

Description

This function identifies whether the SamplingAutoStart feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_SampleAutoStartEnable
- PLIB_ADC_SampleAutoStartDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingAutoStart(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingControl Function

Identifies whether the SamplingControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingControl(ADC_MODULE_ID index);
```

Returns

- true - The SamplingControl feature is supported on the device
- false - The SamplingControl feature is not supported on the device

Description

This function identifies whether the SamplingControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_SamplingStart
- PLIB_ADC_SamplingStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingModeControl Function

Identifies whether the SamplingModeControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingModeControl(ADC_MODULE_ID index);
```

Returns

- true - The SamplingModeControl feature is supported on the device
- false - The SamplingModeControl feature is not supported on the device

Description

This function identifies whether the SamplingModeControl feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplingModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingModeControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingStatus Function

Identifies whether the SamplingStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingStatus(ADC_MODULE_ID index);
```

Returns

- true - The SamplingStatus feature is supported on the device
- false - The SamplingStatus feature is not supported on the device

Description

This function identifies whether the SamplingStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplingIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsStopInIdleControl(ADC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_StopInIdleEnable
- PLIB_ADC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsStopInIdleControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsVoltageReference Function

Identifies whether the VoltageReference feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsVoltageReference(ADC_MODULE_ID index);
```

Returns

- true - The VoltageReference feature is supported on the device
- false - The VoltageReference feature is not supported on the device

Description

This function identifies whether the VoltageReference feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_VoltageReferenceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsVoltageReference(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanSelectExtended Function

Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanSelectExtended(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanSelectExtended feature is supported on the device
- false - The MuxInputScanSelectExtended feature is not supported on the device

Description

This function identifies whether the MuxInputScanSelectExtended feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_InputScanMaskAddExtended
- PLIB_ADC_InputScanMaskRemoveExtended

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanSelectExtended(ADC_MODULE_ID index)

i) Data Types & Constants

ADC_MODULE_ID Enumeration

Identifies the available ADC modules.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_ID_1,
  ADC_ID_2,
  ADC_ID_3,
  ADC_ID_4,
  ADC_ID_5,
  ADC_NUMBER_OF_MODULES
} ADC_MODULE_ID;
```

Members

Members	Description
ADC_ID_1	ADC Module 1 ID
ADC_ID_2	ADC Module 2 ID
ADC_ID_3	ADC Module 3 ID
ADC_ID_4	ADC Module 4 ID
ADC_ID_5	ADC Module 5 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Description

ADC Module ID

This enumeration identifies the available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADC_VOLTAGE_REFERENCE Enumeration

Defining the different ADC Voltage Reference by which the ADC can be configured.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_REFERENCE_VDD_TO_AVSS,
  ADC_REFERENCE_VREFPLUS_TO_AVSS,
  ADC_REFERENCE_AVDD_TO_VREF_NEG,
  ADC_REFERENCE_VREFPLUS_TO_VREFNEG
} ADC_VOLTAGE_REFERENCE;
```

Members

Members	Description
ADC_REFERENCE_VDD_TO_AVSS	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_VREFPLUS_TO_AVSS	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_AVDD_TO_VREF_NEG	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied externally through VREF- pin
ADC_REFERENCE_VREFPLUS_TO_VREFNEG	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied externally through VREF- pin

Description

ADC Voltage Reference Enumeration

This data type defines the different ADC Voltage Reference by which the ADC can be configured.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_NEGATIVE Enumeration

Defines the different ADC Negative Input Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_NEGATIVE_VREF_MINUS,
  ADC_INPUT_NEGATIVE_AN1,
  ADC_INPUT_NEGATIVE_AN2,
  ADC_INPUT_NEGATIVE_AN3,
  ADC_INPUT_NEGATIVE_AN4,
  ADC_INPUT_NEGATIVE_AN5,
  ADC_INPUT_NEGATIVE_AN6
} ADC_INPUTS_NEGATIVE;
```

Members

Members	Description
ADC_INPUT_NEGATIVE_VREF_MINUS	Negative input is VREF
ADC_INPUT_NEGATIVE_AN1	Negative input is AN1
ADC_INPUT_NEGATIVE_AN2	Negative input is AN2
ADC_INPUT_NEGATIVE_AN3	Negative input is AN3
ADC_INPUT_NEGATIVE_AN4	Negative input is AN4
ADC_INPUT_NEGATIVE_AN5	Negative input is AN5
ADC_INPUT_NEGATIVE_AN6	Negative input is AN6

Description

ADC Negative Input Enumeration

This data type defines the different ADC Negative Input Enumeration.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_SAMPLES_PER_INTERRUPT Enumeration

Defining the Samples Per Interrupt Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_1SAMPLE_PER_INTERRUPT,
  ADC_2SAMPLES_PER_INTERRUPT,
  ADC_3SAMPLES_PER_INTERRUPT,
  ADC_4SAMPLES_PER_INTERRUPT,
  ADC_5SAMPLES_PER_INTERRUPT,
  ADC_6SAMPLES_PER_INTERRUPT,
  ADC_7SAMPLES_PER_INTERRUPT,
  ADC_8SAMPLES_PER_INTERRUPT,
  ADC_9SAMPLES_PER_INTERRUPT,
  ADC_10SAMPLES_PER_INTERRUPT,
  ADC_11SAMPLES_PER_INTERRUPT,
  ADC_12SAMPLES_PER_INTERRUPT,
  ADC_13SAMPLES_PER_INTERRUPT,
  ADC_14SAMPLES_PER_INTERRUPT,
  ADC_15SAMPLES_PER_INTERRUPT,
  ADC_16SAMPLES_PER_INTERRUPT
} ADC_SAMPLES_PER_INTERRUPT;
```

Members

Members	Description
ADC_1SAMPLE_PER_INTERRUPT	Interrupts at the completion of conversion for each sample/convert sequence
ADC_2SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 2nd sample/convert sequence
ADC_3SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 3rd sample/convert sequence
ADC_4SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 4th sample/convert sequence
ADC_5SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 5th sample/convert sequence
ADC_6SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 6th sample/convert sequence
ADC_7SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 7th sample/convert sequence
ADC_8SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 8th sample/convert sequence
ADC_9SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 9th sample/convert sequence
ADC_10SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 10th sample/convert sequence
ADC_11SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 11th sample/convert sequence
ADC_12SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 12th sample/convert sequence
ADC_13SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 13th sample/convert sequence
ADC_14SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 14th sample/convert sequence
ADC_15SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 15th sample/convert sequence
ADC_16SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 16th sample/convert sequence

Description

ADC samples per Interrupt Enumeration

This data type defines the Samples Per Interrupt Enumeration.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_CLOCK_SOURCE Enumeration

Defines the ADC Clock Source Select.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK,
  ADC_CLOCK_SOURCE_SYSTEM_CLOCK,
  ADC_CLOCK_SOURCE_INTERNAL_RC
} ADC_CLOCK_SOURCE;
```

Members

Members	Description
ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK	A/D Peripheral clock
ADC_CLOCK_SOURCE_SYSTEM_CLOCK	This option is deprecated, use ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK
ADC_CLOCK_SOURCE_INTERNAL_RC	A/D internal RC clock

Description

ADC Clock Source Select

This data type defines the ADC Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_CONVERSION_TRIGGER_SOURCE Enumeration

Defines the ADC Conversion Trigger Source.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_CONVERSION_TRIGGER_SAMP_CLEAR,
  ADC_CONVERSION_TRIGGER_INT0_TRANSITION,
  ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH,
  ADC_CONVERSION_TRIGGER_CTMU_EVENT,
  ADC_CONVERSION_TRIGGER_INTERNAL_COUNT
} ADC_CONVERSION_TRIGGER_SOURCE;
```

Members

Members	Description
ADC_CONVERSION_TRIGGER_SAMP_CLEAR	Clearing SAMP bit (full program control)
ADC_CONVERSION_TRIGGER_INT0_TRANSITION	Active transition on INT0 pin (basic sync convert)
ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH	Timer3 compare match
ADC_CONVERSION_TRIGGER_CTMU_EVENT	CTMU event
ADC_CONVERSION_TRIGGER_INTERNAL_COUNT	Internal counter (auto-convert)

Description

ADC Conversion Trigger Source

This data type defines the ADC Conversion Trigger Source.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_BUFFER_MODE Enumeration

Defines the ADC Buffer Mode.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_BUFFER_MODE_ONE_16WORD_BUFFER,
  ADC_BUFFER_MODE_TWO_8WORD_BUFFERS
} ADC_BUFFER_MODE;
```

Members

Members	Description
ADC_BUFFER_MODE_ONE_16WORD_BUFFER	Buffer configured as one 16-word buffer (ADC1BUF0 to ADC1BUFF)
ADC_BUFFER_MODE_TWO_8WORD_BUFFERS	Buffer configured as two 8-word buffers (ADC1BUF0 to ADC1BUF7 and ADC1BUF8 to ADC1BUFF)

Description

ADC Buffer Mode

This data type defines the ADC Buffer Mode.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_RESULT_BUF_STATUS Enumeration

Defines the ADC Result Buffer Status

File

help_plib_adc.h

C

```
typedef enum {
  ADC_FILLING_BUF_0TO7,
  ADC_FILLING_BUF_8TOF
} ADC_RESULT_BUF_STATUS;
```

Members

Members	Description
ADC_FILLING_BUF_0TO7	Buffers 0x0 to 0x7 are getting filled
ADC_FILLING_BUF_8TOF	Buffers 0x8 to 0xF are getting filled

Description

ADC Result Buffer Status

This data type defines the elements that specify which group of eight buffers the ADC is currently filling.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_MUX Enumeration

Defining the different ADC MUX Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_MUX_A,
  ADC_MUX_B
} ADC_MUX;
```

Members

Members	Description
ADC_MUX_A	ADC Mux A Selection
ADC_MUX_B	ADC Mux B Selection

Description

ADC MUX Enumeration

This data type defines the different ADC MUX Enumeration.

Remarks

None.

ADC_SAMPLING_MODE Enumeration

Defines the ADC Sampling Mode Select.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_SAMPLING_MODE_MUXA,
  ADC_SAMPLING_MODE_ALTERNATE_INPUT
} ADC_SAMPLING_MODE;
```

Members

Members	Description
ADC_SAMPLING_MODE_MUXA	Always Mux A Sampling Mode
ADC_SAMPLING_MODE_ALTERNATE_INPUT	Alternate Input Sampling Mode

Description

ADC Sampling Mode Select

This data type defines the ADC Sampling Mode Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_SCAN Enumeration

Defines the ADC Scan inputs.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_SCAN_AN0,
  ADC_INPUT_SCAN_AN1,
  ADC_INPUT_SCAN_AN2,
  ADC_INPUT_SCAN_AN3,
  ADC_INPUT_SCAN_AN4,
  ADC_INPUT_SCAN_AN5,
  ADC_INPUT_SCAN_AN6,
  ADC_INPUT_SCAN_AN7,
  ADC_INPUT_SCAN_AN8,
  ADC_INPUT_SCAN_AN9,
  ADC_INPUT_SCAN_AN10,
  ADC_INPUT_SCAN_AN11,
  ADC_INPUT_SCAN_AN12,
  ADC_INPUT_SCAN_AN13,
  ADC_INPUT_SCAN_AN14,
  ADC_INPUT_SCAN_AN15,
  ADC_INPUT_SCAN_AN16,
```

```
  ADC_INPUT_SCAN_AN17,
  ADC_INPUT_SCAN_AN18,
  ADC_INPUT_SCAN_AN19,
  ADC_INPUT_SCAN_AN20,
  ADC_INPUT_SCAN_AN21,
  ADC_INPUT_SCAN_AN22,
  ADC_INPUT_SCAN_AN23,
  ADC_INPUT_SCAN_AN24,
  ADC_INPUT_SCAN_AN25,
  ADC_INPUT_SCAN_AN26,
  ADC_INPUT_SCAN_AN27,
  ADC_INPUT_SCAN_AN28,
  ADC_INPUT_SCAN_AN29,
  ADC_INPUT_SCAN_AN30,
  ADC_INPUT_SCAN_AN31,
  ADC_INPUT_SCAN_IVREF,
  ADC_INPUT_SCAN_CTMU,
  ADC_INPUT_SCAN_VSS
} ADC_INPUTS_SCAN;
```

Members

Members	Description
ADC_INPUT_SCAN_AN0	Scan input is AN0
ADC_INPUT_SCAN_AN1	Scan input is AN1
ADC_INPUT_SCAN_AN2	Scan input is AN2
ADC_INPUT_SCAN_AN3	Scan input is AN3
ADC_INPUT_SCAN_AN4	Scan input is AN4
ADC_INPUT_SCAN_AN5	Scan input is AN5
ADC_INPUT_SCAN_AN6	Scan input is AN6
ADC_INPUT_SCAN_AN7	Scan input is AN7
ADC_INPUT_SCAN_AN8	Scan input is AN8
ADC_INPUT_SCAN_AN9	Scan input is AN9
ADC_INPUT_SCAN_AN10	Scan input is AN10
ADC_INPUT_SCAN_AN11	Scan input is AN11
ADC_INPUT_SCAN_AN12	Scan input is AN12
ADC_INPUT_SCAN_AN13	Scan input is AN13
ADC_INPUT_SCAN_AN14	Scan input is AN14
ADC_INPUT_SCAN_AN15	Scan input is AN15
ADC_INPUT_SCAN_AN16	Scan input is AN16
ADC_INPUT_SCAN_AN17	Scan input is AN17
ADC_INPUT_SCAN_AN18	Scan input is AN18
ADC_INPUT_SCAN_AN19	Scan input is AN19
ADC_INPUT_SCAN_AN20	Scan input is AN20
ADC_INPUT_SCAN_AN21	Scan input is AN21
ADC_INPUT_SCAN_AN22	Scan input is AN22
ADC_INPUT_SCAN_AN23	Scan input is AN23
ADC_INPUT_SCAN_AN24	Scan input is AN24
ADC_INPUT_SCAN_AN25	Scan input is AN25
ADC_INPUT_SCAN_AN26	Scan input is AN26
ADC_INPUT_SCAN_AN27	Scan input is AN27
ADC_INPUT_SCAN_AN28	Scan input is AN28
ADC_INPUT_SCAN_AN29	Scan input is AN29
ADC_INPUT_SCAN_AN30	Scan input is AN30
ADC_INPUT_SCAN_AN31	Scan input is AN31
ADC_INPUT_SCAN_IVREF	Scan input is IVref
ADC_INPUT_SCAN_CTMU	Scan input is CTMU input
ADC_INPUT_SCAN_VSS	Scan input is VSS

Description

ADC Scan Inputs Enumeration

This data type defines the ADC Scan inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_RESULT_FORMAT Enumeration

Defines the ADC Result Format.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_RESULT_FORMAT_INTEGER_16BIT,
  ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT,
  ADC_RESULT_FORMAT_FRACTIONAL_16BIT,
  ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT,
  ADC_RESULT_FORMAT_INTEGER_32BIT,
  ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT,
  ADC_RESULT_FORMAT_FRACTIONAL_32BIT,
  ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT
} ADC_RESULT_FORMAT;
```

Members

Members	Description
ADC_RESULT_FORMAT_INTEGER_16BIT	Integer 16-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT	Signed Integer 16-bit
ADC_RESULT_FORMAT_FRACTIONAL_16BIT	Fractional 16-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT	Signed Fractional 16-bit
ADC_RESULT_FORMAT_INTEGER_32BIT	Integer 32-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT	Signed Integer 32-bit
ADC_RESULT_FORMAT_FRACTIONAL_32BIT	Fractional 32-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT	Signed Fractional 32-bit

Description

ADC Result Format

This data type defines the ADC Result Format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_POSITIVE Enumeration

Defines the ADC inputs.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_POSITIVE_AN0,
  ADC_INPUT_POSITIVE_AN1,
  ADC_INPUT_POSITIVE_AN2,
  ADC_INPUT_POSITIVE_AN3,
  ADC_INPUT_POSITIVE_AN4,
  ADC_INPUT_POSITIVE_AN5,
  ADC_INPUT_POSITIVE_AN6,
  ADC_INPUT_POSITIVE_AN7,
  ADC_INPUT_POSITIVE_AN8,
  ADC_INPUT_POSITIVE_AN9,
  ADC_INPUT_POSITIVE_AN10,
  ADC_INPUT_POSITIVE_AN11,
  ADC_INPUT_POSITIVE_AN12,
  ADC_INPUT_POSITIVE_AN13,
  ADC_INPUT_POSITIVE_AN14,
```

```
    ADC_INPUT_POSITIVE_AN15,
    ADC_INPUT_POSITIVE_AN16,
    ADC_INPUT_POSITIVE_AN17,
    ADC_INPUT_POSITIVE_AN18,
    ADC_INPUT_POSITIVE_AN19,
    ADC_INPUT_POSITIVE_AN20,
    ADC_INPUT_POSITIVE_AN21,
    ADC_INPUT_POSITIVE_AN22,
    ADC_INPUT_POSITIVE_AN23,
    ADC_INPUT_POSITIVE_AN24,
    ADC_INPUT_POSITIVE_AN25,
    ADC_INPUT_POSITIVE_AN26,
    ADC_INPUT_POSITIVE_AN27,
    ADC_INPUT_POSITIVE_AN28,
    ADC_INPUT_POSITIVE_AN29,
    ADC_INPUT_POSITIVE_AN30,
    ADC_INPUT_POSITIVE_AN31,
    ADC_INPUT_POSITIVE_AN32,
    ADC_INPUT_POSITIVE_AN33,
    ADC_INPUT_POSITIVE_AN34,
    ADC_INPUT_POSITIVE_AN35,
    ADC_INPUT_POSITIVE_AN36,
    ADC_INPUT_POSITIVE_AN37,
    ADC_INPUT_POSITIVE_AN38,
    ADC_INPUT_POSITIVE_AN39,
    ADC_INPUT_POSITIVE_AN40,
    ADC_INPUT_POSITIVE_AN41,
    ADC_INPUT_POSITIVE_AN42,
    ADC_INPUT_POSITIVE_AN43,
    ADC_INPUT_POSITIVE_AN44,
    ADC_INPUT_POSITIVE_AN45,
    ADC_INPUT_POSITIVE_AN46,
    ADC_INPUT_POSITIVE_AN47,
    ADC_INPUT_POSITIVE_CTMU,
    ADC_INPUT_POSITIVE_IVREF,
    ADC_INPUT_POSITIVE_OPEN
} ADC_INPUTS_POSITIVE;
```

Members

Members	Description
ADC_INPUT_POSITIVE_AN0	Positive input is AN0
ADC_INPUT_POSITIVE_AN1	Positive input is AN1
ADC_INPUT_POSITIVE_AN2	Positive input is AN2
ADC_INPUT_POSITIVE_AN3	Positive input is AN3
ADC_INPUT_POSITIVE_AN4	Positive input is AN4
ADC_INPUT_POSITIVE_AN5	Positive input is AN5
ADC_INPUT_POSITIVE_AN6	Positive input is AN6
ADC_INPUT_POSITIVE_AN7	Positive input is AN7
ADC_INPUT_POSITIVE_AN8	Positive input is AN8
ADC_INPUT_POSITIVE_AN9	Positive input is AN9
ADC_INPUT_POSITIVE_AN10	Positive input is AN10
ADC_INPUT_POSITIVE_AN11	Positive input is AN11
ADC_INPUT_POSITIVE_AN12	Positive input is AN12
ADC_INPUT_POSITIVE_AN13	Positive input is AN13
ADC_INPUT_POSITIVE_AN14	Positive input is AN14
ADC_INPUT_POSITIVE_AN15	Positive input is AN15
ADC_INPUT_POSITIVE_AN16	Positive input is AN16
ADC_INPUT_POSITIVE_AN17	Positive input is AN17
ADC_INPUT_POSITIVE_AN18	Positive input is AN18
ADC_INPUT_POSITIVE_AN19	Positive input is AN19
ADC_INPUT_POSITIVE_AN20	Positive input is AN20
ADC_INPUT_POSITIVE_AN21	Positive input is AN21
ADC_INPUT_POSITIVE_AN22	Positive input is AN22
ADC_INPUT_POSITIVE_AN23	Positive input is AN23
ADC_INPUT_POSITIVE_AN24	Positive input is AN24

ADC_INPUT_POSITIVE_AN25	Positive input is AN25
ADC_INPUT_POSITIVE_AN26	Positive input is AN26
ADC_INPUT_POSITIVE_AN27	Positive input is AN27
ADC_INPUT_POSITIVE_AN28	Positive input is AN28
ADC_INPUT_POSITIVE_AN29	Positive input is AN29
ADC_INPUT_POSITIVE_AN30	Positive input is AN30
ADC_INPUT_POSITIVE_AN31	Positive input is AN31
ADC_INPUT_POSITIVE_AN32	Positive input is AN32
ADC_INPUT_POSITIVE_AN33	Positive input is AN33
ADC_INPUT_POSITIVE_AN34	Positive input is AN34
ADC_INPUT_POSITIVE_AN35	Positive input is AN35
ADC_INPUT_POSITIVE_AN36	Positive input is AN36
ADC_INPUT_POSITIVE_AN37	Positive input is AN37
ADC_INPUT_POSITIVE_AN38	Positive input is AN38
ADC_INPUT_POSITIVE_AN39	Positive input is AN39
ADC_INPUT_POSITIVE_AN40	Positive input is AN40
ADC_INPUT_POSITIVE_AN41	Positive input is AN41
ADC_INPUT_POSITIVE_AN42	Positive input is AN42
ADC_INPUT_POSITIVE_AN43	Positive input is AN43
ADC_INPUT_POSITIVE_AN44	Positive input is AN44
ADC_INPUT_POSITIVE_AN45	Positive input is AN45
ADC_INPUT_POSITIVE_AN46	Positive input is AN46
ADC_INPUT_POSITIVE_AN47	Positive input is AN47
ADC_INPUT_POSITIVE_CTMU	Positive input is from CTMU
ADC_INPUT_POSITIVE_IVREF	Positive input is IVref
ADC_INPUT_POSITIVE_OPEN	Positive input is Open

Description

ADC Inputs Enumeration

This data type defines the ADC inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_ACQUISITION_TIME Type

Data type defining the different ADC acquisition times by which the ADC can be configured.

File

plib_adc.h

C

```
typedef uint32_t ADC_ACQUISITION_TIME;
```

Description

ADC Acquisition Time Selection Data Type

This data type defines the different ADC acquisition times by which the ADC can be configured.

Remarks

None.

ADC_CONVERSION_CLOCK Type

Data type defines the different ADC Conversion clock

File

plib_adc.h

C

```
typedef uint32_t ADC_CONVERSION_CLOCK;
```

Description

ADC Conversion Clock Selection Data Type

This data type defines the different ADC Conversion clock

Remarks

None.

ADC_SAMPLE Type

Data type defining the size of the ADC sample register.

File

plib_adc.h

C

```
typedef uint32_t ADC_SAMPLE;
```

Description

ADC Sample size

This data type defines the size of the ADC sample register.

Remarks

None.

Files

Files

Name	Description
plib_adc.h	ADC PLIB interface header for definitions common to the ADC peripheral.
help_plib_adc.h	This is file help_plib_adc.h.

Description

This section lists the source and header files used by the library.

plib_adc.h

ADC PLIB interface header for definitions common to the ADC peripheral.

Functions

	Name	Description
	PLIB_ADC_CalibrationDisable	Normal ADC module operation (no calibration is performed).
	PLIB_ADC_CalibrationEnable	Calibration is performed on the next ADC conversion.
	PLIB_ADC_ConversionClockGet	Obtains the conversion clock.
	PLIB_ADC_ConversionClockSet	Sets the ADC module conversion clock.
	PLIB_ADC_ConversionClockSourceSelect	Selects the ADC module conversion clock source.
	PLIB_ADC_ConversionHasCompleted	Provides the conversion completion status of the ADC.
	PLIB_ADC_ConversionStart	Starts ADC module manual conversion process.
	PLIB_ADC_ConversionStopSequenceDisable	Normal conversion sequence.
	PLIB_ADC_ConversionStopSequenceEnable	Stop conversion sequence (when the first ADC module interrupt is generated).
	PLIB_ADC_ConversionTriggerSourceSelect	Selects the conversion trigger source.
	PLIB_ADC_Disable	ADC module is disabled (turned OFF).
	PLIB_ADC_Enable	ADC module is enabled (turned ON).
	PLIB_ADC_ExistsCalibrationControl	Identifies whether the CalibrationControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionClock	Identifies whether the ConversionClock feature exists on the ADC module

	PLIB_ADC_ExistsConversionClockSource	Identifies whether the ConversionClockSource feature exists on the ADC module
	PLIB_ADC_ExistsConversionControl	Identifies whether the ConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionStatus	Identifies whether the ConversionStatus feature exists on the ADC module
	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0NegativeInput	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0PositiveInput	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanControl	Identifies whether the MuxInputScanControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelect	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelectExtended	Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferFillStatus	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferMode	Identifies whether the ResultBufferMode feature exists on the ADC module
	PLIB_ADC_ExistsResultFormat	Identifies whether the ResultFormat feature exists on the ADC module
	PLIB_ADC_ExistsResultGetByIndex	Identifies whether the ResultGetByIndex feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskAddExtended	Select extended ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemoveExtended	Omits extended ADC analog channel for input scan.
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.

Types

	Name	Description
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the ADC sample register.

Description

ADC Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADC PLIB for all families of Microchip microcontrollers. The definitions in this file are common to the ADC peripheral.

File Name

plib_adc.h

Company

Microchip Technology Inc.

help_plib_adc.h

Enumerations

	Name	Description
	ADC_BUFFER_MODE	Defines the ADC Buffer Mode.
	ADC_CLOCK_SOURCE	Defines the ADC Clock Source Select.
	ADC_CONVERSION_TRIGGER_SOURCE	Defines the ADC Conversion Trigger Source.
	ADC_INPUTS_NEGATIVE	Defines the different ADC Negative Input Enumeration.
	ADC_INPUTS_POSITIVE	Defines the ADC inputs.
	ADC_INPUTS_SCAN	Defines the ADC Scan inputs.
	ADC_MODULE_ID	Identifies the available ADC modules.
	ADC_MUX	Defining the different ADC MUX Enumeration.
	ADC_RESULT_BUF_STATUS	Defines the ADC Result Buffer Status
	ADC_RESULT_FORMAT	Defines the ADC Result Format.
	ADC_SAMPLES_PER_INTERRUPT	Defining the Samples Per Interrupt Enumeration.
	ADC_SAMPLING_MODE	Defines the ADC Sampling Mode Select.
	ADC_VOLTAGE_REFERENCE	Defining the different ADC Voltage Reference by which the ADC can be configured.

Description

This is file help_plib_adc.h.

ADCHS Peripheral Library

This section describes the 12-bit High-Speed Successive Approximation Register (SAR) Analog-to-Digital Converter (ADCHS) Peripheral Library.

Introduction

This library provides a low-level abstraction of the High-Speed SAR ADC (ADCHS) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the registers, thereby hiding differences from one microcontroller variant to another.

Description

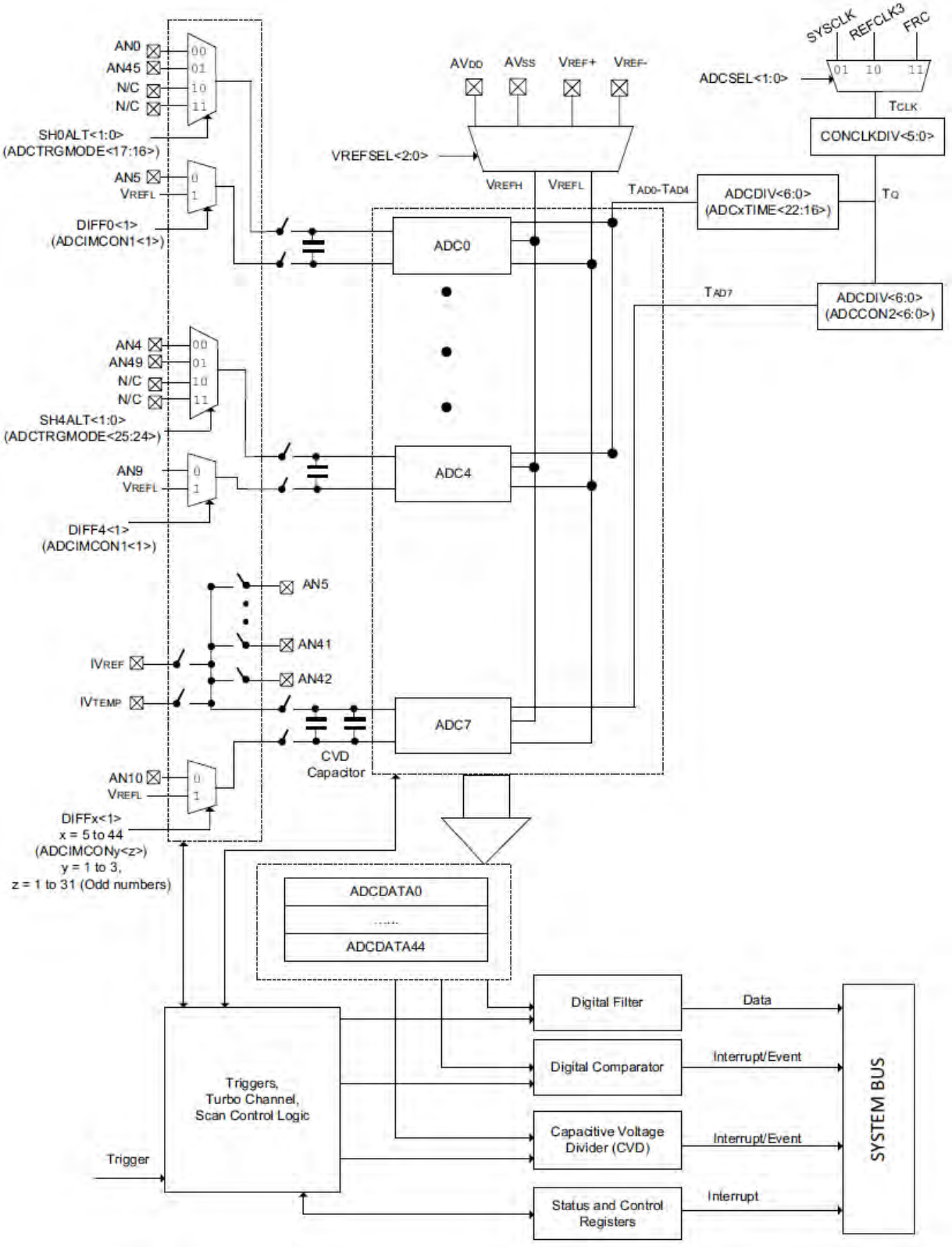
The ADCHS is used to convert an analog input into a digital number that represents that input voltage. When the input is periodically converted, the result is a sequence of digital values that have converted a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal.

This particular ADC architecture consists of multiple SAR channels (maximum of eight). The SAR channels are numbered from Channel 0 to Channel 7. Out of these channels, Channels 0 through 6 have dedicated sample and hold (S&H) circuits connected to a single (selectable) analog input. These SAR channels work independently from each other and are designed to capture time sensitive or transitory analog signals. The remaining channel, Channel 7, has its S&H circuit connected to multiple analog inputs through a multiplexer. Channel 7 is used to convert analog signals that do not need a high conversion rate.

Features of the ADCHS module include:

- Channels 0 through 6 have selectable analog inputs (default, alternate), which can be connected to a S&H circuit
- Channel 7 has multiple analog inputs connected to a S&H circuit
- Analog inputs connected to Channels 0 through 6 are Class 1 analog inputs. Analog inputs connected to Channel 7 are classified as Class 2 and Class 3 analog inputs.
  - Class 1 analog inputs have their individual trigger source and are converted by dedicated Channels (0 through 6).
  - Class 2 analog inputs have their individual trigger source, but are converted by shared channel-7
  - Class 3 analog inputs do not have individual trigger source and are converted by scan mode using Channel 7
- Analog input scan is available, which allows a single trigger to start a sample/conversion sequence of multiple analog inputs using a predefined scan list
- The inputs to a S&H can be configured as unipolar/bipolar and single-ended/differential mode
- For each channel, the converted data resolution can be set as 6, 8, 10, or 12-bits
- Each channel (0 through 7) can have its individual conversion clock and sample time setting
- A voltage reference is required for the ADCHS module. The voltage reference setting is a global setting for the ADCHS module and does not vary across channels.
- Result registers store the conversion results and can be read by the software application. Software is notified of ready results by either polling or through the use of interrupts.
- For Channels 0 through 6, the converted data can be stored in an optional FIFO or directly to RAM using the dedicated DMA interface
- Optional Digital Filters can be used to provide increased resolution at the sacrifice of sample rate. Digital Filters also have an averaging mode.
- Optional Digital Comparators can be used to test conversion results against set high and low limits, which are independent of software, and generate interrupts based on predefined conditions
- Digital Comparators also have Capacitive Voltage Divider (CVD) mode, which can be used to detect a touch event in touch sense application
- Early interrupt generation, providing extremely fast processing of converted data

A block diagram of the ADCHS module is provided in the following figure.



Using the Library

This topic describes the basic architecture of the ADCHS Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adchs.h`

The interface to the ADCHS Peripheral library is defined in the `plib_adchs.h` header file, which is included by the `peripheral.h` file. Any C

language source (`.c`) file that uses the ADCHS Peripheral library must include `peripheral.h`.

Library File:

The ADCHS Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

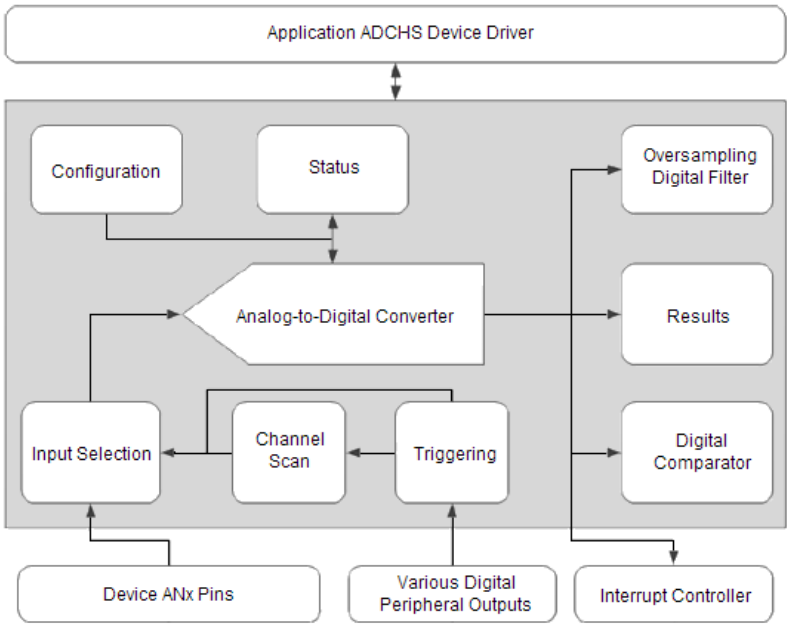
Hardware Abstraction Model

This library provides the low-level abstraction of the ADCHS module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available ADCHS module on the device. Refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each ADCHS module on the device.

Description

High-Speed SAR Analog-to-Digital Converter (ADCHS) Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADCHS module.

Library Interface Section	Description
Configuration Functions	These functions are used to configure, enable, and disable the ADCHS.
Analog Input Functions	These functions are used to configure analog input features.
Analog Input Selection Functions	These functions select and configure the ANx inputs to the ADCHS.
CVD Functions	These functions configure the Capacitive Voltage Divider (CVD) mode.
DMA Functions	These functions are used to configure and enable/disable the DMA, enable/ disable interrupts linked to the DMA, and to get the DMA status.
Channel Related Functions	These functions configure Channels 0 through 7, enable/disable the analog and digital circuitry, and control interrupt generation.
Digital Comparator Functions	These functions are used to configure and query the digital comparators.
Digital Filter Functions	These functions are used to configure the Oversampling Digital Filter and fetch results from it.
FIFO Functions	These functions configure the FIFO and control the interrupt generation.
Status Functions	These functions return status information related to the ADCHS.
Interrupt Functions	These functions control interrupt generation.

Turbo Mode Functions	These functions are used to configure, enable, and disable Turbo mode.
Triggering Functions	These functions configure the trigger source for various cores.
Voltage Reference Functions	These functions control the Voltage Reference (VREF) input for the ADC cores and query VREF status and interrupt generation.
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Functionality

This topic describes the functionality of the ADCHS Peripheral Library.

Description

Analog-to-Digital conversion using the ADCHS involves the following three steps:

1. Sampling of the input signal.
2. Capture of the input signal by the S&H circuit and transfer to the converter.
3. Conversion of the analog signal to its digital representation.

Sampling of the input signal involves charging of the S&H capacitor. The sampling time must be adequate so that the capacitor charges to a value equal to the input voltage. At the appropriate time, the input is disconnected and the capacitor voltage is transferred to the converter. The converter then digitizes the analog signal and provides the result. The converter requires a clock source (common input to individual clock setting of all channels) and a reference voltage. The common input clock is referred to as the control clock and has a period of TQ. The control clock and reference voltage sources are selectable, as well as the clock prescaling that determines the TQ.

The ADCHS has multiple SAR channels. Each channel has independent clocks named TAD0 through TAD7. The analog inputs connected to the SAR channels can be divided as Class 1, Class 2, and Class 3.

- Class 1 inputs are associated with a SAR Channel 0 through 6. Each SAR channel has a single (selectable) Class 1 input associated with it. Each Class 1 input has a unique trigger selection register. Class 1 inputs can also be part of a channel scan list, triggered by the common scan trigger source.
- Class 2 inputs are connected to Channel 7 and are either individually triggered or as part of a channel scan list. When used individually their trigger source is selected by their unique trigger register.
- Class 3 inputs are connected to Channel 7 and are used in channel scan exclusively. They share a common trigger source. When using channel scan it is possible to combine Class 1, Class 2, and Class 3 inputs in the scan list.

Note: For details regarding configuration and triggering options was well as sampling requirements, refer to the **"ADC"** chapter in the specific device data sheet and the family reference manual section specified in that chapter.

Example - Simultaneous Sampling and Conversion of Three Class 1 Inputs

Simultaneous sampling and conversion is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling and conversion requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The respective SAR channels are configured for single-ended input and a unipolar (unsigned) output.

No interrupts are used so the results are polled for ready status.

```
{
    int result[3];                              // storage for results
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                   // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,               // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,    // Use fractional format
        true,                                   // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,    // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,  // vector shift unused, interrupt not used
        0,                                      // vector base address unused, interrupt not used
```

```
    ADCHS_CLOCK_SOURCE_SYSCLK,              // SYSCLK is the clock source
    1,                                      // TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                   // Warm-up time = 32 clocks
);
// Configure the ADC SAR Channel-0
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_0,                        // Channel - 0
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD0 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-0
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0,                    // channel - 0
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Configure the ADC SAR Channel-1
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_1, // Channel - 1
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD1 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-1
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                    // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Configure the ADC SAR Channel-2
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_2,                        // Channel - 2
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD2 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-2
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2,                    // channel - 2
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Select inputs for Channel
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_0,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN0    // Select AN0 for channel-0
            ))
            {
                // error has occurred
                while(1);
            }
```

```
    if(false == PLIB_ADCHS_ChannelInputSelect
                (
                    MY_ADCHS_INSTANCE,
                    ADCHS_CHANNEL_1,
                    ADCHS_CHANNEL_0_DEFAULT_INP_AN1    // Select AN1 for channel-1
                ))
                {
                    // error has occurred
                    while(1);
                }
    if(false == PLIB_ADCHS_ChannelInputSelect
                (
                    MY_ADCHS_INSTANCE,
                    ADCHS_CHANNEL_2,
                    ADCHS_CHANNEL_0_DEFAULT_INP_AN2    // Select AN2 for channel-2
                ))
                {
                    // error has occurred
                    while(1);
                }
    // Select input mode for AN0, AN1, AN2
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN0,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN1,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN2,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    // Select AN0, AN1 and AN2 as edge trigger
    PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN0 );
    PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN1 );
    PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN2 );
    // Select AN0, AN1 and AN2 to be software triggered
    PLIB_ADCHS_AnalogInputTriggerSourceSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CLASS12_AN0,
        ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    PLIB_ADCHS_AnalogInputTriggerSourceSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CLASS12_AN1,
        ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    PLIB_ADCHS_AnalogInputTriggerSourceSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CLASS12_AN2,
        ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-0, 1 and 2
```

```
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1
    );
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2
    );
    // Wait for the channels to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_0
        )
    );
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_1
        )
    );
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_2
        )
    );
    // Enable digital circuit for channels 0, 1, 2
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1
    );
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for conversion complete for AN0
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN0));
        result[0] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN0 );
        // Wait for conversion complete for AN1
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN1));
        result[1] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN1 );
        // Wait for conversion complete for AN2
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN2));
        result[2] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN2 );
    }
    return(1);
}
```

Example - Channel Scanning

Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs.

The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using Channel 7. This example uses inputs AN8, AN31 through AN33. AN8 is Class 2 inputs. AN31 to AN33 are Class 3 inputs. The global software trigger is used to initiate the scan. A 3 TAD7 sample time is specified for Channel 7, which is configured for single-ended operation and a unipolar output. No interrupts are used so the results are polled for ready status.

```
{
    int result[4]; // storage for results
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift unused, interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7,                         // Channel - 7
        ADCHS_DATA_RESOLUTION_12BIT,             // resolution is set to 12bits
        1,                                       // clock divider bit is, TAD7 = 2 * TQ
        1,                                       // Sample time is 3 * TAD7
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1      // 1 clock early interrupt (ignored, interrupt not used)
    );
    // Select analog channel AN8 and scan sequence to be triggered with global scan trigger
    PLIB_ADCHS_AnalogInputScanSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN8,
        ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
        ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    // Now, add further analog inputs, AN31, AN32, AN33 to scan list
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN31);
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN32);
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN33);
    // Select input mode for AN8, AN31, AN32, AN33
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN8,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN31,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN32,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
```

```
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN33,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    // Select individual trigger for class-2 channels viz. AN8 to be scan trigger.
    // AN31, AN32 and AN33 are class-3 channels, and do not have individual trigger
    PLIB_ADCHS_AnalogInputTriggerSourceSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CLASS12_AN8,
        ADCHS_TRIGGER_SOURCE_SCAN
    );
    // Select AN8 to be edge trigger (not level trigger).
    // Calling this function is needed for class-1 and 2 analog inputs
    PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN8);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-7
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    // Wait for the channel-7 to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_7
        )
    );
    // Enable digital circuit for channels 7
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for conversion complete for AN8
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN8));
        result[0] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN8 );
        // Wait for conversion complete for AN31
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN31));
        result[1] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN31 );
        // Wait for conversion complete for AN32
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN32));
        result[2] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN32 );
        // Wait for conversion complete for AN33
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN33));
        result[3] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN33 );
    }
    return(1);
}
```

Example - Digital Filter

The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The following example shows how two extra bits of resolution can be obtained using 16x oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3 TAD0.

```
{
```

```
int result; // storage for results
bool eventFlag = false; // Digital comparator event flag
// Configure the ADC
PLIB_ADCHS_Setup
(
    MY_ADCHS_INSTANCE,
    ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
    ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
    ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
    true,                                    // Do stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift not used since interrupt not used
    0,                                       // vector base address unused, interrupt not used
    ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
    1,                                       // TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
);
// Configure the ADC SAR Channel-0
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_0,                         // Channel - 0
    ADCHS_DATA_RESOLUTION_12BIT,             // resolution is set to 12bits
    1,                                       // clock divider bit is, TAD0 = 2 * TQ
    1,                                       // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1      // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-0
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0,                     // channel - 0
        ADCHS_CHANNEL_SYNC_SAMPLING))        // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Select inputs for Channel
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_0,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN0 // Select AN0 for channel-0
            ))
            {
                // error has occurred
                while(1);
            }
// Select input mode for AN0
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN0,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
// Select AN0 as edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN0 );
// Select AN0 to be software triggered
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN0,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
// Configure digital filter in oversampling mode for 16x sampling
PLIB_ADCHS_DigitalFilterOversamplingModeSetup
(
    MY_ADCHS_INSTANCE, // ADCHS channel ID
    ADCHS_DIGITAL_FILTER_1, // Filter ID
    ADCHS_AN0, // Oversample AN4
```

```
        ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
        ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X, // 16 x oversampling
        false // No Global Int Enable
    );
    // Enable digital filter
    PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-0
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    // Wait for the channels to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
        )
    );
    // Enable digital circuit for channels 0
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for filter result to be ready
        while(!PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1));
        result = PLIB_ADCHS_DigitalFilterDataGet(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    }
    return(1);
}
```

Example - Digital Comparator

The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated test of AN8 for values that are greater than or equal to 80% of full scale or less than 20% of full scale. A count is incremented each time an event occurs.

```
{
    int result; // storage for results
    bool eventFlag = false; // Digital comparator event flag
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift not used since interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
```

```
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7,                        // Channel - 7
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD7 = 2 * TQ
    1,                                      // Sample time is 3 * TAD7
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Select input mode for AN8
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN8,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
// Configure the digital comparator - 1
PLIB_ADCHS_DigitalComparatorSetup
(
    MY_ADCHS_INSTANCE,          // ADCHS channel ID
    ADCHS_DIGITAL_COMPARATOR_1, // Comparator ID
    false,                      // Global Int Enable
    true,                       // test for between low and high
    false,                      // no test for greater than equal to high
    false,                      // no test for less than high
    false,                      // no test for greater than equal to low
    false,                      // no test for less than low
    ADCHS_AN8,                  // select AN8
    0xFFF - (0xFFF/5),          // high limit, 80% of full scale
    (0xFFF/5)                   // low limit, 20% of full scale
);
// Select individual trigger for class-2 channels viz. AN8 to be global software trigger.
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN8,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
// Select edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN8 );
// Enable the digital comparator
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
// Enable ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
// Check VREF to be ready
While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
// Check for VREF Fault
While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
// Enable analog circuit for channel-7
PLIB_ADCHS_ChannelAnalogFeatureEnable
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7
);
// Wait for the channel-7 to be ready
while(!PLIB_ADCHS_ChannelIsReady
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    )
);
// Enable digital circuit for channels 7
PLIB_ADCHS_ChannelDigitalFeatureEnable
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7
);
while(1)
{
    // Enable global software trigger
```

```
            PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
            // Wait for conversion complete for AN8
            while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN8));
            result = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN8 );
            // Note: It is not necessary to read the conversion result for digital comparator
            // to work
            // See if we have comparator event
            if(PLIB_ADCHS_DigitalComparatorEventHasOccurred( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1 ))
            {
                eventFlag = true;
            }
        }
        return(1);
    }
```

Example - CVD Mode

The CVD mode is used to detect a touch event by measuring the voltage across external capacitor using the capacitive voltage divider method. The Digital Comparator used in conjunction with CVD mode automatically compares the voltage with set limits. Once the voltage is beyond the set limit, a comparator event is generated. In the following example, AN40 is used as an analog input. Physically, the AN40 pin should be connected to a touch sense pad. Once a touch is detected, a comparator event will be generated.

```
{
    int result; // storage for result
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift unused, interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7,                    // Channel - 7
        ADCHS_DATA_RESOLUTION_12BIT,        // resolution is set to 12bits
        1,                                  // clock divider bit is, TAD7 = 2 * TQ
        1,                                  // Sample time is 3 * TAD7
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 // 1 clock early interrupt (ignored, interrupt not used)
    );
    // Select analog channels as bipolar and single ended
    // Since CVD measures difference in voltages, the setting should be bipolar
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN40,
        ADCHS_INPUT_MODE_SINGLE_ENDED_BIPOLAR
    );
    // Select AN40 for scanning, as CVD needs scanning to be enabled
    PLIB_ADCHS_AnalogInputScanSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN40,
        ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
        ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    PLIB_ADCHS_CVDSetup
    (
        MY_ADCHS_INSTANCE,
```

```
        ADCHS_CVD_CAPACITOR_5PF,
        false,              // no test for between low and high
        true,               // Once touch event occurs, CVD output will
                            // be higher than the set limits
        false,              // no test for less than high
        false,              // no test for greater than equal to low
        false,              // no test for less than low
        ADCHS_AN40,         // select AN40
        0xFFF - (0xFFF/5), // high limit, 80% of full scale
        (0xFFF/5)           // low limit, 20% of full scale
    );
    // Enable the digital comparator-1, since SVD works with comparator-1
    PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
    // Enable CVD mode
    PLIB_ADCHS_CVDEnable(MY_ADCHS_INSTANCE);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-7
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    // Wait for the channel-7 to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_7
        )
    );
    // Enable digital circuit for channels 7
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait, if there is no touch event sensed by CVD + comparator
        while(!PLIB_ADCHS_DigitalComparatorEventHasOccurred
            (
            MY_ADCHS_INSTANCE,
            ADCHS_DIGITAL_COMPARATOR_1
            )
        );
        // Verify of the CVD selected input is AN40
        // This is not required, but provided as an example
        if(ADCHS_AN40 == PLIB_ADCHS_DigitalComparatorAnalogInputGet( MY_ADCHS_INSTANCE,
        ADCHS_DIGITAL_COMPARATOR_1 ))
        {
            result == PLIB_ADCHS_CVDResultGet( MY_ADCHS_INSTANCE );
        }
    }
    return(1);
}
```

Other Functionality

This topic provides information on additional functionality.

Description

The ADCHS also implements the following additional functionalities:

- Digital Filter
- Digital Comparator
- Capacitive Voltage Divider (CVD) mode
- Turbo mode
- FIFO and DMA Interface for dedicated cores

Digital Filter

The Digital Filter consists of an accumulator and a decimator (downsampler), which function together as a low-pass filter. By sampling an analog input at a higher-than-required sample rate (oversampling), and then processing the data through the oversampling digital filter, the number of usable bits of the ADC channel can be increased at the expense of decreased conversion throughput. For example, using 4x oversampling yields one extra usable bit, 16x oversampling yields two extra usable bits, 64x oversampling provides three extra usable bits, and 256x oversampling provides four extra usable bits. Once configured, the application provides a single trigger to the analog input specified for oversampling, and then fetches the result when the process is complete.

Digital Comparator

The ADCHS module features multiple Digital Comparators that can be used to monitor selected analog input conversion results and generate an interrupt when a conversion result matches the user-specified limits. Conversion triggers are still required to initiate conversions. The comparison occurs automatically once the conversion is complete. When using a hardware source as a periodic trigger, it is possible to monitor analog inputs and create an interrupt when the converted level matches specified criteria without any software overhead.

CVD Mode

The CVD mode allows the ADCHS module to detect a touch event by measuring the voltage difference of internal and external capacitors by alternately connecting it to VDD and GND. The CVD mode works in conjunction with the Digital Comparator and can generate an event (interrupt) once the voltage difference is more than the set value (indicating a touch event).

Turbo Mode

The Turbo mode allows two Class 1 analog inputs to work in an interleaved manner to generate converted data at almost double the maximum throughput of a single Class 1 analog input.

FIFO and DMA Interfaces for Channels 0 through 6

The FIFO and DMA interfaces are applicable only for Channel 0 through 6. Using the FIFO, Channel 0 through 6 can save converted data into a FIFO. Using the DMA interface, Channel 0 through 6 can save the converted data directly into RAM.

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Configuration Functions

	Name	Description
	PLIB_ADCHS_Disable	High-Speed SAR ADC module is disabled (turned OFF).
	PLIB_ADCHS_Enable	Enables the High-Speed SAR ADC module.
	PLIB_ADCHS_Setup	Configures the High-Speed SAR ADC converter.
	PLIB_ADCHS_ControlRegistersCanBeUpdated	Returns the status of update-ready.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable	Disables the update-ready interrupt.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable	Enables the update-ready interrupt.
	PLIB_ADCHS_ExternalConversionRequestDisable	Disables the external conversion request.
	PLIB_ADCHS_ExternalConversionRequestEnable	Enables the external conversion request.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable	Disables the global level software trigger.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable	Initiates a global level software trigger.
	PLIB_ADCHS_GlobalSoftwareTriggerEnable	Initiate a Global Software Trigger.
	PLIB_ADCHS_ScanCompleteInterruptDisable	Disables the end of scan interrupt.
	PLIB_ADCHS_ScanCompleteInterruptEnable	Enables the end of scan interrupt.
	PLIB_ADCHS_SoftwareConversionInputSelect	Selects the analog input of Channel 7 for manual conversion.
	PLIB_ADCHS_SoftwareConversionStart	Triggers the conversion of analog input signal connected to Channel 7.
	PLIB_ADCHS_SoftwareSamplingStart	Enables sampling of analog input for Channel 7.

	PLIB_ADCHS_SoftwareSamplingStop	Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.
	PLIB_ADCHS_TriggerSuspendDisable	Disables trigger suspend.
	PLIB_ADCHS_TriggerSuspendEnable	Suspends all trigger for all ADCHS channels.

b) CVD Functions

	Name	Description
	PLIB_ADCHS_CVDDisable	Disables the CVD feature.
	PLIB_ADCHS_CVDEnable	Enables the CVD feature.
	PLIB_ADCHS_CVDResultGet	Returns a CVD result measured by an ADCHS instance.
	PLIB_ADCHS_CVDSetup	Configures the CVD related setting of ADCHS channel.

c) Analog Input Functions

	Name	Description
	PLIB_ADCHS_AnalogInputDataIsReady	Returns the data ready status of analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptDisable	Disables the data ready interrupt for the selected analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptEnable	Enables the data ready interrupt for the selected analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptDisable	Disables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptEnable	Enables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptIsReady	Returns the early interrupt ready status of analog input.
	PLIB_ADCHS_AnalogInputIsAvailable	Returns the analog input configuration of ADCHS channel.
	PLIB_ADCHS_AnalogInputModeGet	Returns the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputResultGet	Returns a ADC conversion result.
	PLIB_ADCHS_AnalogInputScanIsComplete	Returns the state of End of scan completion.
	PLIB_ADCHS_AnalogInputScanIsSelected	Returns whether or note an analog input is selected for scanning.
	PLIB_ADCHS_AnalogInputScanRemove	Removes the analog input from scanning selection.

d) Analog Input Selection Functions

	Name	Description
	PLIB_ADCHS_AnalogInputModeSelect	Selects the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputScanSelect	Selects the analog input for scanning.
	PLIB_ADCHS_ChannelTriggerSampleSelect	Selects the trigger and sampling modes for channels of High-Speed SAR ADC
	PLIB_ADCHS_AnalogInputEdgeTriggerSet	Sets the trigger as edge sensitive for selected class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputLevelTriggerSet	Sets the trigger as level sensitive for selected Class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputScanSetup	Selects input to include in Analog Input Scan mode.
	PLIB_ADCHS_AnalogInputTriggerSourceSelect	Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).

e) Digital Comparator Functions

	Name	Description
	PLIB_ADCHS_DigitalComparatorAnalogInputGet	Returns the analog input ID used by the digital comparator.
	PLIB_ADCHS_DigitalComparatorAnalogInputSelect	Selects analog inputs, whose converted data will be processed by the comparator.
	PLIB_ADCHS_DigitalComparatorDisable	Disables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEnable	Enables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEventHasOccurred	Returns the status of the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptDisable	Disables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptEnable	Enables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorLimitSet	Sets the limit for the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorSetup	Configures the Digital Comparator on the High-Speed SAR ADC converter.

f) Digital Filter Functions

	Name	Description
	PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect	Selects the number of samples which are averaged by the Digital Filter.
	PLIB_ADCHS_DigitalFilterAveragingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.
	PLIB_ADCHS_DigitalFilterDataGet	Used to fetch the data result from the Digital Filter.

	PLIB_ADCHS_DigitalFilterDataIsReady	Used to determine if the Digital Filter has data ready.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable	Disables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable	Enables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDisable	Disables the Digital Filter.
	PLIB_ADCHS_DigitalFilterEnable	Enables the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect	Selects the oversampling ratio for the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

g) DMA Functions

	Name	Description
	PLIB_ADCHS_DMABuffer_A_InterruptDisable	Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_InterruptEnable	Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_IsFull	Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptDisable	Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptEnable	Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_IsFull	Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMADisable	Disables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAEnable	Enables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAOverflowErrorHasOccurred	Used to determine if the DMA Buff had an overflow error.
	PLIB_ADCHS_DMASetup	Configures the DMA on the High-Speed SAR ADC.
	PLIB_ADCHS_DMASourceRemove	Disables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_DMASourceSelect	Enables the DMA for the specified Channel 0 to 6.

h) Channel Related Functions

	Name	Description
	PLIB_ADCHS_ChannelAnalogFeatureDisable	Disables the analog circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelAnalogFeatureEnable	Enables the analog circuit for High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelConfigurationGet	Used to get the configuration for the specified channel.
	PLIB_ADCHS_ChannelConfigurationSet	Used to set the configuration for the specified channel.
	PLIB_ADCHS_ChannelDigitalFeatureDisable	Disables the digital circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelDigitalFeatureEnable	Enables (turns ON) the digital circuit for channels.
	PLIB_ADCHS_ChannelIsReady	Returns the state of the channel.
	PLIB_ADCHS_ChannelIsReadyInterruptDisable	Disables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelIsReadyInterruptEnable	Enables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelSetup	Configures the High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelInputSelect	Selects the analog input for Channel 0 to 6.

i) FIFO Functions

	Name	Description
	PLIB_ADCHS_FIFODataCountGet	Returns the number of data to be read from FIFO.
	PLIB_ADCHS_FIFODataIsAvailable	Used to determine if the FIFO has data ready.
	PLIB_ADCHS_FIFODataIsNegative	Returns the sign of data stored in FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptDisable	Disables the interrupt for FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptEnable	Enables the interrupt for FIFO.
	PLIB_ADCHS_FIFODisable	Disables the FIFO in the High-Speed SAR ADC.
	PLIB_ADCHS_FIFOEnable	Enables the FIFO in the High-Speed SAR ADC
	PLIB_ADCHS_FIFOErrorHasOccurred	Used to determine if the FIFO has encountered an overflow error.
	PLIB_ADCHS_FIFORead	Used to fetch the data result from the FIFO.
	PLIB_ADCHS_FIFOSourceGet	Returns the channel ID using the FIFO.
	PLIB_ADCHS_FIFOSourceSelect	Sets the Channel 0 to 6 using the FIFO.

j) Interrupt Functions

	Name	Description
	PLIB_ADCHS_EarlyInterruptDisable	Disables the early interrupt for the ADCHS.
	PLIB_ADCHS_EarlyInterruptEnable	Enables the early interrupt for the ADCHS.

k) Turbo Mode Functions

	Name	Description
	PLIB_ADCHS_TurboModeChannelSelect	Configures the channels for Turbo mode.
	PLIB_ADCHS_TurboModeDisable	Disables Turbo mode for High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeEnable	Enables Turbo mode for the High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeErrorHasOccurred	Returns the error state of Turbo mode.

l) Voltage Reference Functions

	Name	Description
	PLIB_ADCHS_VREFFaultHasOccurred	Returns the state of VREF fault.
	PLIB_ADCHS_VREFFaultInterruptDisable	Disables the VREF Fault interrupt.
	PLIB_ADCHS_VREFFaultInterruptEnable	Enables the VREF fault interrupt.
	PLIB_ADCHS_VREFIsReady	Returns the state of VREF.
	PLIB_ADCHS_VREFReadyInterruptDisable	Disables the VREF ready interrupt.
	PLIB_ADCHS_VREFReadyInterruptEnable	Enables the VREF ready interrupt.

m) Feature Existence Functions

	Name	Description
	PLIB_ADCHS_ExistsAnalogInputCheck	Identifies whether the System Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputModeControl	Identifies whether the analog input mode control exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputScan	Identifies whether the Analog input Scan exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelAnalogControl	Identifies whether the Channel Analog control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelConfiguration	Identifies whether the Channel Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelDigitalControl	Identifies whether the Channel Digital control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelInputSelectControl	Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConversionResults	Identifies whether the Conversion Results feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsCVD	Identifies whether the CVD exists on the ADCHS module.
	PLIB_ADCHS_ExistsDigitalComparator	Identifies whether the Digital Comparator feature exists on the ADCHS module .
	PLIB_ADCHS_ExistsDigitalFilter	Identifies whether the Digital Filter feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsDMA	Identifies whether the DMA exists on the ADCHS module.
	PLIB_ADCHS_ExistsEarlyInterruptControl	Identifies whether the Early Interrupt control exists on the ADCHS module.
	PLIB_ADCHS_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCHS module
	PLIB_ADCHS_ExistsExternalConversionRequestControl	Identifies whether the External Convert feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsFIFO	Identifies whether the FIFO exists on the ADCHS module.
	PLIB_ADCHS_ExistsManualControl	Identifies whether the Manual control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerControl	Identifies whether the Trigger control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerSampleControl	Identifies whether the Trigger Sample control feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsTurboMode	Identifies whether the Turbo mode feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsUpdateReadyControl	Identifies whether the Update Ready feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsVREFControl	Identifies whether the VREF control exists on the ADCHS module.

n) Data Types and Constants

	Name	Description
	ADCHS_AN_INPUT_ID	Type for identifying the available ADC Inputs
	ADCHS_CHARGEPUMP_MODE	Defines the selection for the charge pump.
	ADCHS_CLOCK_SOURCE	Defines the ADCHS Clock Source Select.
	ADCHS_CVD_CAPACITOR	Defines the value of the internal capacitor during CVD mode.
	ADCHS_DATA_RESOLUTION	Identifies the resolution of the ADC output.
	ADCHS_DIGITAL_COMPARATOR_ID	Identifies the supported Digital Comparators.
	ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT	Identifies the Digital Filter averaging sample count.
	ADCHS_DIGITAL_FILTER_ID	Identifies the supported Digital Filters.
	ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO	Identifies the supported Digital Filter oversampling ratios.
	ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS	Data length of digital filter.
	ADCHS_DMA_BUFFER_LENGTH	Defines the length of the DMA buffer length.
	ADCHS_DMA_COUNT	Defines the enable/disable of the count feature for DMA.
	ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK	Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.
	ADCHS_FAST_SYNC_PERIPHERAL_CLOCK	Defines the selection of the fast synchronous peripheral clock.
	ADCHS_FAST_SYNC_SYSTEM_CLOCK	Defines the selection of the fast synchronous system clock.
	ADCHS_INPUT_MODE	Defines the available modes for the selected input.
	ADCHS_INTERRUPT_BIT_SHIFT_LEFT	Identifies the bits shift for calculating the interrupt vector.
	ADCHS_OUTPUT_DATA_FORMAT	Defines the selection for the output data format.
	ADCHS_SCAN_TRIGGER_SENSITIVE	Trigger level for scan trigger.
	ADCHS_SCAN_TRIGGER_SOURCE	Defines the ADCHS Channel Scan Trigger Source Selections.
	ADCHS_TRIGGER_SOURCE	Defines the ADCHS Trigger Source Selections.
	ADCHS_VREF_SOURCE	Defines the ADCHS VREF Source Select.
	ADCHS_WARMUP_CLOCK	Identifies the number of clocks before the channel can be ready
	ADCHS_MODULE_ID	Identifies the number of ADC supported.
	ADCHS_CHANNEL_INP_SEL	Defines the alternate input selection for channels 0 to 6.
	ADCHS_CHANNEL_ID	Identifies the ADC channel from 0 to 7.
	ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL	Defines the selection of trigger and sampling for channels 0 to 6.
	ADCHS_CLASS12_AN_INPUT_ID	Type for identifying the available Class 1 and 2 analog Inputs.

Description

This section describes the Application Programming Interface (API) functions of the ADCHS Peripheral Library.

Refer to each section for a detailed description.

a) Configuration Functions

PLIB_ADCHS_Disable Function

High-Speed SAR ADC module is disabled (turned OFF).

File

plib_adchs.h

C

```
void PLIB_ADCHS_Disable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected High-Speed SAR ADC module.

Remarks

Not all functionality is available on all devices. Please refer to the specific device data sheet for the list of available features.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_Disable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_Disable(ADCHS_MODULE_ID index)

PLIB_ADCHS_Enable Function

Enables the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_Enable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected High-Speed SAR ADC module. There are many ADCHS functionalities which should be set/ selected before the ADCHS module is turned ON. Once the ADCHS is turned ON, the application code should check if the VREF is ready and without any fault. Subsequently, the analog bias circuitry for the required channel should be enabled followed by enabling the digital section.

Remarks

None.

Preconditions

All channels and analog input related selections should be made before enabling the ADCHS module.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_Enable(ADCHS_MODULE_ID index)

PLIB_ADCHS_Setup Function

Configures the High-Speed SAR ADC converter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_Setup(ADCHS_MODULE_ID index, ADCHS_VREF_SOURCE voltageRefSelect, ADCHS_CHARGEPUMP_MODE
chargePump, ADCHS_OUTPUT_DATA_FORMAT outputFormat, bool stopInIdle, ADCHS_FAST_SYNC_SYSTEM_CLOCK sysClk,
```

```
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK periClk, ADCHS_INTERRUPT_BIT_SHIFT_LEFT intVectorShift, uint16_t
intVectorBase, ADCHS_CLOCK_SOURCE adcClockSource, int8_t adcClockDivider, ADCHS_WARMUP_CLOCK warmUpClock);
```

Returns

None.

Description

Configures all ADC parameters which are common to all ADC channels (from 0 to 7). This configuration must occur prior to enabling the ADC and therefore must be called when the ADC is disabled.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The ADCHS module is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_Setup
(
    MY_ADCHS_INSTANCE,
    ADCHS_VREF_AVDD_AVSS,                   // AVDD and AVSS as reference
    ADCHS_CHARGEPUMP_DISABLE,               // Charge pump is disabled
    ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,    // Use fractional format
    true,                                   // Stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,    // Enable Fast synchronous system clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS,  // vector left shift set to 2
    0x1600,                                 // vector base address set to 0x1600
    ADCHS_CLOCK_SOURCE_SYSCLK,              // SYSCLK is the clock source
    1,                                      // Control clock, TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                   // Warm-up up time = 32 clocks
);

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
voltageRefSelect	VREF Source Selection.
chargePump	Enables/ disables the charge pump. This variable is of ADCHS_CHARGEPUMP_MODE type. Use when
VREFH	VREFL < .64 (AVDD - AVSS). This feature is not available on all devices.
outputFormat	Sets output data format to be integer or fractional. This parameter is of ADCHS_OUTPUT_DATA_FORMAT type.
stopInIdle	Sets ADC to stop when device is in Idle mode if true
sysClk	Sets the Fast synchronous system clock to ADC control clock. This variable is of type ADCHS_FAST_SYNC_SYSTEM_CLOCK.
periClk	Sets the Fast synchronous peripheral clock to ADC control clock. This variable is of type ADCHS_FAST_SYNC_PERIPHERAL_CLOCK.
intVectorShift	Sets the interrupt vector shift left shift. This variable is of ADCHS_INTERRUPT_BIT_SHIFT_LEFT type.
intVectorBase	Sets the interrupt vector base address.
adcClockSource	Clock source selection. This variable is of type ADCHS_CLOCK_SOURCE.
adcClockDivider	Clock source divider. Values range from 0 to 63. This divider determines the input clock frequency which goes as input to all ADC channels (all ADC channels, 0 to 7). This clock is also known as the "control clock" of ADC. The frequency of control
clock for ADC, as per the following equation	Control clock frequency = (frequency of "adcClockSource")/(adcClockDivider * 2) For adcClockDivider = 0, control clock frequency is same as frequency of "adcClockSource".

warmUpClock	This parameter determines the number of channel clock (not control clock) which is required as warm up time for the ADC channel, once the analog feature of that ADC channel is enabled. The variable is of type ADCHS_WARMUP_CLOCK.

Function

```
void PLIB_ADCHS_Setup
(
    ADCHS_MODULE_ID index,
    ADCHS_VREF_SOURCE voltageRefSelect,    // voltage reference
    ADCHS_CHARGEPUMP_MODE chargePump,      // Charge pump enabled/disabled
    ADCHS_OUTPUT_DATA_FORMAT outputFormat,  // result format fractional
bool stopInIdle,                   // stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK sysClk,   // Fast synchronous system clock
// to ADC control clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK periClk,      // Fast synchronous peripheral
// clock to ADC control clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT intVectorShift,  // Interrupt vector shift bits
uint16_t intVectorBase,            // Interrupt vector base address
    ADCHS_CLOCK_SOURCE adcClockSource,     // clock source
int8_t adcClockDivider,            // clock divider
    ADCHS_WARMUP_CLOCK warmUpClock         // Analog channel warm-up clocks.
)
```

PLIB_ADCHS_ControlRegistersCanBeUpdated Function

Returns the status of update-ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ControlRegistersCanBeUpdated(ADCHS_MODULE_ID index);
```

Returns

- true - The ADCHS module can be safely updated (settings can be configured)
- false - The ADCHS module is not yet ready to be safely updated

Description

Once the triggers are suspended, the ADCHS raises an update-ready flag indicating that the updates to ADCHS registers can be performed. This function returns the status of update ready.

Remarks

None.

Preconditions

None.

Example

```
// Suspend trigger
PLIB_ADCHS_TriggerSuspendEnable(MY_ADCHS_INSTANCE);

// Wait until the channel registers can be updated
while(!PLIB_ADCHS_ControlRegistersCanBeUpdated(MY_ADCHS_INSTANCE));

// Once the function returns true, update the configurations of ADCHS
//....

// Enable triggers
PLIB_ADCHS_TriggerSuspendDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_ControlRegistersCanBeUpdated(ADCHS_MODULE_ID index)

PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable Function

Disables the update-ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the update-ready interrupt.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable Function

Enables the update-ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

Once the triggers are suspended, the ADC channel raises an update ready flag indicating that the updates to ADCHS registers can be performed. This function enables the interrupt which occurs once the flag is raised.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExternalConversionRequestDisable Function

Disables the external conversion request.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ExternalConversionRequestDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the external conversion request. Once disabled, the external sources such as PTG cannot request a conversion.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable external conversion request
PLIB_ADCHS_ExternalConversionRequestDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ExternalConversionRequestDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExternalConversionRequestEnable Function

Enables the external conversion request.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ExternalConversionRequestEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the external conversion request. Once enabled, the ADCHS can accept conversion request from external sources

such as the PTG module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable external conversion request
PLIB_ADCHS_ExternalConversionRequestEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ExternalConversionRequestEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable Function

Disables the global level software trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the global level software trigger on the specified channel.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable Function

Initiates a global level software trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function initiates a global level software trigger. All inputs or the scan list configured to trigger on the global level software trigger are triggered. Once initiated, the trigger continues until the global level trigger is disabled using the function PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalSoftwareTriggerEnable Function

Initiate a Global Software Trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalSoftwareTriggerEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

All inputs or scan list that is configured to trigger on the global software trigger are triggered. Once enabled, the trigger automatically gets disabled (edge sensitive).

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalSoftwareTriggerEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalSoftwareTriggerEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ScanCompleteInterruptDisable Function

Disables the end of scan interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ScanCompleteInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the end of scan interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable EOS interrupt
PLIB_ADCHS_ScanCompleteInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ScanCompleteInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ScanCompleteInterruptEnable Function

Enables the end of scan interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ScanCompleteInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the end of scan interrupt. The interrupt is generated when scanning of all of the selected analog inputs has completed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
```

```
// application developer.

// Enable EOS interrupt
PLIB_ADCHS_ScanCompleteInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ScanCompleteInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareConversionInputSelect Function

Selects the analog input of Channel 7 for manual conversion.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareConversionInputSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID anInput);
```

Returns

None.

Description

This function selects the analog input for Channel 7, which is used for manual conversion (using software control).

Remarks

None.

Preconditions

None.

Example

```
// Select AN24 for manual conversion
PLIB_ADCHS_SoftwareConversionInputSelect(MY_ADCHS_INSTANCE, ADCHS_AN24);

// Place S&H into Sampling mode
PLIB_ADCHS_SoftwareSamplingStart(MY_ADCHS_INSTANCE);

// Wait for the required sampling time
int del;
for(del = 0; del<=20; del++);

// Now, make the S&H into Hold mode
PLIB_ADCHS_SoftwareSamplingStop(MY_ADCHS_INSTANCE);

// Now, start conversion
PLIB_ADCHS_SoftwareConversionStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
anInput	Analog input of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_SoftwareConversionInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID anInput

)

PLIB_ADCHS_SoftwareConversionStart Function

Triggers the conversion of analog input signal connected to Channel 7.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareConversionStart(ADCHS_MODULE_ID index);
```

Returns

None.

Description

After a signal is held on the S&H of Channel 7, the conversion can be triggered manually using software by using this function.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareConversionStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareConversionStart(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareSamplingStart Function

Enables sampling of analog input for Channel 7.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareSamplingStart(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function can be used if Channel 7 needs to be brought to Sampling mode manually (i.e., through software).

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareSamplingStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareSamplingStart(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareSamplingStop Function

Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareSamplingStop(ADCHS_MODULE_ID index);
```

Returns

None.

Description

After sampling for a significant time (using the function PLIB_ADCHS_SoftwareSamplingStart), the sampling can be disabled, which will bring the channel to Hold mode.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareSamplingStop(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareSamplingStop(ADCHS_MODULE_ID index)

PLIB_ADCHS_TriggerSuspendDisable Function

Disables trigger suspend.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TriggerSuspendDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

By calling this function, all triggers are allowed to reach the channel and triggers are not suspended.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_TriggerSuspendDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_TriggerSuspendDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TriggerSuspendEnable Function

Suspends all trigger for all ADCHS channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TriggerSuspendEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function suspends all triggers for all channels on the ADCHS module. The trigger needs to be suspended so that the channels could be configured without disabling (and subsequently enabling) the channels.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_TriggerSuspendEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_TriggerSuspendEnable(ADCHS_MODULE_ID index)

b) CVD Functions

PLIB_ADCHS_CVDDisable Function

Disables the CVD feature.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the CVD feature of an ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_CVDDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_CVDDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDEnable Function

Enables the CVD feature.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

ADCHS channel supports the Capacitive Voltage Divider (CVD) mode. This function enables the CVD feature of an ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_CVDEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_CVDEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDResultGet Function

Returns a CVD result measured by an ADCHS instance.

File

plib_adchs.h

C

```
int16_t PLIB_ADCHS_CVDResultGet(ADCHS_MODULE_ID index);
```

Returns

The conversion result expressed as a signed 16-bit integer.

Description

During CVD measurement, the difference between the voltage measured during positive and negative phases is calculated. This function return the difference value.

Remarks

None.

Preconditions

CVD must be enabled and a comparator event linked to CVD should have occurred.

Example

```
int16_t     result;
...
// fetch result for CVD
result = PLIB_ADCHS_CVDResultGet( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

int16_t PLIB_ADCHS_CVDResultGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDSetup Function

Configures the CVD related setting of ADCHS channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDSetup(ADCHS_MODULE_ID index, ADCHS_CVD_CAPACITOR capSel, bool inBetweenOrEqual, bool
greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool lessThanLo, ADCHS_AN_INPUT_ID inputEnable,
int16_t hiLimit, int16_t loLimit);
```

Returns

None.

Description

While selecting the CVD mode, the internal capacitor should be similar in magnitude to the external capacitor being sensed. This function configures the internal capacitance. Additionally, the functions sets the limits for CVD measurement and the condition to detect a touch event. Since, CVD is implemented with digital comparator 0, using the CVD functionality also needs that digital comparator is enabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the CVD mode
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output. This will be the touch event.

PLIB_ADCHS_CVDSetup(    MY_ADCHS_INSTANCE,
                          ADCHS_CVD_CAPACITOR_5PF,
                          false,            // no test for between low and high
                          true,             // Once touch event occurs, CVD output will
                                            // be higher than the set limits
                          false,            // no test for less than high
```

```
                                false,          // no test for greater than equal to low
                                false,          // no test for less than low
                                ADCHS_AN3,      // enable AN3
                                0x0CCD,         // high limit, 80% of full scale
                                0x0333);        // low limit, 20% of full scale
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
capSel	Selection of value of input capacitor of type ADCHS_CVD_CAPACITOR.
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit.
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit.
lessThanHi	Event is generated when result is less than hiLimit.
greaterEqualLo	Event is generated when result is greater than or equal to loLimit.
lessThanLo	Event is generated when result is less than loLimit.
inputEnable	Bit field which determines which analog inputs are tested by this comparator. This parameter is of type ADCHS_AN_INPUT_ID.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCHS_CVDSetup
(
    ADCHS_MODULE_ID index,
    ADCHS_CVD_CAPACITOR capSel,     // Input capacitor
bool inBetweenOrEqual,          // between low and high
bool greaterEqualHi,            // greater than equal to high
bool lessThanHi,                // less than high
bool greaterEqualLo,            // greater than equal to low
bool lessThanLo,                // less than low
    ADCHS_AN_INPUT_ID inputEnable,  // input enable bit
int16_t  hiLimit,               // high limit
int16_t  loLimit                // low limit
)
```

c) Analog Input Functions

PLIB_ADCHS_AnalogInputDataIsReady Function

Returns the data ready status of analog inputs.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputDataIsReady(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Converted data is ready for selected analog input
- false - Conversion is not yet complete and data is not ready

Description

This function returns if the converted data is ready for the analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Check if data is ready for AN10
if(PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10) == true)
{
    // Do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputDataIsReady
(
    ADCHS_MODULE_ID index,
    ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputDataReadyInterruptDisable Function

Disables the data ready interrupt for the selected analog inputs.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function disables (turns OFF) the data ready interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputDataReadyInterruptDisable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputDataReadyInterruptEnable Function

Enables the data ready interrupt for the selected analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function enables (turns ON) the data ready interrupt for the selected analog input. The interrupt is generated when the converted data is ready for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputDataReadyInterruptEnable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputEarlyInterruptDisable Function

Disables the early interrupt for the analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEarlyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function disables (turns OFF) the early interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputEarlyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputEarlyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputEarlyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_AnalogInputEarlyInterruptEnable Function

Enables the early interrupt for the analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEarlyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function enables (turns ON) the early interrupt for the selected analog input. The interrupt is generated at a set number of clock prior to conversion complete. The number of clock for early interrupt is set using the "configure" function for a specific channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
```

```
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputEarlyInterruptEnable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputEarlyInterruptIsReady Function

Returns the early interrupt ready status of analog input.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputEarlyInterruptIsReady(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Early interrupt event is ready
- false - Early interrupt event has not occurred

Description

This function returns the status of early interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.


// Check if early interrupt is ready for AN10
if(PLIB_ADCHS_AnalogInputEarlyInterruptIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    // Do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputEarlyInterruptIsReady
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputIsAvailable Function

Returns the analog input configuration of ADCHS channel.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputIsAvailable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Selected analog input is available.
- false - Selected analog input is not available in the ADCHS channel.

Description

This function returns the if the selected analog input is available in the ADCHS channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.


// Check if AN10 is available in the ADCHS channel
if(PLIB_ADCHS_AnalogInputIsAvailable(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    // do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputIsAvailable
(
    ADCHS_MODULE_ID index,
    ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputModeGet Function

Returns the mode for the specified analog input.

File

plib_adchs.h

C

```
ADCHS_INPUT_MODE PLIB_ADCHS_AnalogInputModeGet(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

Mode of the selected analog input of type ADCHS_INPUT_MODE type.

Description

This function returns the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified analog input.

When getting the mode of analog input which is meant to be alternate input for an ADC channel, the default analog input ID should be passed to this function.

For example, for CHANNEL_0, the default analog input is AN0 and alternate analog input is AN45. If the mode for AN45 needs to be obtained, the default input ID (i.e., AN0) should be passed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Declare a variable of type "ADCHS_INPUT_MODE"
ADCHS_INPUT_MODE mode;

// Get mode for AN10
mode = PLIB_ADCHS_AnalogInputModeGet(MY_ADCHS_INSTANCE, ADCHS_AN10);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID

Function

```
 ADCHS_INPUT_MODE PLIB_ADCHS_AnalogInputModeGet
(
    ADCHS_MODULE_ID index,
    ADCHS_AN_INPUT_ID analogInput
)
```

PLIB_ADCHS_AnalogInputResultGet Function

Returns a ADC conversion result.

File

plib_adchs.h

C

```
uint32_t PLIB_ADCHS_AnalogInputResultGet(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

The conversion result expressed as a 32-bit integer.

Description

This function returns the converted result for the specified analog input.

Remarks

None.

Preconditions

A valid conversion is ready to be fetched.

Example

```
uint32_t        result;
//...
```

```
// Check if data is ready for AN10
if(PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    result = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN10 );
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

uint32_t PLIB_ADCHS_AnalogInputResultGet
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputScanIsComplete Function

Returns the state of End of scan completion.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputScanIsComplete(ADCHS_MODULE_ID index);
```

Returns

- true - Analog input scanning is complete for the selected analog inputs
- false - Analog input scanning is not complete

Description

This function returns 'true', if all of the selected Analog Inputs have completed scanning.

Remarks

None.

Preconditions

The ADCHS module should have Analog Inputs Scanning mode selected and enabled for scanning to occur.

Example

```
bool EOSState = PLIB_ADCHS_AnalogInputScanIsComplete(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_AnalogInputScanIsComplete(ADCHS_MODULE_ID index)

PLIB_ADCHS_AnalogInputScanIsSelected Function

Returns whether or note an analog input is selected for scanning.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputScanIsSelected(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Analog input is in selected for scanning
- false - Analog input is not selected for scanning

Description

This function returns whether or not an analog input is included in the scanning queue.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Determine if AN10 is selected for scanning or not
if(PLIB_ADCHS_AnalogInputScanIsSelected(MY_ADCHS_INSTANCE, ADCHS_AN10)  == true)
{
    // Perform further operations
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputScanIsSelected

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_AnalogInputScanRemove Function

Removes the analog input from scanning selection.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanRemove(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function removes the analog input for scanning queue. Therefore, the analog input is no longer used for scanning.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Remove AN10 and AN11 from scanning
PLIB_ADCHS_AnalogInputScanRemove(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputScanRemove(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputScanRemove

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

d) Analog Input Selection Functions

PLIB_ADCHS_AnalogInputModeSelect Function

Selects the mode for the specified analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputModeSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput,
ADCHS_INPUT_MODE mode);
```

Returns

None.

Description

This function selects the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified analog input.

When selecting the mode of analog input which is meant to be alternate input for an ADC channel, the default analog input ID should be passed to this function.

For example, for CHANNEL_0, the default analog input is AN0 and the alternate analog input is AN45. If the mode for AN45 needs to be changed, the default input ID (i.e., AN0) should be passed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set AN10 and AN11 to use the single ended, two's complement mode
// selection.
PLIB_ADCHS_AnalogInputModeSelect(MY_ADCHS_INSTANCE,
                        ADCHS_AN10,
                        ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP);

PLIB_ADCHS_AnalogInputModeSelect(MY_ADCHS_INSTANCE,
```

```
                        ADCHS_AN11,
                        ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance
analogInput	Analog input ID for which the input mode is to be selected. It is of type ADCHS_AN_INPUT_ID.
mode	An ADCHS_INPUT_MODE type indicating the mode selection

Function

void PLIB_ADCHS_AnalogInputModeSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput,

ADCHS_INPUT_MODE mode

)

PLIB_ADCHS_AnalogInputScanSelect Function

Selects the analog input for scanning.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function selects the analog input for scanning.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Select for AN10 and AN11 for scanning
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputScanSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_ChannelTriggerSampleSelect Function

Selects the trigger and sampling modes for channels of High-Speed SAR ADC

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelTriggerSampleSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID,
ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL sampSel);
```

Returns

- true - The function successfully selected the sampling and trigger mode
- false - The function could not select the sampling and trigger mode because Channel 7 was selected and Channel 7 does not implement this feature

Description

The channels from 0 through 6 have features to select the trigger as presynchronized or unsynchronized. Also, the sampling can be selected as synchronous. This function provides the functionality to select the trigger and sampling for ADC channels. The valid channels for selecting trigger and sampling is 0 to 6. Channel 7 does not have the feature. Therefore, calling this function for Channel 7 will result in failure. This selection must occur prior to enabling the ADC and therefore must be called when the ADC is disabled. Also, this selection must occur prior to enabling the digital circuit and analog bias circuit for the specified ADC channel.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channel is disabled when calling this function.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_ChannelSetup( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
        1,                                      // channel clock divider bit is, TAD =  2 * TQ
        1,                                      // Sample time is 3 * TAD
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1);   // Interrupt, 1 clock early

status = PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING);           // Synchronous sampling selected

if( false == status)
{
    // error has occurred
    while(1);
}

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);

// Wait until the reference voltage is ready
while(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));

//Check if there is a fault in reference voltage
if(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE))
{
    // process fault accordingly
    while(1);
}
```

```
// Enable the analog circuit
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// Wait until the channel wakes up after warm-up
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));

// Enable the digital circuit
PLIB_ADCHS_ChannelDigitalFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// ADC will begin conversion now, after a valid trigger
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.
sampSel	Sets the presync trigger or sync sampling options. This variable is of type ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL.

Function

bool PLIB_ADCHS_ChannelTriggerSampleSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID,

ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL sampSel

)

PLIB_ADCHS_AnalogInputEdgeTriggerSet Function

Sets the trigger as edge sensitive for selected class 1 and 2 analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEdgeTriggerSet(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function sets the trigger as edge sensitive for selected Class 1 and 2 analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set for AN10 and AN11 for edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN10 );
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN11 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. Since the trigger level can be selected only for Class 1 and 2 analog inputs.

Function

void PLIB_ADCHS_AnalogInputEdgeTriggerSet
(
ADCHS_MODULE_ID index,
ADCHS_CLASS12_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputLevelTriggerSet Function

Sets the trigger as level sensitive for selected Class 1 and 2 analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputLevelTriggerSet(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function sets the trigger as level sensitive for selected Class 1 and 2 analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set for AN5 and AN6 for level trigger
PLIB_ADCHS_AnalogInputLevelTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN5 );
PLIB_ADCHS_AnalogInputLevelTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN6 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. Since the trigger level can be selected only for Class 1 and 2 analog inputs.

Function

void PLIB_ADCHS_AnalogInputLevelTriggerSet
(
ADCHS_MODULE_ID index,
ADCHS_CLASS12_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputScanSetup Function

Selects input to include in Analog Input Scan mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanSetup(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput,
ADCHS_SCAN_TRIGGER_SENSITIVE trgSensitive, ADCHS_SCAN_TRIGGER_SOURCE triggerSource);
```

Returns

None.

Description

Selects inputs, as determined by the low and high enable scan list for inclusion in the analog input scan sequence. If the input is a Class 1 or Class 2 type, it will also select the trigger source for that input to be the scan trigger, which is required if included in analog input scanning. This function configures the scan functionality with a single ADC input. If more inputs are required to be added for scanning, the PLIB_ADCHS_AnalogInputScanSelect function should be used.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure analog input Scanning
// Analog inputs 10 through 13
// Trigger on Timer 1 match.

PLIB_ADCHS_AnalogInputScanSetup(MY_ADCHS_INSTANCE,
                                ADCHS_AN10,
                                ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,   // Edge sensitive
                                ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH);

PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN11);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN12);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN13);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.
trgSensitive	Enables level/edge sensitive scan trigger.
triggerSource	Trigger source used to initiate analog input scan.

Function

void PLIB_ADCHS_AnalogInputScanSetup

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput,

ADCHS_SCAN_TRIGGER_SENSITIVE trgSensitive,

ADCHS_SCAN_TRIGGER_SOURCE triggerSource

)

PLIB_ADCHS_AnalogInputTriggerSourceSelect Function

Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputTriggerSourceSelect(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID inputId,
ADCHS_TRIGGER_SOURCE triggerSource);
```

Returns

None.

Description

This function selects the trigger source for Class 1 or Class 2 inputs. A call to this function is not required when the input is being used as part of an analog input scan, as the input scan configure function also configures all trigger sources.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure AN4 for triggering from INT0.
PLIB_ADCHS_AnalogInputTriggerSourceSelect( MY_ADCHS_INSTANCE,
                                   ADCHS_CLASS12_AN4,
                                   ADCHS_TRIGGER_SOURCE_INT0 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
inputId	Class 1 or Class 2 input trigger to be configured.
triggerSource	Trigger source to use for this input of type ADCHS_TRIGGER_SOURCE.

Function

void PLIB_ADCHS_AnalogInputTriggerSourceSelect

(

ADCHS_MODULE_ID index,

ADCHS_CLASS12_AN_INPUT_ID inputId,

ADCHS_TRIGGER_SOURCE triggerSource

)

e) Digital Comparator Functions

PLIB_ADCHS_DigitalComparatorAnalogInputGet Function

Returns the analog input ID used by the digital comparator.

File

plib_adchs.h

C

```
ADCHS_AN_INPUT_ID PLIB_ADCHS_DigitalComparatorAnalogInputGet(ADCHS_MODULE_ID index,
ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

An analog input ID of type ADCHS_AN_INPUT_ID.

Description

This function returns the analog input ID, whose converted data is being compared by the specified digital comparator. In a typical application, when a comparator event is recorded, the application will need to know the analog input which caused the event. This function returns the value of type ADCHS_AN_INPUT_ID, indicating the analog input ID.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

ADCHS_AN_INPUT_ID input;

// Get analog input tied to digital comparator 2
input = PLIB_ADCHS_DigitalComparatorAnalogInputGet( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
digComparator	Digital comparator ID of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

```
ADCHS_AN_INPUT_ID PLIB_ADCHS_DigitalComparatorAnalogInputGet
(
    ADCHS_MODULE_ID index,
    ADCHS_DIGITAL_COMPARATOR_ID digComparator
)
```

PLIB_ADCHS_DigitalComparatorAnalogInputSelect Function

Selects analog inputs, whose converted data will be processed by the comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorAnalogInputSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function selects the analog input, whose converted data (once available) will be processed by the specified digital comparator. This function should be called only when the comparator is in a disabled state.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set digital comparator 2 to process the output of AN2
PLIB_ADCHS_DigitalComparatorAnalogInputSelect( MY_ADCHS_INSTANCE,
                                    ADCHS_DIGITAL_COMPARATOR_2, ADCHS_AN2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator ID of type ADCHS_DIGITAL_COMPARATOR_ID.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. The valid input range for "analogInput" is from ADCHS_AN0 to ADCHS_AN31. Values from ADCHS_AN32 and higher will be ignored.

Function

void PLIB_ADCHS_DigitalComparatorAnalogInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_DigitalComparatorDisable Function

Disables the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

None.

Description

This function disables (turns OFF) the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_DigitalComparatorDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorEnable Function

Enables the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

None.

Description

This function enables (turns ON) the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorEventHasOccurred Function

Returns the status of the selected digital comparator.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DigitalComparatorEventHasOccurred(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id);
```

Returns

None.

Description

This function returns the status of the selected digital comparator, whether or not an event related to this comparator has occurred. A comparator event will occur if the analog input selected for the comparator is outside the set limits.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
```

```
// Check, if comparator event has occurred for digital comparator 2
if(PLIB_ADCHS_DigitalComparatorEventHasOccurred( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 ))
{
    // Perform related tasks
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator in this device.

Function

bool PLIB_ADCHS_DigitalComparatorEventHasOccurred

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID id

)

PLIB_ADCHS_DigitalComparatorInterruptDisable Function

Disables the interrupt for the selected digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorInterruptDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for the selected digital comparator. The interrupt will not be generated, once a comparator event occurs.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for comparator 2
PLIB_ADCHS_DigitalComparatorInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorInterruptEnable Function

Enables the interrupt for the selected digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorInterruptEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for the selected digital comparator. The interrupt is generated when the converted analog data for the related analog input goes out of the specified limits (set for comparator).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for comparator 2
PLIB_ADCHS_DigitalComparatorInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorLimitSet Function

Sets the limit for the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorLimitSet(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id, int16_t
hiLimit, int16_t loLimit);
```

Returns

None.

Description

This function sets the high and low limits for the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCHS_DigitalComparatorLimitSet( MY_ADCHS_INSTANCE,    // ADCHS module ID
                      ADCHS_DIGITAL_COMPARATOR_1,           // Comparator ID
                      0x0CCD,                               // high limit, 80% of full scale
                      0x0333);                              // low limit, 20% of full scale
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator in this device.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

void PLIB_ADCHS_DigitalComparatorLimitSet

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID id,

int16_t hiLimit,

int16_t loLimit

)

PLIB_ADCHS_DigitalComparatorSetup Function

Configures the Digital Comparator on the High-Speed SAR ADC converter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id, bool
intEnable, bool inBetweenOrEqual, bool greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool
lessThanLo, ADCHS_AN_INPUT_ID analogInput, int16_t hiLimit, int16_t loLimit);
```

Returns

None.

Description

This function configures all parameters for the Digital Comparator of the ADCHS module.

Remarks

This function must be called when the ADC is disabled. The format of hiLimit and loLimit must match the output format of analog input specified in analogInput.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCHS_DigitalComparatorSetup( MY_ADCHS_INSTANCE,   // ADCHS module ID
                       ADCHS_DIGITAL_COMPARATOR_1,      // Comparator ID
                       false,                           // No Int Enable
                       true,                            // test for between low and high
                       false,                           // no test for greater than equal to high
                       false,                           // no test for less than high
                       false,                           // no test for greater than equal to low
                       false,                           // no test for less than low
                       ADCHS_AN3,                       // select AN3
                       0x0CCD,                          // high limit, 80% of full scale
                       0x0333);                         // low limit, 20% of full scale

// Enable Digital Comparator 1
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator specified.
intEnable	When true, comparator events generates interrupt.
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit.
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit.
lessThanHi	Event is generated when result is less than hiLimit.
greaterEqualLo	Event is generated when result is greater than or equal to loLimit.
lessThanLo	Event is generated when result is less than loLimit.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. The valid input range for "analogInput" is from ADCHS_AN0 to ADCHS_AN31. Values from ADCHS_AN32 and higher will be ignored.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCHS_DigitalComparatorSetup
(
    ADCHS_MODULE_ID index,          // ADCHS channel ID
    ADCHS_DIGITAL_COMPARATOR_ID id,    // Comparator ID
bool intEnable,                 // Int Enable
bool inBetweenOrEqual,          // between low and high
bool greaterEqualHi,            // greater than equal to high
bool lessThanHi,                // less than high
bool greaterEqualLo,            // greater than equal to low
bool lessThanLo,                // less than low
    ADCHS_AN_INPUT_ID analogInput,     // input enable bits
int16_t  hiLimit,               // high limit
int16_t  loLimit                // low limit
)
```

f) Digital Filter Functions

PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect Function

Selects the number of samples which are averaged by the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
id, ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count);
```

Returns

None.

Description

This function selects the number of samples which are averaged by the Digital Filter, when the Digital Filter is configured for Averaging mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);

// Select the sample count to be 64 samples
PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect( MY_ADCHS_INSTANCE,      // ADCHS module ID
                              ADCHS_DIGITAL_FILTER_1,                          // Filter ID
                              ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64);   // 64 samples averaged

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
count	Sets the number of samples which will be averaged by the Digital Filter. This variable is of type. ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT.

Function

void PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect

(

ADCHS_MODULE_ID index, // ADCHS module ID

ADCHS_DIGITAL_FILTER_ID id, // Filter ID

ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count // Sample count

)

PLIB_ADCHS_DigitalFilterAveragingModeSetup Function

Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterAveragingModeSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_AN_INPUT_ID analogInput, ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count, bool intEnable);
```

Returns

None.

Description

This function configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Filter in Averaging mode
// AN4 is averaged with 64 samples.  No global interrupt is enabled.
PLIB_ADCHS_DigitalFilterAveragingModeSetup( MY_ADCHS_INSTANCE,               // ADCHS module ID
                                ADCHS_DIGITAL_FILTER_1,                     // Filter ID
                                ADCHS_AN4,                                  // Oversample AN4
                                ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
                                ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64, // 64 samples averaged
                                false );                                    // No Int Enable

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
analogInput	Identifier for the analog input to be filtered.
length	Sets the significant date length.
count	Sets the number of samples which will be averaged by Digital Filter. This variable is of type. ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT.
intEnable	When set, Filter events generated interrupt.

Function

```
void PLIB_ADCHS_DigitalFilterAveragingModeSetup
(
    ADCHS_MODULE_ID index,                  // ADCHS module ID
    ADCHS_DIGITAL_FILTER_ID id,               // Filter ID
    ADCHS_AN_INPUT_ID analogInput,              // analog input ID
    ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,    // Significant data bit
    ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count, // Sample count
bool intEnable                          // Int Enable
)
```

PLIB_ADCHS_DigitalFilterDataGet Function

Used to fetch the data result from the Digital Filter.

File

plib_adchs.h

C

```
int16_t PLIB_ADCHS_DigitalFilterDataGet(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID dfltrID);
```

Returns

A 16-bit result in the format specified by the filter's configuration setting.

Description

This function is used to fetch data from the Digital Filter.

Remarks

None.

Preconditions

This function will function only if the Digital Filter was already configured (in Oversampling mode or Averaging mode) and the Digital Filter was enabled.

Example

```
int16_t myDFLTRResult;

if (PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1)) {
    // fetch data
    myDFLTRResult = PLIB_ADCHS_DigitalFilterDataGet(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    // process result
    ...
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
dfltrID	Identifier for the Digital Filter in this device.

Function

int16_t PLIB_ADCHS_DigitalFilterDataGet

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID dfltrID

)

PLIB_ADCHS_DigitalFilterDataIsReady Function

Used to determine if the Digital Filter has data ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DigitalFilterDataIsReady(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

- true - Digital Filter has filtered data ready to be read
- false - Digital Filter data is not yet ready

Description

This function can be used to determine if the ADCHS Digital Filter has data ready. A 'true' is returned when data is available, which can be fetched using PLIB_ADCHS_DigitalFilterDataGet.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.

Function

bool PLIB_ADCHS_DigitalFilterDataIsReady

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id

)

PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable Function

Disables the interrupt for the selected Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
digFilter);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for the selected Digital Filter. The interrupt will not be generated, once filter data is ready.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for filter 2
PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_DFLTR2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digFilter	Digital Filter of type ADCHS_DIGITAL_FILTER_ID.

Function

void PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID digFilter

)

PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable Function

Enables the interrupt for the selected Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
digFilter);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for the selected Digital Filter. The interrupt is generated when the Digital Filter completes the filtering and data is ready in the register (to be read).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for filter - 2
PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_DFLTR2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digFilter	Digital Filter of type ADCHS_DIGITAL_FILTER_ID.

Function

void PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID digFilter

)

PLIB_ADCHS_DigitalFilterDisable Function

Disables the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Digital Filter.

Remarks

None.

Preconditions

None..

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter 1
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in the ADCHS channel.

Function

void PLIB_ADCHS_DigitalFilterDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id

)

PLIB_ADCHS_DigitalFilterEnable Function

Enables the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Digital Filter.

Remarks

None.

Preconditions

The ADC channel should be configured using the PLIB_ADCHS_Setup function prior to enabling.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital Filter in the ADCHS channel.

Function

void PLIB_ADCHS_DigitalFilterEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id
)

PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect Function

Selects the oversampling ratio for the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio);
```

Returns

None.

Description

This function selects the oversampling ratio for the Digital Filter. This function can be used to change the oversampling ratio of the Digital Filter, when set in Oversampling mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);

// Select the oversampling ratio
PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect( MY_ADCHS_INSTANCE,     // ADCHS module ID
                              ADCHS_DIGITAL_FILTER_1,                      // Filter ID
                              ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X);  // 16 x oversampling


// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
ratio	Sets the oversampling filter ratio. This variable is of type ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO.

Function

void PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect
(
ADCHS_MODULE_ID index, // ADCHS module ID
ADCHS_DIGITAL_FILTER_ID id, // Filter ID
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio // Oversampling ratio
)

PLIB_ADCHS_DigitalFilterOversamplingModeSetup Function

Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterOversamplingModeSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_AN_INPUT_ID analogInput, ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio, bool intEnable);
```

Returns

None.

Description

This function configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter channel is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Filter in Oversampling mode
// AN4 is oversampled at a 16X rate.  No global interrupt is enabled.
PLIB_ADCHS_DigitalFilterOversamplingModeSetup( MY_ADCHS_INSTANCE,           // ADCHS module ID
                               ADCHS_DIGITAL_FILTER_1,                     // Filter ID
                               ADCHS_AN4,                                  // Oversample AN4
                               ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
                               ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X,   // 16 x oversampling
                               false );                                    // No Int Enable

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
analogInput	Identifier for the analog input to be filtered.
length	Sets the significant data length.
ratio	Sets the oversampling filter ratio. This variable is of type ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO.
intEnable	When set, Filter events generated interrupt.

Function

```
void PLIB_ADCHS_DigitalFilterOversamplingModeSetup
(
    ADCHS_MODULE_ID index,                  // ADCHS ID
    ADCHS_DIGITAL_FILTER_ID id,             // Filter ID
    ADCHS_AN_INPUT_ID analogInput,          // analog input ID
    ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,   // Significant data bit
    ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio,    // Oversampling ratio
bool intEnable                          // Int Enable
)
```

g) DMA Functions

PLIB_ADCHS_DMABuffer_A_InterruptDisable Function

Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_A_InterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully disabled interrupt
- false - The function could not disable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA Buffer A full interrupt for the specified Channel 0 to 6. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_A_InterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_A_InterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_A_InterruptEnable Function

Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_A_InterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully enabled interrupt
- false - The function could not enable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA Buffer A full interrupt for the specified Channel 0 to 6. The interrupt is generated when the DMA buffer A is full. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_A_InterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_A_InterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_A_IsFull Function

Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
int8_t PLIB_ADCHS_DMABuffer_A_IsFull(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- 1 - Buffer is full
- 0 - Buffer is empty
- -1 - Invalid channel ID passed

Description

This function can be used to determine if the ADCHS DMA Buffer A is full for the specified Channel 0 to 6.

A value of '1' is returned when the Buffer A is full. A value of '0' is returned when the Buffer A is empty.

This feature is not implemented for Channel 7. Therefore, passing Channel 7 returns an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (1 == PLIB_ADCHS_DMABuffer_A_IsFull(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // process data, if DMA Buffer A is full
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.

Function

int8_t PLIB_ADCHS_DMABuffer_A_IsFull
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID   // channel ID 0 to 6
)

PLIB_ADCHS_DMABuffer_B_InterruptDisable Function

Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_B_InterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully disabled interrupt
- false - The function could not disable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA Buffer B full interrupt for specified Channel 0 to 6. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_B_InterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_B_InterruptDisable
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID   // Channel ID 0 to 6
)

PLIB_ADCHS_DMABuffer_B_InterruptEnable Function

Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_B_InterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully enabled interrupt
- false - The function could not enable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA Buffer B full interrupt for specified Channel 0 to 6. The interrupt is generated when the DMA Buffer B is full. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_B_InterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

void PLIB_ADCHS_DMABuffer_B_InterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_B_IsFull Function

Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
int8_t PLIB_ADCHS_DMABuffer_B_IsFull(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- 1 - Buffer is full
- 0 - Buffer is empty
- -1 - Invalid channel ID passed

Description

This functino is used to determine if the ADCHS DMA Buffer B is full for the specified Channel 0 to 6.

A value of '1' is returned when Buffer B is full. A value of '0' is returned when Buffer B is empty.

This feature is not implemented for Channel 7. Therefore, passing Channel 7 returns an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (1 == PLIB_ADCHS_DMABuffer_B_IsFull(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // process data, if DMA Buffer B is full
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.

Function

int8_t PLIB_ADCHS_DMABuffer_B_IsFull
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6
)

PLIB_ADCHS_DMADisable Function

Disables the DMA in the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMADisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the DMA in the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable DMA
PLIB_ADCHS_DMADisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_DMADisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMAEnable Function

Enables the DMA in the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMAEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the DMA in the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA
PLIB_ADCHS_DMAEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_DMAEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMAOverflowErrorHasOccurred Function

Used to determine if the DMA Buff had an overflow error.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMAOverflowErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If an overflow error occurred
- false - If an overflow error has not occurred

Description

DMA Buffers are circular in nature. If reading the data from DMA buffer is slower as compared to filling the buffer, the newer data will overwrite the previous data (before the data is read). This function returns 'true' if an overflow error has occurred.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (PLIB_ADCHS_DMAOverflowErrorHasOccurred(MY_ADCHS_INSTANCE)
{
    // Error occurred
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_DMAOverflowErrorHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMASetup Function

Configures the DMA on the High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMASetup(ADCHS_MODULE_ID index, ADCHS_DMA_BUFFER_LENGTH bufferLengthBytes, uint32_t
baseAddress, ADCHS_DMA_COUNT countMode, uint32_t countBaseAddress);
```

Returns

None.

Description

This functin configures all parameters for the DMA of the High-Speed SAR ADC.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The ADC channel and the Channel 0 to 6 using DMA are disabled.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the DMA
void PLIB_ADCHS_DMASetup(   MY_ADCHS_INSTANCE,              // ADCHS module ID
                            ADCHS_DMA_BUFFER_LENGTH_8BYTES, // buffer length is 8 Bytes
                            0xA002,                         // Base address is 0xA002
                            ADCHS_DMA_COUNT_ENABLE,         // count is enabled
                            0xC400  );                      // Count base address is 0xC400

// Enable DMA
PLIB_ADCHS_DMAEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
bufferLengthBytes	Length of DMA Buffer in number of bytes. Buffer is saved on RAM. The variable is of type ADCHS_DMA_BUFFER_LENGTH.

baseAddress	Address of RAM, which holds the buffer.
countMode	Enable/ Disable the DMA feature to keep count of number of converted data saved by DMA.
countBaseAddress	Address of RAM, which holds the counts.

Function

```
void PLIB_ADCHS_DMASetup
(
    ADCHS_MODULE_ID index,                  // ADCHS module ID
    ADCHS_DMA_BUFFER_LENGTH bufferLengthBytes,    // Buffer length
uint32_t baseAddress,                   // Base address for data
    ADCHS_DMA_COUNT countMode,              // Enable/ Disable count
uint32_t countBaseAddress               // Base address for count
)
```

PLIB_ADCHS_DMASourceRemove Function

Disables the DMA for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMASourceRemove(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully removed the DMA source
- false - The function could not remove the DMA source, as selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA for specified Channel 0 to 6. If Channel 7 is selected, this function returns error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// DMA for Channel 1
if(!PLIB_ADCHS_DMASourceRemove( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

```
bool PLIB_ADCHS_DMASourceRemove
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID      // Channel ID 0 to 6
)
```

PLIB_ADCHS_DMASourceSelect Function

Enables the DMA for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMASourceSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully selected the DMA source
- false - The function could not select the DMA source, as selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA for specified Channel 0 to 6. If selected Channel is 7, this function return error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// DMA for Channel 1
if(!PLIB_ADCHS_DMASourceSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMASourceSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

h) Channel Related Functions

PLIB_ADCHS_ChannelAnalogFeatureDisable Function

Disables the analog circuit for channels of High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelAnalogFeatureDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the analog circuit for channels. The analog circuit for unused channels can be disabled to reduce current consumption.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelAnalogFeatureDisable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelAnalogFeatureDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelAnalogFeatureEnable Function

Enables the analog circuit for High-Speed SAR ADC channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelAnalogFeatureEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the analog circuit for channels. The analog circuit can be disabled (when not required) to reduce current consumption by the channel. Since, each channel has the feature to individually enable/disable the analog circuit, the unused channel's analog circuit can be disabled. When the analog circuit for a channel is enabled, it needs a minimum warm-up time before which the channel is ready to perform conversion. Once the channel analog feature (circuit) is enabled, its ready status can be check with PLIB_ADCHS_ChannelIsReady function.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

```
// wait until the channel is ready
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelAnalogFeatureEnable
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID
)

PLIB_ADCHS_ChannelConfigurationGet Function

Used to get the configuration for the specified channel.

File

plib_adchs.h

C

```
uint32_t PLIB_ADCHS_ChannelConfigurationGet(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

32-bit value of the configuration for ADCHS channel.

Description

This functions returns the configuration for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

uint32_t config;

config = PLIB_ADCHS_ChannelConfigurationGet(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.

Function

uint32_t PLIB_ADCHS_ChannelConfigurationGet
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID // Channel ID
)

PLIB_ADCHS_ChannelConfigurationSet Function

Used to set the configuration for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelConfigurationSet(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, uint32_t config);
```

Returns

None.

Description

This functions sets the configuration for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

uint32_t config = 0x12345678;

PLIB_ADCHS_ChannelConfigurationSet(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1, config);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.
config	32-bit value for the configuration of channel.

Function

void PLIB_ADCHS_ChannelConfigurationSet
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID,
uint32_t config
)

PLIB_ADCHS_ChannelDigitalFeatureDisable Function

Disables the digital circuit for channels of High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelDigitalFeatureDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the digital circuit for channels. The digital feature (circuit) of unused channels can be disabled to reduce current

consumption. The advantage of disabling the digital feature is that re-enabling the digital feature does not require any warm up time and the channel can be immediately used for conversion.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelDigitalFeatureDisable( MY_ADCHS_INSTANCE,
                                         ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelDigitalFeatureDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelDigitalFeatureEnable Function

Enables (turns ON) the digital circuit for channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelDigitalFeatureEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the digital circuit for channels. Unlike enabling the analog feature, enabling the digital feature does not require any warm-up time.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling PLIB_ADCHS_Enable function. Also, for the channel to perform conversion, the analog features of the channel should be enabled using PLIB_ADCHS_ChannelAnalogFeatureEnable, prior to enabling the digital feature.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable analog feature for Channel 1
PLIB_ADCHS_ChannelAnalogFeatureEnable( MY_ADCHS_INSTANCE,
                                       ADCHS_CHANNEL_1 );

// wait until channel is ready
while(!PLIB_ADCHS_ChannelIsReady( MY_ADCHS_INSTANCE,
                                  ADCHS_CHANNEL_1 ));
```

```
// Enable the digital Channel 1
PLIB_ADCHS_ChannelDigitalFeatureEnable( MY_ADCHS_INSTANCE,
                                        ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelDigitalFeatureEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelIsReady Function

Returns the state of the channel.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelIsReady(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - Channel is ready after warm-up time has elapsed
- false - Channel is not ready

Description

This function returns the state, indicating whether the channel is awake and ready after the warm-up time is complete. When a analog feature (circuit) of a channel is enabled, there is a warm-up time required before the channel is ready and can be used for conversion. The ready state of the channel can be acquired using this function.

Remarks

None.

Preconditions

The analog feature for the channel should be enabled using the PLIB_ADCHS_ChannelAnalogFeatureEnable function. Also, the ADCHS module should be enabled before calling this function.

Example

```
// Enable the analog feature for Channel 1
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// wait until the channel is ready
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured
channelID	ID of the required channel

Function

bool PLIB_ADCHS_ChannelIsReady

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelIsReadyInterruptDisable Function

Disables the Channel ready interrupt for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelIsReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the channel ready interrupt for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable wakeup interrupt for Channel 1
PLIB_ADCHS_ChannelIsReadyInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID.

Function

void PLIB_ADCHS_ChannelIsReadyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // channel ID

)

PLIB_ADCHS_ChannelIsReadyInterruptEnable Function

Enables the Channel ready interrupt for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelIsReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the channel ready interrupt for the specified channel. The interrupt is generated when the channel is ready after wake-up (following warm-up time).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable wakeup interrupt for Channel 1
PLIB_ADCHS_ChannelIsReadyInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID.

Function

void PLIB_ADCHS_ChannelIsReadyInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // channel ID

)

PLIB_ADCHS_ChannelSetup Function

Configures the High-Speed SAR ADC channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelSetup(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, ADCHS_DATA_RESOLUTION res,
uint8_t channelClockDivider, uint16_t sampleTimeCount, ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK earlyInterruptClk);
```

Returns

None.

Description

This function configures all ADC parameters that are common to the specified ADC Channel 0 to 7.

This configuration must occur prior to enabling the ADC, and therefore, must be called when the ADC is disabled. Also, this configuration must occur prior to enabling the digital circuit and analog bias circuit for the specified ADC Channel 0 to 7.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channel is disabled when calling this function.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_ChannelSetup( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
        1,                                      // channel clock divider bit is, TAD =  2 * TQ
        1,                                      // Sample time is 3 * TAD
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1);   // Interrupt, 1 clock early

status = PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
```

```
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING);           // Synchronous sampling selected

if( false == status)
{
    // error has occurred
    while(1);
}
// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);

// Wait until the reference voltage is ready
while(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));

//Check if there is a fault in reference voltage
if(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE))
{
    // process fault accordingly
    while(1);
}

// Enable the analog circuit
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// Wait until the channel wakes up after warm-up
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));

// Enable the digital circuit
PLIB_ADCHS_ChannelDigitalFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// ADC will begin conversion now, after a valid trigger
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID of the required ADC channel.
res	Sets the output data resolution.
channelClockDivider	Channel Clock source divider. Values range from 1 to 127. This divider determines the channel clock (clock frequency at which the channel 0 to 7 is running). The frequency of
channel clock, as per following equation	Channel clock frequency = (control clock frequency)/(channelClockDivider * 2). Please note that value of "0" for channelClockDivider is not allowed. The minimum value of channelClockDivider is "1", which also means that the highest channel clock frequency is half of control clock frequency.
sampleTimeCount	Sets the sample time for ADC channel, 0 to 7. Values range from
0 to 1023. The sample time is given by the equation	Sample time = ((1/Channel clock frequency) * (sampleTimeCount + 2)).
earlyInterruptClk	Sets the number of channel clocks prior to the availability of actual data that the early interrupt is generated. This variable if of type ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK. The selection of this parameter is dependent on the resolution of output data.

Function

```
void PLIB_ADCHS_ChannelSetup
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID,                 // channel ID 0 to 7
    ADCHS_DATA_RESOLUTION res,                  // Output data resolution
uint8_t channelClockDivider,                    // Clock divider
uint16_t sampleTimeCount,                       // Sample time
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK earlyInterruptClk // early interrupt clock setting
)
```

PLIB_ADCHS_ChannelInputSelect Function

Selects the analog input for Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelInputSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, ADCHS_CHANNEL_INP_SEL
sel);
```

Returns

- true - The function successfully selected the analog input for the channel
- false - The function could not select the input, as selected channel is 7, which does not implement this feature. Otherwise, the selected analog input is not available for the selected channel.

Description

Selects the analog input for Channel 0 to 6. The inputs for Channel 0 to 6 can be changed between different inputs. This feature is not available for Channel 7; therefore, calling this function for Channel 7 will result in an error. Also, the input selected should match the ones available for the selected channel. Selecting an input which is not available for the channel will return an error.

Remarks

None.

Preconditions

None.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set Channel-0 to use the first input.
status = PLIB_ADCHS_ChannelInputSelect
                    (
                        MY_ADCHS_INSTANCE,
                        ADCHS_CHANNEL_0,
                        ADCHS_CHANNEL_0_DEFAULT_INP_AN0
                    );

if(false == status)
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.
sel	Selection of inputs of type ADCHS_CHANNEL_INP_SEL.

Function

bool PLIB_ADCHS_ChannelInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID,

ADCHS_CHANNEL_INP_SEL sel

)

i) FIFO Functions

PLIB_ADCHS_FIFODataCountGet Function

Returns the number of data to be read from FIFO.

File

plib_adchs.h

C

```
uint8_t PLIB_ADCHS_FIFODataCountGet(ADCHS_MODULE_ID index);
```

Returns

The number of the data stored in the FIFO.

Description

This function returns the number of data which can be read from FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// variable to save data
uint32_t convData;

// Check, if FIFO has some data
while(!PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE));

// Once data is available, check if the data belongs to channel ID -1
if(PLIB_ADCHS_FIFOSourceGet(MY_ADCHS_INSTANCE) == ADCHS_CHANNEL_1)
{
    // Continue reading FIFO, until the count is zero
    while(PLIB_ADCHS_FIFODataCountGet(MY_ADCHS_INSTANCE) != 0)
    {
        // Read from FIFO. For each read, the count is decremented
        convData = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    }
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

uint8_t PLIB_ADCHS_FIFODataCountGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataIsAvailable Function

Used to determine if the FIFO has data ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFODataIsAvailable(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If data is ready
- false - If data is not ready

Description

Used to determine if the ADCHS FIFO has data ready. A 'true' is returned when data is available, which can be fetched using PLIB_ADCHS_FIFORead.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFODataIsAvailable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataIsNegative Function

Returns the sign of data stored in FIFO.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFODataIsNegative(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If sign is negative
- false - If sign is not negative

Description

This function returns the sign of data stored in the FIFO.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_FIFODataIsNegative(MY_ADCHS_INSTANCE)) {
            //process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFODataIsNegative(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataReadyInterruptDisable Function

Disables the interrupt for FIFO.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODataReadyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for FIFO
PLIB_ADCHS_FIFODataReadyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_FIFODataReadyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataReadyInterruptEnable Function

Enables the interrupt for FIFO.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODataReadyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for FIFO. The interrupt is generated when the FIFO has data to be read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for FIFO
PLIB_ADCHS_FIFODataReadyInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_FIFODataReadyInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODisable Function

Disables the FIFO in the High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable FIFO
PLIB_ADCHS_FIFODisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_FIFODisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOEnable Function

Enables the FIFO in the High-Speed SAR ADC

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFOEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable FIFO
PLIB_ADCHS_FIFOEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_FIFOEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOErrorHasOccurred Function

Used to determine if the FIFO has encountered an overflow error.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFOErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - An error has occurred
- false - No error occurred has in the FIFO

Description

This function can be used to determine if the ADCHS FIFO had a data, which was overwritten by the next round of a FIFO write (overflow error). A 'true' is returned when an overflow error has occurred.

Remarks

None.

Preconditions

None.

Example

```
if (!PLIB_ADCHS_FIFOErrorHasOccurred(MY_ADCHS_INSTANCE)) {
            // process data, if overflow error did not occur
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFOErrorHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFORead Function

Used to fetch the data result from the FIFO.

File

plib_adchs.h

C

```
int32_t PLIB_ADCHS_FIFORead(ADCHS_MODULE_ID index);
```

Returns

A 32-bit result in the format specified by the ADC configuration for the module using the FIFO.

Description

This function is used to fetch data from the FIFO. The FIFO can only be assigned to a Class 1 ADC analog input.

Remarks

None.

Preconditions

The FIFO should have data ready to be read from it.

Example

```
int32_t myFIFOResult;

if (PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE)) {
    // fetch data
    myDFLTRResult = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    // process result
    ...
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

int32_t PLIB_ADCHS_FIFORead(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOSourceGet Function

Returns the channel ID using the FIFO.

File

plib_adchs.h

C

```
ADCHS_CHANNEL_ID PLIB_ADCHS_FIFOSourceGet(ADCHS_MODULE_ID index);
```

Returns

The Channel ID of data type ADCHS_CHANNEL_ID.

Description

This function returns the channel ID of Channel 0 to 6, whose data is stored on the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// variable to save data
uint32_t convData;

// Check, if FIFO has some data
while(!PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE));

// Once data is available, check if the data belongs to channel ID -1
if(PLIB_ADCHS_FIFOSourceGet(MY_ADCHS_INSTANCE) == ADCHS_CHANNEL_1)
{
    // Continue reading FIFO, until the count is zero
    while(PLIB_ADCHS_FIFODataCountGet(MY_ADCHS_INSTANCE) != 0)
    {
        // Read from FIFO. For each read, the count is decremented
        convData = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    }
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

ADCHS_CHANNEL_ID PLIB_ADCHS_FIFOSourceGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOSourceSelect Function

Sets the Channel 0 to 6 using the FIFO.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFOSourceSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully selected the FIFO source
- false - The function could not select the FIFO source, as selected Channel is 7, which does not implement this feature

Description

This function sets the Channel 0 to 6 ID, which would be storing data into FIFO. If this function is called with Channel ID as 7, the function will return an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set Channel 1 to be storing data into FIFO
if(!PLIB_ADCHS_FIFOSourceSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error has occurred
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_FIFOSourceSelect
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID // Channel ID from 0 to 6
)

j) Interrupt Functions

PLIB_ADCHS_EarlyInterruptDisable Function

Disables the early interrupt for the ADCHS.

File

plib_adchs.h

C

```
void PLIB_ADCHS_EarlyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function Disables (turns OFF) the early interrupt for the entire ADCHS. Since early interrupt function is disabled, setting early interrupt enable for individual analog input will not work.

Further selection (enabling/disabling) of interrupt feature for individual analog input is possible through the functions PLIB_ADCHS_AnalogInputDataReadyInterruptEnable or PLIB_ADCHS_AnalogInputDataReadyInterruptDisable.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable early interrupt
PLIB_ADCHS_EarlyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_EarlyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_EarlyInterruptEnable Function

Enables the early interrupt for the ADCHS.

File

plib_adchs.h

C

```
void PLIB_ADCHS_EarlyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the early interrupt for the entire ADCHS. Further selection (enabling/disabling) of early interrupt feature for individual analog input through the functions PLIB_ADCHS_AnalogInputEarlyInterruptEnable or PLIB_ADCHS_AnalogInputEarlyInterruptDisable.

The interrupt is generated at a set number of clock prior to conversion complete. The number of clock for early interrupt is set using the "configure" function for specific channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable early interrupt
PLIB_ADCHS_EarlyInterruptEnable( MY_ADCHS_INSTANCE );

// Now, enable the early interrupt for each analog inputs
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_EarlyInterruptEnable(ADCHS_MODULE_ID index)

k) Turbo Mode Functions

PLIB_ADCHS_TurboModeChannelSelect Function

Configures the channels for Turbo mode.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_TurboModeChannelSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID masterChannelID,
ADCHS_CHANNEL_ID slaveChannelID);
```

Returns

- true - The function successfully selected the channels for Turbo mode
- false - The function could not select the channels for Turbo mode for the following reasons:
  - Both the master and slave channels are the same
  - Channel 7 was used to set up Turbo mode (either as a master or a slave or both)

Description

While running the High-Speed SAR ADC in Turbo mode, two channels (from 0 to 6) are interleaved to get higher throughput. This function configures the two channels. This configuration must occur prior to enabling the ADC and prior to enabling Turbo mode. Only valid channels from 0 to 6 can be made as master or slave. If Channel 7 is used, the function will return error

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channels are disabled and Turbo mode is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC with Channel 0 as master and Channel 1 as slave
if(!PLIB_ADCHS_TurboModeChannelSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_0, ADCHS_CHANNEL_1))
{
    // Error occurred while setting up Turbo mode
    while(1);
}

// Enable Turbo mode
PLIB_ADCHS_TurboModeEnable(MY_ADCHS_INSTANCE);

// Check for Turbo mode error
if(PLIB_ADCHS_TurboModeErrorHasOccurred(MY_ADCHS_INSTANCE);
{
    // Configuration of Turbo mode has caused an error
    // Process error here or reconfigure Turbo Mode
    while(1);
}

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
masterChannelID	Channel ID of master channel. Valid channel IDs are from 0 to 6.
slaveChannelID	Channel ID of slave channel. Valid channel IDs are from 0 to 6.

Function

bool PLIB_ADCHS_TurboModeChannelSelect
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID masterChannelID,
ADCHS_CHANNEL_ID slaveChannelID
)

PLIB_ADCHS_TurboModeDisable Function

Disables Turbo mode for High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TurboModeDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) Turbo mode on the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_TurboModeDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_TurboModeDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TurboModeEnable Function

Enables Turbo mode for the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TurboModeEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) Turbo mode on the High-Speed SAR ADC module.

Remarks

None.

Preconditions

The master and slave channels must be selected to work in Turbo mode, before enabling Turbo mode.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_TurboModeEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_TurboModeEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TurboModeErrorHasOccurred Function

Returns the error state of Turbo mode.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_TurboModeErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - Turbo mode error has occurred
- false - Turbo mode error has not occurred

Description

Returns the state of error bit for Turbo mode. When Turbo mode is enabled, some parameters of master and slave channels should be same. They are ADC channel clock divisor, ADC resolution, and Sampling time. Also, both master and slave channels should use synchronous sampling. If any or all these conditions are not met, the Turbo mode error will be set, which can be read by calling

Remarks

None.

Preconditions

None.

Example

```
bool errorState = PLIB_ADCHS_TurboModeErrorHasOccurred(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_TurboModeErrorHasOccurred(ADCHS_MODULE_ID index)

l) Voltage Reference Functions

PLIB_ADCHS_VREFFaultHasOccurred Function

Returns the state of VREF fault.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_VREFFaultHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - A VREF Fault has occurred
- false - No Fault occurred on VREF

Description

This function returns the Fault state for VREF, Band Gap, or AVDD BOR.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
bool vrefFaultState = PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_VREFFaultHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFFaultInterruptDisable Function

Disables the VREF Fault interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFFaultInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the VREF Fault interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable VREF fault interrupt
PLIB_ADCHS_VREFFaultInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFFaultInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFFaultInterruptEnable Function

Enables the VREF fault interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFFaultInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the VREF fault interrupt. The interrupt is generated when a fault is encountered with VREF (mostly by BOR of the analog supply).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable VREF fault interrupt
PLIB_ADCHS_VREFFaultInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFFaultInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFIsReady Function

Returns the state of VREF.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_VREFIsReady(ADCHS_MODULE_ID index);
```

Returns

- true - Both band gap reference voltage and ADC VREF are ready
- false - Either band gap reference voltage or ADC VREF is not ready

Description

This function returns the VREF state.

Remarks

None.

Preconditions

The ADCHS module should be enabled before checking for VREF status. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
bool vrefState = PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_VREFIsReady(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFReadyInterruptDisable Function

Disables the VREF ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFReadyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the VREF ready interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable VREF interrupt
PLIB_ADCHS_VREFReadyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFReadyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFReadyInterruptEnable Function

Enables the VREF ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFReadyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the VREF ready interrupt. The interrupt is generated when the Band gap voltage/VREF is ready to be used by the ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable VREF ready interrupt
PLIB_ADCHS_VREFReadyInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFReadyInterruptEnable(ADCHS_MODULE_ID index)

m) Feature Existence Functions

PLIB_ADCHS_ExistsAnalogInputCheck Function

Identifies whether the System Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputCheck(ADCHS_MODULE_ID index);
```

Returns

Existence of the System Configuration feature:

- true - The System Configuration feature is supported on the device
- false - The System Configuration feature is not supported on the device

Description

This function identifies whether the System Configuration feature is available on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_AnalogInputIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputCheck(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsAnalogInputModeControl Function

Identifies whether the analog input mode control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputModeControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the input mode control feature:

- true - The analog input mode control feature is supported on the device
- false - The analog input mode control feature is not supported on the device

Description

This function identifies whether the analog input mode control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputModeSelect
- PLIB_ADCHS_AnalogInputModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputModeControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsAnalogInputScan Function

Identifies whether the Analog input Scan exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputScan(ADCHS_MODULE_ID index);
```

Returns

Existence of the Scan feature:

- true - The Analog Input Scan feature is supported on the device
- false - The Analog Input Scan feature is not supported on the device

Description

This function identifies whether the Analog Input Scan feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputScanIsComplete
- PLIB_ADCHS_AnalogInputScanSelect
- PLIB_ADCHS_AnalogInputScanRemove
- PLIB_ADCHS_AnalogInputScanIsSelected
- PLIB_ADCHS_AnalogInputScanSetup
- PLIB_ADCHS_ScanCompleteInterruptEnable
- PLIB_ADCHS_ScanCompleteInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputScan(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelAnalogControl Function

Identifies whether the Channel Analog control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelAnalogControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Analog control feature:

- true - The Channel Analog control feature is supported on the device
- false - The Channel Analog control feature is not supported on the device

Description

This function identifies whether the Channel Analog control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelAnalogFeatureEnable
- PLIB_ADCHS_ChannelAnalogFeatureDisable
- PLIB_ADCHS_ChannelIsReady
- PLIB_ADCHS_ChannelIsReadyInterruptEnable
- PLIB_ADCHS_ChannelIsReadyInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelAnalogControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelConfiguration Function

Identifies whether the Channel Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelConfiguration(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Configuration feature:

- true - The Channel Configuration feature is supported on the device
- false - The Channel Configuration feature is not supported on the device

Description

This function identifies whether the Channel Configuration feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelConfigurationGet
- PLIB_ADCHS_ChannelConfigurationSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelConfiguration(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelDigitalControl Function

Identifies whether the Channel Digital control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelDigitalControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Digital control feature:

- true - The Channel Digital control feature is supported on the device
- false - The Channel Digital control feature is not supported on the device

Description

This function identifies whether the Channel Digital control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelDigitalFeatureEnable
- PLIB_ADCHS_ChannelDigitalFeatureDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelDigitalControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelInputSelectControl Function

Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelInputSelectControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel 0 to 6 Input feature:

- true - The Channel 0 to 6 Input Select feature is supported on the device
- false - The Channel 0 to 6 Input Select feature is not supported on the device

Description

This function identifies whether the Channel 0 to 6 Input select feature is available on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_ChannelInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCHS_ExistsChannelInputSelectControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsConfiguration Function

Identifies whether the Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsConfiguration(ADCHS_MODULE_ID index);
```

Returns

Existence of the Configuration feature:

- true - The Configuration feature is supported on the device
- false - The Configuration feature is not supported on the device

Description

This function identifies whether the Configuration feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_Setup
- PLIB_ADCHS_ChannelSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsConfiguration(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsConversionResults Function

Identifies whether the Conversion Results feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsConversionResults(ADCHS_MODULE_ID index);
```

Returns

Existence of the ConversionResults feature:

- true - The ConversionResults feature is supported on the device
- false - The ConversionResults feature is not supported on the device

Description

This function identifies whether the Conversion Results feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputDataReadyInterruptEnable
- PLIB_ADCHS_AnalogInputDataReadyInterruptDisable
- PLIB_ADCHS_AnalogInputDataIsReady
- PLIB_ADCHS_AnalogInputResultGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsConversionResults(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsCVD Function

Identifies whether the CVD exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsCVD(ADCHS_MODULE_ID index);
```

Returns

Existence of the CVD feature:

- true - The CVD feature is supported on the device
- false - The CVD feature is not supported on the device

Description

This function identifies whether the CVD feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_CVDEnable
- PLIB_ADCHS_CVDDisable
- PLIB_ADCHS_CVDSetup
- PLIB_ADCHS_CVDResultGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsCVD(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDigitalComparator Function

Identifies whether the Digital Comparator feature exists on the ADCHS module .

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDigitalComparator(ADCHS_MODULE_ID index);
```

Returns

Existence of the Digital Comparator feature:

- true - The Digital Comparator feature is supported on the device
- false - The Digital Comparator feature is not supported on the device

Description

This function identifies whether the Digital Comparator feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DigitalComparatorAnalogInputSelect
- PLIB_ADCHS_DigitalComparatorAnalogInputGet
- PLIB_ADCHS_DigitalComparatorEnable
- PLIB_ADCHS_DigitalComparatorDisable
- PLIB_ADCHS_DigitalComparatorInterruptEnable
- PLIB_ADCHS_DigitalComparatorInterruptDisable
- PLIB_ADCHS_DigitalComparatorSetup
- PLIB_ADCHS_DigitalComparatorEventHasOccurred
- PLIB_ADCHS_DigitalComparatorLimitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDigitalComparator(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDigitalFilter Function

Identifies whether the Digital Filter feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDigitalFilter(ADCHS_MODULE_ID index);
```

Returns

Existence of the Digital Filter feature:

- true - The Digital Filter feature is supported on the device
- false - The Digital Filter feature is not supported on the device

Description

This function identifies whether the Digital Filter feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DigitalFilterEnable
- PLIB_ADCHS_DigitalFilterDisable
- PLIB_ADCHS_DigitalFilterOversamplingModeSetup
- PLIB_ADCHS_DigitalFilterAveragingModeSetup
- PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect

- PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect
- PLIB_ADCHS_DigitalFilterDataIsReady
- PLIB_ADCHS_DigitalFilterDataGet
- PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable
- PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDigitalFilter(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDMA Function

Identifies whether the DMA exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDMA(ADCHS_MODULE_ID index);
```

Returns

Existence of the DMA feature:

- true - The DMA feature is supported on the device
- false - The DMA feature is not supported on the device

Description

This function identifies whether the DMA feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DMAEnable
- PLIB_ADCHS_DMADisable
- PLIB_ADCHS_DMASetup
- PLIB_ADCHS_DMASourceSelect
- PLIB_ADCHS_DMASourceRemove
- PLIB_ADCHS_DMABuffer_A_InterruptEnable
- PLIB_ADCHS_DMABuffer_A_InterruptDisable
- PLIB_ADCHS_DMABuffer_B_InterruptEnable
- PLIB_ADCHS_DMABuffer_B_InterruptDisable
- PLIB_ADCHS_DMABuffer_A_IsFull
- PLIB_ADCHS_DMABuffer_B_IsFull
- PLIB_ADCHS_DMAOverflowErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDMA(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsEarlyInterruptControl Function

Identifies whether the Early Interrupt control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsEarlyInterruptControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Early Interrupt control feature:

- true - The Early Interrupt control feature is supported on the device
- false - The Early Interrupt control feature is not supported on the device

Description

This function identifies whether the Early Interrupt control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_EarlyInterruptEnable
- PLIB_ADCHS_EarlyInterruptDisable
- PLIB_ADCHS_AnalogInputEarlyInterruptEnable
- PLIB_ADCHS_AnalogInputEarlyInterruptDisable
- PLIB_ADCHS_AnalogInputEarlyInterruptIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsEarlyInterruptControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADCHS module

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsEnableControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_Enable
- PLIB_ADCHS_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsEnableControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsExternalConversionRequestControl Function

Identifies whether the External Convert feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsExternalConversionRequestControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the External Convert feature:

- true - The External Convert feature is supported on the device
- false - The External Convert feature is not supported on the device

Description

This function identifies whether the External Convert feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ExternalConversionRequestEnable
- PLIB_ADCHS_ExternalConversionRequestDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsExternalConversionRequestControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsFIFO Function

Identifies whether the FIFO exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsFIFO(ADCHS_MODULE_ID index);
```

Returns

Existence of the FIFO feature:

- true - The FIFO feature is supported on the device

- false - The FIFO feature is not supported on the device

Description

This function identifies whether the FIFO feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_FIFORead
- PLIB_ADCHS_FIFODataIsAvailable
- PLIB_ADCHS_FIFODataReadyInterruptEnable
- PLIB_ADCHS_FIFODataReadyInterruptDisable
- PLIB_ADCHS_FIFOEnable
- PLIB_ADCHS_FIFODisable
- PLIB_ADCHS_FIFOSourceSelect
- PLIB_ADCHS_FIFODataCountGet
- PLIB_ADCHS_FIFOSourceGet
- PLIB_ADCHS_FIFOErrorHasOccurred
- PLIB_ADCHS_FIFODataIsNegative

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsFIFO(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsManualControl Function

Identifies whether the Manual control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsManualControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Manual control feature:

- true - The Manual control feature is supported on the device
- false - The Manual control feature is not supported on the device

Description

This function identifies whether the Manual control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_SoftwareSamplingStart
- PLIB_ADCHS_SoftwareSamplingStop
- PLIB_ADCHS_SoftwareConversionStart
- PLIB_ADCHS_SoftwareConversionInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsManualControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTriggerControl Function

Identifies whether the Trigger control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTriggerControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Trigger control feature:

- true - The Trigger control feature is supported on the device
- false - The Trigger control feature is not supported on the device

Description

This function identifies whether the Trigger control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputLevelTriggerSet
- PLIB_ADCHS_AnalogInputEdgeTriggerSet
- PLIB_ADCHS_AnalogInputTriggerSourceSelect
- PLIB_ADCHS_GlobalSoftwareTriggerEnable
- PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable
- PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable
- PLIB_ADCHS_TriggerSuspendEnable
- PLIB_ADCHS_TriggerSuspendDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTriggerControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTriggerSampleControl Function

Identifies whether the Trigger Sample control feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTriggerSampleControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Trigger Sample control feature:

- true - The Trigger Sample control feature is supported on the device

- false - The Trigger Sample control feature is not supported on the device

Description

This function whether the Trigger Sample control feature exists on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_ChannelTriggerSampleSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTriggerSampleControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTurboMode Function

Identifies whether the Turbo mode feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTurboMode(ADCHS_MODULE_ID index);
```

Returns

Existence of the Turbo mode feature:

- true - The Turbo mode feature is supported on the device
- false - The Turbo mode feature is not supported on the device

Description

This function identifies whether the Turbo mode feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_TurboModeEnable
- PLIB_ADCHS_TurboModeDisable
- PLIB_ADCHS_TurboModeErrorHasOccurred
- PLIB_ADCHS_TurboModeChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTurboMode(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsUpdateReadyControl Function

Identifies whether the Update Ready feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsUpdateReadyControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Update Ready feature:

- true - The Update Ready feature is supported on the device
- false - The Update Ready feature is not supported on the device

Description

This function identifies whether the Update Ready feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable
- PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable
- PLIB_ADCHS_ControlRegistersCanBeUpdated

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsUpdateReadyControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsVREFControl Function

Identifies whether the VREF control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsVREFControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the VREF control feature:

- true - The VREF control feature is supported on the device
- false - The VREF control feature is not supported on the device

Description

This function identifies whether the VREF control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_VREFIsReady
- PLIB_ADCHS_VREFFaultHasOccurred
- PLIB_ADCHS_VREFReadyInterruptEnable
- PLIB_ADCHS_VREFReadyInterruptDisable
- PLIB_ADCHS_VREFFaultInterruptEnable
- PLIB_ADCHS_VREFFaultInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsVREFControl(ADCHS_MODULE_ID index)

n) Data Types and Constants

ADCHS_AN_INPUT_ID Enumeration

Type for identifying the available ADC Inputs

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_AN0,
  ADCHS_AN1,
  ADCHS_AN2,
  ADCHS_AN3,
  ADCHS_AN4,
  ADCHS_AN5,
  ADCHS_AN6,
  ADCHS_AN7,
  ADCHS_AN8,
  ADCHS_AN9,
  ADCHS_AN10,
  ADCHS_AN11,
  ADCHS_AN12,
  ADCHS_AN13,
  ADCHS_AN14,
  ADCHS_AN15,
  ADCHS_AN16,
  ADCHS_AN17,
  ADCHS_AN18,
  ADCHS_AN19,
  ADCHS_AN20,
  ADCHS_AN21,
  ADCHS_AN22,
  ADCHS_AN23,
  ADCHS_AN24,
  ADCHS_AN25,
  ADCHS_AN26,
  ADCHS_AN27,
  ADCHS_AN28,
  ADCHS_AN29,
  ADCHS_AN30,
  ADCHS_AN31,
  ADCHS_AN32,
  ADCHS_AN33,
  ADCHS_AN34,
  ADCHS_AN35,
  ADCHS_AN36,
  ADCHS_AN37,
  ADCHS_AN38,
  ADCHS_AN39,
  ADCHS_AN40,
  ADCHS_AN41,
  ADCHS_AN42,
  ADCHS_AN43,
  ADCHS_AN44,
  ADCHS_AN50,
```

```
    ADCHS_AN51,
    ADCHS_AN52,
    ADCHS_AN53,
    ADCHS_AN54,
    ADCHS_AN55,
    ADCHS_AN56,
    ADCHS_AN57,
    ADCHS_AN58,
    ADCHS_AN59,
    ADCHS_AN60,
    ADCHS_AN61,
    ADCHS_AN62,
    ADCHS_AN63
  } ADCHS_AN_INPUT_ID;
```

Members

Members	Description
ADCHS_AN0	Analog Input AN0
ADCHS_AN1	Analog Input AN1
ADCHS_AN2	Analog Input AN2
ADCHS_AN3	Analog Input AN3
ADCHS_AN4	Analog Input AN4
ADCHS_AN5	Analog Input AN5
ADCHS_AN6	Analog Input AN6
ADCHS_AN7	Analog Input AN7
ADCHS_AN8	Analog Input AN8
ADCHS_AN9	Analog Input AN9
ADCHS_AN10	Analog Input AN10
ADCHS_AN11	Analog Input AN11
ADCHS_AN12	Analog Input AN12
ADCHS_AN13	Analog Input AN13
ADCHS_AN14	Analog Input AN14
ADCHS_AN15	Analog Input AN15
ADCHS_AN16	Analog Input AN16
ADCHS_AN17	Analog Input AN17
ADCHS_AN18	Analog Input AN18
ADCHS_AN19	Analog Input AN19
ADCHS_AN20	Analog Input AN20
ADCHS_AN21	Analog Input AN21
ADCHS_AN22	Analog Input AN22
ADCHS_AN23	Analog Input AN23
ADCHS_AN24	Analog Input AN24
ADCHS_AN25	Analog Input AN25
ADCHS_AN26	Analog Input AN26
ADCHS_AN27	Analog Input AN27
ADCHS_AN28	Analog Input AN28
ADCHS_AN29	Analog Input AN29
ADCHS_AN30	Analog Input AN30
ADCHS_AN31	Analog Input AN31
ADCHS_AN32	Analog Input AN32
ADCHS_AN33	Analog Input AN33
ADCHS_AN34	Analog Input AN34
ADCHS_AN35	Analog Input AN35
ADCHS_AN36	Analog Input AN36
ADCHS_AN37	Analog Input AN37
ADCHS_AN38	Analog Input AN38
ADCHS_AN39	Analog Input AN39
ADCHS_AN40	Analog Input AN40
ADCHS_AN41	Analog Input AN41

ADCHS_AN42	Analog Input AN42
ADCHS_AN43	Analog Input AN43
ADCHS_AN44	Analog Input AN44
ADCHS_AN50	Analog Input AN50
ADCHS_AN51	Analog Input AN51
ADCHS_AN52	Analog Input AN52
ADCHS_AN53	Analog Input AN53
ADCHS_AN54	Analog Input AN54
ADCHS_AN55	Analog Input AN55
ADCHS_AN56	Analog Input AN56
ADCHS_AN57	Analog Input AN57
ADCHS_AN58	Analog Input AN58
ADCHS_AN59	Analog Input AN59
ADCHS_AN60	Analog Input AN60
ADCHS_AN61	Analog Input AN61
ADCHS_AN62	Analog Input AN62
ADCHS_AN63	Analog Input AN63

Description

ADC Analog Input ID

Type for identifying the available ADC Inputs, regardless of Class.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHARGEPUMP_MODE Enumeration

Defines the selection for the charge pump.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHARGEPUMP_DISABLE,
  ADCHS_CHARGEPUMP_ENABLE
} ADCHS_CHARGEPUMP_MODE;
```

Members

Members	Description
ADCHS_CHARGEPUMP_DISABLE	Charge pump is disabled
ADCHS_CHARGEPUMP_ENABLE	Charge pump is enabled

Description

ADCHS Charge pump setting

This data type defines the state of the charge pump as either enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CLOCK_SOURCE Enumeration

Defines the ADCHS Clock Source Select.

File

help_plib_adchs.h

C

```
typedef enum {
```

```
  ADCHS_CLOCK_SOURCE_PBCLK,
  ADCHS_CLOCK_SOURCE_FRC,
  ADCHS_CLOCK_SOURCE_RFCLK3,
  ADCHS_CLOCK_SOURCE_SYSCLK
} ADCHS_CLOCK_SOURCE;
```

Members

Members	Description
ADCHS_CLOCK_SOURCE_PBCLK	TAD clock set to peripheral bus clock (PBCLK)
ADCHS_CLOCK_SOURCE_FRC	TAD clock set to FRC
ADCHS_CLOCK_SOURCE_RFCLK3	TAD clock set to REFCLK3
ADCHS_CLOCK_SOURCE_SYSCLK	TAD clock set to SYSCLK (TCY)

Description

ADCHS Clock source selection

This enumeration data type defines the ADCHS Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CVD_CAPACITOR Enumeration

Defines the value of the internal capacitor during CVD mode.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CVD_CAPACITOR_0PF,
  ADCHS_CVD_CAPACITOR_2_5PF,
  ADCHS_CVD_CAPACITOR_5PF,
  ADCHS_CVD_CAPACITOR_7_5PF,
  ADCHS_CVD_CAPACITOR_10PF,
  ADCHS_CVD_CAPACITOR_12_5PF,
  ADCHS_CVD_CAPACITOR_15PF,
  ADCHS_CVD_CAPACITOR_17_5PF
} ADCHS_CVD_CAPACITOR;
```

Members

Members	Description
ADCHS_CVD_CAPACITOR_0PF	While in CVD mode, internal capacitor is 0pF
ADCHS_CVD_CAPACITOR_2_5PF	While in CVD mode, internal capacitor is 2.5pF
ADCHS_CVD_CAPACITOR_5PF	While in CVD mode, internal capacitor is 5pF
ADCHS_CVD_CAPACITOR_7_5PF	While in CVD mode, internal capacitor is 7.5pF
ADCHS_CVD_CAPACITOR_10PF	While in CVD mode, internal capacitor is 10pF
ADCHS_CVD_CAPACITOR_12_5PF	While in CVD mode, internal capacitor is 12.5pF
ADCHS_CVD_CAPACITOR_15PF	While in CVD mode, internal capacitor is 15pF
ADCHS_CVD_CAPACITOR_17_5PF	While in CVD mode, internal capacitor is 17.5pF

Description

ADCHS CVD capacitor selection

This data type defines the value of the internal capacitor during CVD mode.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DATA_RESOLUTION Enumeration

Identifies the resolution of the ADC output.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DATA_RESOLUTION_6BIT,
  ADCHS_DATA_RESOLUTION_8BIT,
  ADCHS_DATA_RESOLUTION_10BIT,
  ADCHS_DATA_RESOLUTION_12BIT
} ADCHS_DATA_RESOLUTION;
```

Members

Members	Description
ADCHS_DATA_RESOLUTION_6BIT	ADC Output Data resolution is 6 bits
ADCHS_DATA_RESOLUTION_8BIT	ADC Output Data resolution is 8 bits
ADCHS_DATA_RESOLUTION_10BIT	ADC Output Data resolution is 10 bits
ADCHS_DATA_RESOLUTION_12BIT	ADC Output Data resolution is 12 bits

Description

ADC Data Resolution

This data type is used to identify the resolution of the ADC output.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_COMPARATOR_ID Enumeration

Identifies the supported Digital Comparators.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_COMPARATOR_1,
  ADCHS_DIGITAL_COMPARATOR_2,
  ADCHS_DIGITAL_COMPARATOR_3,
  ADCHS_DIGITAL_COMPARATOR_4,
  ADCHS_DIGITAL_COMPARATOR_5,
  ADCHS_DIGITAL_COMPARATOR_6,
  ADCHS_NUMBER_OF_DIGITAL_COMPARATOR
} ADCHS_DIGITAL_COMPARATOR_ID;
```

Members

Members	Description
ADCHS_DIGITAL_COMPARATOR_1	Digital Comparator 1
ADCHS_DIGITAL_COMPARATOR_2	Digital Comparator 2
ADCHS_DIGITAL_COMPARATOR_3	Digital Comparator 3
ADCHS_DIGITAL_COMPARATOR_4	Digital Comparator 4
ADCHS_DIGITAL_COMPARATOR_5	Digital Comparator 5
ADCHS_DIGITAL_COMPARATOR_6	Digital Comparator 6
ADCHS_NUMBER_OF_DIGITAL_COMPARATOR	Number of Digital Comparators

Description

ADCHS Digital Comparator

This enumeration identifies all supported Digital Comparators for this ADCHS channel.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT Enumeration

Identifies the Digital Filter averaging sample count.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_2,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_4,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_8,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_16,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_32,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_128,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_256
} ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_2	Averaging is performed on 2 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_4	Averaging is performed on 4 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_8	Averaging is performed on 8 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_16	Averaging is performed on 16 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_32	Averaging is performed on 32 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64	Averaging is performed on 64 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_128	Averaging is performed on 128 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_256	Averaging is performed on 256 samples

Description

ADCHS Averaging sample

This enumeration identifies all supported digital Filter averaging sample count for this ADCHS channel. Once, the set number of samples are acquired by ADC channel, it starts the averaging process.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_ID Enumeration

Identifies the supported Digital Filters.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_1,
  ADCHS_DIGITAL_FILTER_2,
  ADCHS_DIGITAL_FILTER_3,
  ADCHS_DIGITAL_FILTER_4,
  ADCHS_DIGITAL_FILTER_5,
  ADCHS_DIGITAL_FILTER_6,
  ADCHS_NUMBER_OF_DIGITAL_FILTER
} ADCHS_DIGITAL_FILTER_ID;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_1	Digital Filter 1
ADCHS_DIGITAL_FILTER_2	Digital Filter 2
ADCHS_DIGITAL_FILTER_3	Digital Filter 3
ADCHS_DIGITAL_FILTER_4	Digital Filter 4

ADCHS_DIGITAL_FILTER_5	Digital Filter 5
ADCHS_DIGITAL_FILTER_6	Digital Filter 6
ADCHS_NUMBER_OF_DIGITAL_FILTER	Number of Digital Filters

Description

ADCHS Digital Filter

This enumeration identifies all supported digital Filters for this ADCHS channel.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO Enumeration

Identifies the supported Digital Filter oversampling ratios.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_4X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_64X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_256X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_2X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_8X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_32X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_128X
} ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_4X	4x oversampling, shift sum 1 bit to right, output data is 13 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X	16x oversampling, shift sum 2 bits to right, output data is 14 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_64X	64x oversampling, shift sum 3 bits to right, output data is 15 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_256X	256x oversampling, shift sum 4 bits to right, output data is 16 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_2X	2x oversampling, shift sum 0 bits to right, output data is in 12.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_8X	8x oversampling, shift sum 1 bit to right, output data is in 13.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_32X	32x oversampling, shift sum 2 bits to right, output data is in 14.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_128X	128x oversampling, shift sum 3 bit to right, output data is in 15.1 format

Description

ADCHS Oversampling Ratio

This enumeration identifies all supported digital Filter oversampling ratios for this ADCHS channel. Oversampling ratios determine the number of samples used to generate a single output and the resulting resolution and format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS Enumeration

Data length of digital filter.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_SIGNIFICANT_FIRST_12BITS,
  ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS
} ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_SIGNIFICANT_FIRST_12BITS	Output of digital filter has only first 12 bits significant (remaining 4 zeros)
ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS	Output of digital filter has all 16 bits significant

Description

ADCHS Digital filter data length

The output of digital filter can be set as all 16-bit to be significant or only first 12-bit significant followed by 4 zeros.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DMA_BUFFER_LENGTH Enumeration

Defines the length of the DMA buffer length.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DMA_BUFFER_LENGTH_2BYTES,
  ADCHS_DMA_BUFFER_LENGTH_4BYTES,
  ADCHS_DMA_BUFFER_LENGTH_8BYTES,
  ADCHS_DMA_BUFFER_LENGTH_16BYTES,
  ADCHS_DMA_BUFFER_LENGTH_32BYTES,
  ADCHS_DMA_BUFFER_LENGTH_64BYTES,
  ADCHS_DMA_BUFFER_LENGTH_128BYTES,
  ADCHS_DMA_BUFFER_LENGTH_256BYTES
} ADCHS_DMA_BUFFER_LENGTH;
```

Members

Members	Description
ADCHS_DMA_BUFFER_LENGTH_2BYTES	Allocate 2 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_4BYTES	Allocate 4 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_8BYTES	Allocate 8 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_16BYTES	Allocate 16 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_32BYTES	Allocate 32 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_64BYTES	Allocate 64 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_128BYTES	Allocate 128 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_256BYTES	Allocate 256 bytes in RAM for each analog input

Description

ADCHS Class-1 DMA buffer length

This data type defines the length of the DMA buffer (in bytes).

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DMA_COUNT Enumeration

Defines the enable/disable of the count feature for DMA.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DMA_COUNT_DISABLE,
  ADCHS_DMA_COUNT_ENABLE
} ADCHS_DMA_COUNT;
```

Members

Members	Description
ADCHS_DMA_COUNT_DISABLE	DMA does not keep count of number of data saved by DMA
ADCHS_DMA_COUNT_ENABLE	DMA keeps count of number of data saved by DMA

Description

ADCHS DMA count enable

This data type defines whether or not the DMA should count the number of samples and store it in DMA RAM location.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK Enumeration

Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_2,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_3,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_5,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_7,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_8
} ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK;
```

Members

Members	Description
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1	The data ready interrupt is generated 1 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_2	The data ready interrupt is generated 2 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_3	The data ready interrupt is generated 3 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4	The data ready interrupt is generated 4 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_5	The data ready interrupt is generated 5 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6	The data ready interrupt is generated 6 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_7	The data ready interrupt is generated 7 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_8	The data ready interrupt is generated 8 ADC clock prior to end of conversion

Description

ADCHS Early interrupt prior clock selection

This data type defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated. All options are available when the selected resolution for converted data is 12-bit or 10-bit. For a resolution of 8-bit, options from ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 to ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6 are valid. For a resolution of 6-bit, options from ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 to ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4 are valid.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_FAST_SYNC_PERIPHERAL_CLOCK Enumeration

Defines the selection of the fast synchronous peripheral clock.

File

help_plib_adchs.h

C

```
typedef enum {
```

```
  ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE,
  ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_ENABLE
} ADCHS_FAST_SYNC_PERIPHERAL_CLOCK;
```

Members

Members	Description
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE	Peripheral clock is not used for ADCHS
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_ENABLE	Peripheral clock is used for ADCHS

Description

ADCHS Fast synchronous peripheral clock mode

This data type defines whether the fast synchronous peripheral clock to the ADCHS channel is enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_FAST_SYNC_SYSTEM_CLOCK Enumeration

Defines the selection of the fast synchronous system clock.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_FAST_SYNC_SYSTEM_CLOCK_DISABLE,
  ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE
} ADCHS_FAST_SYNC_SYSTEM_CLOCK;
```

Members

Members	Description
ADCHS_FAST_SYNC_SYSTEM_CLOCK_DISABLE	System clock is not used for ADCHS
ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE	System clock is used for ADCHS

Description

ADCHS Fast synchronous system clock mode

This data type defines whether the fast synchronous system clock to ADCHS channel is enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_INPUT_MODE Enumeration

Defines the available modes for the selected input.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR,
  ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP,
  ADCHS_INPUT_MODE_DIFFERENTIAL_UNIPOLAR,
  ADCHS_INPUT_MODE_DIFFERENTIAL_TWOS_COMP
} ADCHS_INPUT_MODE;
```

Members

Members	Description
ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR	Single-ended input, Unipolar encoded
ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP	Single-ended input, two's compliment encoded
ADCHS_INPUT_MODE_DIFFERENTIAL_UNIPOLAR	Differential input, Unipolar encoded
ADCHS_INPUT_MODE_DIFFERENTIAL_TWOS_COMP	Differential input, two's compliment encoded

Description

ADCHS Input Mode

This data type defines the available modes for the selected input.

Remarks

None.

ADCHS_INTERRUPT_BIT_SHIFT_LEFT Enumeration

Identifies the bits shift for calculating the interrupt vector.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_1_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_3_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_4_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_5_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_6_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_7_BITS
} ADCHS_INTERRUPT_BIT_SHIFT_LEFT;
```

Members

Members	Description
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS	Bit shift by 0 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_1_BITS	Bit shift by 1 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS	Bit shift by 2 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_3_BITS	Bit shift by 3 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_4_BITS	Bit shift by 4 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_5_BITS	Bit shift by 5 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_6_BITS	Bit shift by 6 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_7_BITS	Bit shift by 7 bits

Description

ADCHS Interrupt vector shift bits

Interrupt vector address, is calculated by (Base_Address + x << vector_shift), where 'x' is the smallest active channel ID from the status register (AD1DSTAT1 or 2) registers (which has highest priority).

This enumeration identifies all the possible bits shift.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_OUTPUT_DATA_FORMAT Enumeration

Defines the selection for the output data format.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_OUTPUT_DATA_FORMAT_INTEGER,
  ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL
} ADCHS_OUTPUT_DATA_FORMAT;
```

Members

Members	Description
ADCHS_OUTPUT_DATA_FORMAT_INTEGER	Output data format is integer
ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL	Output data format is fractional

Description

ADCHS output data format

The output of ADC can be specified as either integer or fractional. This enum defines the setting of the output data format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_SCAN_TRIGGER_SENSITIVE Enumeration

Trigger level for scan trigger.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
  ADCHS_SCAN_TRIGGER_SENSITIVE_LEVEL
} ADCHS_SCAN_TRIGGER_SENSITIVE;
```

Members

Members	Description
ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE	Scan trigger is edge sensitive
ADCHS_SCAN_TRIGGER_SENSITIVE_LEVEL	Scan trigger is level sensitive

Description

ADCHS scan trigger level

The scan trigger can be configured to be sensitive to level or sensitive to edge. This data type defines the different configurations.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_SCAN_TRIGGER_SOURCE Enumeration

Defines the ADCHS Channel Scan Trigger Source Selections.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_SCAN_TRIGGER_SOURCE_NONE,
  ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE,
  ADCHS_SCAN_TRIGGER_SOURCE_SOFTWARE_LEVEL,
  ADCHS_SCAN_TRIGGER_SOURCE_INT0,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR3_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR5_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_OC1,
  ADCHS_SCAN_TRIGGER_SOURCE_OC3,
  ADCHS_SCAN_TRIGGER_SOURCE_OC5,
  ADCHS_SCAN_TRIGGER_SOURCE_COUT1,
  ADCHS_SCAN_TRIGGER_SOURCE_COUT2
} ADCHS_SCAN_TRIGGER_SOURCE;
```

Members

Members	Description
ADCHS_SCAN_TRIGGER_SOURCE_NONE	No scan trigger source is selected
ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE	Global Software edge trigger selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_SOFTWARE_LEVEL	Level Software trigger selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_INT0	Interrupt 0 (INT0) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH	Timer 1 Match (TMR1) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR3_MATCH	Timer 3 Match (TMR3) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR5_MATCH	Timer 5 Match (TMR5) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC1	Output Compare 1(OC1) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC3	Output Compare 3 (OC3) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC5	Output Compare 5 (OC5) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_COUT1	Comparator Output 1 selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_COUT2	Comparator Output 2 selected as scan trigger source

Description

ADCHS Channel Scan Trigger Source Select

This data type defines the ADCHS Channel Scan Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_TRIGGER_SOURCE Enumeration

Defines the ADCHS Trigger Source Selections.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_TRIGGER_SOURCE_NONE,
  ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE,
  ADCHS_TRIGGER_SOURCE_SOFTWARE_LEVEL,
  ADCHS_TRIGGER_SOURCE_STRIG,
  ADCHS_TRIGGER_SOURCE_INT0,
  ADCHS_TRIGGER_SOURCE_TMR1_MATCH,
  ADCHS_TRIGGER_SOURCE_TMR3_MATCH,
  ADCHS_TRIGGER_SOURCE_TMR5_MATCH,
  ADCHS_TRIGGER_SOURCE_OC1,
  ADCHS_TRIGGER_SOURCE_OC3,
  ADCHS_TRIGGER_SOURCE_OC5,
  ADCHS_TRIGGER_SOURCE_COUT1,
  ADCHS_TRIGGER_SOURCE_COUT2
} ADCHS_TRIGGER_SOURCE;
```

Members

Members	Description
ADCHS_TRIGGER_SOURCE_NONE	No trigger source is selected
ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE	Global Software edge trigger selected as trigger source
ADCHS_TRIGGER_SOURCE_SOFTWARE_LEVEL	Level Software trigger selected as trigger source
ADCHS_TRIGGER_SOURCE_STRIG	Scan trigger source
ADCHS_TRIGGER_SOURCE_INT0	Interrupt 0 (INT0) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR1_MATCH	Timer 1 Match (TMR1) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR3_MATCH	Timer 3 Match (TMR3) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR5_MATCH	Timer 5 Match (TMR5) selected as trigger source
ADCHS_TRIGGER_SOURCE_OC1	Output Compare 1(OC1) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_OC3	Output Compare 3 (OC3) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_OC5	Output Compare 5 (OC5) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_COUT1	Comparator Output 1 selected as trigger source

ADCHS_TRIGGER_SOURCE_COUT2	Comparator Output 2 selected as trigger source

Description

ADCHS Trigger Source Select

This data type defines the ADCHS Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_VREF_SOURCE Enumeration

Defines the ADCHS VREF Source Select.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_VREF_AVDD_AVSS,
  ADCHS_VREF_EXTVREFP_AVSS,
  ADCHS_VREF_AVDD_EXTVREFN,
  ADCHS_VREF_EXTVREFP_EXTVREFN,
  ADCHS_VREF_INTVREFP_INTVREFN,
  ADCHS_VREF_INTVREFP_EXTVREFN,
  ADCHS_VREF_INTVREFP_AVSS,
  ADCHS_VREF_AVDD_INTVREFN
} ADCHS_VREF_SOURCE;
```

Members

Members	Description
ADCHS_VREF_AVDD_AVSS	Reference voltage set to AVDD and AVSS
ADCHS_VREF_EXTVREFP_AVSS	Reference voltage set to external VREF positive and AVSS
ADCHS_VREF_AVDD_EXTVREFN	Reference voltage set to AVDD and external VREF negative
ADCHS_VREF_EXTVREFP_EXTVREFN	Reference voltage set to external VREF positive and external VREF negative
ADCHS_VREF_INTVREFP_INTVREFN	Reference voltage set to internal VREF positive and internal VREF negative
ADCHS_VREF_INTVREFP_EXTVREFN	Reference voltage set to internal VREF positive and external VREF negative
ADCHS_VREF_INTVREFP_AVSS	Reference voltage set to internal VREF positive and AVSS
ADCHS_VREF_AVDD_INTVREFN	Reference voltage set to AVDD positive and internal VREF negative

Description

ADCHS VREF Source Select

This data type defines the ADCHS Reference Voltage (VREF) Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_WARMUP_CLOCK Enumeration

Identifies the number of clocks before the channel can be ready

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_WARMUP_CLOCK_16,
  ADCHS_WARMUP_CLOCK_32,
  ADCHS_WARMUP_CLOCK_64,
  ADCHS_WARMUP_CLOCK_128,
  ADCHS_WARMUP_CLOCK_256,
  ADCHS_WARMUP_CLOCK_512,
  ADCHS_WARMUP_CLOCK_1024,
  ADCHS_WARMUP_CLOCK_2048,
```

```
  ADCHS_WARMUP_CLOCK_4096,
  ADCHS_WARMUP_CLOCK_8192,
  ADCHS_WARMUP_CLOCK_16384,
  ADCHS_WARMUP_CLOCK_32768
} ADCHS_WARMUP_CLOCK;
```

Members

Members	Description
ADCHS_WARMUP_CLOCK_16	Warm-up time is 16 ADC clocks
ADCHS_WARMUP_CLOCK_32	Warm-up time is 32 ADC clocks
ADCHS_WARMUP_CLOCK_64	Warm-up time is 64 ADC clocks
ADCHS_WARMUP_CLOCK_128	Warm-up time is 128 ADC clocks
ADCHS_WARMUP_CLOCK_256	Warm-up time is 256 ADC clocks
ADCHS_WARMUP_CLOCK_512	Warm-up time is 512 ADC clocks
ADCHS_WARMUP_CLOCK_1024	Warm-up time is 1024 ADC clocks
ADCHS_WARMUP_CLOCK_2048	Warm-up time is 2048 ADC clocks
ADCHS_WARMUP_CLOCK_4096	Warm-up time is 4096 ADC clocks
ADCHS_WARMUP_CLOCK_8192	Warm-up time is 8192 ADC clocks
ADCHS_WARMUP_CLOCK_16384	Warm-up time is 16384 ADC clocks
ADCHS_WARMUP_CLOCK_32768	Warm-up time is 32768 ADC clocks

Description

ADCHS Wakeup warm-up clocks

When the analog section of an ADC channel is enabled, the channel needs a certain amount of warm-up time before it can be ready to perform conversion. This enumeration contains the various warm-up time in terms of the number of clocks.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_MODULE_ID Enumeration

Identifies the number of ADC supported.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_ID_0,
  ADCHS_NUMBER_OF_MODULES
} ADCHS_MODULE_ID;
```

Members

Members	Description
ADCHS_ID_0	ADC ID 0
ADCHS_NUMBER_OF_MODULES	Number of available ADC.

Description

This enumeration identifies the available ADC.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Refer to the specific device data sheet to determine how many ADC modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADCHS_CHANNEL_INP_SEL Enumeration

Defines the alternate input selection for channels 0 to 6.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_0_DEFAULT_INP_AN0,
  ADCHS_CHANNEL_0_ALTERNATE_INP_AN45,
  ADCHS_CHANNEL_1_DEFAULT_INP_AN1,
  ADCHS_CHANNEL_1_ALTERNATE_INP_AN46,
  ADCHS_CHANNEL_2_DEFAULT_INP_AN2,
  ADCHS_CHANNEL_2_ALTERNATE_INP_AN47,
  ADCHS_CHANNEL_3_DEFAULT_INP_AN3,
  ADCHS_CHANNEL_3_ALTERNATE_INP_AN48,
  ADCHS_CHANNEL_4_DEFAULT_INP_AN4,
  ADCHS_CHANNEL_4_ALTERNATE_INP_AN49
} ADCHS_CHANNEL_INP_SEL;
```

Members

Members	Description
ADCHS_CHANNEL_0_DEFAULT_INP_AN0	AN0 is the default input for Channel 0
ADCHS_CHANNEL_0_ALTERNATE_INP_AN45	AN45 is the alternate input for Channel 0
ADCHS_CHANNEL_1_DEFAULT_INP_AN1	AN1 is the default input for Channel 1
ADCHS_CHANNEL_1_ALTERNATE_INP_AN46	AN46 is the alternate input for Channel 1
ADCHS_CHANNEL_2_DEFAULT_INP_AN2	AN2 is the default input for Channel 2
ADCHS_CHANNEL_2_ALTERNATE_INP_AN47	AN47 is the alternate input for Channel 2
ADCHS_CHANNEL_3_DEFAULT_INP_AN3	AN3 is the default input for Channel 3
ADCHS_CHANNEL_3_ALTERNATE_INP_AN48	AN48 is the alternate input for Channel 3
ADCHS_CHANNEL_4_DEFAULT_INP_AN4	AN4 is the default input for Channel 4
ADCHS_CHANNEL_4_ALTERNATE_INP_AN49	AN49 is the alternate input for Channel 4

Description

ADCHS Alternate input selection for channel- 0 to 6.

This data type defines the possible alternate input selections for channels 0 to 6. For channel 7, alternate selection does not exist. Passing the values for channel 7 will return an error.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHANNEL_ID Enumeration

Identifies the ADC channel from 0 to 7.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_0,
  ADCHS_CHANNEL_1,
  ADCHS_CHANNEL_2,
  ADCHS_CHANNEL_3,
  ADCHS_CHANNEL_4,
  ADCHS_CHANNEL_5,
  ADCHS_CHANNEL_6,
  ADCHS_CHANNEL_7,
  ADCHS_NUMBER_OF_CHANNEL
} ADCHS_CHANNEL_ID;
```

Members

Members	Description
ADCHS_CHANNEL_0	ADCHS channel 0
ADCHS_CHANNEL_1	ADCHS channel 1
ADCHS_CHANNEL_2	ADCHS channel 2
ADCHS_CHANNEL_3	ADCHS channel 3

ADCHS_CHANNEL_4	ADCHS channel 4
ADCHS_CHANNEL_5	ADCHS channel 5
ADCHS_CHANNEL_6	ADCHS channel 6
ADCHS_CHANNEL_7	ADCHS channel 7
ADCHS_NUMBER_OF_CHANNEL	Number of channels

Description

ADCHS channel Select

This enumeration identifies the ADC channels from 0 to 7. Not all devices have all ADCHS channels. Refer to the specific device data sheet for the number of available channels. Some features that are available in channels 0 to 6 may not be available on channel 7.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL Enumeration

Defines the selection of trigger and sampling for channels 0 to 6.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_UNSYNC_TRIGGER_UNSYNC_SAMPLING,
  ADCHS_CHANNEL_SYNC_SAMPLING,
  ADCHS_CHANNEL_SYNC_TRIGGER_UNSYNC_SAMPLING
} ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL;
```

Members

Members	Description
ADCHS_CHANNEL_UNSYNC_TRIGGER_UNSYNC_SAMPLING	Unsynchronized trigger and unsynchronous sampling
ADCHS_CHANNEL_SYNC_SAMPLING	Synchronous sampling (trigger selection is ignored)
ADCHS_CHANNEL_SYNC_TRIGGER_UNSYNC_SAMPLING	Synchronized trigger and unsynchronous sampling

Description

ADCHS channel trigger and sampling selection

This data type defines the selection of trigger and sampling for ADCHS channels 0 to 6. Passing this value for channel 7 will result in an error.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CLASS12_AN_INPUT_ID Enumeration

Type for identifying the available Class 1 and 2 analog Inputs.

File

help_plib_adchs.h

C

```
typedef enum {
} ADCHS_CLASS12_AN_INPUT_ID;
```

Description

ADC Analog Input ID of type class-1 and class-2

These enums are passed to functions meant to control or enable/disable "triggers". This is because only Class 1 and Class 2 analog inputs have individual triggers associated with them.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

Files

Files

Name	Description
help_plib_adchs.h	ADCHS Peripheral Library interface header file template.
plib_adchs.h	ADCHS Peripheral Library Interface Header for ADCHS common definitions.

Description

This section lists the source and header files used by the library.

help_plib_adchs.h

ADCHS Peripheral Library interface header file template.

Enumerations

	Name	Description
	ADCHS_AN_INPUT_ID	Type for identifying the available ADC Inputs
	ADCHS_CHANNEL_ID	Identifies the ADC channel from 0 to 7.
	ADCHS_CHANNEL_INP_SEL	Defines the alternate input selection for channels 0 to 6.
	ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL	Defines the selection of trigger and sampling for channels 0 to 6.
	ADCHS_CHARGEPUMP_MODE	Defines the selection for the charge pump.
	ADCHS_CLASS12_AN_INPUT_ID	Type for identifying the available Class 1 and 2 analog Inputs.
	ADCHS_CLOCK_SOURCE	Defines the ADCHS Clock Source Select.
	ADCHS_CVD_CAPACITOR	Defines the value of the internal capacitor during CVD mode.
	ADCHS_DATA_RESOLUTION	Identifies the resolution of the ADC output.
	ADCHS_DIGITAL_COMPARATOR_ID	Identifies the supported Digital Comparators.
	ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT	Identifies the Digital Filter averaging sample count.
	ADCHS_DIGITAL_FILTER_ID	Identifies the supported Digital Filters.
	ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO	Identifies the supported Digital Filter oversampling ratios.
	ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS	Data length of digital filter.
	ADCHS_DMA_BUFFER_LENGTH	Defines the length of the DMA buffer length.
	ADCHS_DMA_COUNT	Defines the enable/disable of the count feature for DMA.
	ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK	Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.
	ADCHS_FAST_SYNC_PERIPHERAL_CLOCK	Defines the selection of the fast synchronous peripheral clock.
	ADCHS_FAST_SYNC_SYSTEM_CLOCK	Defines the selection of the fast synchronous system clock.
	ADCHS_INPUT_MODE	Defines the available modes for the selected input.
	ADCHS_INTERRUPT_BIT_SHIFT_LEFT	Identifies the bits shift for calculating the interrupt vector.
	ADCHS_MODULE_ID	Identifies the number of ADC supported.
	ADCHS_OUTPUT_DATA_FORMAT	Defines the selection for the output data format.
	ADCHS_SCAN_TRIGGER_SENSITIVE	Trigger level for scan trigger.
	ADCHS_SCAN_TRIGGER_SOURCE	Defines the ADCHS Channel Scan Trigger Source Selections.
	ADCHS_TRIGGER_SOURCE	Defines the ADCHS Trigger Source Selections.
	ADCHS_VREF_SOURCE	Defines the ADCHS VREF Source Select.
	ADCHS_WARMUP_CLOCK	Identifies the number of clocks before the channel can be ready

Description

ADCHS Peripheral Library Interface Header File Template

This file is provided for documentation purposes.

File Name

help_plib_adchs.h

Company

Microchip Technology Inc.

plib_adchs.h

ADCHS Peripheral Library Interface Header for ADCHS common definitions.

Functions

	Name	Description
	PLIB_ADCHS_AnalogInputDataIsReady	Returns the data ready status of analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptDisable	Disables the data ready interrupt for the selected analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptEnable	Enables the data ready interrupt for the selected analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptDisable	Disables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptEnable	Enables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptIsReady	Returns the early interrupt ready status of analog input.
	PLIB_ADCHS_AnalogInputEdgeTriggerSet	Sets the trigger as edge sensitive for selected class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputIsAvailable	Returns the analog input configuration of ADCHS channel.
	PLIB_ADCHS_AnalogInputLevelTriggerSet	Sets the trigger as level sensitive for selected Class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputModeGet	Returns the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputModeSelect	Selects the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputResultGet	Returns a ADC conversion result.
	PLIB_ADCHS_AnalogInputScanIsComplete	Returns the state of End of scan completion.
	PLIB_ADCHS_AnalogInputScanIsSelected	Returns whether or note an analog input is selected for scanning.
	PLIB_ADCHS_AnalogInputScanRemove	Removes the analog input from scanning selection.
	PLIB_ADCHS_AnalogInputScanSelect	Selects the analog input for scanning.
	PLIB_ADCHS_AnalogInputScanSetup	Selects input to include in Analog Input Scan mode.
	PLIB_ADCHS_AnalogInputTriggerSourceSelect	Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).
	PLIB_ADCHS_ChannelAnalogFeatureDisable	Disables the analog circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelAnalogFeatureEnable	Enables the analog circuit for High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelConfigurationGet	Used to get the configuration for the specified channel.
	PLIB_ADCHS_ChannelConfigurationSet	Used to set the configuration for the specified channel.
	PLIB_ADCHS_ChannelDigitalFeatureDisable	Disables the digital circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelDigitalFeatureEnable	Enables (turns ON) the digital circuit for channels.
	PLIB_ADCHS_ChannelInputSelect	Selects the analog input for Channel 0 to 6.
	PLIB_ADCHS_ChannelIsReady	Returns the state of the channel.
	PLIB_ADCHS_ChannelIsReadyInterruptDisable	Disables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelIsReadyInterruptEnable	Enables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelSetup	Configures the High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelTriggerSampleSelect	Selects the trigger and sampling modes for channels of High-Speed SAR ADC
	PLIB_ADCHS_ControlRegistersCanBeUpdated	Returns the status of update-ready.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable	Disables the update-ready interrupt.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable	Enables the update-ready interrupt.
	PLIB_ADCHS_CVDDisable	Disables the CVD feature.
	PLIB_ADCHS_CVDEnable	Enables the CVD feature.
	PLIB_ADCHS_CVDResultGet	Returns a CVD result measured by an ADCHS instance.
	PLIB_ADCHS_CVDSetup	Configures the CVD related setting of ADCHS channel.
	PLIB_ADCHS_DigitalComparatorAnalogInputGet	Returns the analog input ID used by the digital comparator.
	PLIB_ADCHS_DigitalComparatorAnalogInputSelect	Selects analog inputs, whose converted data will be processed by the comparator.
	PLIB_ADCHS_DigitalComparatorDisable	Disables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEnable	Enables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEventHasOccurred	Returns the status of the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptDisable	Disables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptEnable	Enables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorLimitSet	Sets the limit for the specified digital comparator.

	PLIB_ADCHS_DigitalComparatorSetup	Configures the Digital Comparator on the High-Speed SAR ADC converter.
	PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect	Selects the number of samples which are averaged by the Digital Filter.
	PLIB_ADCHS_DigitalFilterAveragingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.
	PLIB_ADCHS_DigitalFilterDataGet	Used to fetch the data result from the Digital Filter.
	PLIB_ADCHS_DigitalFilterDataIsReady	Used to determine if the Digital Filter has data ready.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable	Disables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable	Enables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDisable	Disables the Digital Filter.
	PLIB_ADCHS_DigitalFilterEnable	Enables the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect	Selects the oversampling ratio for the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.
	PLIB_ADCHS_Disable	High-Speed SAR ADC module is disabled (turned OFF).
	PLIB_ADCHS_DMABuffer_A_InterruptDisable	Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_InterruptEnable	Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_IsFull	Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptDisable	Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptEnable	Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_IsFull	Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMADisable	Disables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAEnable	Enables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAOverflowErrorHasOccurred	Used to determine if the DMA Buff had an overflow error.
	PLIB_ADCHS_DMASetup	Configures the DMA on the High-Speed SAR ADC.
	PLIB_ADCHS_DMASourceRemove	Disables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_DMASourceSelect	Enables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_EarlyInterruptDisable	Disables the early interrupt for the ADCHS.
	PLIB_ADCHS_EarlyInterruptEnable	Enables the early interrupt for the ADCHS.
	PLIB_ADCHS_Enable	Enables the High-Speed SAR ADC module.
	PLIB_ADCHS_ExistsAnalogInputCheck	Identifies whether the System Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputModeControl	Identifies whether the analog input mode control exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputScan	Identifies whether the Analog input Scan exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelAnalogControl	Identifies whether the Channel Analog control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelConfiguration	Identifies whether the Channel Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelDigitalControl	Identifies whether the Channel Digital control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelInputSelectControl	Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConversionResults	Identifies whether the Conversion Results feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsCVD	Identifies whether the CVD exists on the ADCHS module.
	PLIB_ADCHS_ExistsDigitalComparator	Identifies whether the Digital Comparator feature exists on the ADCHS module .
	PLIB_ADCHS_ExistsDigitalFilter	Identifies whether the Digital Filter feature exists on the ADCHS module.

	PLIB_ADCHS_ExistsDMA	Identifies whether the DMA exists on the ADCHS module.
	PLIB_ADCHS_ExistsEarlyInterruptControl	Identifies whether the Early Interrupt control exists on the ADCHS module.
	PLIB_ADCHS_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCHS module
	PLIB_ADCHS_ExistsExternalConversionRequestControl	Identifies whether the External Convert feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsFIFO	Identifies whether the FIFO exists on the ADCHS module.
	PLIB_ADCHS_ExistsManualControl	Identifies whether the Manual control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerControl	Identifies whether the Trigger control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerSampleControl	Identifies whether the Trigger Sample control feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsTurboMode	Identifies whether the Turbo mode feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsUpdateReadyControl	Identifies whether the Update Ready feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsVREFControl	Identifies whether the VREF control exists on the ADCHS module.
	PLIB_ADCHS_ExternalConversionRequestDisable	Disables the external conversion request.
	PLIB_ADCHS_ExternalConversionRequestEnable	Enables the external conversion request.
	PLIB_ADCHS_FIFODataCountGet	Returns the number of data to be read from FIFO.
	PLIB_ADCHS_FIFODataIsAvailable	Used to determine if the FIFO has data ready.
	PLIB_ADCHS_FIFODataIsNegative	Returns the sign of data stored in FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptDisable	Disables the interrupt for FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptEnable	Enables the interrupt for FIFO.
	PLIB_ADCHS_FIFODisable	Disables the FIFO in the High-Speed SAR ADC.
	PLIB_ADCHS_FIFOEnable	Enables the FIFO in the High-Speed SAR ADC
	PLIB_ADCHS_FIFOErrorHasOccurred	Used to determine if the FIFO has encountered an overflow error.
	PLIB_ADCHS_FIFORead	Used to fetch the data result from the FIFO.
	PLIB_ADCHS_FIFOSourceGet	Returns the channel ID using the FIFO.
	PLIB_ADCHS_FIFOSourceSelect	Sets the Channel 0 to 6 using the FIFO.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable	Disables the global level software trigger.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable	Initiates a global level software trigger.
	PLIB_ADCHS_GlobalSoftwareTriggerEnable	Initiate a Global Software Trigger.
	PLIB_ADCHS_ScanCompleteInterruptDisable	Disables the end of scan interrupt.
	PLIB_ADCHS_ScanCompleteInterruptEnable	Enables the end of scan interrupt.
	PLIB_ADCHS_Setup	Configures the High-Speed SAR ADC converter.
	PLIB_ADCHS_SoftwareConversionInputSelect	Selects the analog input of Channel 7 for manual conversion.
	PLIB_ADCHS_SoftwareConversionStart	Triggers the conversion of analog input signal connected to Channel 7.
	PLIB_ADCHS_SoftwareSamplingStart	Enables sampling of analog input for Channel 7.
	PLIB_ADCHS_SoftwareSamplingStop	Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.
	PLIB_ADCHS_TriggerSuspendDisable	Disables trigger suspend.
	PLIB_ADCHS_TriggerSuspendEnable	Suspends all trigger for all ADCHS channels.
	PLIB_ADCHS_TurboModeChannelSelect	Configures the channels for Turbo mode.
	PLIB_ADCHS_TurboModeDisable	Disables Turbo mode for High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeEnable	Enables Turbo mode for the High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeErrorHasOccurred	Returns the error state of Turbo mode.
	PLIB_ADCHS_VREFFaultHasOccurred	Returns the state of VREF fault.
	PLIB_ADCHS_VREFFaultInterruptDisable	Disables the VREF Fault interrupt.
	PLIB_ADCHS_VREFFaultInterruptEnable	Enables the VREF fault interrupt.
	PLIB_ADCHS_VREFIsReady	Returns the state of VREF.
	PLIB_ADCHS_VREFReadyInterruptDisable	Disables the VREF ready interrupt.
	PLIB_ADCHS_VREFReadyInterruptEnable	Enables the VREF ready interrupt.

Description

High-Speed Successive Approximation Register Analog-to-Digital Converter (ADCHS) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADCHS PLIB for all families of Microchip microcontrollers.The definitions in this file are common to ADCHS peripheral.

File Name

plib_adchs.h

Company

Microchip Technology Inc.

ADCP Peripheral Library

This section describes the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library.

Introduction

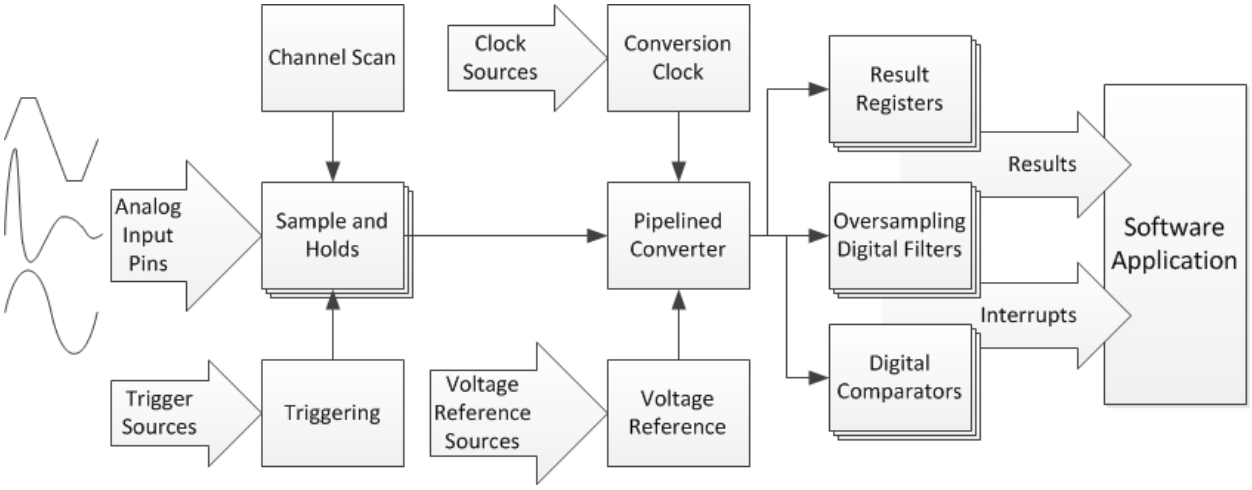
This library provides a low-level abstraction of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Pipelined Analog-to-Digital Converter (ADCP) peripheral is used to convert an analog input into a digital number that represents that input voltage. When the input is periodically converted, the result is a sequence of digital values that have converted a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. This particular ADC architecture is pipelined allowing it to simultaneously convert up to six samples, each at a different stage of conversion resulting in a low overall latency of multiple sampled inputs.

The structure of the ADCP is shown in the following diagram.

- Analog input pins connect to the sample and hold (S&H) circuits. These circuits capture the analog signal so that it can be passed to the pipeline converter.
- The S&H circuits are controlled by predefined trigger sources. These trigger sources switch the sample and hold circuit to hold and queue the conversion of the sample.
- Channel scan is available which allows a single trigger to start a sample/conversion sequence of multiple analog inputs using a predefined scan list
- A conversion clock is required for the pipeline converter. This clock determines the conversion speed and latency and affects power consumption.
- A voltage reference is required for the pipeline converter. This voltage reference determines allowable input range of the analog signal.
- Result registers store the conversion results and can be read by the software application. Software is notified of ready results by either polling or through the use of interrupts.
- Optional Oversampling Digital Filters can be used to provide increased measurement accuracy at the sacrifice of sample rate. Software is notified of ready oversampling results by either polling or through the use of interrupts.
- Optional digital comparators can be used to test conversion results independent of software and generate interrupts based on predefined conditions



Using the Library

This topic describes the basic architecture of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adcp.h`

The interface to the ADCP Peripheral library is defined in the `plib_adcp.h` header file, which is included by the `peripheral.h` file. Any C language source (`.c`) file that uses the Pipelined Analog-to-Digital Converter (ADCP) Peripheral library must include `peripheral.h`.

Library File:

The ADCP Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

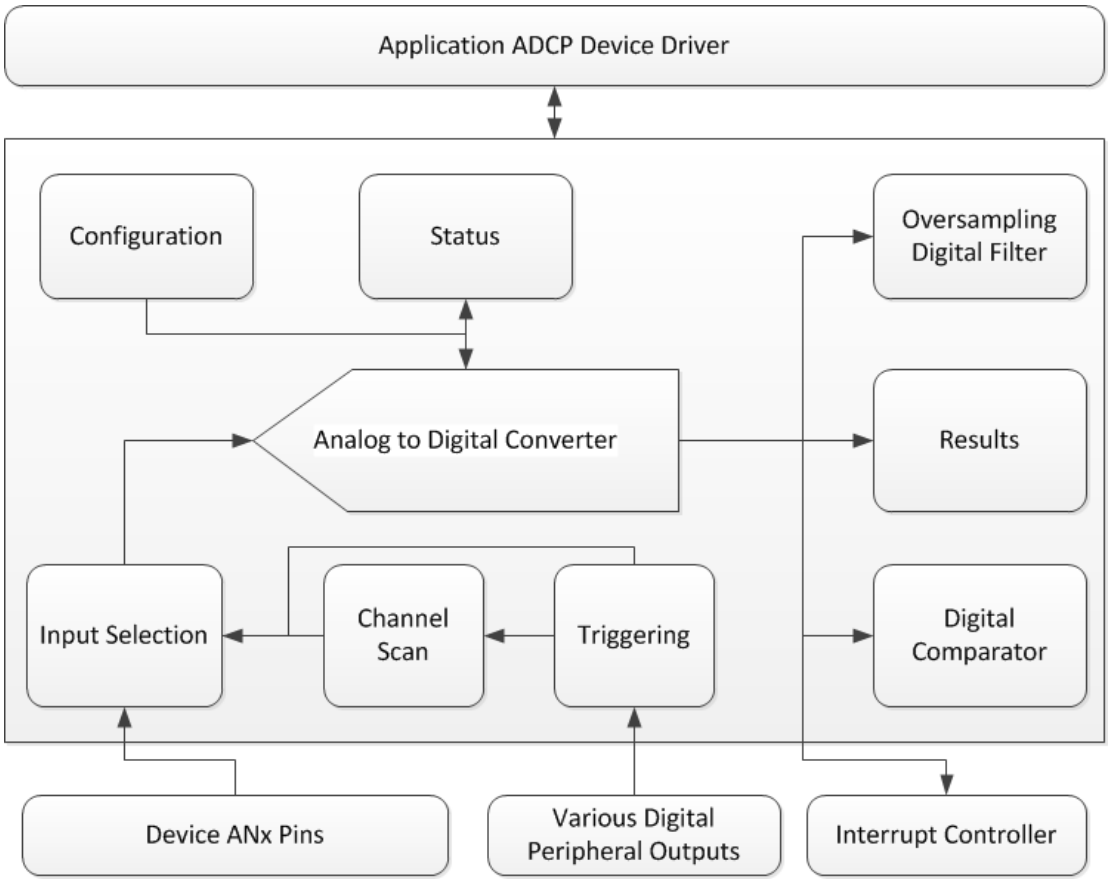
Hardware Abstraction Model

This library provides the low-level abstraction of the ADCP module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a super set of all the functionality of the available Pipelined Analog-to-Digital Converter (ADCP) on the device. Refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each ADCP module on the device.

Description

Pipelined Analog-to-Digital Converter (ADCP) Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADCP module.

Library Interface Section	Description
Configuration Functions	These library functions are used to configure the ADC, enable and disable it, initiate a calibration and manage low power states.
Status Functions	These library functions return status information related to the ADC.
Input Selection Functions	These library functions select and configure the ANx inputs to the ADC.
Channel Scan Functions	These library functions are used to configure the channel scan feature.
Triggering Functions	These library functions are used to configure and initiate conversion triggers.
Analog Input Conversion Results Functions	These library function are used to retrieve individual conversion results for ANx pins.
Digital Comparator Functions	These library functions are used to configure and query the digital comparators.
Oversampling Digital Filter Functions	These library functions are used to configure the Oversampling Digital Filter and fetch results from it.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Core Functionality

This topic describes the core functionality of the ADCP Peripheral Library.

Description

Analog-to-Digital conversion using the ADCP involves the following three steps:

1. Sampling of the input signal.
2. Capture of the input signal by the sample and hold (S&H) circuit and transfer to the converter.
3. Conversion of the analog signal to its digital representation.

Sampling of the input signal involves charging of the S&H capacitor. The sampling time must be adequate so that the capacitor charges to a value equal to the input voltage. At the appropriate time, the input is disconnected and the capacitor voltage is transferred to the converter. The converter then digitizes the analog signal and provides the result.

The converter requires a clock source and a reference voltage. The clock is referred to as the ADC clock and has a period of TAD. The clock and reference voltage sources are selectable, as well as the clock prescaling that determines the TAD.

The ADCP uses two types of S&H circuits: dedicated and shared. Each ADC implementation includes up to five dedicated S&H circuits and a single shared S&H. Analog inputs connected to the S&H circuits are categorized into three types: Class 1, Class 2, and Class 3.

- Class 1 inputs are associated with a dedicated S&H circuit. Each dedicated S&H has a single Class 1 input associated with it. Each Class 1 input has a unique trigger selection register. Class 1 inputs can also be part of a channel scan list, triggered by the common scan trigger source.
- Class 2 inputs are sampled only on the shared S&H and are either individually triggered or as part of a channel scan list. When used individually their trigger source is selected by their unique trigger register.
- Class 3 inputs are sampled only on the shared S&H and are used in channel scan exclusively. They share a common trigger source. When using channel scan it is possible to combine Class 1, Class 2, and Class 3 inputs in the scan list.

Note: For details regarding configuration and triggering options was well as sampling requirements, refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Example - Simultaneous Sampling Three Class 1 Inputs

Simultaneous sampling is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The dedicated Sample and Holds (S&H) circuits are configured for single-ended input and a unipolar (unsigned) output. No interrupts are used so the results are polled for ready status.

Example:

```
int32_t results[3] ;  // storage for results

// This structure is used to simplify testing for a ready channel
AN_READY anReady ;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for class 2/3 inputs or oversampling
// sample time as Class 2/3 inputs and oversampling is not used.
PLIB_ADCP_Configure ( ADCP_ID_1 ,
                      ADCP_VREF_AVDD_AVSS ,    // AVDD and AVSS as reference
                      FALSE ,                  // No VREF boost
                      FALSE ,                  // Do not use fractional format
                      FALSE ,                  // Do not stop in idle
                      ADCP_CLK_SRC_SYSCLK ,    // SYSCLK is the clock source
                      4 ,                      // TAD = 1/SYSCLK * 2 * 4
                                               // or ADC Clock = SYSCLK / (2 * 4)
                      0 ,                      // Oversampling is not used.
                      0 ,                      // No early interrupt.
                      0 );                     // No class 2 or 3 channels are used

// Set up the triggers
// Configure AN0 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
```

```
                                    ADCP_CLASS12_AN0 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;
    // Configure AN1 for triggering from software.
    PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                    ADCP_CLASS12_AN1 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;
    // Configure AN2 for triggering from software.
    PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                    ADCP_CLASS12_AN2 ,
                                    ADCP_TRG_SRC_SOFTWARE ) ;
    // Set S&H modes
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH0 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH1 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH2 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    // Enable the ADC Module
    PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

    // Wait for a ready status
    while (!PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 )) ;

    while (1)
    {
        // Initiate a trigger
        PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

        // Wait for a data results to be ready
        // if lowest priority channel is done then all are done.
        do
        {
            anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
        }
        while (anReady.AN2 == 0) ;

        // fetch AN0
        if (anReady.AN0 == 1)
            results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN0 ) ;

        // fetch AN1
        if (anReady.AN1 == 1)
            results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN1 ) ;

        // fetch AN2
        if (anReady.AN2 == 1)
            results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN2 ) ;
     }
```

Example - Channel Scanning

Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs. The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using the shared S&H. This example uses inputs AN5, AN11 - AN14. AN5 and AN11 are Class 2 inputs. AN13 and AN 14 are Class 3 inputs. The global software trigger is used to initiate the scan. A 4 TAD sample time is specified for the shared S&H. The shared S&H is configured for single-ended operation and a unipolar output. No interrupts are used so the results are polled for ready status.

Example:

```
    int32_t results[5] ;  // storage for results

    // This structure is used to simplify testing of the ready status
    AN_READY anReady ;

    // This structure is used to simplify specifying channels for scan
    // AN5, AN11, AN12, AN13 and AN14 are selected for scanning.
    AN_SELECT anSelect ;
    anSelect.highWord = 0 ;
    anSelect.lowWord = 0 ;
    anSelect.AN5 = 1 ;
    anSelect.AN11 = 1 ;
    anSelect.AN12 = 1 ;
```

```
anSelect.AN13 = 1 ;
anSelect.AN14 = 1 ;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure ( ADCP_ID_1 ,
                      ADCP_VREF_AVDD_AVSS ,    // AVDD and AVSS as reference
                      FALSE ,                  // No VREF boost
                      FALSE ,                  // Do not use fractional format
                      FALSE ,                  // Do not stop in idle
                      ADCP_CLK_SRC_SYSCLK ,    // SYSCLK is the clock source
                      4 ,                      // TAD = 1/SYSCLK * 2 * 4
                                               // or ADC Clock = SYSCLK / (2 * 4)
                      0 ,                      // Oversampling is not used.
                      0 ,                      // No early interrupt.
                      3 ) ;                    // 3 + 1 = 4 TAD for Class 2 and 3
                                               // Sample Time.
// Specify the inputs to include in the Channel Scan using the selections made when
// initializing anSelect.  Use the software trigger to initiate the scan.
PLIB_ADCP_ChannelScanConfigure ( ADCP_ID_1 ,
                                 anSelect.lowWord , anSelect.highWord ,
                                 ADCP_SCAN_TRG_SRC_SOFTWARE ) ;

// Set S&H 5 (shared S&H) to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH5 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;

// Enable the ADC Module
PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 )) ;

while (1)
{
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

    // Wait for a data results to be ready
    // if lowest priority channel is done then all are done.
    do
    {
        anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
    }
    while (anReady.AN14 == 0) ;

    // fetch AN5
    if (anReady.AN5 == 1)
        results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN5 ) ;

    // fetch AN11
    if (anReady.AN11 == 1)
        results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN11 ) ;

    // fetch AN12
    if (anReady.AN12 == 1)
        results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN12 ) ;

    // fetch AN13
    if (anReady.AN13 == 1)
        results[3] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN13 ) ;

    // fetch AN14
    if (anReady.AN14 == 1)
        results[4] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN14 ) ;
}
```

Other Functionality

This topic provides information on additional functionality.

Description

The ADCP also implements a Digital Oversampling Filter and Digital Comparator.

The oversampling digital filter consists of an accumulator and a decimator (down-sampler), which function together as a low-pass filter. By sampling an analog input at a higher-than-required sample rate, and then processing the data through the oversampling digital filter, the number of usable bits of the ADCP module can be increased at the expense of decreased conversion throughput. For example, using 4x oversampling yields one extra usable bit , 16x oversampling yields two extra usable bits, 64x oversampling provides three extra usable bits, and 256x oversampling provides four extra usable bits. Once configured, the application provides a single trigger to the analog input specified for oversampling, it then fetches the result when the process is complete.

The ADCP module features a digital comparator that can be used to monitor selected analog input conversion results and generate an interrupt when a conversion result matches the user-specified limits. Conversion triggers are still required to initiate conversions. The comparison occurs automatically once the conversion is complete. When using a hardware source as a periodic trigger, it is possible to monitor analog inputs and create an interrupt when the converted level matches specified criteria without any software overhead.

Example - Digital Oversampling Filter

The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The following example shows how two extra bits of resolution can be obtained using 16X oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3.5 TAD.

Example:

```
int32_t result; // storage for result

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for Class 2/3 inputs as they are not used.
// Oversampling sample time is set to 3.5 TAD.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVDD and AVSS as reference
                    FALSE,               // No VREF boost
                    FALSE,               // Do not use fractional format
                    FALSE,               // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
                    4,                   // TAD = 1/SYSCLK * 2 * 4
                                         // or ADC Clock = SYSCLK / (2 * 4)
                    2,                   // 2 + 1.5 = 3.5 TAD between
                                         // oversampling triggers
                    0,                   // No early interrupt.
                    0);                  // No Class 2 and 3 are used.

// Set up the triggers
// Configure AN0 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN0, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 0 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH0, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);

// Configure the Oversampling Filter
PLIB_ADCP_OsampDigFilterConfig(ADCP_ID_1,            // ADCP module ID
                               ADCP_ODFLTR1,         // Filter ID
                               ADCP_AN0,             // Oversample AN0
                               ADCP_ODFLTR_16X,      // 16 x oversampling
                               FALSE);               // No Global Int Enable


// Enable ODFLTR0
PLIB_ADCP_OsampDigFilterEnable(ADCP_ID_1, ADCP_ODFLTR1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
```

```
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // Wait for data to be ready
    while (PLIB_ADCP_OsampDigFilterDataRdy(ADCP_ID_1, ADCP_ODFLTR1) == FALSE);

    // get the result
    result = PLIB_ADCP_OsampDigFilterDataGet(ADCP_ID_1, ADCP_ODFLTR1);

}
```

Example - Digital Comparator

The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated

test of AN5 for values >= 80% of full scale or < 20% of full scale. A count is incremented each time an event occurs.

Example:

```
int32_t result; // storage for result
int32_t count = 0;

/* This structure is used to simplify testing for a ready channel */
AN_READY anReady;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVDD and AVSS as reference
                    FALSE,               // No VREF boost
                    FALSE,               // Do not use fractional format
                    FALSE,               // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
                    4,                   // TAD = 1/SYSCLK * 2 * 4 or
                                         // ADC Clock = SYSCLK / (2 * 4)
                    0,                   // no oversampling
                    0,                   // No early interrupt.
                    3);                  // 3 + 1 = 4 TAD for Class 2 and 3 Sample Time.


// Set up the triggers
// Configure AN5 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN5, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 5 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH5, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);

// Configure the Digital Comparator
// Creates an event when the reading of AN5 is less than 20% or
// greater than or equal to 80% of the full scale 12-bit output.
PLIB_ADCP_DigCmpConfig(ADCP_ID_1,           // ADCP module ID
                       ADCP_DCMP1,          // Comparator ID
                       FALSE,               // No Global Int Enable
                       FALSE,               // no test for between low and high
                       TRUE,                // test for greater than or equal to high
                       FALSE,               // no test for less than high
                       FALSE,               // no test for greater than or equal to low
                       TRUE,                // test for less than low
                       1 << 5,              // enable AN5 for comparison
                       0xFFF - (0xFFF / 5), // high limit is 80% of full scale
                       0xFFF / 5);          // low limit is 20% of full scale

// Enable DigCmp1
```

```
PLIB_ADCP_DigCmpEnable(ADCP_ID_1, ADCP_DCMP1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // wait for conversion to complete
    do {
        anReady = PLIB_ADCP_ResultReady(ADCP_ID_1);
    } while (anReady.AN5 == 0);

    // Check result and increment count if an out of range event
    // occurred on digital comparator 1 for AN5
    if (PLIB_ADCP_DigCmpAIdGet(ADCP_ID_1, ADCP_DCMP1) == 5) {
        count++;
    }
}
```

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Configuration Functions

	Name	Description
	PLIB_ADCP_Configure	Configures the Pipelined ADC module including the ADC calibration registers.
	PLIB_ADCP_Enable	Enables the Pipelined ADC module.
	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module.
	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module.

b) Status Functions

	Name	Description
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.
	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module.

c) Input Selection Functions

	Name	Description
	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module.
	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

d) Channel Scan Functions

	Name	Description
	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module.

e) Triggering Functions

	Name	Description
	PLIB_ADCP_GlobalSoftwareTrigger	Initiates a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module.

f) Individual Analog Input Conversion Results Functions

	Name	Description
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.
	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module.

g) Digital Comparator Functions

	Name	Description
	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpAIdGet	Returns the Analog Input ID of for the Comparator Event
	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.
	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module.

h) Oversampling Digital Filter Functions

	Name	Description
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module.
	PLIB_ADCP_OsampDigFilterDataRdy	Determines if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDataGet	Fetches the data result from the Oversampling Digital Filter.
	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module.
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.

i) Data Types and Constants

	Name	Description
	ADCP_MODULE_ID	Identifies the number of ADC Modules Supported.
	ADCP_VREF_SOURCE	Defines the ADCP VREF Source Select.
	ADCP_CLOCK_SOURCE	Defines the ADCP Clock Source Select.
	ADCP_CLASS12_INPUT_ID	Identifies the Class 1 and Class 2 ADC Inputs.
	ADCP_SH_ID	Identifies the supported S&H circuits regardless of type.
	ADCP_DSH_ID	Identifies the supported Dedicated Sample and Hold (S&H) circuits.
	ADCP_SH_MODE	Defines the available modes for the S&H.
	ADCP_INPUT_ID	Identifies the available ADC Inputs.
	ADCP_DCMP_ID	Identifies the supported Digital Comparators.
	ADCP_ODFLTR_ID	Identifies the supported Oversampling Digital Filters.
	ADCP_ODFLTR_OSR	Identifies the supported Digital Filter oversampling ratios.
	ADCP_SCAN_TRG_SRC	Defines the ADCP Channel Scan Trigger Source Selections.
	ADCP_TRG_SRC	Defines the ADCP Trigger Source Selections.
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

Description

This section describes the Application Programming Interface (API) functions of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library.

Refer to each section for a detailed description.

a) Configuration Functions

PLIB_ADCP_Configure Function

Configures the Pipelined ADC module including the ADC calibration registers.

File

plib_adcp.h

C

```
void PLIB_ADCP_Configure(ADCP_MODULE_ID index, ADCP_VREF_SOURCE voltageRefSelect, bool boostVref, bool
fractionalOutputOn, bool stopInIdle, ADCP_CLOCK_SOURCE adcClockSource, int8_t adcClockDivider, int8_t
oversampleDigFilterSamTime, int8_t earlyIntEnable, int8_t class2and3SampleTime);
```

Returns

None.

Description

This function configures all ADC parameters common to all inputs. This configuration must occur prior to enabling the ADC and therefore must be called when the ADC is disabled.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCP_Configure( MY_ADCP_INSTANCE,
        ADCP_VREF_VREFP_VREFN,  // VREF+ and VREF- as reference
        FALSE,          // No VREF boost
        TRUE,           // Use fractional format
        TRUE,           // Do stop in idle
        ADCP_CLK_SRC_SYSCLK,    // SYSCLK is the clock source
        3,              // TAD = 1/SYSCLK * 2 * 3
                        // or ADC Clock = SYSCLK / (2 * 3)
        2,              // 2 + 1.5 = 3.5 TAD between
                        // oversampling triggers
        0,              // No early interrupt.
        3 );            // 3 + 1 = 4 TAD for Class 2 and 3
                        // Sample Time.

// Enable the ADC
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
voltageRefSelect	VREF Source Selection
boostVref	Enables the VREF boost if TRUE.
Use when VREFH	VREFL < .64 (AVDD - AVSS)
fractionalOutputOn	sets output to a fractional format if TRUE

stopInIdle	sets ADC to stop when device is in idle mode if TRUE
adcClockSource	Clock source selection
adcClockDivider	Clock source divider. Values range from 0 to 254.
oversampleDigFilterSamTime	Sets the delay (sample time) between automatically generated oversampling filter triggers. Values range from 0 to 31 for 1.5 TAD to 32.5 TAD respectively.
earlyIntEnable	Sets the number of clocks prior to the actual arrival of data that the ADRDY bit is set. Range is 0 to 7. Used to generate an early interrupt.
class2and3SampleTime	Set the sample time for Class 2 and Class 3 inputs. Range is 0 to 255 for a sample time of 1 to 256, respectively.

Function

void PLIB_ADCP_Configure(ADCP_MODULE_ID index,
ADCP_VREF_SOURCE voltageRefSelect, // voltage reference
bool boostVref, // VREF boost
bool fractionalOutputOn, // result format fractional
bool stopInIdle, // stop in idle
ADCP_CLOCK_SOURCE adcClockSource, // clock source
int8_t adcClockDivider, // clock divider
int8_t oversampleDigFilterSamTime, // sample time between digital filter samples
int8_t earlyIntEnable, // early interrupt enable
int8_t class2and3SampleTime) // Class 2&3 Sample time

PLIB_ADCP_Enable Function

Enables the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_Enable(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected Pipelined ADC module.

Remarks

None

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_Enable(ADCP_MODULE_ID index)

PLIB_ADCP_Disable Function

Pipelined ADC module is disabled (turned OFF).

File

plib_adcp.h

C

```
void PLIB_ADCP_Disable(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected Pipelined ADC module.

Remarks

Not all functionality is available on all devices. Please refer to the specific device data sheet for the list of available features.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_Disable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_Disable(ADCP_MODULE_ID index)

PLIB_ADCP_CalibrationStart Function

Initiates Pipelined ADC Calibration.

File

plib_adcp.h

C

```
void PLIB_ADCP_CalibrationStart(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function initiates calibration of the module. During calibration the module cannot perform conversion.

Remarks

Calibration is complete when PLIB_ADCP_ModuleIsReady() returns a TRUE result.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_CalibrationStart(MY_ADCP_INSTANCE);
    while (!PLIB_ADCP_ModuleIsRead(MY_ADCP_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_CalibrationStart(ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateEnter Function

Places the Pipelined ADC module in a low power state.

File

plib_adcp.h

C

```
void PLIB_ADCP_LowPowerStateEnter(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function places the Pipelined ADC module in a low power state. The feature is used in place of disabling the ADC when power reduction is needed. The Pipelined ADC can come out of low power state and be operational much faster since recalibration is not required.

Remarks

None.

Preconditions

The Pipelined ADC must be enabled.

Example

```
PLIB_ADCP_LowPowerStateEnter(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_LowPowerStateEnter (ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateExit Function

Takes the Pipelined ADC module out of low power state and puts it into an operational mode.

File

plib_adcp.h

C

```
void PLIB_ADCP_LowPowerStateExit(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function takes the Pipelined ADC module out of a low power state and places it into an operational mode.

Remarks

The first five conversions following the exit from ADC Low-power mode may be subject to lower accuracy than specified in the device data sheet.

Preconditions

None.

Example

```
PLIB_ADCP_LowPowerStateExit(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_LowPowerStateExit(ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateGet Function

Returns the state of the low power setting.

File

plib_adcp.h

C

```
bool PLIB_ADCP_LowPowerStateGet(ADCP_MODULE_ID index);
```

Returns

A Boolean indicating the state of the low power setting.

Description

This function returns the state of the low power setting.

Remarks

None.

Preconditions

None.

Example

```
bool lowPowerSate = PLIB_ADCP_LowPowerStateGet(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

bool PLIB_ADCP_LowPowerStateGet(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsCalibration Function

Identifies whether the Calibration feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsCalibration(ADCP_MODULE_ID index);
```

Returns

Existence of the Calibration feature:

- true - When Calibration feature is supported on the device
- false - When Calibration feature is not supported on the device

Description

This function identifies whether the Calibration feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_CalibrationStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsCalibration(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsEnableControl(ADCP_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_Enable
- PLIB_ADCP_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsEnableControl(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsLowPowerControl Function

Identifies whether the LowPowerControl feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsLowPowerControl(ADCP_MODULE_ID index);
```

Returns

Existence of the LowPowerControl feature:

- true - When LowPowerControl feature is supported on the device

- false - When LowPowerControl feature is not supported on the device

Description

This function identifies whether the LowPowerControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_LowPowerStateEnter
- PLIB_ADCP_LowPowerStateExit
- PLIB_ADCP_LowPowerStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsLowPowerControl(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsConfiguration Function

Identifies whether the Configuration feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsConfiguration(ADCP_MODULE_ID index);
```

Returns

Existence of the Configuration feature:

- true - When Configuration feature is supported on the device
- false - When Configuration feature is not supported on the device

Description

This function identifies whether the Configuration feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_Configure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsConfiguration(ADCP_MODULE_ID index)

b) Status Functions

PLIB_ADCP_ModuleIsReady Function

Returns the overall ready status of the module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ModuleIsReady(ADCP_MODULE_ID index);
```

Returns

Boolean indicating ready status.

Description

This function returns the ready status of the Pipelined ADC. The ADC must be ready before operation.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable the module
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);

// wait for calibration to complete
while (!PLIB_ADCP_ModuleIsReady(MY_ADCP_INSTANCE));

// begin sampling
...
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

bool PLIB_ADCP_ModuleIsReady(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsReadyStatus Function

Identifies whether the ReadyStatus feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsReadyStatus(ADCP_MODULE_ID index);
```

Returns

Existence of the ReadyStatus feature:

- true - When ReadyStatus feature is supported on the device
- false - When ReadyStatus feature is not supported on the device

Description

This function identifies whether the ReadyStatus feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_ModuleIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsReadyStatus(ADCP_MODULE_ID index)

c) Input Selection Functions

PLIB_ADCP_AlternateInputSelect Function

Selects the alternate physical input for the specified dedicated (Class 1) S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_AlternateInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id);
```

Returns

None.

Description

This function selects the alternate physical input for the specified S&H.

Remarks

None.

Preconditions

The function only applies to dedicated S&H circuits (Class 1 Inputs).

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the alternate input.
PLIB_ADCP_AlternateInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Function

void PLIB_ADCP_AlternateInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id)

PLIB_ADCP_DefaultInputSelect Function

Selects the default physical input for the specified dedicated (Class 1) S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_DefaultInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id);
```

Returns

None.

Description

This function selects the default physical input for the specified S&H.

Remarks

None.

Preconditions

The function only applies to dedicated S&H circuits (Class 1 Inputs).

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the default input.
PLIB_ADCP_DefaultInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Function

void PLIB_ADCP_DefaultInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id)

PLIB_ADCP_ExistsInputSelect Function

Identifies whether the InputSelect feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsInputSelect(ADCP_MODULE_ID index);
```

Returns

Existence of the InputSelect feature:

- true - When InputSelect feature is supported on the device
- false - When InputSelect feature is not supported on the device

Description

This function identifies whether the InputSelect feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_DedicatedSHInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsInputSelect(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsModeSelect Function

Identifies whether the ModeSelect feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsModeSelect(ADCP_MODULE_ID index);
```

Returns

Existence of the ModeSelect feature:

- true - When ModeSelect feature is supported on the device
- false - When ModeSelect feature is not supported on the device

Description

This function identifies whether the ModeSelect feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_SHModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsModeSelect(ADCP_MODULE_ID index)

PLIB_ADCP_SHModeSelect Function

Selects the mode for the specified S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_SHModeSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id, ADCP_SH_MODE mode);
```

Returns

None.

Description

This function selects the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified S&H.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the single ended, two's complement mode
// selection.
```

```
PLIB_ADCP_SHModeSelect(MY_ADCP_INSTANCE,
                       ADCP_SH1,
                       ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated S&H to configure to use its alternate source
mode	An ADCP_SH_MODE type indicating the mode selection

Function

void PLIB_ADCP_SHModeSelect(ADCP_MODULE_ID index,
ADCP_SH_ID id,
ADCP_SH_MODE mode)

d) Channel Scan Functions

PLIB_ADCP_ChannelScanConfigure Function

Selects input to include in Channel Scan Mode

File

plib_adcp.h

C

```
void PLIB_ADCP_ChannelScanConfigure(ADCP_MODULE_ID index, uint32_t lowEnable, uint32_t highEnable,
ADCP_SCAN_TRG_SRC triggerSource);
```

Returns

None.

Description

This function selects inputs, as determined by the low and high enable scan list for inclusion in the channel scan sequence. If the input is a Class 1 or Class 2 type, it will also select the trigger source for that input to be the scan trigger, which is required if included in channel scanning.

Remarks

The type def AN_SELECT can be used to create a variable to simplify selection of the inputs to include in the channel scan. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure Channel Scanning
// Channels 10 through 13
// Trigger on Timer 1 match.
PLIB_ADCP_ChannelScanConfigure(MY_ADCP_INSTANCE,
                               0XF000, //   AN12 - AN15
                               0X0F00, //  AN24 - AN27
                               ADCP_SCAN_TRG_SRC_TMR1_MATCH)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels

highEnable	bit mask for selecting high order scan channels
triggerSource	Trigger source used to initiate channel scan

Function

void PLIB_ADCP_ChannelScanConfigure(ADCP_MODULE_ID index,
uint32_t lowEnable,
uint32_t highEnable,
ADCP_SCAN_TRG_SRC triggerSource)

PLIB_ADCP_ExistsChannelScan Function

Identifies whether the ChannelScan feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsChannelScan(ADCP_MODULE_ID index);
```

Returns

Existence of the ChannelScan feature:

- true - When ChannelScan feature is supported on the device
- false - When ChannelScan feature is not supported on the device

Description

This function identifies whether the ChannelScan feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_ChannelScanConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsChannelScan(ADCP_MODULE_ID index)

e) Triggering Functions

PLIB_ADCP_GlobalSoftwareTrigger Function

Initiates a Global Software Trigger on the specified module.

File

plib_adcp.h

C

```
void PLIB_ADCP_GlobalSoftwareTrigger(ADCP_MODULE_ID index);
```

Returns

None.

Description

All inputs or scan list that is configured to trigger on the global software trigger are triggered by this function.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCP_GlobalSoftwareTrigger(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_GlobalSoftwareTrigger(ADCP_MODULE_ID index)

PLIB_ADCP_IndividualTrigger Function

Triggers an individual channel independent of the configured trigger source

File

plib_adcp.h

C

```
void PLIB_ADCP_IndividualTrigger(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId);
```

Returns

None.

Description

This function forces a trigger of an individual Class 1 or Class 2 channel independent of its configured trigger source.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Force a trigger of AN5
PLIB_ADCP_IndividualTrigger(MY_ADCP_INSTANCE, ADCP_CLASS12_AN5)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	An ADCP_INPUT_ID type indicating which input to trigger.

Function

void PLIB_ADCP_IndividualTrigger(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId)

PLIB_ADCP_Class12TriggerConfigure Function

Configures a Class 1 or Class 2 Trigger Source.

File

plib_adcp.h

C

```
void PLIB_ADCP_Class12TriggerConfigure(ADCP_MODULE_ID index, ADCP_CLASS12_INPUT_ID inputId, ADCP_TRG_SRC
triggerSource);
```

Returns

none.

Description

This function configures the trigger source for Class 1 or Class 2 inputs. A call to this function is not required when the input is being used as part of a channel scan as the channel scan configure function also configures all trigger sources.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure AN0 for triggering from INT0.
PLIB_ADCP_Class12TriggerConfigure(MY_ADCP_INSTANCE,
                                  ADCP_CLASS12_AN0,
                                  ADCP_TRG_SRC_INT0);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Class 1 or Class 2 input to configure the trigger for.
triggerSource	Trigger source to use for this input.

Function

void PLIB_ADCP_Class12TriggerConfigure(ADCP_MODULE_ID index,
ADCP_CLASS12_INPUT_ID inputId,
ADCP_TRG_SRC triggerSource)

PLIB_ADCP_ExistsTriggering Function

Identifies whether the Triggering feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsTriggering(ADCP_MODULE_ID index);
```

Returns

Existence of the Triggering feature:

- true - When Triggering feature is supported on the device
- false - When Triggering feature is not supported on the device

Description

This function identifies whether the Triggering feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_Class12TriggerConfigure
- PLIB_ADCP_GlobalSoftwareTrigger
- PLIB_ADCP_IndividualTrigger

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsTriggering(ADCP_MODULE_ID index)

f) Individual Analog Input Conversion Results Functions

PLIB_ADCP_ResultReady Function

Returns the ADC conversion result ready bits for the module.

File

plib_adcp.h

C

```
AN_READY PLIB_ADCP_ResultReady(ADCP_MODULE_ID index);
```

Returns

A AN_READY type indicating conversion result status.

Description

This function returns a result indicating which analog inputs have conversion results ready.

Remarks

This function returns individual conversion results. It does not return results from the module.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
AN_READY     MyRdyStatus;

// check for data and process it
if ((MyRdyStatus = PLIB_ADCP_ResultReady(MY_ADCP_INSTANCE)) != 0) {
    // fetch the results and process
}
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

AN_READY PLIB_ADCP_ResultReady(ADCP_MODULE_ID index)

PLIB_ADCP_ResultGet Function

Returns a Pipelined ADC conversion result.

File

plib_adcp.h

C

```
int32_t PLIB_ADCP_ResultGet(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId);
```

Returns

The conversion result expressed as a 32-bit integer.

Description

This function returns the specified Pipelined ADC analog input conversion result.

Remarks

None.

Preconditions

A valid conversion is ready to be fetched.

Example

```
int32_t     result;
...
// fetch result for AN31
result = PLIB_ADCP_ResultGet( MY_ADCP_INSTANCE, ADCP_AN31 );
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Identifier for the input

Function

int32_t PLIB_ADCP_ResultGet(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId)

PLIB_ADCP_ResultReadyGrpIntConfigure Function

Selects input to include in global interrupt.

File

plib_adcp.h

C

```
void PLIB_ADCP_ResultReadyGrpIntConfigure(ADCP_MODULE_ID index, uint32_t lowEnable, uint32_t highEnable);
```

Returns

None.

Description

This function selects inputs, as determined by the input mask channel scan list for inclusion in the global or global interrupt.

Remarks

The type def AN_SELECT can be used to create a variable to simplify selection of the inputs to include in the global interrupt. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure Global Interrupts
// Analog inputs 10 through 13 are included in the global interrupt.
```

```
PLIB_ADCP_ResultReadyGrpIntConfigure(MY_ADCP_INSTANCE,
                                     0x0F00,    // inputs AN8 - AN11
                                     0x000F );  // inputs AN16 - AN19
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels
highEnable	bit mask for selecting high order scan channels

Function

void PLIB_ADCP_ResultReadyGrpIntConfigure(ADCP_MODULE_ID index,

uint32_t lowEnable,

uint32_t highEnable)

PLIB_ADCP_ExistsConversionResults Function

Identifies whether the ConversionResults feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsConversionResults(ADCP_MODULE_ID index);
```

Returns

Existence of the ConversionResults feature:

- true - When ConversionResults feature is supported on the device
- false - When ConversionResults feature is not supported on the device

Description

This function identifies whether the ConversionResults feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_ResultReady
- PLIB_ADCP_ResultGet
- PLIB_ADCP_ResultReadyGrpIntConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsConversionResults(ADCP_MODULE_ID index)

g) Digital Comparator Functions

PLIB_ADCP_DigCmpEnable Function

Enables the Digital Comparator in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpEnable(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Digital Comparator in the specified Pipelined ADC module.

Remarks

None

Preconditions

The digital comparator should be configured using the PLIB_ADCP_Configure() function prior to enabling.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable Digital Comparator 1
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Function

void PLIB_ADCP_DigCmpEnable(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpDisable Function

Disables the Digital Comparator in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpDisable(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Digital Comparator in the specified Pipelined ADC module.

Remarks

None

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Disable Digital Comparator 1
PLIB_ADCP_DigCmpDisable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Function

void PLIB_ADCP_DigCmpDisable(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpAIdGet Function

Returns the Analog Input ID of for the Comparator Event

File

plib_adcp.h

C

```
int16_t PLIB_ADCP_DigCmpAIdGet(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

Value 'x' of type int16_t where 'x' is the index to the analog input ANx, which caused the event or -1 if no event has occurred.

Description

This function tests the DigCmp Event Flag and if true, returns the ANx Identifier for the input that caused the event, or -1 if there was no pending DigCmp event. The ID value returned corresponds to numeric portion of the analog input ID, ANx.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
int16_t     DigCmpResult;

// Get Digital Comparator 1 result.
DigCmpResult = PLIB_ADCP_DigCmpAIdGet(MY_ADCP_INSTANCE, ADCP_DCMP1);
if (DigCmpResult != -1) {
    // process event
}
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module

Function

int16_t PLIB_ADCP_DigCmpAIdGet(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpConfig Function

Configures the Digital Comparator on the Pipelined ADC converter.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpConfig(ADCP_MODULE_ID index, ADCP_DCMP_ID id, bool globalIntEnable, bool
inBetweenOrEqual, bool greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool lessThanLo, uint32_t
inputEnable, int32_t hiLimit, int32_t loLimit);
```

Returns

None.

Description

This function configures all parameters for the Digital Comparator module of the pipelined ADC.

Remarks

This function must be called when the ADC is disabled. The format of hiLimit and loLimit must match the output format of the channel(s) specified in inputEnable.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN0 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCP_DigCmpConfig( MY_ADCP_INSTANCE,   // ADCP module ID
                        ADCP_DCMP1,       // Comparator ID
                        FALSE,            // Global Int Enable
                        TRUE,             // test for between low and high
                        FALSE,            // no test for greater than equal to high
                        FALSE,            // no test for less than high
                        FALSE,            // no test for greater than equal to low
                        FALSE,            // no test for less than low
                        1 << 3,           // enable AN3
                        0x0CCD,           // high limit, 80% of full scale
                        0x0333);          // low limit, 20% of full scale

// Enable Digital Comparator 1
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in this device
globalIntEnable	When set, comparator events are included in the Global Interrupt
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit
lessThanHi	Event is generated when result is less than hiLimit
greaterEqualLo	Event is generated when result is greater than or equal to loLimit
lessThanLo	Event is generated when result is less than loLimit
inputEnable	Bit field which determines which analog inputs are tested by this comparator module. Bit 0 applies to AN0 and bit 31 to AN31.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCP_DigCmpConfig(   ADCP_MODULE_ID index,  // ADCP module ID
                               ADCP_DCMP_ID id,       // Comparator ID
bool globalIntEnable,  // Global Int Enable
bool inBetweenOrEqual, // between low and high
bool greaterEqualHi,   // greater than equal to high
bool lessThanHi,       // less than high
bool greaterEqualLo,   // greater than equal to low
bool lessThanLo,       // less than low
uint32_t inputEnable,  // input enable bits
```

```
int32_t  hiLimit,     // high limit
int32_t  loLimit )    // low limit
```

PLIB_ADCP_ExistsDigCmp Function

Identifies whether the DigitalComparator feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsDigCmp(ADCP_MODULE_ID index);
```

Returns

Existence of the DigitalComparator feature:

- true - When DigitalComparator feature is supported on the device
- false - When DigitalComparator feature is not supported on the device

Description

This function identifies whether the DigitalComparator feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_DigCmpConfig
- PLIB_ADCP_DigCmpEnable
- PLIB_ADCP_DigCmpDisable
- PLIB_ADCP_DigCmpAIdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsDigCmp(ADCP_MODULE_ID index)

h) Oversampling Digital Filter Functions

PLIB_ADCP_OsampDigFilterEnable Function

Enables the Oversampling Digital Filter in the Pipelined ADC module

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterEnable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Remarks

None.

Preconditions

The Oversampling Digital Filter should be configured using the PLIB_ADCP_Configure() function prior to enabling.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable OsampDigFilter1
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module

Function

void PLIB_ADCP_OsampDigFilterEnable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDisable Function

Disables the Oversampling Digital Filter in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterDisable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Disable OsampDigFilter1
PLIB_ADCP_OsampDigFilterDisable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module

Function

void PLIB_ADCP_OsampDigFilterDisable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDataRdy Function

Determines if the Oversampling Digital Filter has data ready.

File

plib_adcp.h

C

```
bool PLIB_ADCP_OsampDigFilterDataRdy(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

Boolean:

- true - Data is ready
- false - Data is not ready

Description

This function can be used to determine if the ADCP digital filter has data ready. A TRUE is returned when data is available, which can be fetched using PLIB_ADCP_OsampDigFilterDataGet.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR_ID_0)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in this device

Function

bool PLIB_ADCP_OsampDigFilterDataRdy(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDataGet Function

Fetches the data result from the Oversampling Digital Filter.

File

plib_adcp.h

C

```
int16_t PLIB_ADCP_OsampDigFilterDataGet(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

A 16-bit result in the format specified by the filter's oversampling setting.

Description

This function is used to fetch data from the Oversampling Digital Filter. The format of the data is determined by the oversampling setting configuration defined in the call of PLIB_ADCP_OsampDigFilterConfig.

Remarks

None.

Preconditions

None.

Example

```
int16_t myODFLTRResult;

if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR1)) {
    // fetch data
    myODFLTRResult = PLIB_ADCP_OsampDigFilterDataGet(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
    // process result
    ...
```

}

Parameters

Parameters	Description
index	Identifier for the ADCP instance
dfltrID	Identifier for the digital Filter in this device

Function

int16_t PLIB_ADCP_OsampDigFilterDataGet(ADCP_MODULE_ID index, ADCP_ODFLTR_ID dfltrID)

PLIB_ADCP_ExistsOsampDigFilter Function

Identifies whether the OsampDigitalFilter feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsOsampDigFilter(ADCP_MODULE_ID index);
```

Returns

Existence of the OsampDigitalFilter feature:

- true - When OsampDigitalFilter feature is supported on the device
- false - When OsampDigitalFilter feature is not supported on the device

Description

This function identifies whether the OsampDigitalFilter feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_OsampDigFilterConfig
- PLIB_ADCP_OsampDigFilterEnable
- PLIB_ADCP_OsampDigFilterDisable
- PLIB_ADCP_OsampDigFilterDataRdy
- PLIB_ADCP_OsampDigFilterDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsOsampDigFilter(ADCP_MODULE_ID index)

PLIB_ADCP_OsampDigFilterConfig Function

Configures the Oversampling Digital Filter on the Pipelined ADC converter.

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterConfig(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id, ADCP_INPUT_ID inputId,
ADCP_ODFLTR_SAMPLE_RATIO oversampleRatio, bool globalIntEnable);
```

Returns

None.

Description

Configures all parameters for the Oversampling Digital Filter module of the pipelined ADC.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Oversampling Digital Filter module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Oversampling Digital Filter
// AN4 is oversampled at a 16X rate.  No global interrupt is enabled.
PLIB_ADCP_OsampDigFilterConfig( MY_ADCP_INSTANCE,   // ADCP module ID
                                ADCP_ODFLTR1,       // Filter ID
                                ADCP_AN4,           // Oversample AN4
                                ADCP_ODFLTR_16X,    // 16 x oversampling
                                FALSE );            // No Global Int Enable

// Enable OsampDigFilter1
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in this device
inputId	Identifier for the analog input to be oversampled
oversampleRatio	Enumerator specifying the oversampling ratio
globalIntEnable	When set, Filter events are included in the Global Interrupt

Function

void PLIB_ADCP_OsampDigFilterConfig(ADCP_MODULE_ID index, // ADCP module ID
ADCP_ODFLTR_ID id, // Filter ID
ADCP_INPUT_ID inputId, // Input Id
ADCP_ODFLTR_OSR oversampleRatio, // Oversampling ratio
bool globalIntEnable) // Global Int Enable

i) Data Types and Constants

ADCP_MODULE_ID Enumeration

Identifies the number of ADC Modules Supported.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ID_1,
  ADC_NUMBER_OF_MODULES
} ADCP_MODULE_ID;
```

Members

Members	Description
ADCP_ID_1	ADC Module 1 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Description

This enumeration identifies the available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADCP_VREF_SOURCE Enumeration

Defines the ADCP VREF Source Select.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_VREF_AVDD_AVSS,
  ADCP_VREF_VREFP_AVSS,
  ADCP_VREF_AVDD_VREFN,
  ADCP_VREF_VREFP_VREFN
} ADCP_VREF_SOURCE;
```

Members

Members	Description
ADCP_VREF_AVDD_AVSS	Reference voltage set to AVDD and AVSS
ADCP_VREF_VREFP_AVSS	Reference voltage set to VREF positive and AVSS
ADCP_VREF_AVDD_VREFN	Reference voltage set to AVDD and VREF negative
ADCP_VREF_VREFP_VREFN	Reference voltage set to VREF positive and VREF negative

Description

ADCP VREF Source Select

This data type defines the ADCP Reference Voltage (VREF) Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_CLOCK_SOURCE Enumeration

Defines the ADCP Clock Source Select.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_CLK_SRC_NONE,
  ADCP_CLK_SRC_SYSCLK,
  ADCP_CLK_SRC_RFCLK3,
  ADCP_CLK_SRC_FRC
} ADCP_CLOCK_SOURCE;
```

Members

Members	Description
ADCP_CLK_SRC_NONE	TAD clock disabled
ADCP_CLK_SRC_SYSCLK	TAD clock set to SYSCLK (TCY)
ADCP_CLK_SRC_RFCLK3	TAD clock set to REFCLK3
ADCP_CLK_SRC_FRC	TAD clock set to FRC

Description

This enumeration data type defines the ADCP Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_CLASS12_INPUT_ID Enumeration

Identifies the Class 1 and Class 2 ADC Inputs.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_CLASS12_AN0,
  ADCP_CLASS12_AN1,
  ADCP_CLASS12_AN2,
  ADCP_CLASS12_AN3,
  ADCP_CLASS12_AN4,
  ADCP_CLASS12_AN5,
  ADCP_CLASS12_AN6,
  ADCP_CLASS12_AN7,
  ADCP_CLASS12_AN8,
  ADCP_CLASS12_AN9,
  ADCP_CLASS12_AN10,
  ADCP_CLASS12_AN11
} ADCP_CLASS12_INPUT_ID;
```

Members

Members	Description
ADCP_CLASS12_AN0	Analog Input AN0
ADCP_CLASS12_AN1	Analog Input AN1
ADCP_CLASS12_AN2	Analog Input AN2
ADCP_CLASS12_AN3	Analog Input AN3
ADCP_CLASS12_AN4	Analog Input AN4
ADCP_CLASS12_AN5	Analog Input AN5
ADCP_CLASS12_AN6	Analog Input AN6
ADCP_CLASS12_AN7	Analog Input AN7
ADCP_CLASS12_AN8	Analog Input AN8
ADCP_CLASS12_AN9	Analog Input AN9
ADCP_CLASS12_AN10	Analog Input AN10
ADCP_CLASS12_AN11	Analog Input AN11

Description

ADC Class 1 and Class 2 Input ID

This data type identifies the Class 1 and Class 2 ADC analog inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SH_ID Enumeration

Identifies the supported S&H circuits regardless of type.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SH0,
  ADCP_SH1,
  ADCP_SH2,
  ADCP_SH3,
  ADCP_SH4,
```

```
  ADCP_SH5,
  ADCP_NUMBER_OF_SH
} ADCP_SH_ID;
```

Members

Members	Description
ADCP_SH0	S&H 0
ADCP_SH1	S&H 1
ADCP_SH2	S&H 2
ADCP_SH3	S&H 3
ADCP_SH4	S&H 4
ADCP_SH5	S&H 5
ADCP_NUMBER_OF_SH	Number of S&H circuits

Description

ADCP S&H Select

This enumeration identifies all supported S&H circuits.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_DSH_ID Enumeration

Identifies the supported Dedicated Sample and Hold (S&H) circuits.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_DSH0,
  ADCP_DSH1,
  ADCP_DSH2,
  ADCP_DSH3,
  ADCP_DSH4,
  ADCP_NUMBER_OF_DSH
} ADCP_DSH_ID;
```

Members

Members	Description
ADCP_DSH0	Dedicated S&H 0
ADCP_DSH1	Dedicated S&H 1
ADCP_DSH2	Dedicated S&H 2
ADCP_DSH3	Dedicated S&H 3
ADCP_DSH4	Dedicated S&H 4
ADCP_NUMBER_OF_DSH	Number of Dedicated S&H circuits

Description

ADCP Dedicated S&H Select

This enumeration identifies the supported Dedicated S&H circuits.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SH_MODE Enumeration

Defines the available modes for the S&H.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR,
  ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP,
  ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR,
  ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP
} ADCP_SH_MODE;
```

Members

Members	Description
ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR	Single-ended input, Unipolar encoded
ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP	Single-ended input, two's compliment encoded
ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR	Differential input, Unipolar encoded
ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP	Differential input, two's compliment encoded

Description

ADCP S&H Mode

This data type defines the available modes for the S&H.

Remarks

None.

ADCP_INPUT_ID Enumeration

Identifies the available ADC Inputs.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_AN0,
  ADCP_AN1,
  ADCP_AN2,
  ADCP_AN3,
  ADCP_AN4,
  ADCP_AN5,
  ADCP_AN6,
  ADCP_AN7,
  ADCP_AN8,
  ADCP_AN9,
  ADCP_AN10,
  ADCP_AN11,
  ADCP_AN12,
  ADCP_AN13,
  ADCP_AN14,
  ADCP_AN15,
  ADCP_AN16,
  ADCP_AN17,
  ADCP_AN18,
  ADCP_AN19,
  ADCP_AN20,
  ADCP_AN21,
  ADCP_AN22,
  ADCP_AN23,
  ADCP_AN24,
  ADCP_AN25,
  ADCP_AN26,
  ADCP_AN27,
  ADCP_AN28,
  ADCP_AN29,
  ADCP_AN30,
  ADCP_AN31,
  ADCP_AN32,
  ADCP_AN33,
  ADCP_AN34,
  ADCP_AN35,
  ADCP_AN36,
```

```
  ADCP_AN37,
  ADCP_AN38,
  ADCP_AN39,
  ADCP_AN40,
  ADCP_AN41,
  ADCP_AN42,
  ADCP_AN43,
  ADCP_AN44,
  ADCP_AN45,
  ADCP_AN46,
  ADCP_AN47,
  ADCP_AN48,
  ADCP_AN49,
  ADCP_AN50,
  ADCP_AN51,
  ADCP_AN52,
  ADCP_AN53,
  ADCP_AN54,
  ADCP_AN55,
  ADCP_AN56,
  ADCP_AN57,
  ADCP_AN58,
  ADCP_AN59,
  ADCP_AN60,
  ADCP_AN61,
  ADCP_AN62,
  ADCP_AN63,
  ADCP_IVREF,
  ADCP_IVTEMP
} ADCP_INPUT_ID;
```

Members

Members	Description
ADCP_AN0	Analog Input AN0
ADCP_AN1	Analog Input AN1
ADCP_AN2	Analog Input AN2
ADCP_AN3	Analog Input AN3
ADCP_AN4	Analog Input AN4
ADCP_AN5	Analog Input AN5
ADCP_AN6	Analog Input AN6
ADCP_AN7	Analog Input AN7
ADCP_AN8	Analog Input AN8
ADCP_AN9	Analog Input AN9
ADCP_AN10	Analog Input AN10
ADCP_AN11	Analog Input AN11
ADCP_AN12	Analog Input AN12
ADCP_AN13	Analog Input AN13
ADCP_AN14	Analog Input AN14
ADCP_AN15	Analog Input AN15
ADCP_AN16	Analog Input AN16
ADCP_AN17	Analog Input AN17
ADCP_AN18	Analog Input AN18
ADCP_AN19	Analog Input AN19
ADCP_AN20	Analog Input AN20
ADCP_AN21	Analog Input AN21
ADCP_AN22	Analog Input AN22
ADCP_AN23	Analog Input AN23
ADCP_AN24	Analog Input AN24
ADCP_AN25	Analog Input AN25
ADCP_AN26	Analog Input AN26
ADCP_AN27	Analog Input AN27
ADCP_AN28	Analog Input AN28
ADCP_AN29	Analog Input AN29

ADCP_AN30	Analog Input AN30
ADCP_AN31	Analog Input AN31
ADCP_AN32	Analog Input AN32
ADCP_AN33	Analog Input AN33
ADCP_AN34	Analog Input AN34
ADCP_AN35	Analog Input AN35
ADCP_AN36	Analog Input AN36
ADCP_AN37	Analog Input AN37
ADCP_AN38	Analog Input AN38
ADCP_AN39	Analog Input AN39
ADCP_AN40	Analog Input AN40
ADCP_AN41	Analog Input AN41
ADCP_AN42	Analog Input AN42
ADCP_AN43	Analog Input AN43
ADCP_AN44	Analog Input AN44
ADCP_AN45	Analog Input AN45
ADCP_AN46	Analog Input AN46
ADCP_AN47	Analog Input AN47
ADCP_AN48	Analog Input AN48
ADCP_AN49	Analog Input AN49
ADCP_AN50	Analog Input AN50
ADCP_AN51	Analog Input AN51
ADCP_AN52	Analog Input AN52
ADCP_AN53	Analog Input AN53
ADCP_AN54	Analog Input AN54
ADCP_AN55	Analog Input AN55
ADCP_AN56	Analog Input AN56
ADCP_AN57	Analog Input AN57
ADCP_AN58	Analog Input AN58
ADCP_AN59	Analog Input AN59
ADCP_AN60	Analog Input AN60
ADCP_AN61	Analog Input AN61
ADCP_AN62	Analog Input AN62
ADCP_AN63	Analog Input AN63
ADCP_IVREF	Analog Input for Internal Voltage Reference
ADCP_IVTEMP	Analog Input for Internal Temperature Sensor

Description

ADC Input ID

This data type identifies the available ADC Inputs, regardless of Class.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all device.

ADCP_DCMP_ID Enumeration

Identifies the supported Digital Comparators.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_DCMP1,
  ADCP_DCMP2,
  ADCP_DCMP3,
  ADCP_DCMP4,
  ADCP_DCMP5,
```

```
  ADCP_DCMP6,
  ADCP_NUMBER_OF_DCMP
} ADCP_DCMP_ID;
```

Members

Members	Description
ADCP_DCMP1	DCMP1
ADCP_DCMP2	DCMP2
ADCP_DCMP3	DCMP3
ADCP_DCMP4	DCMP4
ADCP_DCMP5	DCMP5
ADCP_DCMP6	DCMP6
ADCP_NUMBER_OF_DCMP	Number of Digital Comparators

Description

ADCP Digital Comparator

This enumeration identifies all supported Digital Comparators for this ADCP module.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_ODFLTR_ID Enumeration

Identifies the supported Oversampling Digital Filters.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ODFLTR1,
  ADCP_ODFLTR2,
  ADCP_ODFLTR3,
  ADCP_ODFLTR4,
  ADCP_ODFLTR5,
  ADCP_ODFLTR6,
  ADCP_NUMBER_OF_ODFLTR
} ADCP_ODFLTR_ID;
```

Members

Members	Description
ADCP_ODFLTR1	ODFLTR1
ADCP_ODFLTR2	ODFLTR2
ADCP_ODFLTR3	ODFLTR3
ADCP_ODFLTR4	ODFLTR4
ADCP_ODFLTR5	ODFLTR5
ADCP_ODFLTR6	ODFLTR6
ADCP_NUMBER_OF_ODFLTR	Number of Oversampling Digital Filters

Description

ADCP Oversampling Digital Filter

This enumeration identifies all supported Digital Filters for this ADCP module.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_ODFLTR_OSR Enumeration

Identifies the supported Digital Filter oversampling ratios.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ODFLTR_4X,
  ADCP_ODFLTR_16X,
  ADCP_ODFLTR_64X,
  ADCP_ODFLTR_256X,
  ADCP_ODFLTR_2X,
  ADCP_ODFLTR_8X,
  ADCP_ODFLTR_32X,
  ADCP_ODFLTR_128X
} ADCP_ODFLTR_OSR;
```

Members

Members	Description
ADCP_ODFLTR_4X	4x oversampling, shift sum 1 bit to right, output data is 13 bits
ADCP_ODFLTR_16X	16x oversampling, shift sum 2 bits to right, output data is 14 bits
ADCP_ODFLTR_64X	64x oversampling, shift sum 3 bits to right, output data is 15 bits
ADCP_ODFLTR_256X	256x oversampling, shift sum 4 bits to right, output data is 16 bits
ADCP_ODFLTR_2X	2x oversampling, shift sum 0 bits to right, output data is in 12.1 format
ADCP_ODFLTR_8X	8x oversampling, shift sum 1 bit to right, output data is in 13.1 format
ADCP_ODFLTR_32X	32x oversampling, shift sum 2 bits to right, output data is in 14.1 format
ADCP_ODFLTR_128X	128x oversampling, shift sum 3 bit to right, output data is in 15.1 format

Description

ADCP Oversampling Ratio

This enumeration identifies all supported Digital Filter oversampling ratios for this ADCP module. Oversampling ratios determine the number of samples used to generate a single output and the resulting resolution and format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SCAN_TRG_SRC Enumeration

Defines the ADCP Channel Scan Trigger Source Selections.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SCAN_TRG_SRC_NONE,
  ADCP_SCAN_TRG_SRC_SOFTWARE,
  ADCP_SCAN_TRG_SRC_INT0,
  ADCP_SCAN_TRG_SRC_TMR1_MATCH,
  ADCP_SCAN_TRG_SRC_TMR3_MATCH,
  ADCP_SCAN_TRG_SRC_TMR5_MATCH,
  ADCP_SCAN_TRG_SRC_OCMP1,
  ADCP_SCAN_TRG_SRC_OCMP3,
  ADCP_SCAN_TRG_SRC_OCMP5,
  ADCP_SCAN_TRG_SRC_COMP1_COUT,
  ADCP_SCAN_TRG_SRC_COMP2_COUT
} ADCP_SCAN_TRG_SRC;
```

Members

Members	Description
ADCP_SCAN_TRG_SRC_NONE	No scan trigger source is selected
ADCP_SCAN_TRG_SRC_SOFTWARE	Global Software trigger selected as scan trigger source
ADCP_SCAN_TRG_SRC_INT0	Interrupt 0 (INT0) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as scan trigger source

ADCP_SCAN_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as scan trigger source

Description

ADCP Channel Scan Trigger Source Select

This data type defines the ADCP Channel Scan Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_TRG_SRC Enumeration

Defines the ADCP Trigger Source Selections.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_TRG_SRC_NONE,
  ADCP_TRG_SRC_SOFTWARE,
  ADCP_TRG_SRC_SCAN_TRG,
  ADCP_TRG_SRC_INT0,
  ADCP_TRG_SRC_TMR1_MATCH,
  ADCP_TRG_SRC_TMR3_MATCH,
  ADCP_TRG_SRC_TMR5_MATCH,
  ADCP_TRG_SRC_OCMP1,
  ADCP_TRG_SRC_OCMP3,
  ADCP_TRG_SRC_OCMP5,
  ADCP_TRG_SRC_COMP1_COUT,
  ADCP_TRG_SRC_COMP2_COUT
} ADCP_TRG_SRC;
```

Members

Members	Description
ADCP_TRG_SRC_NONE	No trigger source is selected
ADCP_TRG_SRC_SOFTWARE	Global Software trigger selected as the trigger source
ADCP_TRG_SRC_SCAN_TRG	Use the Channel Scan Trigger as the trigger source (input is part of ch scan)
ADCP_TRG_SRC_INT0	Interrupt 0 (INT0) selected as the trigger source
ADCP_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as the trigger source
ADCP_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as the trigger source
ADCP_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as the trigger source
ADCP_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as the trigger source
ADCP_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as the trigger source
ADCP_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as the trigger source
ADCP_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as the trigger source
ADCP_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as the trigger source

Description

ADCP Trigger Source Select

This data type defines the ADCP Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

AN_SELECT Union

This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.

File

plib_adcp.h

C

```
union AN_SELECT {
  int ret_val;
  struct {
    uint32_t lowWord;
    uint32_t highWord;
  };
  struct {
    uint32_t AN0 : 1;
    uint32_t AN1 : 1;
    uint32_t AN2 : 1;
    uint32_t AN3 : 1;
    uint32_t AN4 : 1;
    uint32_t AN5 : 1;
    uint32_t AN6 : 1;
    uint32_t AN7 : 1;
    uint32_t AN8 : 1;
    uint32_t AN9 : 1;
    uint32_t AN10 : 1;
    uint32_t AN11 : 1;
    uint32_t AN12 : 1;
    uint32_t AN13 : 1;
    uint32_t AN14 : 1;
    uint32_t AN15 : 1;
    uint32_t AN16 : 1;
    uint32_t AN17 : 1;
    uint32_t AN18 : 1;
    uint32_t AN19 : 1;
    uint32_t AN20 : 1;
    uint32_t AN21 : 1;
    uint32_t AN22 : 1;
    uint32_t AN23 : 1;
    uint32_t AN24 : 1;
    uint32_t AN25 : 1;
    uint32_t AN26 : 1;
    uint32_t AN27 : 1;
    uint32_t AN28 : 1;
    uint32_t AN29 : 1;
    uint32_t AN30 : 1;
    uint32_t AN31 : 1;
    uint32_t AN32 : 1;
    uint32_t AN33 : 1;
    uint32_t AN34 : 1;
    uint32_t AN35 : 1;
    uint32_t AN36 : 1;
    uint32_t AN37 : 1;
    uint32_t AN38 : 1;
    uint32_t AN39 : 1;
    uint32_t AN40 : 1;
    uint32_t AN41 : 1;
    uint32_t AN42 : 1;
    uint32_t AN43 : 1;
    uint32_t AN44 : 1;
    uint32_t AN45 : 1;
    uint32_t AN46 : 1;
    uint32_t AN47 : 1;
    uint32_t AN48 : 1;
    uint32_t AN49 : 1;
    uint32_t AN50 : 1;
    uint32_t AN51 : 1;
    uint32_t AN52 : 1;
    uint32_t AN53 : 1;
    uint32_t AN54 : 1;
    uint32_t AN55 : 1;
    uint32_t AN56 : 1;
```

```
    uint32_t AN57 : 1;
    uint32_t AN58 : 1;
    uint32_t AN59 : 1;
    uint32_t AN60 : 1;
    uint32_t AN61 : 1;
    uint32_t AN62 : 1;
    uint32_t AN63 : 1;
  };
};
```

Description

ADCP AN Select Union

The structure labels identify the analog inputs associated with each bit when initializing the structure. Unsigned 32-bit integers, lowWord and highWord are used as arguments to the function.

Remarks

See the Channel Scanning example in the ADCP Help documentation regarding use of this type define.

AN_READY Type

This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

File

plib_adcp.h

C

```
typedef AN_SELECT AN_READY;
```

Description

ADCP AN Ready Union

The structure labels identify the analog inputs associated with each bit for testing ready status of individual inputs. Unsigned 32-bit integers, lowWord and highWord allow for testing ready status of groups of bits using a mask.

Remarks

See the Simultaneous Sampling or Channel Scanning examples in the ADCP Help documentation regarding use of this type define.

Files

Files

Name	Description
plib_adcp.h	ADCP Peripheral Library Interface Header for ADCP common definitions
help_plib_adcp.h	ADCP Peripheral Library interface header file template.

Description

This section lists the source and header files used by the library.

plib_adcp.h

ADCP Peripheral Library Interface Header for ADCP common definitions

Functions

	Name	Description
	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
	PLIB_ADCP_Configure	Configures the Pipelined ADC module including the ADC calibration registers.
	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_DigCmpAIdGet	Returns the Analog Input ID of for the Comparator Event
	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.

	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
	PLIB_ADCP_Enable	Enables the Pipelined ADC module.
	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module.
	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module.
	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module.
	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module.
	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module.
	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module.
	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module.
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module.
	PLIB_ADCP_GlobalSoftwareTrigger	Initiates a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.
	PLIB_ADCP_OsampDigFilterDataGet	Fetches the data result from the Oversampling Digital Filter.
	PLIB_ADCP_OsampDigFilterDataRdy	Determines if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module.
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.
	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

Types

	Name	Description
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

Unions

	Name	Description
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.

Description

Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADCP PLIB for all families of Microchip microcontrollers. The definitions in this file are common to the ADCP peripheral.

File Name

plib_adcp.h

Company

Microchip Technology Inc.

help_plib_adcp.h

ADCP Peripheral Library interface header file template.

Enumerations

	Name	Description
	ADCP_CLASS12_INPUT_ID	Identifies the Class 1 and Class 2 ADC Inputs.
	ADCP_CLOCK_SOURCE	Defines the ADCP Clock Source Select.
	ADCP_DCMP_ID	Identifies the supported Digital Comparators.
	ADCP_DSH_ID	Identifies the supported Dedicated Sample and Hold (S&H) circuits.
	ADCP_INPUT_ID	Identifies the available ADC Inputs.
	ADCP_MODULE_ID	Identifies the number of ADC Modules Supported.
	ADCP_ODFLTR_ID	Identifies the supported Oversampling Digital Filters.
	ADCP_ODFLTR_OSR	Identifies the supported Digital Filter oversampling ratios.
	ADCP_SCAN_TRG_SRC	Defines the ADCP Channel Scan Trigger Source Selections.
	ADCP_SH_ID	Identifies the supported S&H circuits regardless of type.
	ADCP_SH_MODE	Defines the available modes for the S&H.
	ADCP_TRG_SRC	Defines the ADCP Trigger Source Selections.
	ADCP_VREF_SOURCE	Defines the ADCP VREF Source Select.

Description

ADCP Peripheral Library Interface Header File Template

This file is used by the DOM project.

File Name

help_plib_adcp.h

Company

Microchip Technology Inc.

BMX Peripheral Library

This section describes the Bus Matrix (BMX) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Bus Matrix (BMX) modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Bus Matrix is essentially a high-speed switch that establishes connections between devices. Devices can be either master (initiator) or slave (target).

The Bus Matrix also performs these other important functions:

- Partition memory - The Bus Matrix provides the ability to partition both RAM and Flash memory into kernel and user areas. It also controls the type of access, program or data, for each memory area.
- Enable/disable program Flash memory cache for Direct Memory Access (DMA)
- Enable/disable bus error exceptions - The Bus Matrix provides the ability to disable bus error exceptions for debugging. They are enabled by default.
- Set Bus Arbitration mode - The Bus Matrix provides the ability to select between arbitration modes. The arbitration modes control the order and priority of initiator access to peripherals.

Using the Library

This topic describes the basic architecture of the BMX Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_bmx.h`

The interface to the BMX library is defined in the `plib_bmx.h` header file, which is included by the `peripheral.h` peripheral library header file. Any C language source (`.c`) file that uses the BMX library must include `peripheral.h`.

Library File:

The BMX Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

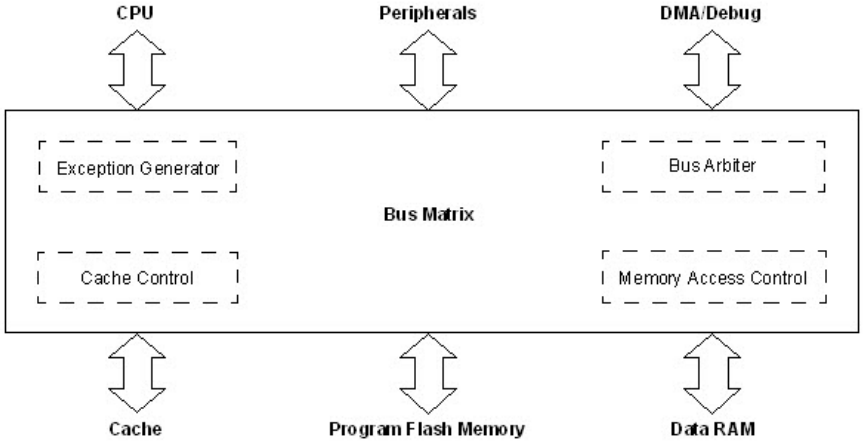
Hardware Abstraction Model

This library provides a low-level abstraction of the BMX module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure shows the Bus Matrix hardware abstraction model.

Hardware Abstraction Model



The BMX Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram.

The **Exception Generator** generates a bus error exception for unmapped address accesses from all Bus Matrix initiators:

- Initiator Expansion Interface (IXI) shared bus
- In-Circuit Debug (ICD) unit
- Direct Memory Access (DMA) controller
- CPU data bus
- CPU instruction bus

Each of the initiators can be individually enabled or disabled with Exception Generator peripheral library routines. All are enabled by default.

The **Cache Control** simply enables/disables program Flash memory (PFM) data cacheability for DMA accesses. The default setting is disabled. When enabled, the PCACHE module must have data caching enabled.

The **Bus Arbitrator** assigns priority levels to bus initiators following one of three priority schemes:

- Fixed, with CPU higher than DMA
- Fixed, with DMA higher than CPU (default)
- Rotating between four priority sequences

The **Memory Access Control** partitions program Flash memory and RAM into kernel and user areas. The RAM can be further partitioned into data and program segments for both kernel and user areas. By default, the full PFM and RAM are mapped to Kernel mode.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the BMX module.

Library Interface Section	Description
Bus Exception Routines	Provide an interface to individually enable or disable initiator access to unmapped memory.
Program Flash Cache Routines	Provide an interface to enable or disable caching of DMA accesses to Program Flash Memory.
Arbiter Routines	Provide an interface to set or change the bus initiator priority sequence.
Memory Access Control Routines	Provide an interface to partition Program Flash and Data Memories into user and kernel regions. Provide an interface to read the size of the Boot Flash, Program Flash and Data SRAM memories.
Feature Existence Routines	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

The Bus Matrix is initialized in the start-up code, before an application is invoked. Most applications will not need to modify the Bus Matrix default settings. The BMX Peripheral Library is available for those applications requiring more complex memory partitioning, including operating systems and applications requiring specialized memory access. The ability to disable exceptions for unmapped memory access may be useful for debugging, but is not recommended for normal use. Changes to the arbitration scheme may improve performance for CPU or DMA intensive applications.

Exception Generator

Any unmapped address access by a Bus Matrix initiator will by default generate an exception. These exceptions may be individually enabled or disabled with the peripheral library.

Example: Enable CPU instruction exception

```
// Enable CPU instruction exception
PLIB_BMX_ISBusExceptionEnable();
```

Cache Control

This peripheral library function enables program Flash memory (data) cacheability for DMA accesses. Data caching must be enabled in the Pcache. Hits are still read from the cache, but misses do not update the cache. This function is disabled by default.

Example: Enable PFM Cacheability for DMA access

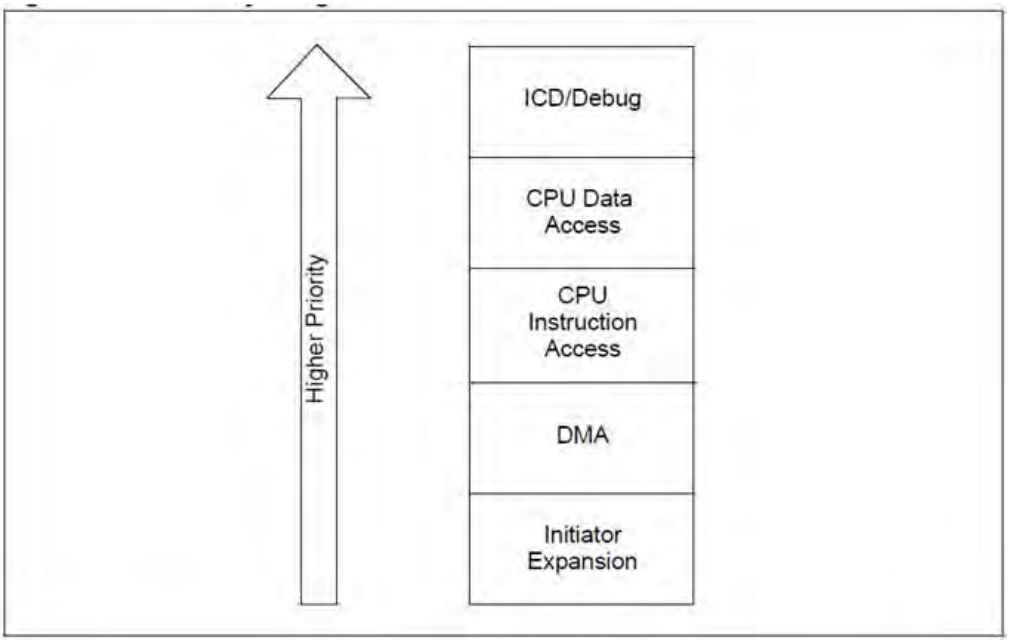
```
PLIB_BMX_EnablePfmCacheDma()
```

Bus Arbiter

Since there can be more than one initiator attempting to access the same target, an arbitration scheme must be used to control access to the target. The arbitration modes assign priority levels to all the initiators. The initiator with the higher priority level will always win target access over a lower priority initiator. The BMX Bus Arbitrator allows the programmer to select from one of three arbitration modes.

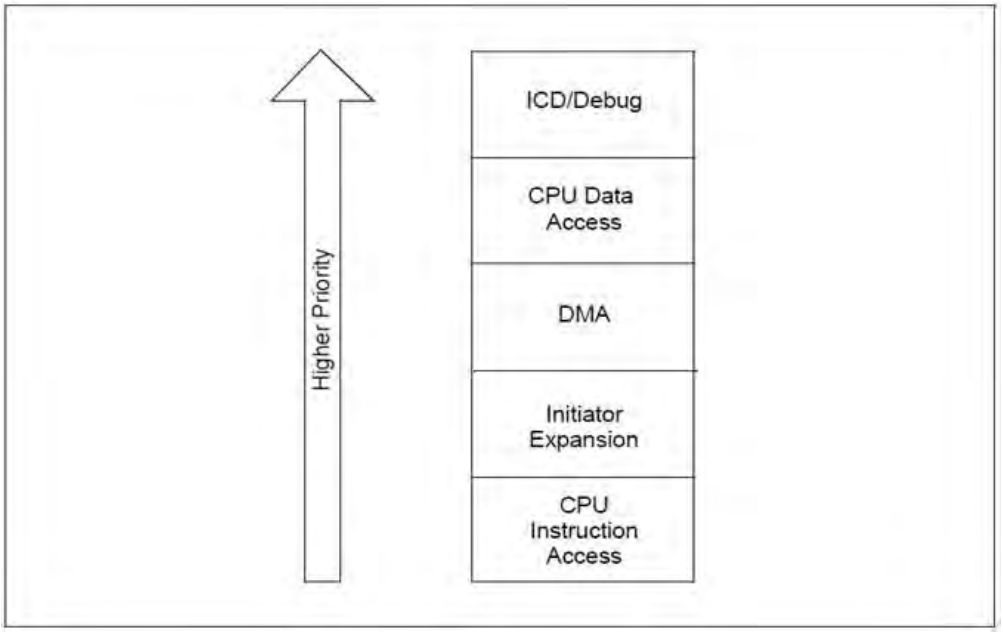
PLIB_BMX_ARB_MODE_INST

Arbitration mode 0 is a fixed priority scheme, with the CPU given higher priority than DMA. This mode can starve the DMA, so choose this mode when DMA is not being used.



PLIB_BMX_ARB_MODE_DMA

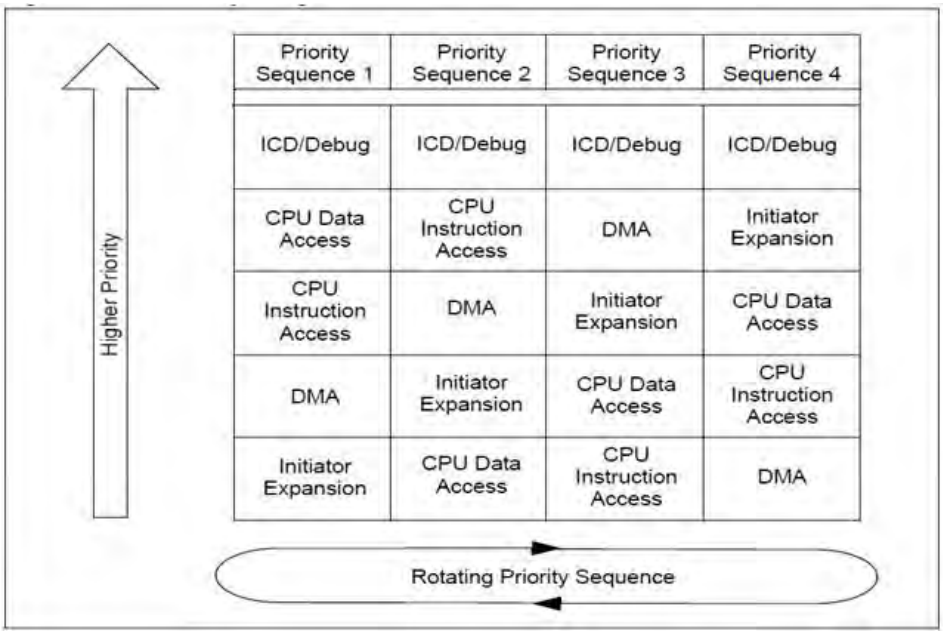
Arbitration Mode 1 is a fixed priority scheme like Mode 0, except that the CPU is always the lowest priority. Mode 1 is the default mode.



PLIB_BMX_ARB_MODE_ROT

Mode 2 arbitration supports rotating priority assignments to all initiators. Instead of a fixed priority assignment, each initiator is assigned the highest priority in a rotating fashion. In this mode, the rotating priority is applied with the following exceptions:

1. CPU data is always selected over CPU instruction.
2. ICD is always the highest priority.
3. When the CPU is processing an exception (EXL = 1) or an error (ERL = 1), the arbitrator temporarily reverts to Mode 0.



Note: Priority Sequence 2 is not selected in the rotation priority scheme if there is a pending CPU data access. In this case, once the data access is complete, Sequence 2 is selected.

Example: Set Bus Arbitration Mode to Rotating Priority

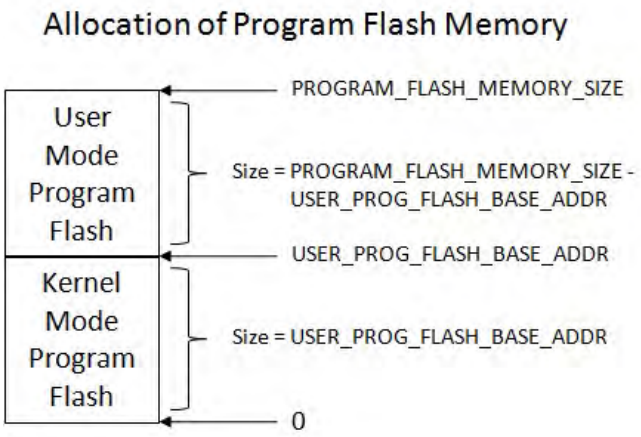
```
PLIB_BMX_SetArbitrationMode(PLIB_BMX_ARB_ROT);
```

Memory Access Control

The Bus Matrix allows the programmer to partition program Flash memory into user and kernel partitions. It also allows the programmer to partition data RAM into user and kernel partitions, each of which can be sub-divided into program and data partitions. At reset, the entire program Flash memory is one kernel partition and the entire data RAM is one kernel data partition.

Program Flash Memory Partitioning

Partitioning of the program Flash memory is accomplished by setting the offset of the user partition within the program Flash memory using the PLIB_BMX_ProgramFlashPartitionSet function. The size of the program Flash memory varies among devices, but can be retrieved with the PLIB_BMX_ProgramFlashMemorySizeGet function. The offset must be a multiple of the program Flash memory block size. The program Flash memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the value will automatically be truncated to a block size boundary. At reset, the entire program Flash memory is mapped to Kernel mode.



Example: Create a User Mode Partition of 12K in Program Flash Memory

```
size_t pfmSize;
size_t userSize = (6 * PLIB_PCACHE_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet();
userOffset = pfmSize - userSize;
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(userOffset);
}
```

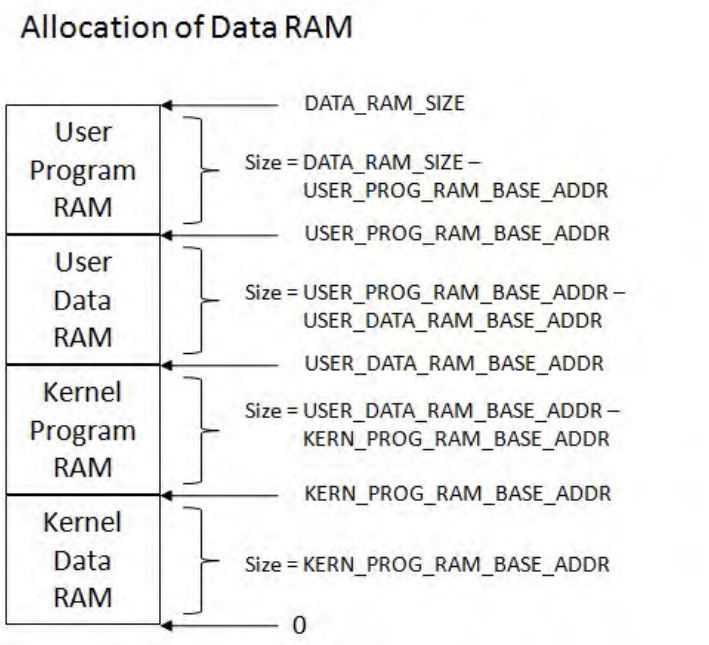
Data RAM Memory Partitioning

The Data RAM Can be partitioned into four sections:

- Kernel Data
- Kernel Program
- User Data
- User Program

Similar to the program Flash memory partitioning, the data RAM partitions are created by setting the base addresses of the various partitions. The BMX Peripheral Library provides a function to set all of the partitions at once. The size of the data RAM varies among devices, but may be retrieved with the PLIB_BMX_DataRAMSizeGet function. The partitions must be a multiple of the Data RAM memory block size. The data RAM memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the partition will automatically be truncated to a block size boundary. At reset, the entire data RAM is mapped to Kernel mode.

Note: At Reset, the entire data RAM is mapped to Kernel mode and all of the offsets are set to zero. To partition the data RAM, all of the offsets must be programmed. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data.



Example: RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

```
//Total Data RAM = 32KB.
size_t totalRamSize;
size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

size_t kernDataRamSize = (3 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Data RAM = 6KB
size_t kernProgRamSize = (2 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Program RAM = 4KB
size_t userDataRamSize = (6 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Data RAM = 12KB
size_t userProgRamSize = (5 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Program RAM = 10KB

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet();

//Verify our partition sizes fit our RAM
if ((kernDataRamSize + kernProgRamSize + userDataRamSize + userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Error\n");
    return -1;
}

kernProgOffset = kernDataRamSize;
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(kernProgOffset, userDataOffset, userProgOffset);
```

Example: Determine the size of Data RAM partitions

```
size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;
size_t kernDataPart;
size_t kernProgPart;
size_t userDataPart;
size_t userProgPart;
size_t totalRamSize;
totalRamSize = PLIB_BMX_DataRAMSizeGet();
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet();
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet();
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet();

kernDataPart = kernProgOffset;
kernProgPart = userDataOffset - kernProgOffset;
```

```
userDataPart = userProgOffset - userDataOffset;
userProgPart = totalRamSize - userProgOffset;
```

Configuring the Library

The library is configured for the supported BMX module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Bus Exception Functions

	Name	Description
	PLIB_BMX_BusExceptionDataDisable	Disables the data bus exception.
	PLIB_BMX_BusExceptionDataEnable	Enables the Data bus exception.
	PLIB_BMX_BusExceptionDMADisable	Disables the DMA bus exception.
	PLIB_BMX_BusExceptionDMAEnable	Enables the DMA bus exception.
	PLIB_BMX_BusExceptionICDDisable	Disables the ICD bus exception.
	PLIB_BMX_BusExceptionICDEnable	Enables the ICD bus exception.
	PLIB_BMX_BusExceptionInstructionDisable	Disables the instruction bus exception.
	PLIB_BMX_BusExceptionInstructionEnable	Enables the instruction bus exception.
	PLIB_BMX_BusExceptionIXIDisable	Disables the IXI bus exception.
	PLIB_BMX_BusExceptionIXIEnable	Enables the IXI bus exception.

b) Program Flash Cache Functions

	Name	Description
	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.

c) Arbiter Functions

	Name	Description
	PLIB_BMX_ArbitrationModeGet	Returns the bus matrix arbitration mode.
	PLIB_BMX_ArbitrationModeSet	Sets the bus matrix arbitration mode.

d) Memory Access Control Functions

	Name	Description
	PLIB_BMX_DataRAMKernelProgramOffsetGet	Gets the offset (from start of RAM) of the kernel program partition.
	PLIB_BMX_DataRAMPartitionSet	Sets the size of the kernel and user partitions in data RAM memory.
	PLIB_BMX_DataRAMSizeGet	Gets the size of data RAM memory.
	PLIB_BMX_DataRAMUserDataOffsetGet	Gets the offset (from start of RAM) of the user data partition.
	PLIB_BMX_DataRAMUserProgramOffsetGet	Gets the offset (from start of RAM) of the user program partition.
	PLIB_BMX_DataRamWaitStateGet	Returns the number of data RAM Wait states.
	PLIB_BMX_DataRamWaitStateSet	Sets the number of data RAM Wait states.
	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

e) Feature Existence Functions

	Name	Description
	PLIB_BMX_ExistsArbitrationMode	Identifies that the ArbitrationMode feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionData	Identifies that the BusExceptionData feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionDMA	Identifies that the BusExceptionDMA feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionICD	Identifies that the BusExceptionICD feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionInstruction	Identifies that the BusExceptionInstruction feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.

	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.

f) Data Types and Constants

	Name	Description
	BMX_MODULE_ID	Lists the possible Module IDs for the bus matrix.
	PLIB_BMX_ARB_MODE	Lists the possible arbitration modes for the bus matrix.
	PLIB_BMX_DATA_RAM_WAIT_STATES	Defines the number of data RAM wait states for address setup.
	PLIB_BMX_EXCEPTION_SRC	Defines which events trigger a bus exception.
	PLIB_BMX_DRM_BLOCK_SIZE	Defines the minimum partition block size in data RAM memory.
	PLIB_BMX_PFM_BLOCK_SIZE	Defines the minimum partition block size in program Flash memory.

Description

This section describes the Application Programming Interface (API) functions of the BMX Peripheral Library.

Refer to each section for a detailed description.

a) Bus Exception Functions

PLIB_BMX_BusExceptionDataDisable Function

Disables the data bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDataDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the data bus exception.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionData in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDataDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionDataDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDataEnable Function

Enables the Data bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDataEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the Data bus exception.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionData in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDataEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionDataEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDMADisable Function

Disables the DMA bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDMADisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA bus exception.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionDMA in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDMADisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionDMADisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDMAEnable Function

Enables the DMA bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDMAEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA bus exception.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionDMA in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDMAEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionDMAEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionICDDisable Function

Disables the ICD bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionICDDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the ICD bus exception.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionICD in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionICDDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionICDDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionICDEnable Function

Enables the ICD bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionICDEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the ICD bus exception.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionICD in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionICDEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionICDEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionInstructionDisable Function

Disables the instruction bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionInstructionDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the instruction bus exception.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionInstruction in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionInstructionDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionInstructionDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionInstructionEnable Function

Enables the instruction bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionInstructionEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the instruction bus exception.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionInstruction in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionInstructionEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionInstructionEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionIXIDisable Function

Disables the IXI bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionIXIDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the IXI bus exception.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionIXI in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionIXIDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionIXIDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionIXIEnable Function

Enables the IXI bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionIXIEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the IXI bus exception.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionIXI in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionIXIEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionIXIEnable(BMX_MODULE_ID index)

b) Program Flash Cache Functions

PLIB_BMX_ProgramFlashMemoryCacheDmaDisable Function

Disables the bus matrix program Flash cacheability for DMA.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the program Flash cacheability for DMA.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemoryCacheDma in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashMemoryCacheDmaEnable Function

Enables the bus matrix program Flash cacheability for DMA.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the program Flash cacheability for DMA.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemoryCacheDma in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_MODULE_ID index)

c) Arbiter Functions

PLIB_BMX_ArbitrationModeGet Function

Returns the bus matrix arbitration mode.

File

plib_bmx.h

C

```
PLIB_BMX_ARB_MODE PLIB_BMX_ArbitrationModeGet(BMX_MODULE_ID index);
```

Returns

PLIB_BMX_ARB_MODE enumerator value representing the arbitration mode.

Description

This function returns the bus matrix arbitration mode.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsArbitrationMode in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ARB_MODE mode;
mode = PLIB_BMX_ArbitrationModeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_BMX_ARB_MODE PLIB_BMX_ArbitrationModeGet (BMX_MODULE_ID index)

PLIB_BMX_ArbitrationModeSet Function

Sets the bus matrix arbitration mode.

File

plib_bmx.h

C

```
void PLIB_BMX_ArbitrationModeSet(BMX_MODULE_ID index, PLIB_BMX_ARB_MODE mode);
```

Returns

None.

Description

This function sets the bus matrix arbitration mode.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsArbitrationMode in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ArbitrationModeSet(BMX_ID_0, PLIB_BMX_ARB_MODE_DMA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Identifies the desired arbitration mode

Function

void PLIB_BMX_ArbitrationModeSet (BMX_MODULE_ID index,
PLIB_BMX_ARB_MODE mode)

d) Memory Access Control Functions

PLIB_BMX_DataRAMKernelProgramOffsetGet Function

Gets the offset (from start of RAM) of the kernel program partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of kernel program partition from base of RAM.

Description

This function returns the offset of start address of the kernel program RAM partition. This represents the size of the kernel data RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t kernProgOffset;
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRAMPartitionSet Function

Sets the size of the kernel and user partitions in data RAM memory.

File

plib_bmx.h

C

```
void PLIB_BMX_DataRAMPartitionSet(BMX_MODULE_ID index, size_t kernProgOffset, size_t userDataOffset, size_t
userProgOffset);
```

Returns

None.

Description

This function sets the size of the kernel and user partitions in the data RAM memory (DRM). By default, the entire data RAM is mapped to Kernel mode and all of the offsets are zero. To partition the data RAM, all of the offsets must be set to a minimum of one DRM block size. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data. The partitions must be a multiple of PLIB_BMX_DRM_BLOCK_SIZE. If programmed with a value that is not a multiple of PLIB_BMX_DRM_BLOCK_SIZE, the value will be automatically truncated to PLIB_BMX_DRM_BLOCK_SIZE.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Total Data RAM = 32 KB.
size_t totalRamSize;
size_t kernDataRamSize = (3 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t kernProgRamSize = (2 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userDataRamSize = (6 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userProgRamSize = (5 * PLIB_BMX_DRM_BLOCK_SIZE);

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);

//Verify our partition sizes fit our RAM
if ((kernDataRamSize + kernProgRamSize + userDataRamSize +
     userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Errorn");
}

size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

kernProgOffset = kernDataRamSize;
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(BMX_ID_0, kernProgOffset,
                             userDataOffset, userProgOffset);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
kernProgOffset	Size of the offset of the Kernel Program partition
userDataOffset	Size of the offset of the User Data partition
userProgOffset	Size of the offset of the User Program partition

Function

void PLIB_BMX_DataRAMPartitionSet(BMX_MODULE_ID index,
size_t kernProgOffset,
size_t userDataOffset,
size_t userProgOffset)

PLIB_BMX_DataRAMSizeGet Function

Gets the size of data RAM memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMSizeGet(BMX_MODULE_ID index);
```

Returns

Size of data RAM memory.

Description

This function returns the size of the data RAM memory.

Remarks

This function implements an operation of the DataRAMSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMSize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t drmSize;
drmSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMSizeGet (BMX_MODULE_ID index)

PLIB_BMX_DataRAMUserDataOffsetGet Function

Gets the offset (from start of RAM) of the user data partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMUserDataOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of user data partition from base of RAM.

Description

This function returns the offset of start address of the user data RAM partition. Subtracting the kernel program offset from the user data offset gives the size of the kernel program RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine

whether this feature is available.

Preconditions

None.

Example

```
size_t userDataOffset;
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

size_t PLIB_BMX_DataRAMUserDataOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRAMUserProgramOffsetGet Function

Gets the offset (from start of RAM) of the user program partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of user data partition from base of RAM.

Description

This function returns the offset of start address of the user program RAM partition. Subtracting the user data offset from the user program offset gives the size of the user data RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t userProgOffset;
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRamWaitStateGet Function

Returns the number of data RAM Wait states.

File

plib_bmx.h

C

```
PLIB_BMX_DATA_RAM_WAIT_STATES PLIB_BMX_DataRamWaitStateGet(BMX_MODULE_ID index);
```

Returns

PLIB_BMX_DATA_RAM_WAIT_STATES enumeration representing the number of wait states.

Description

This function returns the number of data RAM Wait states.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRamWaitState in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_DATA_RAM_WAIT_STATES wait;
wait = PLIB_BMX_DataRamWaitStateGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_BMX_DATA_RAM_WAIT_STATES PLIB_BMX_DataRamWaitStateGet(BMX_MODULE_ID index)

PLIB_BMX_DataRamWaitStateSet Function

Sets the number of data RAM Wait states.

File

plib_bmx.h

C

```
void PLIB_BMX_DataRamWaitStateSet(BMX_MODULE_ID index, PLIB_BMX_DATA_RAM_WAIT_STATES wait);
```

Returns

None.

Description

This function sets the number of data RAM Wait states.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRamWaitState in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_DataRamWaitStateSet(BMX_ID_0, PLIB_BMX_DATA_RAM_WAIT_ONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PLIB_BMX_DATA_RAM_WAIT_STATES	Enumeration representing the number of Wait states

Function

void PLIB_BMX_DataRamWaitStateSet(BMX_MODULE_ID index,
PLIB_BMX_DATA_RAM_WAIT_STATES wait)

PLIB_BMX_ProgramFlashBootSizeGet Function

Gets the size of boot program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashBootSizeGet(BMX_MODULE_ID index);
```

Returns

Size of boot program Flash memory.

Description

This function returns the size of the boot program Flash memory.

Remarks

This function implements an operation of the ProgramFlashBootSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashBootSize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t bootSize;
bootSize = PLIB_BMX_ProgramFlashBootSizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashBootSizeGet (BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashMemorySizeGet Function

Gets the size of program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashMemorySizeGet(BMX_MODULE_ID index);
```

Returns

Size of program Flash memory.

Description

This function returns the size of the program Flash memory.

Remarks

This function implements an operation of the ProgramFlashMemorySize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemorySize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfmSize;
```

```
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashMemorySizeGet (BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashPartitionGet Function

Gets the size of the kernel partition in program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashPartitionGet(BMX_MODULE_ID index);
```

Returns

Size of the kernel partition in program Flash memory.

Description

This function gets the size of the kernel partition in the program Flash memory. The remaining Flash is set to user mode, and may be accessed by user programs. On reset, the entire Flash is mapped to Kernel mode, and this function will return zero.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfmSize;
size_t userPfmSize;
size_t kernPfmSize;

// Get size of PFM
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
kernPfmSize = PLIB_BMX_ProgramFlashPartitionGet(BMX_ID_0);
userPfmSize = pfmSize - kernPfmSize;
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashPartitionGet(BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashPartitionSet Function

Sets the size of the kernel and user partitions in program Flash memory.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashPartitionSet(BMX_MODULE_ID index, size_t user_size);
```

Returns

None.

Description

This function sets the size of the kernel and user partitions in program Flash memory (PFM). Kernel programs may access both partitions, but user programs may not access the kernel partition (including peripheral registers). By default, the entire Flash is mapped to Kernel mode. If called with a non-zero value, a user partition of size user_size is created, and the remaining PFM remains in Kernel mode. The user partition must be a multiple of PLIB_BMX_PFM_BLOCK_SIZE. If programmed with a value that is not a multiple of PLIB_BMX_PFM_BLOCK_SIZE, the value will be truncated to a PLIB_BMX_PFM_BLOCK_SIZE boundary.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfm_size;
size_t userSize = (6 * PLIB_BMX_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
size_t pfmSize;
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
userOffset = pfmSize - userSize;
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(BMX_ID_0, userOffset);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
userSize	Size of the user partition in PFM

Function

void PLIB_BMX_ProgramFlashPartitionSet(BMX_MODULE_ID index,
size_t userSize)

e) Feature Existence Functions

PLIB_BMX_ExistsArbitrationMode Function

Identifies that the ArbitrationMode feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsArbitrationMode(BMX_MODULE_ID index);
```

Returns

- true = The ArbitrationMode feature is supported on the device
- false = The ArbitrationMode feature is not supported on the device

Description

This interface identifies that the ArbitrationMode feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ArbitrationModeSet

- PLIB_BMX_ArbitrationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsArbitrationMode(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionData Function

Identifies that the BusExceptionData feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionData(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionData feature is supported on the device
- false = The BusExceptionData feature is not supported on the device

Description

This interface identifies that the BusExceptionData feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionDataEnable
- PLIB_BMX_BusExceptionDataDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionData(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionDMA Function

Identifies that the BusExceptionDMA feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionDMA(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionDMA feature is supported on the device
- false = The BusExceptionDMA feature is not supported on the device

Description

This interface identifies that the BusExceptionDMA feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionDMAEnable
- PLIB_BMX_BusExceptionDMADisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionDMA(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionICD Function

Identifies that the BusExceptionICD feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionICD(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionICD feature is supported on the device
- false = The BusExceptionICD feature is not supported on the device

Description

This interface identifies that the BusExceptionICD feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionICDEnable
- PLIB_BMX_BusExceptionICDDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionICD(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionInstruction Function

Identifies that the BusExceptionInstruction feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionInstruction(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionInstruction feature is supported on the device
- false = The BusExceptionInstruction feature is not supported on the device

Description

This interface identifies that the BusExceptionInstruction feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionInstructionEnable
- PLIB_BMX_BusExceptionInstructionDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionInstruction(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionIXI Function

Identifies that the BusExceptionIXI feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionIXI(BMX_MODULE_ID index);
```

Returns

true = The BusExceptionIXI feature is supported on the device

- false = The BusExceptionIXI feature is not supported on the device

Description

This interface identifies that the BusExceptionIXI feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionIXIEnable
- PLIB_BMX_BusExceptionIXIDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionIXI(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRAMPartition Function

Identifies that the DataRAMPartition feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRAMPartition(BMX_MODULE_ID index);
```

Returns

- true = The DataRAMPartition feature is supported on the device
- false = The DataRAMPartition feature is not supported on the device

Description

This interface identifies that the DataRAMPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_DataRAMPartitionSet
- PLIB_BMX_DataRAMKernelProgramOffsetGet
- PLIB_BMX_DataRAMUserDataOffsetGet
- PLIB_BMX_DataRAMUserProgramOffsetGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRAMPartition(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRAMSize Function

Identifies that the DataRAMSize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRAMSize(BMX_MODULE_ID index);
```

Returns

- true = The DataRAMSize feature is supported on the device
- false = The DataRAMSize feature is not supported on the device

Description

This interface identifies that the DataRAMSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_DataRAMSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRAMSize(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRamWaitState Function

Identifies that the DataRamWaitState feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRamWaitState(BMX_MODULE_ID index);
```

Returns

- true = The DataRamWaitState feature is supported on the device
- false = The DataRamWaitState feature is not supported on the device

Description

This interface identifies that the DataRamWaitState feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_DataRamWaitStateSet
- PLIB_BMX_DataRamWaitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRamWaitState(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashBootSize Function

Identifies that the ProgramFlashBootSize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashBootSize(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashBootSize feature is supported on the device
- false = The ProgramFlashBootSize feature is not supported on the device

Description

This interface identifies that the ProgramFlashBootSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_ProgramFlashBootSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashBootSize(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashMemoryCacheDma Function

Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashMemoryCacheDma(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashMemoryCacheDma feature is supported on the device
- false = The ProgramFlashMemoryCacheDma feature is not supported on the device

Description

This interface identifies that the ProgramFlashMemoryCacheDma feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ProgramFlashMemoryCacheDmaEnable
- PLIB_BMX_ProgramFlashMemoryCacheDmaDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashMemoryCacheDma(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashMemorySize Function

Identifies that the ProgramFlashMemorySize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashMemorySize(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashMemorySize feature is supported on the device
- false = The ProgramFlashMemorySize feature is not supported on the device

Description

This interface identifies that the ProgramFlashMemorySize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_ProgramFlashMemorySizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashMemorySize(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashPartition Function

Identifies that the ProgramFlashPartition feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashPartition(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashPartition feature is supported on the device
- false = The ProgramFlashPartition feature is not supported on the device

Description

This interface identifies that the ProgramFlashPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ProgramFlashPartitionGet
- PLIB_BMX_ProgramFlashPartitionSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashPartition(BMX_MODULE_ID index)

f) Data Types and Constants

BMX_MODULE_ID Enumeration

Lists the possible Module IDs for the bus matrix.

File

help_plib_bmx.h

C

```
typedef enum {
  BMX_ID_0,
  BMX_NUMBER_OF_MODULES
} BMX_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the bus matrix.

Remarks

Refer to the specific device data sheet to obtain the correct number of modules defined for the desired device.

PLIB_BMX_ARB_MODE Enumeration

Lists the possible arbitration modes for the bus matrix.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_ARB_MODE_INST,
  PLIB_BMX_ARB_MODE_DMA,
  PLIB_BMX_ARB_MODE_ROT
} PLIB_BMX_ARB_MODE;
```

Members

Members	Description
PLIB_BMX_ARB_MODE_INST	Arbitration Mode 0
PLIB_BMX_ARB_MODE_DMA	Arbitration Mode 1
PLIB_BMX_ARB_MODE_ROT	Arbitration Mode 2

Description

Arbitration Mode

This enumeration lists the possible arbitration modes for the bus matrix.

PLIB_BMX_DATA_RAM_WAIT_STATES Enumeration

Defines the number of data RAM wait states for address setup.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_DATA_RAM_WAIT_ZERO,
  PLIB_BMX_DATA_RAM_WAIT_ONE
} PLIB_BMX_DATA_RAM_WAIT_STATES;
```

Members

Members	Description
PLIB_BMX_DATA_RAM_WAIT_ZERO	Zero wait states for address setup
PLIB_BMX_DATA_RAM_WAIT_ONE	One wait state for address setup

Description

Wait States

This definition specifies the number of data RAM wait states for address setup.

PLIB_BMX_EXCEPTION_SRC Enumeration

Defines which events trigger a bus exception.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_ERR_IXI,
  PLIB_BMX_ERR_ICD,
  PLIB_BMX_ERR_DMA,
  PLIB_BMX_ERR_DATA,
  PLIB_BMX_ERR_INST
} PLIB_BMX_EXCEPTION_SRC;
```

Members

Members	Description
PLIB_BMX_ERR_IXI	IXI Shared Bus
PLIB_BMX_ERR_ICD	In-Circuit Debugger
PLIB_BMX_ERR_DMA	DMA Controller
PLIB_BMX_ERR_DATA	CPU Data Bus
PLIB_BMX_ERR_INST	CPU Instruction Bus

Description

Exception Bits

This definition specifies which events trigger a bus exception.

PLIB_BMX_DRM_BLOCK_SIZE Macro

Defines the minimum partition block size in data RAM memory.

File

plib_bmx.h

C

```
#define PLIB_BMX_DRM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in data RAM memory.

PLIB_BMX_PFM_BLOCK_SIZE Macro

Defines the minimum partition block size in program Flash memory.

File

plib_bmx.h

C

```
#define PLIB_BMX_PFM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in program Flash memory.

Files

Files

Name	Description
plib_bmx.h	Defines the Bus Matrix (BMX) Peripheral Library interface
help_plib_bmx.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_bmx.h

Defines the Bus Matrix (BMX) Peripheral Library interface

Enumerations

	Name	Description
	PLIB_BMX_ARB_MODE	Lists the possible arbitration modes for the bus matrix.
	PLIB_BMX_DATA_RAM_WAIT_STATES	Defines the number of data RAM wait states for address setup.
	PLIB_BMX_EXCEPTION_SRC	Defines which events trigger a bus exception.

Functions

	Name	Description
	PLIB_BMX_ArbitrationModeGet	Returns the bus matrix arbitration mode.
	PLIB_BMX_ArbitrationModeSet	Sets the bus matrix arbitration mode.
	PLIB_BMX_BusExceptionDataDisable	Disables the data bus exception.
	PLIB_BMX_BusExceptionDataEnable	Enables the Data bus exception.
	PLIB_BMX_BusExceptionDMADisable	Disables the DMA bus exception.
	PLIB_BMX_BusExceptionDMAEnable	Enables the DMA bus exception.
	PLIB_BMX_BusExceptionICDDisable	Disables the ICD bus exception.
	PLIB_BMX_BusExceptionICDEnable	Enables the ICD bus exception.
	PLIB_BMX_BusExceptionInstructionDisable	Disables the instruction bus exception.
	PLIB_BMX_BusExceptionInstructionEnable	Enables the instruction bus exception.
	PLIB_BMX_BusExceptionIXIDisable	Disables the IXI bus exception.
	PLIB_BMX_BusExceptionIXIEnable	Enables the IXI bus exception.
	PLIB_BMX_DataRAMKernelProgramOffsetGet	Gets the offset (from start of RAM) of the kernel program partition.
	PLIB_BMX_DataRAMPartitionSet	Sets the size of the kernel and user partitions in data RAM memory.
	PLIB_BMX_DataRAMSizeGet	Gets the size of data RAM memory.
	PLIB_BMX_DataRAMUserDataOffsetGet	Gets the offset (from start of RAM) of the user data partition.
	PLIB_BMX_DataRAMUserProgramOffsetGet	Gets the offset (from start of RAM) of the user program partition.
	PLIB_BMX_DataRamWaitStateGet	Returns the number of data RAM Wait states.
	PLIB_BMX_DataRamWaitStateSet	Sets the number of data RAM Wait states.
	PLIB_BMX_ExistsArbitrationMode	Identifies that the ArbitrationMode feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionData	Identifies that the BusExceptionData feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionDMA	Identifies that the BusExceptionDMA feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionICD	Identifies that the BusExceptionICD feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionInstruction	Identifies that the BusExceptionInstruction feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.
	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.

	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.
	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

Macros

	Name	Description
	PLIB_BMX_DRM_BLOCK_SIZE	Defines the minimum partition block size in data RAM memory.
	PLIB_BMX_PFM_BLOCK_SIZE	Defines the minimum partition block size in program Flash memory.

Description

Bus Matrix Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Bus Matrix (BMX) Peripheral Library (PLIB) for Microchip microcontrollers. The definitions in this file are for the bus matrix controller module.

File Name

plib_bmx.h

Company

Microchip Technology Inc.

help_plib_bmx.h

Enumerations

	Name	Description
	BMX_MODULE_ID	Lists the possible Module IDs for the bus matrix.

Description

This file is used for documentation purposes

CAN Peripheral Library

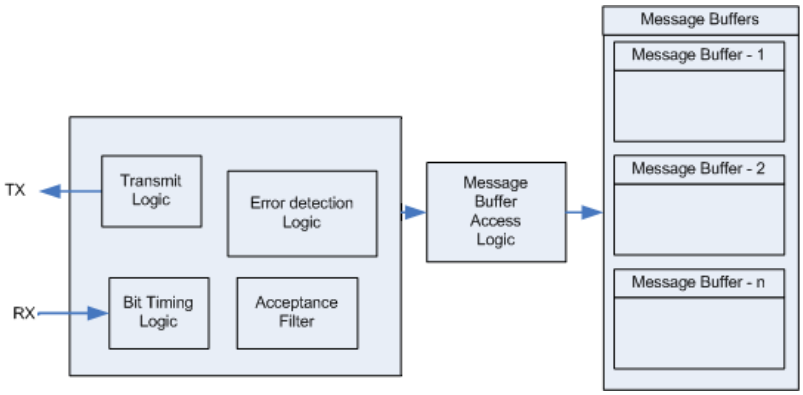
This section describes the CAN Peripheral Library.

Introduction

This library provides a low-level abstraction of the CAN module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Controller Area Network (CAN) is used primarily in industrial and automotive applications. It is an asynchronous serial data communication protocol that provides reliable communication in an electrically noisy environment.



Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the CAN Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_can.h`

The interface to the CAN Peripheral library is defined in the `plib_can.h` header file, which is included in the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the CAN Peripheral library must include `peripheral.h`.

Library File:

The CAN Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

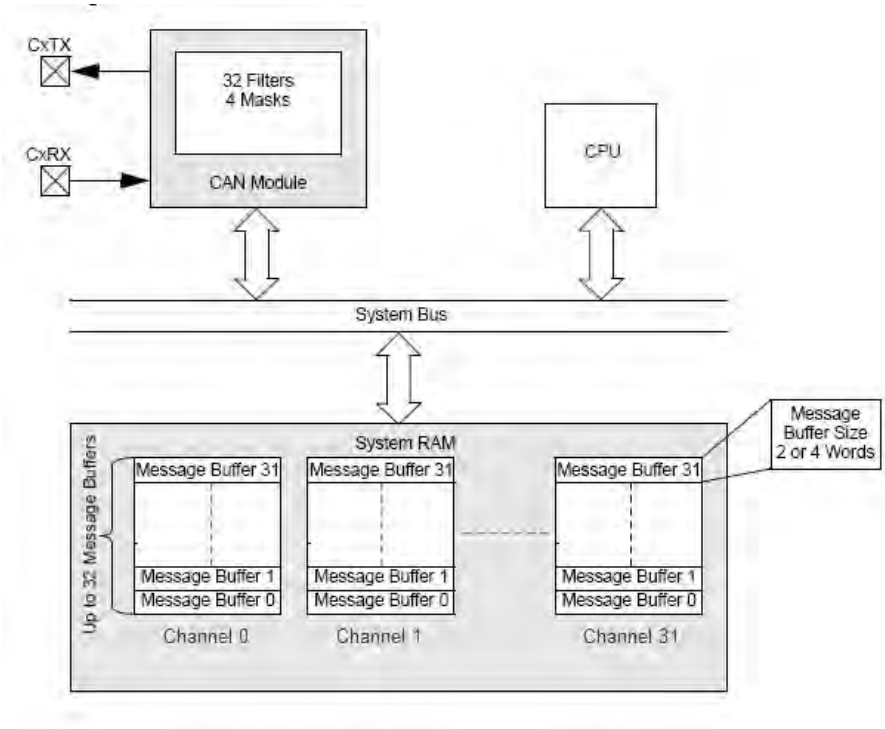
Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

Note: The interface provided is a super set of all of the functionality of the available CAN on the device. Refer to the specific device data sheet or the related family reference manual to determine the set of functions that are supported for each CAN module on the device.

This library provides the low-level abstraction of the CAN module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

CAN Software Abstraction Block Diagram



Software Model:

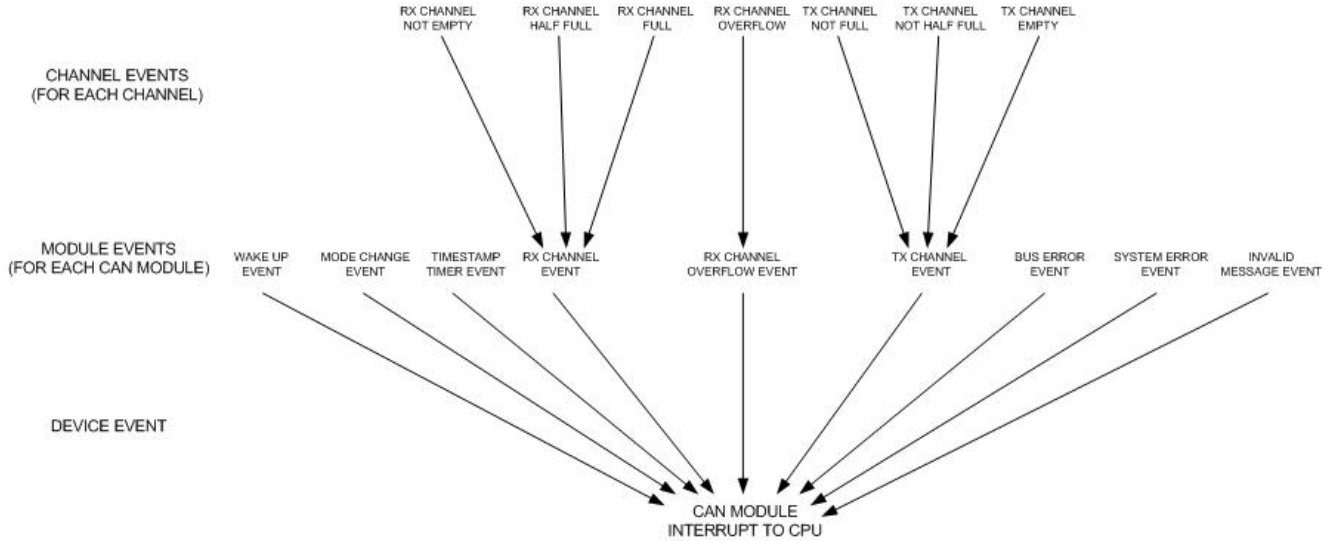
The CAN Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram. A channel can be either a transmit channel or a receive channel. The size of a CAN message is typically 16 bytes (8 bytes for data-only receive messages). The size of a channel (the number of messages it can buffer) is configurable. Each channel has a minimum size of one message buffer. The message buffer area for all channels is assigned at configuration time. The CAN Peripheral Library then provides access to these channels and buffers via the library interface.

The user code must enable and configure message acceptance filters to receive messages. These filters compare the ID field of the incoming message with configured value and accepts the messages if IDs match. The message is then stored in a selectable receive channel. At least one message acceptance filter and one filter mask should be enabled for the CAN module to receive messages. A filter mask allows specified filter bits to be ignored during the comparison process. This allows the filter to accept a message with an ID range.

The CAN Peripheral Library allows the CAN module to generate events. Events can be generated at the channel level and at the module level.

Events:

The CAN module is capable of notifying the user application when specific events occur in the module. The following diagram shows these events and how they are organized.



There are two types of events in the CAN module: Channel events and Module events. Channel events are generated by transmit and receive channels. Module events are generated by various sources (including channels) within the CAN module. Each event can be enabled or disabled. Enabling a channel event will cause the CAN module to generate a module event. An enabled module level event will cause the CAN module to generate an interrupt to the CPU.

The application can also poll channel events to check if these are active. This may be done in cases where a polling scheme (rather than an interrupt scheme) needs to be implemented. In this case, the CAN module provides events status on a channel basis. If a channel event is found active, the application can then use library API calls to inspect which channel condition caused the channel event.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CAN module.

Library Interface Section	Description
Basic Configuration Functions	APIs that are used for basic control of the module, such as enabling and disabling the module.
Advanced Configuration Functions	These functions enable advanced configuration, such as selection of the operation mode.
Bus Speed Configuration Functions	These functions enable bus speed configuration.
Event Management Functions	These functions can be used to manage the events.
Message Transmit Functions	These functions can be used for message transmission.
Message Receive Functions	These functions are related to data reception.
Error State Tracking Functions	These functions are used for error tracking.
Information Functions	These functions provide information about the module.
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

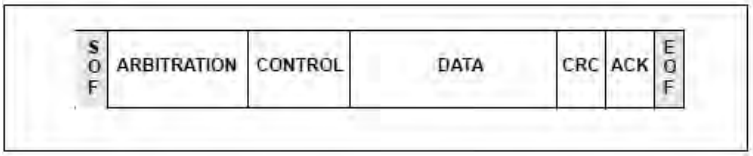
Note: Not all modes are available on all the devices. Please refer to the specific device data sheet to determine the set of modes that are supported for your device.

About CAN Protocol

This section briefly discusses the CAN Protocol briefly. For complete details on the CAN protocol, refer to the CAN 2.0 specification available from Bosch.

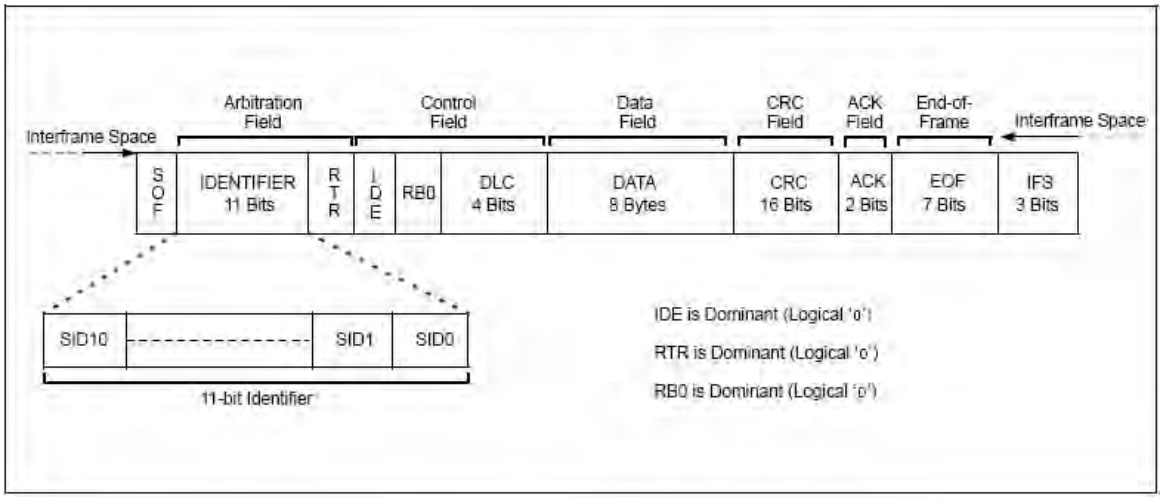
Description

The CAN bus protocol uses asynchronous communication. Information is passed from transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame. This is shown in the following figure.



Each frame begins with a Start-of-Frame (SOF) bit and terminates with an End-of-Frame (EOF) bit field. The SOF is followed by arbitration and control fields, which identify the message type, format, length and priority. This information allows each node on the CAN bus to respond appropriately to the message. The data field conveys the message content and is of variable length, ranging from 0 to 8 bytes. Error protection is provided by the Cyclic Redundancy Check (CRC) and acknowledgement (ACK) fields.

Standard ID Message Format



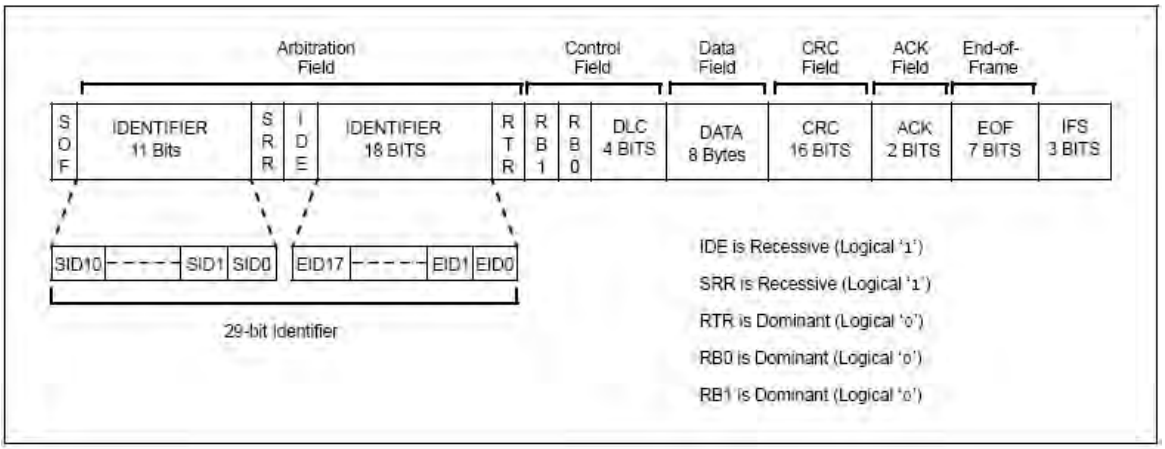
The Standard ID message begins with a Start-of-Frame bit followed by a 12-bit arbitration field. The arbitration field contains an 11-bit identifier and the Remote Transmit Request (RTR) bit. The identifier defines the type of information contained in the message and is used by each receiving node to determine if the message is of interest. The RTR bit distinguishes a data frame from a remote frame. For a standard data frame, the RTR bit is clear.

Following the arbitration field is a 6-bit control field, which provides more information about the contents of the message. The first bit in the control field is an Identifier Extension (IDE) bit, which distinguishes the message as either a standard or extended data frame. A standard data frame is indicated by a dominant state (logic level '0') during transmission of the IDE bit. The second bit in the control field is a reserved (RB0) bit, which is in the dominant state (logic level '0'). The last 4 bits in the control field represent the Data Length Code (DLC), which specifies the number of data bytes present in the message.

The data field follows the control field. This field carries the message data – the actual payload of the data frame. This field is of variable length, ranging from 0 to 8 bytes. The number of bytes is user-selectable.

The data field is followed by the Cyclic Redundancy Check (CRC) field, which is a 15-bit CRC sequence with one delimiter bit. The acknowledgement (ACK) field is sent as a recessive bit (logic level '1') and is overwritten as a dominant bit by any receiver that has received the data correctly. The message is acknowledged by the receiver regardless of the result of the acceptance filter comparison. The last field is the End-of-Frame field, which consists of 7 recessive bits that indicate the end of message.

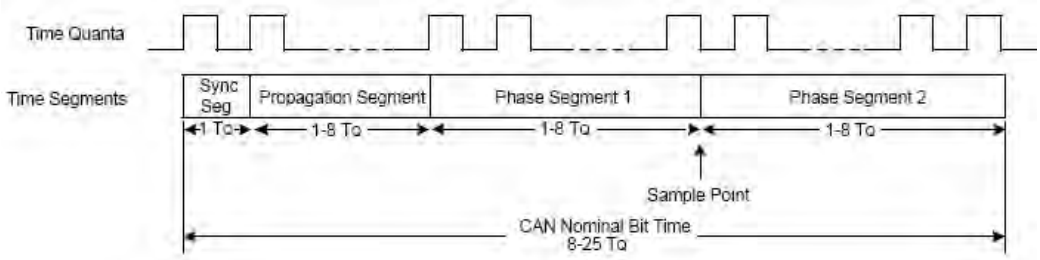
Extended ID Message Format



The Extended ID Message frame begins with an SOF bit followed by a 31-bit arbitration field. The arbitration field for the extended data frame contains 29 identifier bits in two fields separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit determines if the message is a remote frame. SRR = 1 for extended data frames. The IDE bit indicates the data frame type. For the extended data frame, IDE = 1.

The extended data frame Control field consists of 7 bits. The first bit is the RTR. For the extended data frame, RTR = 0. The next two bits, RB1 and RB0, are reserved bits that are in the dominant state (logic level '0'). The last 4 bits in the Control field are the Data Length Code, which specifies the number of data bytes present in the message. The remaining fields in an extended data frame are identical to a standard data frame.

CAN Bit Time Quanta



The nominal bit rate is the number of bits per second transmitted on the CAN bus.

Nominal bit time = 1 ÷ Nominal Bit Rate.

There are four time segments in a bit time to compensate for any phase shifts due to oscillator drifts or propagation delays. These time segments do not overlap each other and are represented in terms of Time Quantum (TQ). One TQ is a fixed unit of time derived from the oscillator clock. The total number of time quanta in a nominal bit time must be programmed between 8 and 25 TQ.

Each bit transmission time consists of four time segments:

Synchronization Segment – This time segment synchronizes the different nodes connected on the CAN bus. A bit edge is expected to be within this segment. Based on CAN protocol, the Synchronization Segment is assumed to be one Time Quantum.

Propagation Segment – This time segment compensates for any time delay that may occur due to the bus line or due to the various transceivers connected on that bus.

Phase Segment 1 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be lengthened during resynchronization to compensate for the phase shift.

Phase Segment 2 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be shortened during resynchronization to compensate for the phase shift. The Phase Segment 2 time can be configured to be either programmable or specified by the Phase Segment 1 time.

Two types of synchronization are used – Hard Synchronization and Resynchronization. A Hard Synchronization occurs once at the start of a frame. Resynchronization occurs inside a frame.

Hard synchronization takes place on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Resynchronization takes place when a bit edge does not occur within the Synchronization Segment in a message. One of the Phase Segments is shortened or lengthened by an amount that depends on the phase error in the signal. The maximum amount that can be used is determined by the Synchronization Jump Width parameter.

The length of Phase Segment 1 and Phase Segment 2 can be changed depending on oscillator tolerances of the transmitting and receiving node. Resynchronization compensates for any phase shifts that may occur due to the different oscillators used by the transmitting and receiving nodes.

Bit Lengthening – If the transmitting node in CAN has a slower oscillator than the receiving node, the next falling edge, and therefore, the sample point, can be delayed by lengthening Phase Segment 1 in the bit time.

Bit Shortening – If the transmitting node in CAN has a faster oscillator than the receiving node, the next falling edge, and therefore, the sample point of the next bit, can be reduced by shortening the Phase Segment 2 in the bit time.

Synchronization Jump Width (SJW) – This determines the synchronization jump width by limiting the amount of lengthening or shortening that can be applied to the Phase Segment 1 and Phase Segment 2 time intervals. This segment should not be longer than Phase Segment 2 time. The width can be 1-4 TQ

Initialization of CAN

This section provides information on initializing CAN, including channel, filters, masks, and mode configuration, setting the bus speed, assigning buffer memory, and enabling events.

Mode Configuration

The CAN module must be enabled before configuration. Enabling the CAN module gives it control over the CAN pins of the device. The following code example shows how the PLIB_CAN_Enable function is used to enable the module.

```
/* This code expects the data direction settings to be done from other modules */
/* Switch the CAN module ON */
PLIB_CAN_Enable(CAN_ID_1);
```

The CAN module must be placed in Configuration mode before attempting to configure it. The following features are available in Configuration mode only:

- Configuring the CAN module clock and CAN bus speed
- Configuring the message acceptance filter mask

- Assigning the channel buffer memory
- Configuring channel size and Receive Channel Data-only mode.

The CAN module operating mode can be changed by using the PLIB_CAN_OperationModeSelect function. The caller should wait until the operating mode is set before proceeding. The PLIB_CAN_OperationModeGet function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Configuration mode. Wait until the switch is complete */
PLIB_CAN_Enable(CAN_ID_1);
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_CONFIGURATION_MODE);
```

The CAN module must be placed in Normal mode to be able to send and receive messages.

The caller should wait until the Normal operating mode is set before proceeding. The PLIB_CAN_OperationModeSelect function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Normal mode. Wait until the switch is complete */
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_NORMAL_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_NORMAL_MODE);
```

Setting Bus Speed

The bus speed at which the CAN node operates is determined by the CAN module clock and the number of Bit Time Quanta (TQ) assigned to the CAN bit.

The following code shows an example of setting the CAN module clock and CAN bus speed. In this example, the number of TQ is 11.

```
/*The CPU core frequency is 80000000 Hz and the desired CAN bus speed is 500 kbps.
  Use ECAN Bit Rate Calculator plug-in to get parameter values for 500 kbps Baud rate*/

    #define MY_CAN_ID       CAN_ID_1
    #define PRESCALE        6
    #define SYNCJUMPWIDTH   1
    #define PROPAGATION     2
    #define SEGMENT1        4
    #define SEGMENT2        4
    #define CAN_CLOCK       80000000ul

    uint32_t baudRate;
    bool result;


    if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
    {
        result = PLIB_CAN_PrecalculatedBitRateSetup( MY_CAN_ID, PRESCALE, SYNCJUMPWIDTH,
                                            PROPAGATION, SEGMENT1, SEGMENT2 );

        if(false == result)
        {
            // Error occurred, handle accordingly
        }
        else
        {
            baudRate = PLIB_CAN_BaudRateGet(MY_CAN_ID, CAN_CLOCK);
        }

    }
```

Assigning Buffer Memory

The CAN module requires RAM memory to store received messages and queue message that need to be transmitted. The total amount of memory required depends on the size of each channel. While each channel has a minimum size of one message buffer (16 bytes), to simplify usage issues, channels should be used in a sequentially ascending order starting with channel 0.

To better understand the buffer memory requirements of the CAN module, consider the following examples:

Example 1: Two transmit channels (each with eight message buffers) and one full receive channel (with 10 message buffers):

- Two transmit channels with eight message buffers: 2 * 8 * 16 = 256 bytes
- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- Total memory required in bytes = 416 bytes

Example 2: Four transmit channels (two with eight message buffers and other two with 10 message buffers) and two full receive channels (one with 10 message buffers and one with 20 message buffers):

- Two transmit channels with eight message buffers: 2 * 8 * 16 = 256 bytes
- Two transmit channels with 10 message buffers: 2 * 10 * 16 = 320 bytes
- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- One receive channel with 20 message buffers: 1 * 20 * 16 = 320 bytes
- Total memory required in bytes = 1056 bytes

Example 3: One transmit channel (with maximum message buffers) and two receive Channels (one full receive with 10 message buffers and one data-only channel with maximum message buffers):

- One transmit channel with maximum (32) message buffers: 1 * 32 * 16 = 512 bytes
- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- One data-only receive channel with 32 message buffers: 1 * 32 * 8 = 256 bytes
- Total memory required in bytes = 928 bytes

The PLIB_CAN_MemoryBufferAssign function is used to assign a memory buffer to the CAN module, which is shown in the following example.

```
    /* Assign the buffer area to the CAN module.
     * In this case assign enough memory for 2
     * channels, each with 8 message buffers.*/

    uint8_t MY_CAN_MessageFifoArea[2 * 8 * 16];
    PLIB_CAN_MemoryBufferAssign(CAN_ID_1, MY_CAN_MessageFifoArea);
```

Channel Configuration

To transmit and receive CAN messages, the user application must configure channels. The PLIB_CAN_ChannelForTransmitSet function is used to configure a channel for transmit operation. The PLIB_CAN_ChannelForReceiveSet function is used to configure a channel for receive operation.

Note: To receive CAN messages, at least one message acceptance filter with a receive channel must be configured.

A CAN Transmit channel can be enabled to process a Remote Transmit Request (RTR). In this case, the transmit channel will automatically transmit all queued messages when an RTR message is accepted by the CAN module.

The following code shows an example of configuring a CAN Transmit channel.

```
/* Configure CAN1 Channel 0 for Transmit operation. Allocate 8 message buffer,
 * disable the RTR feature and assign low medium priority for transmissions. */

 PLIB_CAN_ChannelForTransmitSet(CAN_ID_1, CAN_CHANNEL0, 8, CAN_TX_RTR_DISABLED,
      CAN_LOW_MEDIUM_PRIORITY);
```

A CAN receive channel can be enabled to store the data payload portion of an accepted message. This mode is called the Data-only mode. The CAN module otherwise stores both the ID and data payload portion of the accepted message.

The following code shows an example of configuring a CAN Receive channel.

```
/* Configure CAN1 Channel 1 for Receive operation. Allocate 8 message buffers,
 * disable the Data-only feature (specify full receive mode). */

 PLIB_CAN_ChannelForReceiveSet(CAN_ID_1, CAN_CHANNEL1, 8, CAN_RX_FULL_RECEIVE);
```

Filters and Masks Configuration

The CAN message acceptance filter allows an application to accept only selected messages from the CAN bus. The CAN module achieves this by comparing the Standard or Extended ID of a message with the configured filter ID. The message is accepted on an ID match. The accepted message is stored in the configured destination channel.

The CAN message acceptance filter mask directs the CAN module to ignore specified bits in the filter comparison. This allows the application to accept messages within an ID range.

At least one filter and one mask must be configured for the CAN module to be able to receive messages. The destination channel for a filter must be a receive channel unless the channel is a transmit channel with the RTR feature enabled.

The PLIB_CAN_FilterConfigure function is used to configure a message acceptance filter. The PLIB_CAN_FilterMaskConfigure function is used to configure a message acceptance filter mask. The filter and mask must be linked together to a channel. This is done using the PLIB_CAN_FilterToChannelLink function. The CAN filter should be enabled to allow messages to be processed. This is done using the PLIB_CAN_FilterEnable function.

To better understand how the filter and masks can be used to implement filtering, consider the following code examples:

Example 1: Configure a CAN filter to accept a Standard ID message with ID 0x201

```
/* Only one message ID is to be accepted. All bits are to be compared
 * therefore, the mask value should be 0x7FF. The message type is Standard ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x201, CAN_SID);
```

```
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x7FF, CAN_SID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 2: Configure a CAN filter to accept an Extended ID message with ID 0x8004001

```
/* Only one message ID is to be accepted. All bits are to be compared
 * therefore, the mask value should be 0x8004001. The message type is Extended ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004001, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFF, CAN_EID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 3: Configure a CAN filter to accept an Extended ID message within ID range 0x8004000 - 0x8004003

```
/* A message ID range is to be implemented. The 2 least significant bits can be
 * ignored. Therefore, the mask value should be 0x3FFFFFFC. The message type is
 * Extended ID. Configure CAN1, Filter 0, Mask 0 and set the destination channel
 * as Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004000, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFC,
    CAN_EID, CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Note: Refer to About CAN Protocol for more information on Standard and Extended ID addressing.

Enabling Events

The CAN module can generate events that indicate the status of the module. Events can be generated at the channel level and at the module level. Enabling a channel event will cause the CAN module to generate a module level event when the channel event occurs. Enabling a module event will cause the CAN module to interrupt the CPU.

The following code shows an example of using the PLIB_CAN_ChannelEventEnable and PLIB_CAN_ModuleEventEnable functions to enable channel and module events.

```
    /* Enable the Receive channel not empty and Receive channel
     * full events at the channel level. Enable the Receive event
     * at the module level. */
    PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,
               (CAN_RX_CHANNEL_NOT_EMPTY|CAN_RX_CHANNEL_FULL));
    PLIB_CAN_ModuleEventEnable (CAN_ID_1, CAN_RX_EVENT);
```

Note: The caller must use the Interrupt Peripheral Library to enable the CAN CPU interrupt.

Transmitting a CAN Message

The caller application can transmit a CAN message by queuing the message in a transmit channel, updating the channel, and then flushing it. Flushing a transmit channel will cause the CAN module to transmit all of the CAN messages that are queued in the channel.

The PLIB_CAN_TransmitBufferGet function should be used to obtain a CAN_TX_MSG_BUFFER handle type to an empty message buffer. The application should then populate this buffer. The PLIB_CAN_ChannelUpdate function must be called when the buffer has been populated. Finally, the PLIB_CAN_TransmitChannelFlush function is called to flush the channel.

The following code shows an example of transmitting an Extended ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * My_Message;

/* Get a pointer to the next buffer in the Transmit channel
 * check if the returned value is null. */
My_Message = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(My_Message != NULL)
{
```

```
    /* Form a Extended ID CAN message. Start by clearing the buffer.
    Set the IDE bit to 1 to indicate extended ID message. Clear RTR bit
    as this is not a RTR message. */
    My_Message->messageWord[0] = 0;
    My_Message->messageWord[1] = 0;
    My_Message->messageWord[2] = 0;
    My_Message->messageWord[3] = 0;
    My_Message->msgSID.sid    = 0x200;
    My_Message->msgEID.eid    = 0x4002;
    My_Message->msgEID.ide    = 1;
    My_Message->msgEID.sub_remote_req    = 0;
    My_Message->msgEID.data_length_code  = 1;
    My_Message->data[0]       = 0xAA;

    /* This function lets the CAN module
     know that the message processing is done.*/
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

    /* Direct the CAN module to flush the
     Transmit channel. This will send any pending
     message in the Transmit channel. */
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}
```

Note: The PLIB_CAN_TransmitBufferGet function will return NULL if the transmit channel is full. In this occurs, the PLIB_CAN_TransmitChannelFlush function should be called to empty the channel.

The following code shows an example of transmitting a Standard ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * myMessage;

/* Get a pointer to the next buffer in the channel
 check if the returned value is null. */
myMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(myMessage != NULL)
{

    /* Form a Standard ID CAN message. Start by clearing the buffer.
     Send message to the CAN2 module. IDE = 0 means Standard ID message.
     Send one byte of data.
     */
    myMessage->messageWord[0] = 0;
    myMessage->messageWord[1] = 0;
    myMessage->messageWord[2] = 0;
    myMessage->messageWord[3] = 0;

    myMessage->msgSID.sid     = 0x202;
    myMessage->msgEID.ide     = 0;
    myMessage->msgEID.data_length_code     = 1;
    myMessage->data[0]        = 0x23;

    /* This function lets the CAN module
     know that the message processing is done
     and message is ready to be processed. */
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

    /* Direct the CAN module to flush the
     Transmit channel. This will send any pending
     message in the Transmit channel. */
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}
```

Receiving a CAN Message

The caller application can use either the interrupt or polling technique to detect when a message has been received.

The following code shows how the PLIB_CAN_ReceivedMessageGet function can be used to read a received message.

```
CAN_RX_MSG_BUFFER * message;
```

```
message = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL1);
if(message != NULL)
{
    //Process the message received

   /* Call the PLIB_CAN_ChannelUpdate function to let
    the CAN module know that the message processing
    is done. */
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);
}
```

The PLIB_CAN_ReceivedMessageGet function returns an address pointer after converting it from physical address space to virtual address space.

Handling Events

The CAN module can generate events at the channel level and at the module level. An enabled channel event will cause a module event. An enabled module event will cause a CAN CPU interrupt. See Channel Events and Module Events more information on each type of event.

Channel Events

Each channel in the module has its own set of events. The type of events are common to all channels. While a specific channel event may be active and can be inspected, it will cause a module event only if the channel event is enabled.

The following code shows an example of enabling and disabling channel events using the PLIB_CAN_ChannelEventEnable function.

```
/* CAN 1 Channel 1 and 3 are Transmit channels and Channel 2 and 4 are Receive
 channels */
/* Enable Channel 1 empty event and channel not full event. Disable Channel 2 full
 and overflow event */
/* Enable all Transmit events on Channel 2 and disable all RX events on Channel 4 */

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,
    (CAN_TX_CHANNEL_EMPTY | CAN_TX_CHANNEL_NOT_FULL));
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2,
    (CAN_RX_CHANNEL_FULL | CAN_RX_CHANNEL_OVERFLOW));
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);
```

The PLIB_CAN_ModuleEventGet function can be used to check if a specific event is active in a specific channel.

It is only necessary for the event to be active and not enabled for this to work, as shown in the following code example.

```
// Check if RX Channel 2 of CAN module 1 is not empty or if its full.
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}
```

If a channel event is enabled and is active, it will set a channel event status flag. This flag indicates that some event in the channel is active.

The following code example shows how the PLIB_CAN_AllChannelEventsGet function is used to check if any channel event is active.

```
/* Check if RX Channel 2 of CAN module 1 is not empty or if its full */
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}
```

A receive channel overflow condition is special channel event and is available as a module event as well. The PLIB_CAN_AllChannelOverflowStatusGet function returns the status of all channels that have an active overflow event. Only channels with enabled overflow events will be reflected.

The following code shows an example of how this function is used.

```
/* Check if Channel 2 or 3 of CAN module 1 have any active events */
CAN_CHANNEL_MASK channelEvents;
```

```
channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);

if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}
```

Module Events

The CAN module has various sources of events, which includes the channel events. The PLIB_CAN_ModuleEventEnable function can be used to enable and disable module level events. Enabling a module event will cause the CAN module to generate a CPU interrupt, as shown in the following code example.

```
// Enable CAN1 module Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventDisable(CAN_ID_1,
    (CAN_OPERATION_MODE_CHANGE_EVENT | CAN_RX_OVERFLOW_EVENT));
```

The PLIB_CAN_ModuleEventGet function can be used to check if any module level events are active. This is shown in the following code example.

```
// Check if the CAN Module 1 Receive event
// or if Transmit event is active
if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.
}
```

The CAN module also provides code to indicate the highest priority pending event in the CAN module. The PLIB_CAN_PendingEventsGet function can be used to read and analyze this code, as shown in the following code example.

```
/* Implement a switch to check and process any active CAN module events */
CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
{
    case CAN_NO_EVENT:
        /* Procedure to handle no CAN events */
        break;
    case CAN_WAKEUP_EVENT:
        /* Procedure to handle device wake-up on CAN bus activity event */
        break;
    default:
        break;
}
```

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Basic Configuration Functions

	Name	Description
	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.

	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_TimeStampPrescalerSet	Sets the CAN receive message time stamp timer prescaler.
	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.

b) Advanced Configuration Functions

	Name	Description
	PLIB_CAN_ChannelResetIsComplete	Returns 'true' if the specified channel reset is complete.
	PLIB_CAN_DeviceNetConfigure	Configures the CAN module DeviceNet(TM) filter feature.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.

c) Bus Speed Configuration Functions

	Name	Description
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_BaudRateGet	Returns the current CAN module Baud rate.
	PLIB_CAN_BaudRatePrescaleSetup	Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.
	PLIB_CAN_PrecalculatedBitRateSetup	Sets the desired Baud rate for the respective CAN module.
	PLIB_CAN_BitSamplePhaseSet	Sets the CAN bit-sampling phase parameters.

d) Channel Configuration Functions

	Name	Description
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.

e) Event Management Functions

	Name	Description
	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
	PLIB_CAN_ModuleEventDisable	Disables the module level events.
	PLIB_CAN_ModuleEventEnable	Enables the module level events.
	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.
	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
	PLIB_CAN_ChannelEventEnable	Enables channel level events.
	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.

f) Message Transmit Functions

	Name	Description
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

g) Message Receive Functions

	Name	Description
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.

h) Message Filtering Functions

	Name	Description
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.

i) Error State Tracking Functions

	Name	Description
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.

j) Information Functions

	Name	Description
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.

k) Feature Existence Functions

	Name	Description
	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.
	PLIB_CAN_ExistsFilterConfigure	Identifies whether the FilterConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterEnable	Identifies whether the FilterEnable feature exists on the CAN module.
	PLIB_CAN_ExistsFilterMaskConfigure	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterToChannelLink	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	PLIB_CAN_ExistsLatestFilterMatchGet	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	PLIB_CAN_ExistsMemoryBufferAssign	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventClear	Identifies whether the ModuleEvents feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventEnable	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsModuleInfo	Identifies whether the ModuleInformation feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeRead	Identifies whether the OperationModeRead feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeWrite	Identifies whether the OperationModeSet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingEventsGet	Identifies whether the PendingEventsGet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingTransmissionsAbort	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.

	PLIB_CAN_ExistsReceivedMessageGet	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	PLIB_CAN_ExistsReceiveErrorCount	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module
	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRateGet	Identifies whether the BaudRateGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRatePrescaleSetup	Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.
	PLIB_CAN_ExistsBitSamplePhaseSet	Identifies whether the BitSamplePhaseSet feature exists on the CAN module.
	PLIB_CAN_ExistsPrecalculatedBitRateSetup	Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.

I) Data Types and Constants

	Name	Description
	CAN_CHANNEL	Identifies the supported CAN Channels.
	CAN_CHANNEL_EVENT	Identifies all Layer 3 interrupts.
	CAN_CHANNEL_MASK	Lists the series of useful masks.
	CAN_DNET_FILTER_SIZE	Specifies the size of the DeviceNet filter.
	CAN_ERROR_STATE	Specifies the CAN module error states.
	CAN_FILTER	CAN event code returned by the CAN module.
	CAN_FILTER_MASK	Identifies the available CAN filter masks.
	CAN_FILTER_MASK_TYPE	Specifies the CAN filter mask type.
	CAN_ID_TYPE	Specifies the CAN ID type.
	CAN_MODULE_EVENT	Specifies the CAN module events
	CAN_MODULE_ID	Enumeration: CAN_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	CAN_MSG_EID	Defines the structure of the EID word section of the transmit and receive message.
	CAN_OPERATION_MODES	Lists all possible CAN module operational modes.
	CAN_RECEIVE_CHANNEL	Lists all possible CAN module receive channels.
	CAN_RECEIVE_MODES	Lists all possible CAN module receive modes.
	CAN_RX_DATA_MODE	Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.
	CAN_RX_MSG_BUFFER	Defines the structure of the CAN receive message buffer
	CAN_RX_MSG_SID	Defines the structure of the SID word section of the receive message.
	CAN_TIME_SEGMENT_LENGTH	All possible values for the assignable number of Time Quanta.
	CAN_TX_CHANNEL_STATUS	Identifies possible transmit channel-specific conditions.
	CAN_TX_MSG_BUFFER	Defines the structure of the CAN transmit message buffer.
	CAN_TX_MSG_SID	Defines the structure of the SID word section of the transmit message.
	CAN_TX_RTR	Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.
	CAN_TXCHANNEL_PRIORITY	Specifies the priority of a transmit channel.
	CAN_RX_DATA_ONLY_SIZE_BYTES	Used as the size of the CAN data-only receive message buffer in bytes.
	CAN_TX_RX_MESSAGE_SIZE_BYTES	Used as the array size of the CAN transmit and full receive message buffer.

Description

This section describes the Application Programming Interface (API) functions of the CAN Peripheral Library.

Refer to each section for a detailed description.

a) Basic Configuration Functions

PLIB_CAN_BusActivityWakeupDisable Function

Disables the wake-up on bus activity receive line filter.

File

plib_can.h

C

```
void PLIB_CAN_BusActivityWakeupDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the Wake up on bus activity receive line filter.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusActivityWakeup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusActivityWakeupDisable (CAN_MODULE_ID index)

PLIB_CAN_BusActivityWakeupEnable Function

Enables the wake-up on bus activity receive line filter.

File

plib_can.h

C

```
void PLIB_CAN_BusActivityWakeupEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the wake-up on bus activity receive line filter.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusActivityWakeup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusActivityWakeupEnable (CAN_MODULE_ID index)

PLIB_CAN_Disable Function

Disables the specified CAN module.

File

plib_can.h

C

```
void PLIB_CAN_Disable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the specified CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

If the user application clears this bit, it may take a number of cycles before the CAN module completes the current transaction and responds to this request. The user application should check this using the PLIB_CAN_IsActive function to verify that the request has been honored.

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_Disable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_Disable (CAN_MODULE_ID index)

PLIB_CAN_Enable Function

Enables the specified CAN module.

File

plib_can.h

C

```
void PLIB_CAN_Enable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the specified CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

Some of the CAN module settings can be done only when the module is off. Therefore, always call the PLIB_CAN_Enable Function after all other configurations are complete.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_CAN_ID CAN_ID_1

//Do all the other configurations before enabling.

PLIB_CAN_Enable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_Enable (CAN_MODULE_ID index)

PLIB_CAN_IsActive Function

Returns 'true' if the CAN module is active.

File

plib_can.h

C

```
bool PLIB_CAN_IsActive(CAN_MODULE_ID index);
```

Returns

- true = The module is still active
- false = The module is completely disabled

Description

This function returns 'true' if the CAN module is active. This function is typically called after the CAN module is disabled using the PLIB_CAN_Disable function. The CAN module disable request does not complete immediately when requested and depends on the CAN bus status. This function should be called to check whether the module disable is completed.

Remarks

The CAN module disable operation should not be confused with placing the CAN module in the Disable mode using the PLIB_CAN_OperationModeSelect function. The Disable mode is one of the operating modes of the CAN module. The CAN module is still enabled while in Disable mode. The module disable operation switches the CAN module off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsActiveStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CAN module 1. Wait until the module is completely disabled.

PLIB_CAN_Disable(CAN_ID_1);

while(PLIB_CAN_IsActive(CAN_ID_1) == true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

bool PLIB_CAN_IsActive (CAN_MODULE_ID index)

PLIB_CAN_OperationModeGet Function

Obtains the current CAN operating mode.

File

plib_can.h

C

```
CAN_OPERATION_MODES PLIB_CAN_OperationModeGet(CAN_MODULE_ID index);
```

Returns

The current CAN_OP_MODE type of operation mode of the CAN module.

Description

This function obtains the current CAN operating mode. This function is typically called after the PLIB_CAN_OperationModeSelect function to check if the requested operation mode change is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsOperationModeRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check and wait until the CAN module is in Disable Mode.

while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_DISABLE_MODE);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_OPERATION_MODES PLIB_CAN_OperationModeGet (CAN_MODULE_ID index)

PLIB_CAN_OperationModeSelect Function

Sets the CAN operating mode.

File

plib_can.h

C

```
void PLIB_CAN_OperationModeSelect(CAN_MODULE_ID index, CAN_OPERATION_MODES opMode);
```

Returns

None.

Description

This function sets the CAN operating mode. The CAN module requires itself to be in certain modes to gain access to module functionality. For example, the module should be in Normal mode to send and receive messages. Note that after this function is called, it should be checked to determine whether the mode was set by using the PLIB_CAN_OperationModeGet function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsOperationModeWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set the CAN_ID_1 operating mode to Configuration mode.

PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
```

Parameters

Parameters	Description
module	Identifies the desired CAN module
opMode	Desired CAN_OP_MODE type of the mode to be set

Function

void PLIB_CAN_OperationModeSelect (CAN_MODULE_ID index, CAN_OPERATION_MODES opMode)

PLIB_CAN_StopInIdleDisable Function

Disables the Stop in Idle feature.

File

plib_can.h

C

```
void PLIB_CAN_StopInIdleDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the CAN module from stopping operation when the system enters Idle mode. The module continues operation when the system enters Idle mode.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_StopInIdleDisable (CAN_MODULE_ID index)

PLIB_CAN_StopInIdleEnable Function

Enables the CAN module to stop when the processor enters Idle mode.

File

plib_can.h

C

```
void PLIB_CAN_StopInIdleEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function configures the CAN module to stop operation when the system enters Idle mode.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_StopInIdleEnable (CAN_MODULE_ID index)

PLIB_CAN_TimeStampDisable Function

Disables the time stamp feature for the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables time stamp capture feature for the CAN module. This conserves power by stopping the CAN timer.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_TimeStampDisable (CAN_MODULE_ID index)

PLIB_CAN_TimeStampEnable Function

Enables the time stamp feature for the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the time stamp capture feature for the CAN module. The CAN timer value will be stored on valid message reception and is stored with the message.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampEnable in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_TimeStampEnable (CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampPrescaler Function

Identifies whether the TimeStampPrescaler feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampPrescaler(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampPrescaler feature is supported on the device
- false = The TimeStampPrescaler feature is not supported on the device

Description

This function identifies whether the TimeStampPrescaler feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TimeStampPrescalerSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampPrescaler(CAN_MODULE_ID index)

PLIB_CAN_TimeStampPrescalerSet Function

Sets the CAN receive message time stamp timer prescaler.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampPrescalerSet(CAN_MODULE_ID index, unsigned preScaler);
```

Returns

None.

Description

This function sets the CAN receive message time stamp timer prescaler. This prescaler determines the rate at the which the time stamp timer is incremented. For example, if the prescaler value is 0xFF, the time stamp timer is incremented by 1 every 255 system clock periods.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampPrescaler in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set the CAN_ID_1 Receive Message Time Stamp Timer prescaler to increment
//the Time Stamp Timer every 1024 system clock periods.

PLIB_CAN_TimeStampPrescalerSet(CAN_ID_1, 1024);
```

Parameters

Parameters	Description
module	Identifies the desired CAN module
preScaler	Prescaler for the CAN receive message time stamp timer. This value should be between 0x0 and 0xFFFF.

Function

void PLIB_CAN_TimeStampPrescalerSet(CAN_MODULE_ID index, unsigned preScaler)

PLIB_CAN_MemoryBufferAssign Function

Assigns buffer memory to the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_MemoryBufferAssign(CAN_MODULE_ID index, void * buffer);
```

Returns

None.

Description

This function assigns buffer memory address to the CAN module. The CAN module uses this buffer memory to store messages to be transmitted and received. The size of the memory buffer should be enough to accommodate the required number of message buffers and channels.

Remarks

This API is useful only if the CAN module uses device RAM for message buffers and message FIFO location is software configurable. This should not be used if the device message buffer is part of SFR.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsMemoryBufferAssign in your application to determine whether this feature is available.

Preconditions

The module should be in Configuration mode (using the PLIB_CAN_OperationModeSelect function).

Example

```
// Define and assign a CAN 1 memory buffer for 2 Transmit channels and 1 Receive
// channel, each with 4 message buffers

uint8_t canMemoryBuffer [3 * 4 * CAN_TX_RX_MESSAGE_SIZE_BYTES];

PLIB_CAN_MemoryBufferAssign(CAN_ID_1, canMemoryBuffer);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
buffer	Pointer to the buffer memory area

Function

void PLIB_CAN_MemoryBufferAssign (CAN_MODULE_ID index, void * buffer)

b) Advanced Configuration Functions

PLIB_CAN_ChannelResetIsComplete Function

Returns 'true' if the specified channel reset is complete.

File

plib_can.h

C

```
bool PLIB_CAN_ChannelResetIsComplete(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

- true = Channel reset is complete
- false = Channel reset is in progress

Description

This function returns 'true' if the specified channel reset is complete. This function should preferably be called after calling the PLIB_CAN_ChannelReset function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelReset in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset channel 4 of CAN_ID_1 module
// and wait until the reset is complete
PLIB_CAN_ChannelReset(CAN_ID_1,CAN_CHANNEL4);

while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1,CAN_CHANNEL4) != true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN Channel to be inspected for reset

Function

bool PLIB_CAN_ChannelResetIsComplete (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_DeviceNetConfigure Function

Configures the CAN module DeviceNet(TM) filter feature.

File

plib_can.h

C

```
void PLIB_CAN_DeviceNetConfigure(CAN_MODULE_ID index, CAN_DNET_FILTER_SIZE dncnt);
```

Returns

None.

Description

This function configures the CAN module DeviceNet filter feature. DeviceNet filtering allows a portion of the data payload to be used as a filter criteria. This function allows the size of this filter to be configured in bits. The filter can also be disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsDeviceNet in your application to determine whether this feature is available.

Preconditions

The CAN module should preferably be in Configuration mode (using the PLIB_CAN_OperationModeSelect function).

Example

```
// Disable the CAN_ID_1 DeviceNet filter feature.

PLIB_CAN_DeviceNetConfigure(CAN_ID_1, CAN_DNET_FILTER_DISABLE);
```

```
// Set the CAN_ID_2 Device Net Filter Size to 10 bits.

PLIB_CAN_DeviceNetConfigure(CAN_ID_2, CAN_DNET_FILTER_SIZE_10_BIT);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
dncnt	Specifies the CAN_DNET_FILTER_SIZE size of the DeviceNet filter in bits

Function

void PLIB_CAN_DeviceNetConfigure (CAN_MODULE_ID index, CAN_DNET_FILTER_SIZE dncnt)

PLIB_CAN_TimeStampValueGet Function

Returns the current value of the CAN receive message time stamp timer value.

File

plib_can.h

C

```
unsigned PLIB_CAN_TimeStampValueGet(CAN_MODULE_ID index);
```

Returns

The current value of the CAN receive message time stamp timer.

Description

This function gets the current value of the CAN receive message time stamp timer value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
unsigned int timeStampTimerVal;

// Get the current time stamp timer value.
timeStampTimerVal = PLIB_CAN_TimeStampValueGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

unsigned PLIB_CAN_TimeStampValueGet (CAN_MODULE_ID index)

PLIB_CAN_TimeStampValueSet Function

Sets the CAN receive message time stamp timer value.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampValueSet(CAN_MODULE_ID index, unsigned value);
```

Returns

None.

Description

This function sets the CAN receive message time stamp timer value. The timer will then start counting up from this value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set start value of CAN_ID_1 Receive Message Time Stamp Timer to 0x100
PLIB_CAN_TimeStampValueSet(CAN_ID_1, 0x0100);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
value	Start value of the receive message time stamp timer. This value should be between 0x0 and 0xFFFF.

Function

void PLIB_CAN_TimeStampValueSet (CAN_MODULE_ID index, unsigned value)

c) Bus Speed Configuration Functions

PLIB_CAN_BusLine3TimesSamplingDisable Function

Disables the bus line three times sampling.

File

plib_can.h

C

```
void PLIB_CAN_BusLine3TimesSamplingDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

The bus line will be sampled three times at the sampling point. If this is disabled, the bus line will be sampled only once.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This API may not function if the baud rate prescale value is more than a certain limit. Refer to the specific device data sheet to determine the maximum prescale that allows three times sampling.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusLine3TimesSampling in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
PLIB_CAN_BusLine3TimesSamplingDisable(MY_CAN_ID);

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusLine3TimesSamplingDisable (CAN_MODULE_ID index)

PLIB_CAN_BusLine3TimesSamplingEnable Function

Enables the bus line three times sampling.

File

plib_can.h

C

```
void PLIB_CAN_BusLine3TimesSamplingEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

The bus line will be sampled three times at the sampling point. If this is disabled, the bus line will be sampled only once.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This API may not function if the baud rate prescale value is more than a certain limit. Refer to the specific device data sheet to determine the maximum prescale that allows three times sampling.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusLine3TimesSampling in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
PLIB_CAN_BusLine3TimesSamplingEnable(MY_CAN_ID);

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusLine3TimesSamplingEnable (CAN_MODULE_ID index)

PLIB_CAN_BaudRateGet Function

Returns the current CAN module Baud rate.

File

plib_can.h

C

```
uint32_t PLIB_CAN_BaudRateGet(CAN_MODULE_ID index, uint32_t sysclock);
```

Returns

Current Baud rate value in kbps.

Description

This function returns the current baud rate of the CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBaudRateGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID   CAN_ID_1
#define CAN_CLOCK   80000000

uint32_t baudRate;

baudRate = PLIB_CAN_BaudRateGet(MY_CAN_ID, CAN_CLOCK);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
sysclock	CAN input clock frequency

Function

uint32_t PLIB_CAN_BaudRateGet(CAN_MODULE_ID index, uint32_t clock);

PLIB_CAN_BaudRatePrescaleSetup Function

Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.

File

plib_can.h

C

```
bool PLIB_CAN_BaudRatePrescaleSetup(CAN_MODULE_ID index, uint16_t prescale);
```

Returns

- true = BaudRate prescale Setup success
- false = BaudRate prescale Setup failure, arguments passed are beyond the possible limits

Description

Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.

Remarks

The prescale value is the actual input clock divisor value.

baud rate = input clock / divisor

However, the values must be chosen carefully so that the desired resultant baud rate is an integral number (not fractional).

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBaudRatePrescaleSetup in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID   CAN_ID_1

#define PRESCALE    7
bool result;
if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{

    result = PLIB_CAN_BaudRatePrescaleSetup(MY_CAN_ID, PRESCALE);

    if(false == result)
    {
        // Error occurred
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
prescale	Prescale value by which the input clock to the CAN module is divided to determine the CAN baud rate.

Function

bool PLIB_CAN_BaudRatePrescaleSetup(CAN_MODULE_ID index, uint16_t prescale)

PLIB_CAN_PrecalculatedBitRateSetup Function

Sets the desired Baud rate for the respective CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_PrecalculatedBitRateSetup(CAN_MODULE_ID index, uint8_t prescale, uint8_t syncjumpWidth,
uint8_t propagation, uint8_t segment1, uint8_t segment2);
```

Returns

- true = Baud rate setup success
- false = Baud rate setup failure, arguments passed are beyond the possible limits

Description

This function sets the configuration register with the desired Baud rate.

Remarks

This function's parameters must be precalculated for the desired baud rate and properly encoded for the registers of the CAN module that is in use. This function is primarily intended to be used in conjunction with the MPLAB X IDE ECAN Baud Rate calculator or the MPLAB Harmony Configurator (MHC), which precalculates and correctly encodes these values.

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPrecalculatedBitRateSetup in your application to determine whether this feature is available.

Preconditions

Use the "ECAN Bit Rate Calculator" MPLAB X IDE plug-in to get all of the parameters needed for the function.

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
// Use ECAN Bit Rate Calculator plug-in to get parameter values for 500kbps
// Baud rate.
#define MY_CAN_ID       CAN_ID_1
#define PRESCALE        6
#define SYNCJUMPWIDTH   1
```

```
#define PROPAGATION     2
#define SEGMENT1        4
#define SEGMENT2        4
bool result;

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
   result = PLIB_CAN_PrecalculatedBitRateSetup( MY_CAN_ID, PRESCALE, SYNCJUMPWIDTH,
                                        PROPAGATION, SEGMENT1, SEGMENT2 );

   if(false == result)
   {
       // Error occurred, handle accordingly
   }
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
prescale	The CAN module input clock prescale divisor.
syncJumpWidth	The synchronization jump width is the number of time quanta by which the CAN module may adjust phase segment 1 and segment 2 to compensate for jitter in the bit phase.
propagation	Propagation segment length is the number of time quanta allocated to allow for propagation variation in the bit sampling phase.
segment1	Phase segment 1 length(in time quanta) after the propagation segment before sampling of the bit begins.
segment2	Phase segment 2 length (in time quanta) after sampling has begun.

Function

bool PLIB_CAN_PrecalculatedBitRateSetup(CAN_MODULE_ID index,
uint8_t prescale,
uint8_t syncjumpWidth,
uint8_t propagation,
uint8_t segment1,
uint8_t segment2);

PLIB_CAN_BitSamplePhaseSet Function

Sets the CAN bit-sampling phase parameters.

File

plib_can.h

C

```
bool PLIB_CAN_BitSamplePhaseSet(CAN_MODULE_ID index, uint8_t syncJumpWidth, uint8_t propagation, uint8_t
segment1, uint8_t segment2);
```

Returns

- true = BitSamplePhase Setup success
- false = BitSamplePhase Setup failure, arguments passed are beyond the possible limits

Description

This function selects the CAN bit-sampling phase parameters that determine the Baud rate setting.

For a given baud rate setting, a single bit time is divided into a number segments or phases of equal time length called "time quanta". The bit time for a given baud rate (i.e. 1/baud) is divided into a number of time quanta of equal length.

1 time quantum = (one bit time) / (total number of time quanta)

These time quanta are then partitioned into different phases of the bit sample time called synchronization phase (always 1 time quantum long), the propagation phase, and the segment 1 and segment 2 phases. Thus, the total number of time quanta equals the sum of the time quanta allocated to each phase.

total number of time quanta = (1 + propagation + segment1 + segment2)

(Sampling of the bit occurs at the end of the segment 1 phase.)

This function determines the length of each phase (in time quanta) as well as the synchronization jump width, the number of time quanta by which

the CAN module may move time from segment 1 to segment 2 (or vice versa) to adjust for jitter in the bit sampling phase timing.

Remarks

The values of the syncJumpWidth, propagation, segment1 and segment2 parameters should passed as a number of time quanta for each. These values will be encoded by this function correctly for the the part in use before they are written to the registers of the CAN peripheral. To pass in precalculated values that are already encoded for the physical registers of the CAN controller (such as the values provided by the ECAN Baud Rate Calculator or the MPLAB Harmony Configurator (MHC) utility), use the PLIB_CAN_PrecalculatedBitRateSetup function instead.

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBitSamplePhaseSet in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID       CAN_ID_1

#define SYNCJUMPWIDTH   1
#define PROPAGATION     2
#define SEGMENT1        4
#define SEGMENT2        4
bool result;
if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
   result = PLIB_CAN_BitSamplePhaseSet(MY_CAN_ID, SYNCJUMPWIDTH, PROPAGATION,
                                       SEGMENT1, SEGMENT2);

   if(false == result)
   {
       // Error occurred
   }
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
syncJumpWidth	The synchronization jump width is the number of time quanta by which the CAN module may adjust phase segment 1 and segment 2 to compensate for jitter in the bit phase to achieve a valid sampling point.
propagation	Propagation segment length is the number of time quanta allocated to allow for propagation variation in the bit sampling phase.
segment1	Phase segment 1 length(in time quanta) after the propagation segment before sampling of the bit begins.
segment2	Phase segment 2 length (in time quanta) after sampling has begun.

Function

bool PLIB_CAN_BitSamplePhaseSet(CAN_MODULE_ID index,
uint8_t syncJumpWidth,
uint8_t propagation,
uint8_t segment1,
uint8_t segment2)

d) Channel Configuration Functions

PLIB_CAN_ChannelForReceiveSet Function

Configures a CAN channel for receive operation.

File

plib_can.h

C

```
void PLIB_CAN_ChannelForReceiveSet(CAN_MODULE_ID index, CAN_CHANNEL channel, uint32_t channelSize,
CAN_RX_DATA_MODE dataOnly);
```

Returns

None.

Description

This function configures a CAN channel for receive operation. A receive channel can be either a full CAN message receive channel, which receives an entire CAN message (Arbitration field + Data field) or a data-only message channel, which receives only the data payload section of the message. A receive channel can buffer up to 32 messages. The number of messages to buffer (i.e., the size of the channel) is set by the channelSize parameter.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelForReceiveSet in your application to determine whether this feature is available.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure channel 4 of CAN module for receive operation. Set channel
// size to buffer 10 messages.
// Channel should be data only receive channel

PLIB_CAN_ChannelForReceiveSet(CAN_ID_1, CAN_CHANNEL4, 10, CAN_RX_DATA_ONLY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Receive channel
channelSize	Specifies the number of received messages that the channel can buffer before it overflows. This should be a value between 1 and 32.
dataOnly	Specifies a full CAN message Receive channel or a data-only message channel. • CAN_RX_DATA_ONLY - Channel will be a data-only message receive channel • CAN_RX_FULL_RECEIVE - Channel will be a full CAN message receive channel

Function

void PLIB_CAN_ChannelForReceiveSet(CAN_MODULE_ID index, CAN_CHANNEL channel,
uint32_t channelSize, CAN_RX_DATA_MODE dataOnly)

PLIB_CAN_ChannelForTransmitSet Function

Configures a CAN channel for transmission.

File

plib_can.h

C

```
void PLIB_CAN_ChannelForTransmitSet(CAN_MODULE_ID index, CAN_CHANNEL channel, uint8_t channelSize,
CAN_TX_RTR rtren, CAN_TXCHANNEL_PRIORITY priority);
```

Returns

None.

Description

This function configures a CAN Channel for transmission. The size of the channel specifies the number of messages that the channel can buffer. The channel is a First In First Out (FIFO) type of buffer. The remote transmit request feature allows the transmit channel to start transmitting when an associated filter has received a message. The transmit channel priority determines the priority as compared to other transmit channels. If two transmit channels have the same priority, the natural channel priority determines which channel transmits first.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelForTransmitSet in your application to determine whether this feature is available.

Preconditions

The CAN Module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure CAN 1 Channel 2 for Transmit operation. Set the channel size to 15
// and enable remote transmit request. Set the priority to low medium
// priority.

PLIB_CAN_ChannelForTransmitSet (CAN_ID_1, CAN_CHANNEL2, 15,
    CAN_TX_RTR_ENABLED, CAN_LOW_MEDIUM_PRIORITY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
channelSize	Size of the channel in messages. This value should be between 1 and 32.
rtren	Enables disables Remote Transmit Request: • CAN_TX_RTR_ENABLED - Remote Transmit Request is enabled • CAN_TX_RTR_DISABLED - Remote Transmit Request is disabled
priority	Specifies the priority of the Transmit channel

Function

void PLIB_CAN_ChannelForTransmitSet (CAN_MODULE_ID index,
CAN_CHANNEL channel, uint8_t channelSize,
CAN_TX_RTR rtren, CAN_TXCHANNEL_PRIORITY priority)

PLIB_CAN_ChannelReset Function

Resets a CAN channel.

File

plib_can.h

C

```
void PLIB_CAN_ChannelReset(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function resets a CAN channel. Resetting a CAN channel causes it to discard any unread received messages or any messages that have not been transmitted yet. The reset operation will wait if a message is being currently transmitted or is being received. The PLIB_CAN_ChannelResetIsComplete function can be used to check if the channel reset is complete.

Remarks

Attempting to read from a channel that is reset will return messages that may have already been read or may not have been read at all.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelReset in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset channel 4 of CAN_ID_1 module
// and wait until the reset is complete

PLIB_CAN_ChannelReset(CAN_ID_1,CAN_CHANNEL4);
```

```
while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1,CAN_CHANNEL4) != true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN channel to be reset

Function

void PLIB_CAN_ChannelReset(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_ChannelUpdate Function

Updates the CAN Channel internal pointers.

File

plib_can.h

C

```
void PLIB_CAN_ChannelUpdate(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function updates the CAN Channel internal pointers. This function should be called when a message has been read or processed completely from a CAN receive channel (using the PLIB_CAN_ReceivedMessageGet function). It should be called after a new message has been written to a CAN transmit channel (using the PLIB_CAN_TransmitBufferGet function) and before the PLIB_CAN_TransmitChannelFlush function.

Trying to read a CAN receive channel that has not been updated will cause the PLIB_CAN_ReceivedMessageGet function to return the same message. Writing to a CAN transmit channel that has not been updated will cause the last written message to be overwritten.

Remarks

The PLIB_CAN_ChannelUpdate function should be called on a CAN receive channel only after the message obtained using the PLIB_CAN_ReceivedMessageGet function has been read or processed completely. The CAN module peripheral library does not provide access to older messages once the PLIB_CAN_ChannelUpdate function has been called.

When using the PLIB_CAN_TransmitMessageBuffer function to write to a CAN transmit channel the PLIB_CAN_ChannelUpdate function should be called only when a valid message has been written to the channel. Calling the update function without writing to the CAN channel will cause an incorrect message to be output on the CAN channel when the transmit channel is flushed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelUpdate in your application to determine whether this feature is available.

Preconditions

This function is effective only when the CAN module is not in Configuration mode or Disable mode.

Example

```
// CAN_ID_1 Channel 1 is a Receive channel and Channel 2 is a Transmit
// channel. Read and update Channel 1. Write a message to Channel 2 and then
// update and flush the channel.

CAN_TX_MSG_BUFFER * TransmitMessage;
CAN_RX_MSG_BUFFER * rxMessage;

rxMessage = PLIB_CAN_ReceivedMessageGet(CAN_ID_1, CAN_CHANNEL1);

if(rxMessage != NULL)
{
    // Process the received message.

    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);
}

TransmitMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL2);
if(TransmitMessage != NULL)
{
```

```
    // Write a message to buffer

    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL2);
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL2);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN channel to be updated

Function

void PLIB_CAN_ChannelUpdate (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_ChannelEventDisable Function

Enables channel level events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventDisable(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT flags);
```

Returns

None.

Description

This function disables channel level events. Any enabled channel event will cause a CAN module event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CAN_ChannelEventDisable(CAN_ID_1, CAN_CHANNEL1, CAN_TX_CHANNEL_EMPTY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ChannelEventDisable (CAN_MODULE_ID index, CAN_CHANNEL channel,
CAN_CHANNEL_EVENT flags)

e) Event Management Functions

PLIB_CAN_ModuleEventClear Function

Clears the CAN module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventClear(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function clears module level events. If the event condition is persistent, clearing the event will have no effect. The event condition itself should be cleared. The CAN_SYSTEM_ERROR_EVENT can only be cleared by disabling the CAN module using the PLIB_CAN_Enable function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clear CAN_ID_1 Transmit Event and Receive Events.

PLIB_CAN_ModuleEventClear(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be cleared by specifying a logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventClear (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventDisable Function

Disables the module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventDisable(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function disables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.

PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |
```

```
    CAN_RX_OVERFLOW_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventDisable (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventEnable Function

Enables the module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventEnable(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function enables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.

PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |
    CAN_RX_OVERFLOW_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventEnable (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventGet Function

Returns the status of the CAN module events.

File

plib_can.h

C

```
CAN_MODULE_EVENT PLIB_CAN_ModuleEventGet(CAN_MODULE_ID index);
```

Returns

A status word representing the status of the CAN module events.

Description

This function returns the status of the CAN module events. The routine returns a status word. This word should be logically ANDed with the desired CAN_MODULE_EVENT event mask. A non-zero result of such an operation would mean that the events specified in the event mask are active. An event mask can contain one event or can be a logical OR combination of multiple events.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if the CAN_ID_1 Module Receive event
// or if Transmit event is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.

}

// Check if the CAN_ID_2 System Error Event
// is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_2) & CAN_SYSTEM_ERROR_EVENT) != 0)
{
    // CAN_ID_2 System error event is active.

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_MODULE_EVENT PLIB_CAN_ModuleEventGet(CAN_MODULE_ID index)

PLIB_CAN_AllChannelEventsGet Function

Returns a value representing the event status of all CAN channels.

File

plib_can.h

C

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelEventsGet(CAN_MODULE_ID index);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_MASK mask value to check if any event on a channel is active.

Description

This function returns a value representing the event status of all of the CAN channels. The return value can be logically ANDed with a CAN_CHANNEL_MASK type to check whether the channel has any active events. Only an enabled channel event will cause the bit to be updated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsAllChannelEvents in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Channel 2 or 3 of CAN_ID_1 module
// have any active events

CAN_CHANNEL_MASK channelEvents;

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);

if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Channel 31 of CAN_ID_2 module
// has an any active events

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_2);

if((channelEvents & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has an active event.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_CHANNEL_MASK PLIB_CAN_AllChannelEventsGet(CAN_MODULE_ID index)

PLIB_CAN_AllChannelOverflowStatusGet Function

Returns a value representing the receive overflow event status of all CAN channels.

File

plib_can.h

C

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelOverflowStatusGet(CAN_MODULE_ID index);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_MASK mask value to check if a receive channel overflow event is active.

Description

This function returns a value representing the Receive overflow event status of all the CAN Channels. The return value can be logically ANDed with a CAN_CHANNEL_MASK type to check whether a channel has any active receive overflow events. The return value will reflect the channel status only if the receive overflow event of the channel is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsAllChannelOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Receive Channel 2 or 3 of CAN_ID_1 module
// have overflowed
```

```
CAN_CHANNEL_MASK channelOverflowEvent;

channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_1);

if((channelOverflowEvent & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Receive Channel 2 or 3 has  overflowed.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Receive Channel 31 of CAN_ID_2 module
// has overflowed

channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_2);

if((channelOverflowEvent & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has overflowed.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_CHANNEL_MASK PLIB_CAN_AllChannelOverflowStatusGet (CAN_MODULE_ID index)

PLIB_CAN_ChannelEventClear Function

Clears CAN channel events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventClear(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT events);
```

Returns

None.

Description

This function clears channel events. The events to be cleared are specified as mask. Note that only the receive overflow event is clearable. Attempting to clear other events will have no effect since these events are persistent and clear only when the event condition is cleared.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEvent in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clear CAN_ID_2 Receive Channel 3 overflow event

PLIB_CAN_ChannelEventClear(CAN_ID_2, CAN_CHANNEL3, CAN_RX_CHANNEL_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
events	Mask specifying the events to be cleared

Function

void PLIB_CAN_ChannelEventClear (CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT events)

PLIB_CAN_ChannelEventEnable Function

Enables channel level events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventEnable(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT flags);
```

Returns

None.

Description

This function enables channel level events. Any enabled channel event will cause a CAN module event. An event can be active regardless of it being enabled or disabled. Enabling a transmit type of event for a receive channel will have no any effect.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// CAN_ID_1 Channel 1 and 3 are Transmit channels and Channel 2  and 4 are
// Receive channels. Enable Channel 1 empty event and channel not full
// event. Disable Channel 2 full and overflow event.
// Enable all Transmit events on Channel 2 and disable all Receive events on
// on Channel 4.

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1, (CAN_TX_CHANNEL_EMPTY |
    CAN_TX_CHANNEL_NOT_FULL));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2, (CAN_RX_CHANNEL_FULL |
    CAN_RX_CHANNEL_OVERFLOW));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ChannelEventEnable (CAN_MODULE_ID index, CAN_CHANNEL channel,
CAN_CHANNEL_EVENT flags)

PLIB_CAN_ChannelEventGet Function

Returns a value representing the event status of a CAN channel.

File

plib_can.h

C

```
CAN_CHANNEL_EVENT PLIB_CAN_ChannelEventGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_EVENT type to check if specific CAN channel events are active.

Description

This function returns a value representing the event status of a CAN channel. The return value can be logically ANDed with CAN_CHANNEL_EVENT type to check for a specific event(s). Channels events are affected regardless of whether the event itself is enabled or disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEvent in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Receive Channel 2 of CAN_ID_1 module
// is not empty or if its full.

CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1,CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either Receive Channel 2 is not empty
    // or the Channel is full.


}

// Check if Transmit Channel 3 of CAN_ID_2 module
// has any active events

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL3);

if((channelEvent & CAN_TX_CHANNEL_ANY_EVENT) != 0)
{
    // This means that some event is active

}

// Check if Transmit Channel 6 of CAN_ID_2 module is not full

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL6);

if((channelEvent & CAN_TX_CHANNEL_NOT_FULL) != 0)
{
    // This means the Channel is not full.

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Function

CAN_CHANNEL_EVENT PLIB_CAN_ChannelEventGet (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_PendingEventsGet Function

Returns a value representing the highest priority active event in the module.

File

plib_can.h

C

```
CAN_EVENT_CODE PLIB_CAN_PendingEventsGet(CAN_MODULE_ID index);
```

Returns

Returns a CAN_EVENT_CODE type representing the highest priority active event in the module.

Description

This function returns a value representing the highest priority active event in the module. The return value is a CAN_EVENT_CODE type.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPendingEventsGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Implement a switch to check and process
// any active CAN module events.

CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
    {
        case CAN_NO_EVENT:
            // Procedure to handle no CAN events
            break;

        case CAN_WAKEUP_EVENT:
            // Procedure to handle device wake-up
            // on CAN bus activity event
            break;

        default:
            break;
    }
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_EVENT_CODE PLIB_CAN_PendingEventsGet (CAN_MODULE_ID index)

f) Message Transmit Functions

PLIB_CAN_PendingTransmissionsAbort Function

Aborts any pending transmit operations.

File

plib_can.h

C

```
void PLIB_CAN_PendingTransmissionsAbort(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function aborts any pending transmit operations. Any messages that are yet to be transmitted will not be transmitted. The messages will still be present in the respective channel. Any message that is in the process of being transmitted will be transmitted completely. Either one channel or all channels can be specified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPendingTransmissionsAbort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and
// Channel 5.
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL5);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS, transmission on all transmit channels will be aborted.

Function

void PLIB_CAN_PendingTransmissionsAbort (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmissionIsAborted Function

Returns 'true' if the transmit abort operation is complete.

File

plib_can.h

C

```
bool PLIB_CAN_TransmissionIsAborted(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

- true - If channel = CAN_ALL_CHANNELS, Transmit Abort is complete. If channel = CAN_CHANNELx, Transmit Abort was successful.
- false - If channel = CAN_ALL_CHANNELS, Transmit Abort is in progress. If channel = CAN_CHANNELx, Transmit Abort was not successful.

Description

This function returns 'true' if the transmit abort operation is complete. Either individual channels can be specified or all channels can be specified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmissionIsAborted in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and
// check if the current message transmission was aborted.

PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);
```

```
if(PLIB_CAN_TransmissionIsAborted(CAN_ID_1, CAN_CHANNEL4) == false)
{
    // The message was not aborted.
}
else
{
    // The message was aborted.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS the status of transmit abort on all transmit channels will be returned.

Function

bool PLIB_CAN_TransmissionIsAborted (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitBufferGet Function

Returns a pointer to an empty transmit buffer.

File

plib_can.h

C

```
CAN_TX_MSG_BUFFER * PLIB_CAN_TransmitBufferGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a CAN_TX_MSG_BUFFER type pointer to an empty message buffer in the Transmit channel. Returns NULL if the channel is full and there aren't any empty message buffers.

Description

This function returns a pointer to an empty transmit buffer. The routine will return a NULL pointer if there aren't any empty transmit buffers. In such a case, the application should flush the channel and wait until the transmit channel has at least one empty buffer. In order to function correctly, it is essential that the PLIB_CAN_ChannelUpdate function is called in the proper sequence for the PLIB_CAN_TransmitBufferGet function to return a pointer to an empty buffer.

Remarks

Calling the PLIB_CAN_TransmitBufferGet function on a channel that has not been updated after a message was written to the channel, will cause the function to return a pointer to the written message in case of transmit buffer FIFO. Therefore, a non-transmitted message could get overwritten.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitBufferGet in your application to determine whether this feature is available.

Preconditions

PLIB_CAN_MemoryBufferAssign must be called if the 'transmit buffer' should be in the device RAM. It is not required if the 'transmit buffer' is in SFR space.

Example

```
// Transmit a message through CAN_ID_1 Channel 4

CAN_TX_MSG_BUFFER * msgBuffer;

msgBuffer = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL4);

if(msgBuffer != NULL)
{
    // Load the message here

}
else
{
    // No space in the channel
    // wait until a message
```

```
    // buffer is free.

    while((PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL4) &
        CAN_TX_CHANNEL_NOT_FULL) == 0);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Function

CAN_TX_MSG_BUFFER * PLIB_CAN_TransmitBufferGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitChannelFlush Function

Causes all messages in the specified transmit channel to be transmitted.

File

plib_can.h

C

```
void PLIB_CAN_TransmitChannelFlush(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function causes all messages in the specified transmit channel to be transmitted. All messages in the channel at the time of the flush operation will be transmitted. The transmit channel flush operation should preferably be called immediately after the PLIB_CAN_ChannelUpdate function. This will ensure that messages are transmitted promptly.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitChannelFlush in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Flush CAN_ID_1 Transmit Channel 4.

PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL4);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Function

void PLIB_CAN_TransmitChannelFlush (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitChannelStatusGet Function

Returns the condition of the transmit channel.

File

plib_can.h

C

```
CAN_TX_CHANNEL_STATUS PLIB_CAN_TransmitChannelStatusGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a status word that can be logically ANDed with the CAN_TX_CHANNEL_STATUS type to check whether a condition exists.

Description

This function returns the condition of the transmit channel. The return value can be logically ANDed with CAN_TX_CHANNEL_STATUS type values.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitChannelStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 Transmit Channel 4
// is still transmitting

CAN_TX_CHANNEL_STATUS status;

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_1, CAN_CHANNEL4);

if((status & CAN_TX_CHANNEL_TRANSMITTING) != 0)
{
    // This means that channel is still
    // transmitting.

}

// Check if CAN_ID_2 Transmit Channel 5 has lost arbitration
// or other Transmit errors.

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_2, CAN_CHANNEL5);

if((status & (CAN_TX_CHANNEL_ARBITRATION_LOST | CAN_TX_CHANNEL_ERROR)) != 0)
{
    // This means that the Transmit channel has either
    // lost arbitration or a Transmit error has occurred.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Function

CAN_TX_CHANNEL_STATUS PLIB_CAN_TransmitChannelStatusGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitErrorCountGet Function

Returns the CAN transmit error count

File

plib_can.h

C

```
int PLIB_CAN_TransmitErrorCountGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN transmit error count.

Description

This function returns the CAN transmit error count. Refer to CAN 2.0B specification for more details on the CAN transmit error count and its

significance.

Remarks

There are multiple bus conditions, which could cause the transmit error count to increase. Please refer to the CAN 2.0B specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitErrorCountGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if the CAN_ID_1 Transmit error count is more than 200
int errorCount;

errorCount = PLIB_CAN_TransmitErrorCountGet(CAN_ID_1);
if(errorCount > 200)
{
    // This error count is high.
    // Do some diagnostics.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

int PLIB_CAN_TransmitErrorCountGet(CAN_MODULE_ID index)

g) Message Receive Functions

PLIB_CAN_ReceivedMessageGet Function

Returns a pointer to a message to be read from the CAN channel.

File

plib_can.h

C

```
CAN_RX_MSG_BUFFER * PLIB_CAN_ReceivedMessageGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a pointer to a message to be read from the CAN channel; returns a CAN_RX_MSG_BUFFER type pointer to a received CAN message. If the receive channel is a full CAN message receive channel, the caller should use the msgSID, msgEID and data members of the CAN_RX_MSG_BUFFER data structure to access the received CAN message. If the receive channel is a data-only channel, the message will only contain 8 payload data bytes (even if the message was placed on the bus with less than 8 bytes). The caller in this case should use the dataOnlyMsgData member of the CAN_RX_MSG_BUFFER data structure to read the data contained in the received CAN message.

Description

This function returns a CAN_RX_MSG_BUFFER pointer to a message to be read from the CAN channel. The PLIB_CAN_ChannelUpdate routine should be called after the received message has been processed.

Remarks

The CAN receive channel is configured as a full CAN message receive channel or a data-Only channel while configuring the channel using the PLIB_CAN_ChannelForReceiveSet function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsReceivedMessageGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read a message from the CAN_ID_1 Channel 8
```

```
// which is configured as full CAN message
// receive channel.

CAN_RX_MSG_BUFFER * receivedMsg;

receivedMsg = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL8);

if(receivedMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
    // the message and then update
    // the CAN channel.


    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL8);
}

// Read a message from the CAN_ID_2 Channel 9
// which is configured as data only message
// receive channel. Access the message
// as bytes;

CAN_RX_MSG_BUFFER * rxMsg;

rxMsg = PLIB_CAN_ReceivedMessageGet(CAN_ID_2, CAN_CHANNEL9);

if(rxMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
    // the message and then update
    // the CAN channel.

    // rxMsg->dataOnlyMsgData[0] is the first byte of message
    // data payload. rxMsg->dataOnlyMsgData[0] is the second
    // byte of message data payload and so on.


    PLIB_CAN_ChannelUpdate(CAN_ID_2, CAN_CHANNEL9);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module.
channel	Identifies the desired CAN Receive channel.

Function

CAN_RX_MSG_BUFFER * PLIB_CAN_ReceivedMessageGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

h) Message Filtering Functions

PLIB_CAN_FilterConfigure Function

Configures a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterConfigure(CAN_MODULE_ID index, CAN_FILTER filter, uint32_t id, CAN_ID_TYPE filterType);
```

Returns

None.

Description

This function configures a CAN message acceptance filter. The ID field of the incoming message must match the filter bits for the CAN module to accept the message. A filter can be a EID type filter, which filters EID messages or a SID filter, which in turn filters SID messages. The filter mask bits (configured using the PLIB_CAN_FilterMaskConfigure function) additionally allow specified message ID bits to be ignored in the filtering process.

Remarks

A CAN message acceptance filter can be configured in Normal operation mode. The filter must be disabled before this is done.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterConfigure in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Configure CAN_ID_2 filter 4 to accept standard ID messages
// with SID 0x100

PLIB_CAN_FilterConfigure(CAN_ID_2, CAN_FILTER4, 0x100, CAN_SID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN receive filter
id	A value in the range 0x0 to 0x7FF for SID filter type or 0x0 to 0x1FFFFFFF for EID filter type.
filterType	Specifies the type of filter • CAN_EID - Filter is an extended ID message filter • CAN_SID - Filter is an standard ID message filter

Function

void PLIB_CAN_FilterConfigure (CAN_MODULE_ID index, CAN_FILTER filter,
uint32_t id, CAN_ID_TYPE filterType)

PLIB_CAN_FilterDisable Function

Disables a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterDisable(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

None.

Description

This function disables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the PLIB_CAN_FilterIsDisabled function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable filter 6 of CAN_ID_2

PLIB_CAN_FilterDisable (CAN_ID_2, CAN_FILTER6);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Message Acceptance Filter

Function

void PLIB_CAN_FilterDisable(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterEnable Function

Enables a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterEnable(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

None.

Description

This function enables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the PLIB_CAN_FilterIsDisabled function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable filter 6 of CAN_ID_2

PLIB_CAN_FilterEnable (CAN_ID_2, CAN_FILTER6);


while(!PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN message acceptance filter

Function

void PLIB_CAN_FilterEnable(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterIsDisabled Function

Returns 'true' if the specified filter is disabled.

File

plib_can.h

C

```
bool PLIB_CAN_FilterIsDisabled(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

- true = The filter is disabled
- false = The filter is enabled

Description

Returns 'true' if the specified filter is disabled. This function should be called to check if the filter is disabled before calling the PLIB_CAN_FilterConfigure function and PLIB_CAN_FilterToChannelLink function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CAN_ID_1 filter 3 and wait until the filter is disabled.
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN filter

Function

bool PLIB_CAN_FilterIsDisabled(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterMaskConfigure Function

Configures a CAN filter mask.

File

plib_can.h

C

```
void PLIB_CAN_FilterMaskConfigure(CAN_MODULE_ID index, CAN_FILTER_MASK mask, uint32_t maskBits, CAN_ID_TYPE
idType, CAN_FILTER_MASK_TYPE mide);
```

Returns

None.

Description

This function configures a CAN filter mask. The mask bits determine which message ID bits are ignored and compared during the filtering process.

Remarks

A mask bit value of zero essentially means that all messages with any ID are accepted. The mode and idType input parameters are still relevant in such a case.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterMaskConfigure in your application to determine whether this feature is available.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure CAN_ID_1 Filter Mask 2 to accept
// extended ID messages in the range 0x4F1DE8 - 0x4F1DEC.
// On analyzing this address range it can be seen that only
// the last two bits of the incoming CAN message should ignored.
// Therefore the mask value should 0x1FFFFFFC.
// This mask will be used with a extended ID filter.
// Set the masking option to filter IDE type.

PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK2, 0x1FFFFFFC,
                             CAN_EID, CAN_FILTER_MASK_IDE_TYPE);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
mask	Identifies the desired CAN receive filter mask.
maskBits	A value in the range 0x0 to 0x7FF for SID range or 0x0 to 0x1FFFFFFF for the EID range. Each set bit will mean that the corresponding bit in the filter will be compared to the corresponding bit in the message ID. A clear mask bit means the corresponding bit in the incoming message ID field will be ignored.
idType	Specifies the value range of maskBits parameter. • CAN_EID - Value range of maskBits parameter is 0x0 (ignore all 29 bits of the incoming message ID) to 0x1FFFFFFF (compare all 29 bits of the incoming message ID). • CAN_SID - Value range of maskBits parameter is 0x0 (ignore all 11 bits of the incoming message ID) to 0x7FF (compare all 11 bits of the incoming message ID).
mide	Specifies ID masking options • CAN_FILTER_MASK_IDE_TYPE - Masking will be performed by filter type only. If the filter is set up for SID messages, the mask will decline all incoming EID messages. If the filter is set up for EID messages, the mask will decline all incoming SID messages • CAN_FILTER_MASK_ANY_TYPE - Masking will be performed regardless of the incoming message ID type. The message will be accepted on a Filter and Message SID match or a filter and message SID/EID match.

Function

void PLIB_CAN_FilterMaskConfigure(CAN_MODULE_ID index, CAN_FILTER_MASK mask,
uint32_t maskBits, CAN_ID_TYPE idType, CAN_FILTER_MASK_TYPE mide)

PLIB_CAN_FilterToChannelLink Function

Links a filter to a channel.

File

plib_can.h

C

```
void PLIB_CAN_FilterToChannelLink(CAN_MODULE_ID index, CAN_FILTER filter, CAN_FILTER_MASK mask, CAN_CHANNEL
channel);
```

Returns

None.

Description

This function links a filter to a channel. A filter is typically linked to a receive channel. This allows the channel to receive messages accepted by the filter. A filter can also be linked to a transmit channel if the transmit channel is configured for remote request transmit. In this case, a message accepted by the filter will automatically cause the linked transmit channel to transmit CAN messages that are buffered in the channel. Note that a filter should be enabled using the PLIB_CAN_FilterEnable function after the filter has been linked to the desired channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterToChannelLink in your application to determine whether this feature is available.

Preconditions

Filter should have been disabled using the PLIB_CAN_FilterDisable function.

Example

```
// Configure CAN_ID_1 filter 3 to accept extended ID messages
// with EID 0x1D400 and link the filter to CAN_ID_1 Channel 5.
// Note the sequence in which the steps are performed.
// Disable the filter and check if its disabled.
// Configure the filter. Link it to the Channel and then
// enable the filter.

PLIB_CAN_FilterDisable(CAN_ID_1, CAN_FILTER3);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER3, 0x1D400, CAN_EID);

PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER3, CAN_FILTER_MASK0,
    CAN_CHANNEL5);

PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Filter
mask	Identifies the mask to be used with this filter
channel	Identifies the channel to be linked to this filter. If a transmit channel is linked, the transmit channel should have its remote transmit request feature enabled.

Function

void PLIB_CAN_FilterToChannelLink(CAN_MODULE_ID index, CAN_FILTER filter,
CAN_FILTER_MASK mask, CAN_CHANNEL channel)

PLIB_CAN_LatestFilterMatchGet Function

Returns the index of the filter that accepted the latest message.

File

plib_can.h

C

```
CAN_FILTER PLIB_CAN_LatestFilterMatchGet(CAN_MODULE_ID index);
```

Returns

Index of the filter that accepted the latest message

Description

This function returns the index of the filter that accepted the latest message.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsLatestFilterMatchGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_2 module Receive Buffer event
// is active and if so check which filter
// accepted the message.

CAN_FILTER filter;

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & CAN_RX_EVENT) != 0)
{
```

```
    // Check which filter accepted the message

    filter = PLIB_CAN_LatestFilterMatchGet(CAN_ID_1);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_FILTER PLIB_CAN_LatestFilterMatchGet(CAN_MODULE_ID index)

i) Error State Tracking Functions

PLIB_CAN_ReceiveErrorCountGet Function

Returns the CAN receive error count.

File

plib_can.h

C

```
int PLIB_CAN_ReceiveErrorCountGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN receive error count.

Description

This function returns the CAN receive error count. Refer to the CAN 2.0B specification for more details on the CAN receive error count and its significance.

Remarks

There are multiple bus conditions, which could cause the receive error count to increase. Please refer to the CAN 2.0b specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsReceiveErrorCount in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 Receive error count is more than 200.
int errCount;

errCount = PLIB_CAN_ReceiveErrorCountGet(CAN_ID_1);
if(errCount > 200)
{
    // This error count is high.
    // Do some diagnostics.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

int PLIB_CAN_ReceiveErrorCountGet(CAN_MODULE_ID index)

PLIB_CAN_ErrorStateGet Function

Returns the CAN error status word.

File

plib_can.h

C

```
CAN_ERROR_STATE PLIB_CAN_ErrorStateGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN_ERROR_STATE word, which can be logically ANDed with the desired CAN_ERROR_STATE member to check whether the CAN module is in a specific error state.

Description

This function returns the CAN error status word. The return word can be logically ANDed with the desired CAN_ERROR_STATE member to check if the CAN module is in a specific error state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsErrorState in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 is in the Transmit or Receive warning state.
CAN_ERROR_STATE errorState;

errorState = PLIB_CAN_ErrorStateGet(CAN_ID_1);

if((errorState & CAN_TX_RX_WARNING_STATE) != 0)
{
    // CAN_ID_1 is in either  Transmit or Receive warning state.
}
// Check if CAN_ID_2 is in the Receive Bus Passive or Transmit Bus passive
// state.
errorState = PLIB_CAN_ErrorStateGet(CAN_ID_2);

if((errorState & (CAN_TX_BUS_PASSIVE_STATE | CAN_RX_BUS_PASSIVE_STATE)) != 0)
{
    // This means that the CAN module is in Transmit Bus Passive
    // or Receive Bus Passive state.
 }
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_ERROR_STATE PLIB_CAN_ErrorStateGet(CAN_MODULE_ID index)

j) Information Functions

PLIB_CAN_TotalChannelsGet Function

Returns the total number of CAN channels per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalChannelsGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN channels per CAN module.

Description

This function returns the total number of CAN channels per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalChannels;

totalChannels = PLIB_CAN_TotalChannelsGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalChannelsGet(CAN_MODULE_ID index)

PLIB_CAN_TotalFiltersGet Function

Returns the total number of CAN Filters per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalFiltersGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN Filters per CAN module.

Description

This function returns the total number of CAN filters per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalFilters;

totalFilters = PLIB_CAN_TotalFiltersGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalFiltersGet(CAN_MODULE_ID index)

PLIB_CAN_TotalMasksGet Function

Returns the total number of CAN masks per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalMasksGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN Masks per CAN module.

Description

This function returns the total number of CAN masks per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalMasks;

totalMasks = PLIB_CAN_TotalMasksGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalMasksGet(CAN_MODULE_ID index)

k) Feature Existence Functions

PLIB_CAN_ExistsActiveStatus Function

Identifies whether the ActiveStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsActiveStatus(CAN_MODULE_ID index);
```

Returns

- true = The ActiveStatus feature is supported on the device
- false = The ActiveStatus feature is not supported on the device

Description

This function identifies whether the ActiveStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_IsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsActiveStatus(CAN_MODULE_ID index)

PLIB_CAN_ExistsAllChannelEvents Function

Identifies whether the AllChannelEvents feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsAllChannelEvents(CAN_MODULE_ID index);
```

Returns

- true = The AllChannelEvents feature is supported on the device
- false = The AllChannelEvents feature is not supported on the device

Description

This function identifies whether the AllChannelEvents feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_AllChannelEventsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsAllChannelEvents(CAN_MODULE_ID index)

PLIB_CAN_ExistsAllChannelOverflow Function

Identifies whether the AllChannelOverflow feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsAllChannelOverflow(CAN_MODULE_ID index);
```

Returns

- true = The AllChannelOverflow feature is supported on the device
- false = The AllChannelOverflow feature is not supported on the device

Description

This function identifies whether the AllChannelOverflow feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_AllChannelOverflowStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsAllChannelOverflow(CAN_MODULE_ID index)

PLIB_CAN_ExistsBusActivityWakeup Function

Identifies whether the BusActivityWakeup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBusActivityWakeup(CAN_MODULE_ID index);
```

Returns

- true = The BusActivityWakeup feature is supported on the device
- false = The BusActivityWakeup feature is not supported on the device

Description

This function identifies whether the BusActivityWakeup feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_BusActivityWakeupEnable
- PLIB_CAN_BusActivityWakeupDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBusActivityWakeup(CAN_MODULE_ID index)

PLIB_CAN_ExistsBusLine3TimesSampling Function

Identifies whether the BusLine3TimesSampling feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBusLine3TimesSampling(CAN_MODULE_ID index);
```

Returns

- true = The BusLine3TimesSampling feature is supported on the device
- false = The BusLine3TimesSampling feature is not supported on the device

Description

This function identifies whether the BusLine3TimesSampling feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_BusLine3TimesSamplingEnable
- PLIB_CAN_BusLine3TimesSamplingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBusLine3TimesSampling(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelEvent Function

Identifies whether the ChannelEventGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelEvent(CAN_MODULE_ID index);
```

Returns

- true = The ChannelEventGet feature is supported on the device
- false = The ChannelEventGet feature is not supported on the device

Description

This function identifies whether the ChannelEventGet feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelEventGet
- PLIB_CAN_ChannelEventClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelEvent(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelEventEnable Function

Identifies whether the ChannelEventEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelEventEnable(CAN_MODULE_ID index);
```

Returns

- true = The ChannelEventEnable feature is supported on the device

- false = The ChannelEventEnable feature is not supported on the device

Description

This function identifies whether the ChannelEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelEventEnable
- PLIB_CAN_ChannelEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelEventEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelForReceiveSet Function

Identifies whether the ChannelForReceiveSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelForReceiveSet(CAN_MODULE_ID index);
```

Returns

- true = The ChannelForReceiveSet feature is supported on the device
- false = The ChannelForReceiveSet feature is not supported on the device

Description

This function identifies whether the ChannelForReceiveSet feature is available on the CAN module. When this function returns true, this function is are supported on the device:

- PLIB_CAN_ChannelForReceiveSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelForReceiveSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelForTransmitSet Function

Identifies whether the ChannelForTransmitSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelForTransmitSet(CAN_MODULE_ID index);
```

Returns

- true = The ChannelForTransmitSet feature is supported on the device
- false = The ChannelForTransmitSet feature is not supported on the device

Description

This function identifies whether the ChannelForTransmitSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ChannelForTransmitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelForTransmitSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelReset Function

Identifies whether the ChannelReset feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelReset(CAN_MODULE_ID index);
```

Returns

- true = The ChannelReset feature is supported on the device
- false = The ChannelReset feature is not supported on the device

Description

This function identifies whether the ChannelReset feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelReset
- PLIB_CAN_ChannelResetIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelReset(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelUpdate Function

Identifies whether the ChannelUpdate feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelUpdate(CAN_MODULE_ID index);
```

Returns

- true = The ChannelUpdate feature is supported on the device
- false = The ChannelUpdate feature is not supported on the device

Description

This function identifies whether the ChannelUpdate feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ChannelUpdate

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelUpdate(CAN_MODULE_ID index)

PLIB_CAN_ExistsDeviceNet Function

Identifies whether the DeviceNet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsDeviceNet(CAN_MODULE_ID index);
```

Returns

- true = The DeviceNet feature is supported on the device
- false = The DeviceNet feature is not supported on the device

Description

This function identifies whether the DeviceNet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_DeviceNetConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsDeviceNet(CAN_MODULE_ID index)

PLIB_CAN_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsEnableControl(CAN_MODULE_ID index);
```

Returns

- true = The EnableControl feature is supported on the device
- false = The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_Enable
- PLIB_CAN_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsEnableControl(CAN_MODULE_ID index)

PLIB_CAN_ExistsErrorState Function

Identifies whether the ErrorStateGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsErrorState(CAN_MODULE_ID index);
```

Returns

- true = The ErrorStateGet feature is supported on the device
- false = The ErrorStateGet feature is not supported on the device

Description

This function identifies whether the ErrorStateGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ErrorStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsErrorState(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterConfigure Function

Identifies whether the FilterConfigure feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterConfigure(CAN_MODULE_ID index);
```

Returns

- true = The FilterConfigure feature is supported on the device
- false = The FilterConfigure feature is not supported on the device

Description

This function identifies whether the FilterConfigure feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterConfigure(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterEnable Function

Identifies whether the FilterEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterEnable(CAN_MODULE_ID index);
```

Returns

- true = The FilterEnable feature is supported on the device
- false = The FilterEnable feature is not supported on the device

Description

This function identifies whether the FilterEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_FilterEnable
- PLIB_CAN_FilterDisable
- PLIB_CAN_FilterIsDisabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterMaskConfigure Function

Identifies whether the FilterMaskConfigure feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterMaskConfigure(CAN_MODULE_ID index);
```

Returns

- true = The FilterMaskConfigure feature is supported on the device
- false = The FilterMaskConfigure feature is not supported on the device

Description

This function identifies whether the FilterMaskConfigure feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterMaskConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterMaskConfigure(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterToChannelLink Function

Identifies whether the FilterToChannelLink feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterToChannelLink(CAN_MODULE_ID index);
```

Returns

- true = The FilterToChannelLink feature is supported on the device
- false = The FilterToChannelLink feature is not supported on the device

Description

This function identifies whether the FilterToChannelLink feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterToChannelLink

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterToChannelLink(CAN_MODULE_ID index)

PLIB_CAN_ExistsLatestFilterMatchGet Function

Identifies whether the LatestFilterMatchGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsLatestFilterMatchGet(CAN_MODULE_ID index);
```

Returns

- true = The LatestFilterMatchGet feature is supported on the device
- false = The LatestFilterMatchGet feature is not supported on the device

Description

This function identifies whether the LatestFilterMatchGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_LatestFilterMatchGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsLatestFilterMatchGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsMemoryBufferAssign Function

Identifies whether the MemoryBufferAssign feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsMemoryBufferAssign(CAN_MODULE_ID index);
```

Returns

- true = The MemoryBufferAssign feature is supported on the device
- false = The MemoryBufferAssign feature is not supported on the device

Description

This function identifies whether the MemoryBufferAssign feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_MemoryBufferAssign

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsMemoryBufferAssign(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleEventClear Function

Identifies whether the ModuleEvents feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleEventClear(CAN_MODULE_ID index);
```

Returns

- true = The ModuleEvents feature is supported on the device
- false = The ModuleEvents feature is not supported on the device

Description

This function identifies whether the ModuleEvents feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ModuleEventClear
- PLIB_CAN_ModuleEventGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleEventClear(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleEventEnable Function

Identifies whether the ModuleEventEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleEventEnable(CAN_MODULE_ID index);
```

Returns

- true = The ModuleEventEnable feature is supported on the device
- false = The ModuleEventEnable feature is not supported on the device

Description

This function identifies whether the ModuleEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ModuleEventEnable
- PLIB_CAN_ModuleEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleEventEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleInfo Function

Identifies whether the ModuleInformation feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleInfo(CAN_MODULE_ID index);
```

Returns

- true = The ModuleInformation feature is supported on the device
- false = The ModuleInformation feature is not supported on the device

Description

This function identifies whether the ModuleInformation feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TotalChannelsGet
- PLIB_CAN_TotalFiltersGet
- PLIB_CAN_TotalMasksGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleInfo(CAN_MODULE_ID index)

PLIB_CAN_ExistsOperationModeRead Function

Identifies whether the OperationModeRead feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsOperationModeRead(CAN_MODULE_ID index);
```

Returns

- true = The OperationModeRead feature is supported on the device
- false = The OperationModeRead feature is not supported on the device

Description

This function identifies whether the OperationModeRead feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_OperationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsOperationModeRead(CAN_MODULE_ID index)

PLIB_CAN_ExistsOperationModeWrite Function

Identifies whether the OperationModeSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsOperationModeWrite(CAN_MODULE_ID index);
```

Returns

- true = The OperationModeSet feature is supported on the device
- false = The OperationModeSet feature is not supported on the device

Description

This function identifies whether the OperationModeSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_OperationModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsOperationModeWrite(CAN_MODULE_ID index)

PLIB_CAN_ExistsPendingEventsGet Function

Identifies whether the PendingEventsGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPendingEventsGet(CAN_MODULE_ID index);
```

Returns

- true = The PendingEventsGet feature is supported on the device
- false = The PendingEventsGet feature is not supported on the device

Description

This function identifies whether the PendingEventsGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PendingEventsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPendingEventsGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsPendingTransmissionsAbort Function

Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPendingTransmissionsAbort(CAN_MODULE_ID index);
```

Returns

- true = The PendingTransmissionsAbort feature is supported on the device
- false = The PendingTransmissionsAbort feature is not supported on the device

Description

This function identifies whether the PendingTransmissionsAbort feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PendingTransmissionsAbort

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPendingTransmissionsAbort(CAN_MODULE_ID index)

PLIB_CAN_ExistsReceivedMessageGet Function

Identifies whether the ReceivedMessageGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsReceivedMessageGet(CAN_MODULE_ID index);
```

Returns

- true = The ReceivedMessageGet feature is supported on the device
- false = The ReceivedMessageGet feature is not supported on the device

Description

This function identifies whether the ReceivedMessageGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ReceivedMessageGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsReceivedMessageGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsReceiveErrorCount Function

Identifies whether the ReceiveErrorCount feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsReceiveErrorCount(CAN_MODULE_ID index);
```

Returns

- true = The ReceiveErrorCount feature is supported on the device
- false = The ReceiveErrorCount feature is not supported on the device

Description

This function identifies whether the ReceiveErrorCount feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ReceiveErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsReceiveErrorCount(CAN_MODULE_ID index)

PLIB_CAN_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsStopInIdle(CAN_MODULE_ID index);
```

Returns

- true = The StopInIdle feature is supported on the device
- false = The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_StopInIdleEnable
- PLIB_CAN_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsStopInIdle(CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampEnable Function

Identifies whether the TimeStampEnable feature exists on the CAN module

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampEnable(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampEnable feature is supported on the device
- false = The TimeStampEnable feature is not supported on the device

Description

This function identifies whether the TimeStampEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TimeStampEnable
- PLIB_CAN_TimeStampDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampValue Function

Identifies whether the TimeStampValue feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampValue(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampValue feature is supported on the device
- false = The TimeStampValue feature is not supported on the device

Description

This function identifies whether the TimeStampValue feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TimeStampValueSet
- PLIB_CAN_TimeStampValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampValue(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmissionIsAborted Function

Identifies whether the TransmissionAbortStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmissionIsAborted(CAN_MODULE_ID index);
```

Returns

- true = The TransmissionAbortStatus feature is supported on the device
- false = The TransmissionAbortStatus feature is not supported on the device

Description

This function identifies whether the TransmissionAbortStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmissionIsAborted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmissionIsAborted(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitBufferGet Function

Identifies whether the TransmitBufferGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitBufferGet(CAN_MODULE_ID index);
```

Returns

- true = The TransmitBufferGet feature is supported on the device
- false = The TransmitBufferGet feature is not supported on the device

Description

This function identifies whether the TransmitBufferGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitBufferGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitBufferGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitChannelFlush Function

Identifies whether the TransmitChannelFlush feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitChannelFlush(CAN_MODULE_ID index);
```

Returns

- true = The TransmitChannelFlush feature is supported on the device
- false = The TransmitChannelFlush feature is not supported on the device

Description

This function identifies whether the TransmitChannelFlush feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitChannelFlush

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitChannelFlush(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitChannelStatus Function

Identifies whether the TransmitChannelStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitChannelStatus(CAN_MODULE_ID index);
```

Returns

- true = The TransmitChannelStatus feature is supported on the device
- false = The TransmitChannelStatus feature is not supported on the device

Description

This function identifies whether the TransmitChannelStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitChannelStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitChannelStatus(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitErrorCountGet Function

Identifies whether the TransmitErrorCountGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitErrorCountGet(CAN_MODULE_ID index);
```

Returns

- true = The TransmitErrorCountGet feature is supported on the device
- false = The TransmitErrorCountGet feature is not supported on the device

Description

This function identifies whether the TransmitErrorCountGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitErrorCountGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsBaudRateGet Function

Identifies whether the BaudRateGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBaudRateGet(CAN_MODULE_ID index);
```

Returns

- true = The BaudRateGet feature is supported on the device.
- false = The BaudRateGet feature is not supported on the device.

Description

This function identifies whether the BaudRateGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBaudRateGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsBaudRatePrescaleSetup Function

Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBaudRatePrescaleSetup(CAN_MODULE_ID index);
```

Returns

- true = The BaudRatePrescaleSetup feature is supported on the device.
- false = The BaudRatePrescaleSetup feature is not supported on the device.

Description

This function identifies whether the BaudRatePrescaleSetup feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BaudRatePrescaleSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBaudRatePrescaleSetup(CAN_MODULE_ID index)

PLIB_CAN_ExistsBitSamplePhaseSet Function

Identifies whether the BitSamplePhaseSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBitSamplePhaseSet(CAN_MODULE_ID index);
```

Returns

- true = The BitSamplePhaseSet feature is supported on the device.
- false = The BitSamplePhaseSet feature is not supported on the device.

Description

This function identifies whether the BitSamplePhaseSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BitSamplePhaseSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBitSamplePhaseSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsPrecalculatedBitRateSetup Function

Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPrecalculatedBitRateSetup(CAN_MODULE_ID index);
```

Returns

- true = The PrecalculatedBitRateSetup feature is supported on the device.
- false = The PrecalculatedBitRateSetup feature is not supported on the device.

Description

This function identifies whether the PrecalculatedBitRateSetup feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PrecalculatedBitRateSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPrecalculatedBitRateSetup(CAN_MODULE_ID index)

I) Data Types and Constants

CAN_CHANNEL Enumeration

Identifies the supported CAN Channels.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0,
  CAN_CHANNEL1,
  CAN_CHANNEL2,
  CAN_CHANNEL3,
  CAN_CHANNEL4,
  CAN_CHANNEL5,
  CAN_CHANNEL6,
  CAN_CHANNEL7,
  CAN_CHANNEL8,
  CAN_CHANNEL9,
  CAN_CHANNEL10,
```

```
  CAN_CHANNEL11,
  CAN_CHANNEL12,
  CAN_CHANNEL13,
  CAN_CHANNEL14,
  CAN_CHANNEL15,
  CAN_CHANNEL16,
  CAN_CHANNEL17,
  CAN_CHANNEL18,
  CAN_CHANNEL19,
  CAN_CHANNEL20,
  CAN_CHANNEL21,
  CAN_CHANNEL22,
  CAN_CHANNEL23,
  CAN_CHANNEL24,
  CAN_CHANNEL25,
  CAN_CHANNEL26,
  CAN_CHANNEL27,
  CAN_CHANNEL28,
  CAN_CHANNEL29,
  CAN_CHANNEL30,
  CAN_CHANNEL31
} CAN_CHANNEL;
```

Members

Members	Description
CAN_CHANNEL0	Channel 0 ID
CAN_CHANNEL1	Channel 1 ID
CAN_CHANNEL2	Channel 2 ID
CAN_CHANNEL3	Channel 3 ID
CAN_CHANNEL4	Channel 4 ID
CAN_CHANNEL5	Channel 5 ID
CAN_CHANNEL6	Channel 6 ID
CAN_CHANNEL7	Channel 7 ID
CAN_CHANNEL8	Channel 8 ID
CAN_CHANNEL9	Channel 9 ID
CAN_CHANNEL10	Channel 10 ID
CAN_CHANNEL11	Channel 11 ID
CAN_CHANNEL12	Channel 12 ID
CAN_CHANNEL13	Channel 13 ID
CAN_CHANNEL14	Channel 14 ID
CAN_CHANNEL15	Channel 15 ID
CAN_CHANNEL16	Channel 16 ID
CAN_CHANNEL17	Channel 17 ID
CAN_CHANNEL18	Channel 18 ID
CAN_CHANNEL19	Channel 19 ID
CAN_CHANNEL20	Channel 20 ID
CAN_CHANNEL21	Channel 21 ID
CAN_CHANNEL22	Channel 22 ID
CAN_CHANNEL23	Channel 23 ID
CAN_CHANNEL24	Channel 24 ID
CAN_CHANNEL25	Channel 25 ID
CAN_CHANNEL26	Channel 26 ID
CAN_CHANNEL27	Channel 27 ID
CAN_CHANNEL28	Channel 28 ID
CAN_CHANNEL29	Channel 29 ID
CAN_CHANNEL30	Channel 30 ID
CAN_CHANNEL31	Channel 31 ID

Description

CAN Channel

This enumeration identifies the available CAN channels.

CAN_CHANNEL_EVENT Enumeration

Identifies all Layer 3 interrupts.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RX_CHANNEL_NOT_EMPTY,
  CAN_RX_CHANNEL_HALF_FULL,
  CAN_RX_CHANNEL_FULL,
  CAN_RX_CHANNEL_OVERFLOW,
  CAN_RX_CHANNEL_ANY_EVENT,
  CAN_TX_CHANNEL_EMPTY,
  CAN_TX_CHANNEL_HALF_EMPTY,
  CAN_TX_CHANNEL_NOT_FULL,
  CAN_TX_CHANNEL_ANY_EVENT
} CAN_CHANNEL_EVENT;
```

Members

Members	Description
CAN_RX_CHANNEL_NOT_EMPTY	CAN Receive Channel Not Empty Event Mask
CAN_RX_CHANNEL_HALF_FULL	CAN Receive Channel Half Full Event Mask
CAN_RX_CHANNEL_FULL	CAN Receive Channel Full Event Mask
CAN_RX_CHANNEL_OVERFLOW	CAN Receive Channel Overflow Event Mask
CAN_RX_CHANNEL_ANY_EVENT	CAN Receive Channel Any Event Mask
CAN_TX_CHANNEL_EMPTY	CAN Transmit Channel Empty Event Mask
CAN_TX_CHANNEL_HALF_EMPTY	CAN Transmit Channel Half Empty Event Mask
CAN_TX_CHANNEL_NOT_FULL	CAN Transmit Channel Not Full Event Mask
CAN_TX_CHANNEL_ANY_EVENT	CAN Transmit Channel Any Event Mask

Description

CAN Layer 3 Interrupts

This enumerates all the leading interrupts generated due to CAN transmit and receive events. This enumeration can be used to enable or disable channel events and as a mask to check if a channel event is active.

A single or a combination of the interrupts can be logically ORed to specify the interrupt(s) to be enabled disabled or events to check.

This enumeration is used by PLIB_CAN_ChannelEventEnable, PLIB_CAN_ChannelEventDisable, PLIB_CAN_ChannelEventClear, and PLIB_CAN_ChannelEventGet functions.

CAN_CHANNEL_MASK Enumeration

Lists the series of useful masks.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0_MASK,
  CAN_CHANNEL1_MASK,
  CAN_CHANNEL2_MASK,
  CAN_CHANNEL3_MASK,
  CAN_CHANNEL4_MASK,
  CAN_CHANNEL5_MASK,
  CAN_CHANNEL6_MASK,
  CAN_CHANNEL7_MASK,
  CAN_CHANNEL8_MASK,
  CAN_CHANNEL9_MASK,
  CAN_CHANNEL10_MASK,
  CAN_CHANNEL11_MASK,
  CAN_CHANNEL12_MASK,
  CAN_CHANNEL13_MASK,
  CAN_CHANNEL14_MASK,
```

```
    CAN_CHANNEL15_MASK,
    CAN_CHANNEL16_MASK,
    CAN_CHANNEL17_MASK,
    CAN_CHANNEL18_MASK,
    CAN_CHANNEL19_MASK,
    CAN_CHANNEL20_MASK,
    CAN_CHANNEL21_MASK,
    CAN_CHANNEL22_MASK,
    CAN_CHANNEL23_MASK,
    CAN_CHANNEL24_MASK,
    CAN_CHANNEL25_MASK,
    CAN_CHANNEL26_MASK,
    CAN_CHANNEL27_MASK,
    CAN_CHANNEL28_MASK,
    CAN_CHANNEL29_MASK,
    CAN_CHANNEL30_MASK,
    CAN_CHANNEL31_MASK,
    CAN_ANYCHANNEL_MASK
} CAN_CHANNEL_MASK;
```

Members

Members	Description
CAN_CHANNEL0_MASK	Channel 0 Mask
CAN_CHANNEL1_MASK	Channel 1 Mask
CAN_CHANNEL2_MASK	Channel 2 Mask
CAN_CHANNEL3_MASK	Channel 3 Mask
CAN_CHANNEL4_MASK	Channel 4 Mask
CAN_CHANNEL5_MASK	Channel 5 Mask
CAN_CHANNEL6_MASK	Channel 6 Mask
CAN_CHANNEL7_MASK	Channel 7 Mask
CAN_CHANNEL8_MASK	Channel 8 Mask
CAN_CHANNEL9_MASK	Channel 9 Mask
CAN_CHANNEL10_MASK	Channel 10 Mask
CAN_CHANNEL11_MASK	Channel 11 Mask
CAN_CHANNEL12_MASK	Channel 12 Mask
CAN_CHANNEL13_MASK	Channel 13 Mask
CAN_CHANNEL14_MASK	Channel 14 Mask
CAN_CHANNEL15_MASK	Channel 15 Mask
CAN_CHANNEL16_MASK	Channel 16 Mask
CAN_CHANNEL17_MASK	Channel 17 Mask
CAN_CHANNEL18_MASK	Channel 18 Mask
CAN_CHANNEL19_MASK	Channel 19 Mask
CAN_CHANNEL20_MASK	Channel 20 Mask
CAN_CHANNEL21_MASK	Channel 21 Mask
CAN_CHANNEL22_MASK	Channel 22 Mask
CAN_CHANNEL23_MASK	Channel 23 Mask
CAN_CHANNEL24_MASK	Channel 24 Mask
CAN_CHANNEL25_MASK	Channel 25 Mask
CAN_CHANNEL26_MASK	Channel 26 Mask
CAN_CHANNEL27_MASK	Channel 27 Mask
CAN_CHANNEL28_MASK	Channel 28 Mask
CAN_CHANNEL29_MASK	Channel 29 Mask
CAN_CHANNEL30_MASK	Channel 30 Mask
CAN_CHANNEL31_MASK	Channel 31 Mask
CAN_ANYCHANNEL_MASK	Channel any channel Mask

Description

CAN Channel Masks

This enumeration identifies a series of useful masks. Each mask represents a CAN channel. These masks are used with the PLIB_CAN_AllChannelEventsGet and PLIB_CAN_AllChannelOverflowStatusGet functions. The value returned by these functions can be logically ANDed with any of these masks to check if an event or overflow event for that channel is active.

CAN_DNET_FILTER_SIZE Enumeration

Specifies the size of the DeviceNet filter.

File

plib_can_help.h

C

```
typedef enum {
  CAN_DNET_FILTER_SIZE_18_BIT,
  CAN_DNET_FILTER_SIZE_17_BIT,
  CAN_DNET_FILTER_SIZE_16_BIT,
  CAN_DNET_FILTER_SIZE_15_BIT,
  CAN_DNET_FILTER_SIZE_14_BIT,
  CAN_DNET_FILTER_SIZE_13_BIT,
  CAN_DNET_FILTER_SIZE_12_BIT,
  CAN_DNET_FILTER_SIZE_11_BIT,
  CAN_DNET_FILTER_SIZE_10_BIT,
  CAN_DNET_FILTER_SIZE_9_BIT,
  CAN_DNET_FILTER_SIZE_8_BIT,
  CAN_DNET_FILTER_SIZE_7_BIT,
  CAN_DNET_FILTER_SIZE_6_BIT,
  CAN_DNET_FILTER_SIZE_5_BIT,
  CAN_DNET_FILTER_SIZE_4_BIT,
  CAN_DNET_FILTER_SIZE_3_BIT,
  CAN_DNET_FILTER_SIZE_2_BIT,
  CAN_DNET_FILTER_SIZE_1_BIT,
  CAN_DNET_FILTER_DISABLE
} CAN_DNET_FILTER_SIZE;
```

Members

Members	Description
CAN_DNET_FILTER_SIZE_18_BIT	Compare up to data byte 2 bit 6 with filter mask bit 17
CAN_DNET_FILTER_SIZE_17_BIT	Compare up to data byte 2 bit 7 with filter mask bit 16
CAN_DNET_FILTER_SIZE_16_BIT	Compare up to data byte 1 bit 0 with filter mask bit 15
CAN_DNET_FILTER_SIZE_15_BIT	Compare up to data byte 1 bit 1 with filter mask bit 14
CAN_DNET_FILTER_SIZE_14_BIT	Compare up to data byte 1 bit 2 with filter mask bit 13
CAN_DNET_FILTER_SIZE_13_BIT	Compare up to data byte 1 bit 3 with filter mask bit 12
CAN_DNET_FILTER_SIZE_12_BIT	Compare up to data byte 1 bit 4 with filter mask bit 11
CAN_DNET_FILTER_SIZE_11_BIT	Compare up to data byte 1 bit 5 with filter mask bit 10
CAN_DNET_FILTER_SIZE_10_BIT	Compare up to data byte 1 bit 6 with filter mask bit 9
CAN_DNET_FILTER_SIZE_9_BIT	Compare up to data byte 1 bit 7 with filter mask bit 8
CAN_DNET_FILTER_SIZE_8_BIT	Compare up to data byte 0 bit 0 with filter mask bit 7
CAN_DNET_FILTER_SIZE_7_BIT	Compare up to data byte 0 bit 1 with filter mask bit 6
CAN_DNET_FILTER_SIZE_6_BIT	Compare up to data byte 0 bit 2 with filter mask bit 5
CAN_DNET_FILTER_SIZE_5_BIT	Compare up to data byte 0 bit 3 with filter mask bit 4
CAN_DNET_FILTER_SIZE_4_BIT	Compare up to data byte 0 bit 4 with filter mask bit 3
CAN_DNET_FILTER_SIZE_3_BIT	Compare up to data byte 0 bit 5 with filter mask bit 2
CAN_DNET_FILTER_SIZE_2_BIT	Compare up to data byte 0 bit 6 with filter mask bit 1
CAN_DNET_FILTER_SIZE_1_BIT	Compare up to data byte 0 bit 7 with filter mask bit 0
CAN_DNET_FILTER_DISABLE	Do not compare data bytes, Device Net Filtering is disabled

Description

CAN DeviceNet filter bit numbers.

This enumeration identifies the size of the DeviceNet filter in bits. If the size of the DeviceNet filter is "n", the "n" most significant bits of the data payload are compared with the EID field of enabled filters.

CAN_ERROR_STATE Enumeration

Specifies the CAN module error states.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_RX_WARNING_STATE,
  CAN_RX_WARNING_STATE,
  CAN_TX_WARNING_STATE,
  CAN_RX_BUS_PASSIVE_STATE,
  CAN_TX_BUS_PASSIVE_STATE,
  CAN_TX_BUS_OFF_STATE
} CAN_ERROR_STATE;
```

Members

Members	Description
CAN_TX_RX_WARNING_STATE	CAN Module is in a Transmit or Receive warning state.
CAN_RX_WARNING_STATE	CAN Module is in a Receive warning state.
CAN_TX_WARNING_STATE	CAN Module is in a Transmit warning state.
CAN_RX_BUS_PASSIVE_STATE	CAN Receive is in a Bus Passive state.
CAN_TX_BUS_PASSIVE_STATE	CAN Transmit is in a Bus Passive state.
CAN_TX_BUS_OFF_STATE	CAN Transmit is in Bus Off state.

Description

CAN Error States

This enumeration identifies all of the CAN module error states events. The CAN module enters or exits an error state as the transmit and receive error counter values change. Refer to the CAN 2.0B specification for more details on the error states.

This enumeration is used with the PLIB_CAN_ErrorStateGet function.

CAN_FILTER Enumeration

CAN event code returned by the CAN module.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0_EVENT,
  CAN_CHANNEL1_EVENT,
  CAN_CHANNEL2_EVENT,
  CAN_CHANNEL3_EVENT,
  CAN_CHANNEL4_EVENT,
  CAN_CHANNEL5_EVENT,
  CAN_CHANNEL6_EVENT,
  CAN_CHANNEL7_EVENT,
  CAN_CHANNEL8_EVENT,
  CAN_CHANNEL9_EVENT,
  CAN_CHANNEL10_EVENT,
  CAN_CHANNEL11_EVENT,
  CAN_CHANNEL12_EVENT,
  CAN_CHANNEL13_EVENT,
  CAN_CHANNEL14_EVENT,
  CAN_CHANNEL15_EVENT,
  CAN_CHANNEL16_EVENT,
  CAN_CHANNEL17_EVENT,
  CAN_CHANNEL18_EVENT,
  CAN_CHANNEL19_EVENT,
  CAN_CHANNEL20_EVENT,
  CAN_CHANNEL21_EVENT,
  CAN_CHANNEL22_EVENT,
  CAN_CHANNEL23_EVENT,
  CAN_CHANNEL24_EVENT,
  CAN_CHANNEL25_EVENT,
  CAN_CHANNEL26_EVENT,
  CAN_CHANNEL27_EVENT,
  CAN_CHANNEL28_EVENT,
```

```
    CAN_CHANNEL29_EVENT,
    CAN_CHANNEL30_EVENT,
    CAN_CHANNEL31_EVENT,
    CAN_NO_EVENT,
    CAN_ERROR_EVENT,
    CAN_WAKEUP_EVENT,
    CAN_RX_CHANNEL_OVERFLOW_EVENT,
    CAN_ADDRESS_ERROR_EVENT,
    CAN_BUS_BANDWIDTH_ERROR,
    CAN_TIMESTAMP_TIMER_EVENT,
    CAN_MODE_CHANGE_EVENT,
    CAN_FILTER0,
    CAN_FILTER1,
    CAN_FILTER2,
    CAN_FILTER3,
    CAN_FILTER4,
    CAN_FILTER5,
    CAN_FILTER6,
    CAN_FILTER7,
    CAN_FILTER8,
    CAN_FILTER9,
    CAN_FILTER10,
    CAN_FILTER11,
    CAN_FILTER12,
    CAN_FILTER13,
    CAN_FILTER14,
    CAN_FILTER15,
    CAN_FILTER16,
    CAN_FILTER17,
    CAN_FILTER18,
    CAN_FILTER19,
    CAN_FILTER20,
    CAN_FILTER21,
    CAN_FILTER22,
    CAN_FILTER23,
    CAN_FILTER24,
    CAN_FILTER25,
    CAN_FILTER26,
    CAN_FILTER27,
    CAN_FILTER28,
    CAN_FILTER29,
    CAN_FILTER30,
    CAN_FILTER31
} CAN_FILTER;
```

Members

Members	Description
CAN_CHANNEL0_EVENT	An event on Channel 0 is active
CAN_CHANNEL1_EVENT	An event on Channel 1 is active
CAN_CHANNEL2_EVENT	An event on Channel 2 is active
CAN_CHANNEL3_EVENT	An event on Channel 3 is active
CAN_CHANNEL4_EVENT	An event on Channel 4 is active
CAN_CHANNEL5_EVENT	An event on Channel 5 is active
CAN_CHANNEL6_EVENT	An event on Channel 6 is active
CAN_CHANNEL7_EVENT	An event on Channel 7 is active
CAN_CHANNEL8_EVENT	An event on Channel 8 is active
CAN_CHANNEL9_EVENT	An event on Channel 9 is active
CAN_CHANNEL10_EVENT	An event on Channel 10 is active
CAN_CHANNEL11_EVENT	An event on Channel 11 is active
CAN_CHANNEL12_EVENT	An event on Channel 12 is active
CAN_CHANNEL13_EVENT	An event on Channel 13 is active
CAN_CHANNEL14_EVENT	An event on Channel 14 is active
CAN_CHANNEL15_EVENT	An event on Channel 15 is active
CAN_CHANNEL16_EVENT	An event on Channel 16 is active
CAN_CHANNEL17_EVENT	An event on Channel 17 is active
CAN_CHANNEL18_EVENT	An event on Channel 18 is active
CAN_CHANNEL19_EVENT	An event on Channel 19 is active

CAN_CHANNEL20_EVENT	An event on Channel 20 is active
CAN_CHANNEL21_EVENT	An event on Channel 21 is active
CAN_CHANNEL22_EVENT	An event on Channel 22 is active
CAN_CHANNEL23_EVENT	An event on Channel 23 is active
CAN_CHANNEL24_EVENT	An event on Channel 24 is active
CAN_CHANNEL25_EVENT	An event on Channel 25 is active
CAN_CHANNEL26_EVENT	An event on Channel 26 is active
CAN_CHANNEL27_EVENT	An event on Channel 27 is active
CAN_CHANNEL28_EVENT	An event on Channel 28 is active
CAN_CHANNEL29_EVENT	An event on Channel 29 is active
CAN_CHANNEL30_EVENT	An event on Channel 30 is active
CAN_CHANNEL31_EVENT	An event on Channel 31 is active
CAN_NO_EVENT	No Event is active
CAN_ERROR_EVENT	CAN Bus Error Event is active
CAN_WAKEUP_EVENT	CAN Bus Wakeup Event is active
CAN_RX_CHANNEL_OVERFLOW_EVENT	CAN Receive Channel Overflow Event is active
CAN_ADDRESS_ERROR_EVENT	CAN Address Error Event is active
CAN_BUS_BANDWIDTH_ERROR	CAN Bus Bandwidth Error is active
CAN_TIMESTAMP_TIMER_EVENT	CAN Time Stamp Timer Overflow event is active
CAN_MODE_CHANGE_EVENT	CAN Module Mode Change is active

Description

CAN Event Code

This enumeration identifies all of the event codes returned by the PLIB_CAN_PendingEventsGet function.

CAN_FILTER_MASK Enumeration

Identifies the available CAN filter masks.

File

plib_can_help.h

C

```
typedef enum {
  CAN_FILTER_MASK0,
  CAN_FILTER_MASK1,
  CAN_FILTER_MASK2,
  CAN_FILTER_MASK3,
  CAN_NUMBER_OF_FILTER_MASKS
} CAN_FILTER_MASK;
```

Members

Members	Description
CAN_FILTER_MASK0	CAN Filter Mask 0
CAN_FILTER_MASK1	CAN Filter Mask 1
CAN_FILTER_MASK2	CAN Filter Mask 2
CAN_FILTER_MASK3	CAN Filter Mask 3
CAN_NUMBER_OF_FILTER_MASKS	Total number of filter masks in the module

Description

CAN Filter Masks

This enumeration identifies the available filters masks in each CAN module.

CAN_FILTER_MASK_TYPE Enumeration

Specifies the CAN filter mask type.

File

plib_can_help.h

C

```
typedef enum {
  CAN_FILTER_MASK_IDE_TYPE,
  CAN_FILTER_MASK_ANY_TYPE
} CAN_FILTER_MASK_TYPE;
```

Members

Members	Description
CAN_FILTER_MASK_IDE_TYPE	Mask processes only Filter type of message
CAN_FILTER_MASK_ANY_TYPE	Mask processes any type (SID or EID) of message

Description

CAN Filter Mask Type

This enumeration specifies the filter mask type. The filter mask can either process messages with any type of ID (extended or standard) or can only process by ID specified in the filter configuration. For example, if the filter associated with the mask only accepts EID type messages and if the mask is configured to process by ID type, then SID messages will not be accepted. If the mask is configured to process any type of message, both SID and EID messages will be accepted on a filter match.

CAN_ID_TYPE Enumeration

Specifies the CAN ID type.

File

plib_can_help.h

C

```
typedef enum {
  CAN_EID,
  CAN_SID
} CAN_ID_TYPE;
```

Members

Members	Description
CAN_EID	CAN Extended ID
CAN_SID	CAN Standard ID

Description

CAN ID Type

This enumeration specifies the two CAN ID types: Standard and Extended. The Standard Type ID is 11 bits long and the Extended ID is 29 bits long. This enumeration then specifies the type of the ID specified while configuring filters and filter masks.

CAN_MODULE_EVENT Enumeration

Specifies the CAN module events

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_EVENT,
  CAN_RX_EVENT,
  CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT,
  CAN_OPERATION_MODE_CHANGE_EVENT,
  CAN_RX_OVERFLOW_EVENT,
  CAN_SYSTEM_ERROR_EVENT,
  CAN_BUS_ERROR_EVENT,
  CAN_BUS_ACTIVITY_WAKEUP_EVENT,
  CAN_INVALID_RX_MESSAGE_EVENT
} CAN_MODULE_EVENT;
```

Members

Members	Description
CAN_TX_EVENT	Transmit channel event. This event will occur if any of the Transmit Channel events are active.
CAN_RX_EVENT	Receive channel event. This event will occur if any of the Receive Channel events are active.
CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT	CAN Time Stamp Timer Overflow event occurs. This event occurs when the Time Stamp Timer has overflowed.
CAN_OPERATION_MODE_CHANGE_EVENT	CAN Operation Mode Change Event. This event occurs when the CAN module has changed its operating mode successfully.
CAN_RX_OVERFLOW_EVENT	CAN Receive Channel Overflow Event. This event occurs when any of the Receive Channel has overflowed.
CAN_SYSTEM_ERROR_EVENT	CAN System Error Event. This event occurs when CAN module tries to access an invalid Device RAM location.
CAN_BUS_ERROR_EVENT	CAN Bus Error Event. This event occurs when the CAN module cannot access the system bus.
CAN_BUS_ACTIVITY_WAKEUP_EVENT	CAN Bus Activity Wakeup. This event occurs when the device is in Sleep mode and bus activity is detected on the CAN bus.
CAN_INVALID_RX_MESSAGE_EVENT	CAN Bus Invalid Receive Message Event. This event occurs when the CAN module receives an Invalid message.

Description

CAN Module Events

This enumeration identifies all of the CAN module events. A combination of listed events can be logically ORed to enable or disable module level events. Similarly, a combination of events can be checked if active.

This enumeration is used with the PLIB_CAN_ModuleEventEnable function and PLIB_CAN_ModuleEventGet function.

CAN_MODULE_ID Enumeration

File

plib_can_help.h

C

```
typedef enum {
  CAN_ID_1,
  CAN_ID_2,
  CAN_ID_3,
  CAN_NUMBER_OF_MODULES
} CAN_MODULE_ID;
```

Members

Members	Description
CAN_NUMBER_OF_MODULES	The total number of modules available.

Description

Enumeration: CAN_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.

CAN_MSG_EID Structure

Defines the structure of the EID word section of the transmit and receive message.

File

plib_can.h

C

```
typedef struct {
  unsigned data_length_code : 4;
  unsigned reserved0 : 1;
  unsigned unimplemented1 : 3;
  unsigned reserved1 : 1;
```

```
  unsigned remote_request : 1;
  unsigned eid : 18;
  unsigned ide : 1;
  unsigned sub_remote_req : 1;
  unsigned unimplemented2 : 2;
} CAN_MSG_EID;
```

Members

Members	Description
unsigned data_length_code : 4;	Data Length Control. Specifies the size of the data payload section of the CAN packet. Valid values are 0x0 - 0x8
unsigned reserved0 : 1;	Reserved bit. Should be always 0.
unsigned unimplemented1 : 3;	Unimplemented bit. Should be always 0.
unsigned reserved1 : 1;	Reserved bit. Should be always 0.
unsigned remote_request : 1;	Remote Transmit Request bit. Should be set for RTR messages, clear otherwise.
unsigned eid : 18;	CAN Transmit and Receive Extended ID field. Valid values are in range 0x0 - 0x3FFFF
unsigned ide : 1;	Identifier bit. If 0 means that message is SID. If 1 means that message is EID type.
unsigned sub_remote_req : 1;	Substitute Remote request bit. This bit should always be clear for an EID message. It is ignored for an SID message.
unsigned unimplemented2 : 2;	Unimplemented bit. Should be always 0.

Description

CAN Message EID Word

This data structure represents the EID section of the CAN transmit and receive message. The data structure is an element of the CAN_TX_MSG_BUFFER and CAN_RX_MSG_BUFFER data structure.

CAN_OPERATION_MODES Enumeration

Lists all possible CAN module operational modes.

File

plib_can_help.h

C

```
typedef enum {
  CAN_LISTEN_ALL_MESSAGES_MODE,
  CAN_CONFIGURATION_MODE,
  CAN_LISTEN_ONLY_MODE,
  CAN_LOOPBACK_MODE,
  CAN_DISABLE_MODE,
  CAN_NORMAL_MODE
} CAN_OPERATION_MODES;
```

Members

Members	Description
CAN_LISTEN_ALL_MESSAGES_MODE	CAN Listen All Message Mode. The CAN module listens to all messages, regardless of errors
CAN_CONFIGURATION_MODE	CAN Configuration Mode. Various CAN module settings can be configured in this mode
CAN_LISTEN_ONLY_MODE	CAN Listen Only Mode. In this mode, the CAN module will not acknowledge signal and will not participate in error signaling. All messages are captured
CAN_LOOPBACK_MODE	CAN Loopback Mode. In this mode, the CAN module Transmit is internally connected to the Receive line. This mode is useful for debugging operation
CAN_DISABLE_MODE	CAN Disable Mode. The CAN module does not transmit or receive messages in this mode
CAN_NORMAL_MODE	CAN Normal Operation Mode. The CAN module transmits and receives messages in this mode

Description

CAN Operation Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the PLIB_CAN_OperationModeSelect function, and should then use the PLIB_CAN_OperationModeGet function to check if the CAN module has entered the requested mode.

CAN_RECEIVE_CHANNEL Enumeration

Lists all possible CAN module receive channels.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RECEIVE_CHANNEL0,
  CAN_RECEIVE_CHANNEL1
} CAN_RECEIVE_CHANNEL;
```

Members

Members	Description
CAN_RECEIVE_CHANNEL0	CAN receive channel 0
CAN_RECEIVE_CHANNEL1	CAN receive channel 1

Description

CAN Receive Channels

This enumerates all CAN module receive channels(channels that are not allowed to be configured for transmit).

CAN_RECEIVE_MODES Enumeration

Lists all possible CAN module receive modes.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RECEIVE_ALL_MSG_MODE,
  CAN_RECEIVE_VALID_EID_MODE,
  CAN_RECEIVE_VALID_STD_MODE,
  CAN_RECEIVE_VALID_MODE
} CAN_RECEIVE_MODES;
```

Members

Members	Description
CAN_RECEIVE_ALL_MSG_MODE	Receive all messages(Including with errors). The message filters will not be considered
CAN_RECEIVE_VALID_EID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_STD_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with standard identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier/standard identifier Based on the configuration

Description

CAN Receive Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the PLIB_CAN_ReceiveModeSelect function, and should then use the PLIB_CAN_ReceiveModeGet function to check if the CAN module has entered the requested mode.

CAN_RX_DATA_MODE Enumeration

Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.

File

plib_can_help.h

C

```
typedef enum {
```

```
  CAN_RX_DATA_ONLY,
  CAN_RX_FULL_RECEIVE
} CAN_RX_DATA_MODE;
```

Members

Members	Description
CAN_RX_DATA_ONLY	CAN Receive Channel Data Only Mode is enabled
CAN_RX_FULL_RECEIVE	CAN Receive Channel Full Receive Mode is enabled

Description

CAN Receive Channel Data Only Mode

This enumeration specifies the status of the CAN receive channel data-only feature. If the feature is enabled, the CAN module will store only the data payload portion of the received CAN message. If the Full Receive mode is specified, the CAN module stores the entire CAN message (ID field plus data payload). The receive channel can be either in Data-Only mode or Full Receive mode.

CAN_RX_MSG_BUFFER Union

Defines the structure of the CAN receive message buffer

File

plib_can.h

C

```
typedef union {
  struct {
    CAN_RX_MSG_SID msgSID;
    CAN_MSG_EID msgEID;
    uint8_t data[8];
  }
  uint8_t dataOnlyMsgData[8];
  uint32_t messageWord[4];
} CAN_RX_MSG_BUFFER;
```

Members

Members	Description
CAN_RX_MSG_SID msgSID;	This is SID portion of the CAN Receive message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Receive message
uint8_t data[8];	This is the data payload section of the received message
uint8_t dataOnlyMsgData[8];	This can be used if the message buffer is to be read from a Data-Only type of CAN Receive Channel
uint32_t messageWord[4];	This is CAN Receive message organized as a set of 32-bit words

Description

CAN Receive Message Buffer

This data structure represents the CAN receive message buffer. Received messages could be either full-receive messages or data-only messages. A full-receive message contains the message header and data payload section. For a full-receive CAN receive channel, the caller should use the msgSID, msgEID and data members. A data-only message contains only an 8-byte data payload. While using this data structure with a data-only type of CAN receive channel, the caller should use the dataOnlyMsgData member of the structure and should read only 8 bytes of data.

CAN_RX_MSG_SID Structure

Defines the structure of the SID word section of the receive message.

File

plib_can.h

C

```
typedef struct {
  unsigned sid : 11;
  unsigned filter_hit : 5;
  unsigned msg_timestamp : 16;
} CAN_RX_MSG_SID;
```

Members

Members	Description
unsigned sid : 11;	SID of the Received CAN Message
unsigned filter_hit : 5;	Filter which accepted this message
unsigned msg_timestamp : 16;	Time stamp of the received message. This is valid only if the Timestamping is enabled

Description

CAN Receive Message SID Word

This data structure represents the SID section of the CAN receive message. The data structure is an element of the CAN_RX_MSG_BUFFER data structure.

CAN_TIME_SEGMENT_LENGTH Enumeration

All possible values for the assignable number of Time Quanta.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TIME_SEGMENT_LEN_8TQ,
  CAN_TIME_SEGMENT_LEN_7TQ,
  CAN_TIME_SEGMENT_LEN_6TQ,
  CAN_TIME_SEGMENT_LEN_5TQ,
  CAN_TIME_SEGMENT_LEN_4TQ,
  CAN_TIME_SEGMENT_LEN_3TQ,
  CAN_TIME_SEGMENT_LEN_2TQ,
  CAN_TIME_SEGMENT_LEN_1TQ
} CAN_TIME_SEGMENT_LENGTH;
```

Members

Members	Description
CAN_TIME_SEGMENT_LEN_8TQ	Segment length is 8 x TQ
CAN_TIME_SEGMENT_LEN_7TQ	Segment length is 7 x TQ
CAN_TIME_SEGMENT_LEN_6TQ	Segment length is 6 x TQ
CAN_TIME_SEGMENT_LEN_5TQ	Segment length is 5 x TQ
CAN_TIME_SEGMENT_LEN_4TQ	Segment length is 4 x TQ
CAN_TIME_SEGMENT_LEN_3TQ	Segment length is 3 x TQ
CAN_TIME_SEGMENT_LEN_2TQ	Segment length is 2 x TQ
CAN_TIME_SEGMENT_LEN_1TQ	Segment length is 1 x TQ

Description

CAN Segment length

This enumeration defines values that can be assigned while defining the number of Time Quanta in a bit.

CAN_TX_CHANNEL_STATUS Enumeration

Identifies possible transmit channel-specific conditions.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_CHANNEL_TRANSMITTING,
  CAN_TX_CHANNEL_ERROR,
  CAN_TX_CHANNEL_ARBITRATION_LOST
} CAN_TX_CHANNEL_STATUS;
```

Members

Members	Description
CAN_TX_CHANNEL_TRANSMITTING	CAN Transmit Channel is currently Transmitting.
CAN_TX_CHANNEL_ERROR	CAN Transmit Channel Error has occurred.
CAN_TX_CHANNEL_ARBITRATION_LOST	CAN Transmit Channel lost arbitration.

Description

CAN Transmit Channel Condition

This enumeration identifies the possible transmit channel condition. These masks can be logically ANDed with values returned by the PLIB_CAN_TransmitChannelStatusGet function to check if a condition is active.

CAN_TX_MSG_BUFFER Union

Defines the structure of the CAN transmit message buffer.

File

plib_can.h

C

```
typedef union {
  struct {
    CAN_TX_MSG_SID msgSID;
    CAN_MSG_EID msgEID;
    unsigned char data[8];
  }
  uint32_t messageWord[4];
} CAN_TX_MSG_BUFFER;
```

Members

Members	Description
CAN_TX_MSG_SID msgSID;	This is SID portion of the CAN Transmit message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Transmit message
unsigned char data[8];	This is the data portion of the CAN Transmit message
uint32_t messageWord[4];	This is CAN Transmit message organized as a set of 32 bit words

Description

CAN Transmit Message Buffer

This data structure represents the CAN transmit message buffer. This should be used for writing data to a CAN transmit channel and transmitting data. A pointer to this type of data structure is returned by the PLIB_CAN_TransmitBufferGet function.

The data structure allows the transmit message buffer to be accessed as fields of a CAN transmit message and also as an array of four 32-bit words. The latter allows for quick initialization of the message buffer.

CAN_TX_MSG_SID Structure

Defines the structure of the SID word section of the transmit message.

File

plib_can.h

C

```
typedef struct {
  unsigned sid : 11;
} CAN_TX_MSG_SID;
```

Members

Members	Description
unsigned sid : 11;	CAN Transmit Message Standard ID. This value should be between 0x0 - 0x7FF

Description

CAN Transmit Message SID Word

This data structure represents the SID section of the CAN transmit message. The data structure is an element of the CAN_TX_MSG_BUFFER data structure.

CAN_TX_RTR Enumeration

Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_RTR_ENABLED,
  CAN_TX_RTR_DISABLED
} CAN_TX_RTR;
```

Members

Members	Description
CAN_TX_RTR_ENABLED	CAN Transmit Channel RTR Feature is enabled
CAN_TX_RTR_DISABLED	CAN Transmit Channel RTR Feature is disabled

Description

CAN Transmit Channel Remote Transmit Request (RTR)

This enumeration specifies the status of the CAN RTR feature for a CAN transmit channel. The RTR feature allows a node on the CAN Bus to request a transmission from another node on the bus. The responding node in this case should have a RTR enabled Transmit Channel in order to be able to respond to this request.

CAN_TXCHANNEL_PRIORITY Enumeration

Specifies the priority of a transmit channel.

File

plib_can_help.h

C

```
typedef enum {
  CAN_LOWEST_PRIORITY,
  CAN_LOW_MEDIUM_PRIORITY,
  CAN_HIGH_MEDIUM_PRIORITY,
  CAN_HIGHEST_PRIORITY
} CAN_TXCHANNEL_PRIORITY;
```

Members

Members	Description
CAN_LOWEST_PRIORITY	CAN lowest priority
CAN_LOW_MEDIUM_PRIORITY	CAN low medium priority
CAN_HIGH_MEDIUM_PRIORITY	CAN high medium priority
CAN_HIGHEST_PRIORITY	CAN highest priority

Description

CAN Transmit Channel Priority

This enumeration identifies the available transmit channel priorities. A transmit channel has its own natural priority order, which determines priority when two or more transmit channels are assigned the same priority level. Channel 1 has higher natural priority than channel 0 and channel 2 has a natural priority than channel 1, and so on.

CAN_RX_DATA_ONLY_SIZE_BYTES Macro

Used as the size of the CAN data-only receive message buffer in bytes.

File

plib_can.h

C

```
#define CAN_RX_DATA_ONLY_SIZE_BYTES 8
```

Description

CAN Data-Only Receive Message Buffer

This constant is used as the size of the CAN data-only receive message buffer in bytes.

CAN_TX_RX_MESSAGE_SIZE_BYTES Macro

Used as the array size of the CAN transmit and full receive message buffer.

File

plib_can.h

C

```
#define CAN_TX_RX_MESSAGE_SIZE_BYTES 16
```

Description

CAN Transmit and Receive Message Buffer size

This constant is used as the array size of the CAN transmit and full receive message buffer.

Files

Files

Name	Description
plib_can.h	This file contains the interface definition for the CAN Peripheral Library.
plib_can_help.h	This file contains the enumerations for the CAN Peripheral Library help file.

Description

This section lists the source and header files used by the library.

plib_can.h

This file contains the interface definition for the CAN Peripheral Library.

Functions

	Name	Description
	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.
	PLIB_CAN_BaudRateGet	Returns the current CAN module Baud rate.
	PLIB_CAN_BaudRatePrescaleSetup	Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.
	PLIB_CAN_BitSamplePhaseSet	Sets the CAN bit-sampling phase parameters.
	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.
	PLIB_CAN_ChannelEventEnable	Enables channel level events.
	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelResetIsComplete	Returns 'true' if the specified channel reset is complete.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_DeviceNetConfigure	Configures the CAN module DeviceNet(TM) filter feature.

	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.
	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRateGet	Identifies whether the BaudRateGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRatePrescaleSetup	Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.
	PLIB_CAN_ExistsBitSamplePhaseSet	Identifies whether the BitSamplePhaseSet feature exists on the CAN module.
	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.
	PLIB_CAN_ExistsFilterConfigure	Identifies whether the FilterConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterEnable	Identifies whether the FilterEnable feature exists on the CAN module.
	PLIB_CAN_ExistsFilterMaskConfigure	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterToChannelLink	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	PLIB_CAN_ExistsLatestFilterMatchGet	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	PLIB_CAN_ExistsMemoryBufferAssign	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventClear	Identifies whether the ModuleEvents feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventEnable	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsModuleInfo	Identifies whether the ModuleInformation feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeRead	Identifies whether the OperationModeRead feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeWrite	Identifies whether the OperationModeSet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingEventsGet	Identifies whether the PendingEventsGet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingTransmissionsAbort	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.
	PLIB_CAN_ExistsPrecalculatedBitRateSetup	Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.
	PLIB_CAN_ExistsReceivedMessageGet	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	PLIB_CAN_ExistsReceiveErrorCount	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.

	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.
	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
	PLIB_CAN_ModuleEventDisable	Disables the module level events.
	PLIB_CAN_ModuleEventEnable	Enables the module level events.
	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.
	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_PrecalculatedBitRateSetup	Sets the desired Baud rate for the respective CAN module.
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampPrescalerSet	Sets the CAN receive message time stamp timer prescaler.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

Macros

	Name	Description
	CAN_RX_DATA_ONLY_SIZE_BYTES	Used as the size of the CAN data-only receive message buffer in bytes.
	CAN_TX_RX_MESSAGE_SIZE_BYTES	Used as the array size of the CAN transmit and full receive message buffer.

Structures

	Name	Description
	CAN_MSG_EID	Defines the structure of the EID word section of the transmit and receive message.
	CAN_RX_MSG_SID	Defines the structure of the SID word section of the receive message.
	CAN_TX_MSG_SID	Defines the structure of the SID word section of the transmit message.

Unions

	Name	Description
	CAN_RX_MSG_BUFFER	Defines the structure of the CAN receive message buffer
	CAN_TX_MSG_BUFFER	Defines the structure of the CAN transmit message buffer.

Description

CAN Peripheral Library Interface Header

This library provides a low-level abstraction of the Controller Area Network (CAN) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_can.h

Company

Microchip Technology Inc.

plib_can_help.h

This file contains the enumerations for the CAN Peripheral Library help file.

Enumerations

	Name	Description
	CAN_CHANNEL	Identifies the supported CAN Channels.
	CAN_CHANNEL_EVENT	Identifies all Layer 3 interrupts.
	CAN_CHANNEL_MASK	Lists the series of useful masks.
	CAN_DNET_FILTER_SIZE	Specifies the size of the DeviceNet filter.
	CAN_ERROR_STATE	Specifies the CAN module error states.
	CAN_FILTER	CAN event code returned by the CAN module.
	CAN_FILTER_MASK	Identifies the available CAN filter masks.
	CAN_FILTER_MASK_TYPE	Specifies the CAN filter mask type.
	CAN_ID_TYPE	Specifies the CAN ID type.
	CAN_MODULE_EVENT	Specifies the CAN module events
	CAN_MODULE_ID	Enumeration: CAN_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	CAN_OPERATION_MODES	Lists all possible CAN module operational modes.
	CAN_RECEIVE_CHANNEL	Lists all possible CAN module receive channels.
	CAN_RECEIVE_MODES	Lists all possible CAN module receive modes.
	CAN_RX_DATA_MODE	Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.
	CAN_TIME_SEGMENT_LENGTH	All possible values for the assignable number of Time Quanta.
	CAN_TX_CHANNEL_STATUS	Identifies possible transmit channel-specific conditions.
	CAN_TX_RTR	Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.
	CAN_TXCHANNEL_PRIORITY	Specifies the priority of a transmit channel.

Description

CAN Peripheral Library help file

This file contains the enumerations for the CAN Peripheral Library help file.

File Name

plib_can_help.h

Company

Microchip Technology Inc.

Comparator Peripheral Library

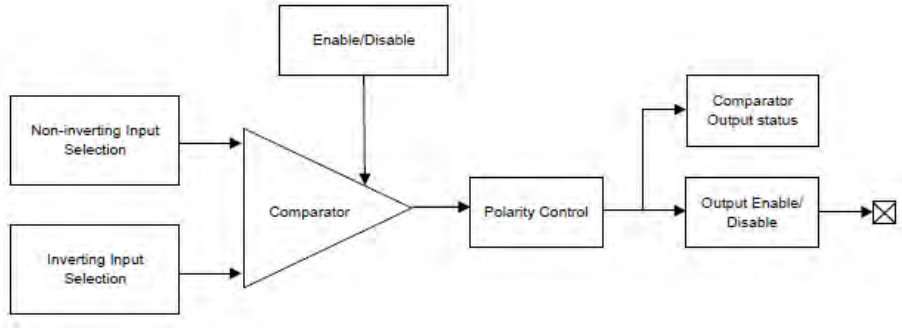
This section describes the Comparator Peripheral Library.

Introduction

The Comparator module is comprised of several identical analog comparator blocks, each complete with its own supporting input selectors and output logic. Each Comparator can be configured in a variety of ways, independent of the other comparators. The input voltage to the module can be either fed externally or it can be configured to use the internal voltage.

Description

This library provides a low-level abstraction of the Comparator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby abstracting hardware differences from one microcontroller variant to another.



Using the Library

This topic describes the basic architecture of the Comparator Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_cmp.h`

The interface to the Comparator Peripheral Library is defined in the `plib_cmp.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Comparator Peripheral Library must include `peripheral.h`.

Library File:

The Comparator Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. The first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the Comparator Peripheral Library, the module instance IDs are defined by the Comparator module enumeration.

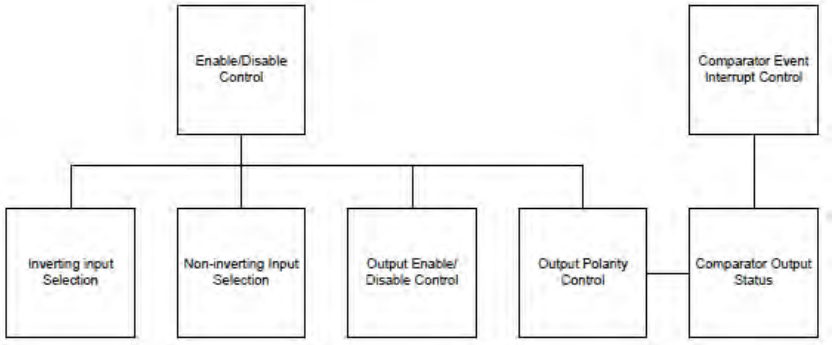
Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Comparator (CMP) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Diagram



Hardware Abstraction Model Description

The Comparator Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram.

The Comparator VIN+ and VIN- block selects the Comparator non-inverting and inverting inputs, which can be either external or internal inputs.

The Comparator module block compares the voltage level on the inputs and produce an output. If the input VIN+ > VIN-, the output will be high. The output will be low when VIN+ < VIN-.

The Comparator Output Polarity block selects the Comparator output polarity.

The Comparator Output Control block enables/disables the Comparator output pin.

The Comparator Interrupt Logic Control block decide the comparator interrupt generation. The interrupt can happen at Low-to-high, High-to-Low, and for both transitions of the Comparator output.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Comparator module

Library Interface Section	Description
General Configuration	Provides the configuration routines for: • Enabling and disabling the module • Enabling and disabling Comparator output • Selecting Comparator input • Comparator status
Comparator Feature Configuration	Provides the configuration routines for various features in the Comparator: • Enabling and disabling Comparator Stop in Idle mode • Selecting Comparator speed/power • Enabling and disabling Comparator hysteresis • Enabling and disabling Comparator filters
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

The following processes are involved while using a Comparator module:

- CVREF Setup
- Initialization
- Configuring the Comparator interrupt (optional)
- Power-Saving modes
- Turning on the Comparator module

CVREF Setup

This topic describes how to configure the Comparator Voltage Reference (CVREF) if it is to be used as one of the inputs for the Comparator module.

Description

The CVREF is one of the internal voltage references available for the Comparator module, which can be configured for the required voltage value.

Do the following to set up the CVREF:

1. Select the voltage source for CVREF using PLIB_CMP_CVREF_SourceVoltageSelect.
2. Select the CVREF voltage range using PLIB_CMP_CVREF_WideRangeEnable and/or PLIB_CMP_CVREF_WideRangeDisable.
3. Select the CVREF value using PLIB_CMP_CVREF_ValueSelect.
4. Enable or disable the CVREF pin output using PLIB_CMP_CVREF_OutputEnable or PLIB_CMP_CVREF_OutputDisable.
5. Turn on the CVREF module using PLIB_CMP_CVREF_Enable.

Example

```
#define MY_CMP_ID           CMP_ID_1

//Select the CVREF source
PLIB_CMP_CVREF_SourceVoltageSelect(MY_CMP_ID, CMP_CVREF_VOLTAGE_SOURCE_VDD);

//Select the CVREF for wide range
PLIB_CMP_CVREF_WideRangeEnable(MY_CMP_ID);

//Select the CVREF value
PLIB_CMP_CVREF_ValueSelect(MY_CMP_ID, CMP_CVREF_VALUE_15);

//Enable the CVREF output pin
PLIB_CMP_CVREF_OutputEnable(MY_CMP_ID);

//Turn on the CVREF module
PLIB_CMP_CVREF_Enable(MY_CMP_ID);
```

Initialization

This section describes Comparator module initialization.

Description

Do the following when using a Comparator module:

1. Select the Comparator inverting input using PLIB_CMP_InvertingInputChannelSelect.
2. Select the Comparator non-inverting input using PLIB_CMP_NonInvertingInputChannelSelect.
3. Select the Comparator output polarity using PLIB_CMP_OutputInvertEnable and/or PLIB_CMP_OutputInvertDisable.
4. Select the Comparator output pin using PLIB_CMP_OutputEnable and/or PLIB_CMP_OutputDisable.
5. Turn on the Comparator module using PLIB_CMP_Enable.

Example

```
#define MY_CMP_ID           CMP_ID_1

//Select the comparator inverting input
PLIB_CMP_InvertingInputChannelSelect(MY_CMP_INSTANCE,CMP_INVERTING_INPUT_CHANNEL_B);

//Select the comparator non-inverting input
PLIB_CMP_NonInvertingInputChannelSelect(MY_CMP_INSTANCE,CMP_NON_INVERTING_INPUT_A);

//Select the comparator output polarity
 PLIB_CMP_OutputInvertDisable(MY_CMP_INSTANCE);

//Enable the comparator output pin
PLIB_CMP_OutputEnable(MY_CMP_INSTANCE);

//Turn on the comparator module
PLIB_CMP_Enable(MY_CMP_INSTANCE);
```

Configuring the Comparator Interrupts

This section describes the configuration of Comparator module interrupts.

Description

The comparator interrupt can happen for three events. The user must configure the event-detection logic to report changes to the output state of the Comparator, which can in turn be used for event or interrupt generation. The options include event generation on rising state changes (low-to-high), falling state changes (high-to-low), and all state changes. When a configured event occurs, it is flagged by the event detection flag.

The user must clear this flag for the next interrupt generation.

The following step is involved while configuring a Comparator interrupt:

1. Select the Comparator interrupt event using PLIB_CMP_InterruptEventSelect.

Example

```
#define MY_CMP_ID          CMP_ID_1

//This code expects the comparator interrupt
//enabled in the interrupt module

//Select the comparator interrupt event
PLIB_CMP_InterruptEventSelect(MY_CMP_INSTANCE, CMP_LOW_TO_HIGH);

// If the event meets, interrupt will occur. Do the respective
   operation in the ISR and clear the interrupt flag.
```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

Power-Saving Modes

This section provides information on the Power-Saving modes for the Comparator module.

Description

Operation in Sleep mode: If the Comparator interrupt is enabled, the device will wake up from Sleep mode on the Comparator interrupt. If the Comparator interrupt is disabled,the comparator will work, but it will not wake up the device from sleep.

Operation in Idle mode: The functions PLIB_CMP_StopInIdleModeEnable and PLIB_CMP_StopInIdleModeDisable determine if the Comparator module stops or continues operation in Idle mode. If the function PLIB_CMP_StopInIdleModeDisable is used, the module will continue operation in the Idle mode. If the Comparator interrupt is enabled, the device will wake up from Idle mode on the Comparator interrupt. If the function PLIB_CMP_StopInIdleModeEnable is used, the module will work in Idle mode but the interrupt will not generate if it is enabled.

#	Description	Function Associated
1	Enable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeEnable
2	Disable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeDisable

Example

```
//Select comparator stop in idle mode
PLIB_CMP_StopInIdleModeEnable(MY_CMP_INSTANCE);
```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

Configuring the Library

The library is configured for the supported Comparator module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value, CVref.
	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.

	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.
	PLIB_CMP_Disable	Disables the Comparator module.
	PLIB_CMP_Enable	Enables the Comparator module.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_OutputInvertDisable	Comparator output is non-inverted.
	PLIB_CMP_OutputInvertEnable	Comparator output is inverted.
	PLIB_CMP_OutputLogicHigh	Comparator output bit will give a 'logic high' on satisfying the input condition.
	PLIB_CMP_OutputLogicLow	Comparator will be set to give 'logic low' on satisfying the input condition.

b) Comparator Feature Configuration Functions

	Name	Description
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.
	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

c) Feature Existence Functions

	Name	Description
	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect	Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the CMP module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CMP module.
	PLIB_CMP_ExistsOutputStatusGet	Identifies whether the OutputStatusGet feature exists on the CMP module.

d) Data Types and Constants

	Name	Description
	CMP_CLOCK_DIVIDE	Defines Comparator Filter Clock Divide.
	CMP_CVREF_BANDGAP_SELECT	Lists the band gap selection options.
	CMP_CVREF_REFERENCE_SELECT	Lists the reference selection options.

	CMP_CVREF_VALUE	Lists the voltage reference value selection options.
	CMP_CVREF_VOLTAGE_SOURCE	Lists the Voltage source options.
	CMP_FILTER_CLOCK	Defines Comparator filter input clock
	CMP_INTERRUPT_EVENT	Defines Comparator interrupt events.
	CMP_INVERTING_INPUT	Defines the list of Comparator inverting Input.
	CMP_MASK_A	Defines Comparator Mask A Input.
	CMP_MASK_B	Defines Comparator Mask B Input.
	CMP_MASK_C	Defines Comparator Mask C Input.
	CMP_MODULE_ID	Identifies the Comparator modules supported
	CMP_NON_INVERTING_INPUT	Defines the list of Comparator non-inverting Input.
	CMP_SPEED_POWER	Defines the Speed/Power of the Comparator

Description

This section describes the Application Programming Interface (API) functions of the Comparator Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_CMP_CVREF_BandGapReferenceSourceSelect Function

Selects the band gap reference voltage source.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_BandGapReferenceSourceSelect(CMP_MODULE_ID index, CMP_CVREF_BANDGAP_SELECT select);
```

Returns

None.

Description

This function selects the band gap reference voltage source from the available options from CMP_CVREF_BANDGAP_SELECT.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_BandGapReferenceSourceSelect ( CMP_ID_1,  CMP_CVREF_BANDGAP_0_6V );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
select	Select a band gap reference source from CMP_CVREF_BANDGAP_SELECT

Function

void PLIB_CMP_CVREF_BandGapReferenceSourceSelect (CMP_MODULE_ID index,
CMP_CVREF_BANDGAP_SELECT bandGap);

PLIB_CMP_CVREF_Enable Function

Enables the voltage reference of the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_Enable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage reference of the Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFEnableControl in your application to determine whether this feature is available.

Preconditions

The Comparator module should be appropriately configured before being enabled.

Example

```
PLIB_CMP_CVREF_Enable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_Enable (CMP_CVREF_MOD index)

PLIB_CMP_CVREF_OutputDisable Function

Disables the output voltage.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_OutputDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the reference voltage output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_OutputDisable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_OutputDisable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_OutputEnable Function

Enables the voltage output.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_OutputEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage output

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_OutputEnable(CMP_ID_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_OutputEnable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_ReferenceVoltageSelect Function

Selects the voltage reference value, CVref.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_ReferenceVoltageSelect(CMP_MODULE_ID index, CMP_CVREF_REFERENCE_SELECT reference);
```

Returns

None.

Description

This function selects the voltage reference value, CVref. This value decides which voltage source should be taken as reference voltage from the set CMP_CVREF_REFERENCE_SELECT.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsCVREFRefVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

Determine the correct value that should be passed.

Example

```
PLIB_CMP_CVREF_ReferenceVoltageSelect ( CMP_ID_1,  CMP_CVREF_RESISTOR_LADDER_VOLTAGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_REFERENCE_SELECT

Function

void PLIB_CMP_CVREF_ReferenceVoltageSelect (CMP_MODULE_ID index,
CMP_CVREF_REFERENCE_SELECT reference);

PLIB_CMP_CVREF_SourceNegativeInputSelect Function

Configures the Comparator module to use the selected input as a negative reference.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_SourceNegativeInputSelect(CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE
negInput);
```

Returns

None.

Description

This function configures the Comparator module to use the selected input as a negative reference for the voltage source.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability. For such devices, selecting the positive source will automatically select the negative input.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_SourceNegativeInputSelect ( CMP_ID_1, CMP_CVREF_VOLTAGE_SOURCE_NEG_REF_GROUND );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
negInput	Select the voltage source negative reference from CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE

Function

void PLIB_CMP_CVREF_SourceNegativeInputSelect (CMP_MODULE_ID index,
CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE negInput)

PLIB_CMP_CVREF_SourceVoltageSelect Function

Connects the Comparator module to the selected voltage source.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_SourceVoltageSelect(CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE source);
```

Returns

None.

Description

This function connects the Comparator module to the selected voltage source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_SourceVoltageSelect ( CMP_ID_1, CMP_CVREF_VOLTAGE_SOURCE_INTERNAL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the voltage source from CMP_CVREF_VOLTAGE_SOURCE

Function

void PLIB_CMP_CVREF_SourceVoltageSelect (CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE source)

PLIB_CMP_CVREF_ValueSelect Function

Selects the voltage reference value.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_ValueSelect(CMP_MODULE_ID index, CMP_CVREF_VALUE value);
```

Returns

None.

Description

This function selects the voltage reference value. This value decides how many resistance units will be added and therefore, decides the output voltage.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFValueSelect in your application to determine whether this feature is available.

Preconditions

Determine the correct value that should be passed.

Example

```
PLIB_CMP_CVREF_ValueSelect ( CMP_ID_1,  CMP_CVREF_VALUE_13 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_VALUE

Function

void PLIB_CMP_CVREF_ValueSelect (CMP_MODULE_ID index,
CMP_CVREF_VALUE value);

PLIB_CMP_CVREF_WideRangeDisable Function

Disables the wide range.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_WideRangeDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the wide range for reference voltage. The range of possible voltages will become narrower, and finer voltage options can be achieved in this case.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_WideRangeDisable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_WideRangeDisable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_WideRangeEnable Function

Enables the wide range.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_WideRangeEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the wide range for reference voltage. The voltage range starts from zero if the wide range is selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_WideRangeEnable(CMP_ID_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_WideRangeEnable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_WideRangeIsEnabled Function

Returns whether the wide range is selected for the reference voltage.

File

plib_cmp.h

C

```
bool PLIB_CMP_CVREF_WideRangeIsEnabled(CMP_MODULE_ID index);
```

Returns

- true = The wide range is enabled
- false = The wide range is not enabled

Description

This function returns whether the wide range is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool range;

range = PLIB_CMP_CVREF_WideRangeIsEnabled ( MY_CMP_CVREF_ID );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_CMP_CVREF_WideRangeIsEnabled (CMP_MODULE_ID index);

PLIB_CMP_CVREF_Disable Function

Disables the voltage reference of the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_Disable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the voltage reference of the Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_Disable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_Disable (CMP_MODULE_ID index)

PLIB_CMP_Disable Function

Disables the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_Disable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_Disable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_Disable (CMP_MODULE_ID index)

PLIB_CMP_Enable Function

Enables the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_Enable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_Enable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_Enable (CMP_MODULE_ID index)

PLIB_CMP_InterruptEventSelect Function

Comparator interrupt event select.

File

plib_cmp.h

C

```
void PLIB_CMP_InterruptEventSelect(CMP_MODULE_ID index, CMP_INTERRUPT_EVENT event);
```

Returns

None.

Description

This function will select when the Comparator interrupt should occur.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsInterruptEventSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// CMP_INTERRUPT_EVENT - CMP_LOW_TO_HIGH
PLIB_CMP_InterruptEventSelect ( MY_CMP_INSTANCE,
                        CMP_INTERRUPT_GENERATION_LOW_TO_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	One of the possible values from CMP_INTERRUPT_EVENT

Function

void PLIB_CMP_InterruptEventSelect (CMP_MODULE_ID index, CMP_INTERRUPT_EVENT event)

PLIB_CMP_InvertingInputChannelSelect Function

Comparator inverting input channel select.

File

plib_cmp.h

C

```
void PLIB_CMP_InvertingInputChannelSelect(CMP_MODULE_ID index, CMP_INVERTING_INPUT channel);
```

Returns

None.

Description

This function will select the inverting input channels for the Comparator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsInvertingInputSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
 PLIB_CMP_InvertingInputChannelSelect ( MY_CMP_INSTANCE,
                            CMP_INVERTING_INPUT_IVREF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from CMP_INVERTING_INPUT

Function

void PLIB_CMP_InvertingInputChannelSelect (CMP_MODULE_ID index,
CMP_INVERTING_INPUT channel)

PLIB_CMP_NonInvertingInputChannelSelect Function

Comparator input channel select.

File

plib_cmp.h

C

```
void PLIB_CMP_NonInvertingInputChannelSelect(CMP_MODULE_ID index, CMP_NON_INVERTING_INPUT input);
```

Returns

None.

Description

This function will select the non-inverting input channels for the Comparator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_CMP_ExistsNonInvertingInputSelect in your application to determine whether this feature is available.s

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_NonInvertingInputChannelSelect ( MY_CMP_INSTANCE, CMP_NON_INVERTING_INPUT_CVREF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from CMP_NON_INVERTING_INPUT

Function

void PLIB_CMP_NonInvertingInputChannelSelect (CMP_MODULE_ID index,
CMP_NON_INVERTING_INPUT input)

PLIB_CMP_OutputDisable Function

Disables the Comparator output.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the Comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputDisable (CMP_MODULE_ID index)

PLIB_CMP_OutputEnable Function

Enables the Comparator output.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the Comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputEnable( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputEnable (CMP_MODULE_ID index)

PLIB_CMP_OutputStatusGet Function

Comparator output status.

File

plib_cmp.h

C

```
bool PLIB_CMP_OutputStatusGet(CMP_MODULE_ID index);
```

Returns

- true - The status flag is set
- false - The status flag is clear

Description

This function will return the current status of the Comparator output.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1
bool cmp_status;

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
```

```
cmp_status=PLIB_CMP_OutputStatusGet ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_CMP_OutputStatusGet (CMP_MODULE_ID index)

PLIB_CMP_OutputInvertDisable Function

Comparator output is non-inverted.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputInvertDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will select the non-inverted comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputInvertDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputInvertDisable (CMP_MODULE_ID index)

PLIB_CMP_OutputInvertEnable Function

Comparator output is inverted.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputInvertEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

Calling this function will set the comparator to make its output inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl in your application to determine whether this feature is available.

Setting this bit will invert the signal to the comparator interrupt generator as well. This will result in an interrupt being generated on the opposite edge from the one selected by PLIB_CMP_InterruptEventSelect function.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputInvertEnable (CMP_MODULE_ID index)

PLIB_CMP_OutputLogicHigh Function

Comparator output bit will give a 'logic high' on satisfying the input condition.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputLogicHigh(CMP_MODULE_ID index);
```

Returns

None.

Description

Calling this API will set the Comparator module to output a 'logic high' on satisfying the input condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputLevelControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputLogicHigh ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputLogicHigh (CMP_MODULE_ID index)

PLIB_CMP_OutputLogicLow Function

Comparator will be set to give 'logic low' on satisfying the input condition.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputLogicLow(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will set the Comparator to give 'logic low' on satisfying the input condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputLevelControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputLogicLow( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputLogicLow (CMP_MODULE_ID index)

b) Comparator Feature Configuration Functions

PLIB_CMP_StopInIdleModeDisable Function

Disables Stop in Idle mode.

File

plib_cmp.h

C

```
void PLIB_CMP_StopInIdleModeDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will continue operation of all enabled comparators when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_StopInIdleModeDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_StopInIdleModeDisable (CMP_MODULE_ID index)

PLIB_CMP_StopInIdleModeEnable Function

Enables Stop in Idle mode.

File

plib_cmp.h

C

```
void PLIB_CMP_StopInIdleModeEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will discontinue operation of all comparators when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_StopInIdleModeEnable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_StopInIdleModeEnable (CMP_MODULE_ID index)

c) Feature Existence Functions

PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect Function

Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFBGRefVoltageRangeSelect feature is supported on the device
- false - The CVREFBGRefVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFBGRefVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_BandGapReferenceSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFEnableControl Function

Identifies whether the CVREFEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFEnableControl feature is supported on the device
- false - The CVREFEnableControl feature is not supported on the device

Description

This function identifies whether the CVREFEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_Enable
- PLIB_CMP_CVREF_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFOutputEnableControl Function

Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFOutputEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFOutputEnableControl feature is supported on the device
- false - The CVREFOutputEnableControl feature is not supported on the device

Description

This function identifies whether the CVREFOutputEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_OutputEnable
- PLIB_CMP_CVREF_OutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFOutputEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFRefVoltageRangeSelect Function

Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFRefVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFRefVoltageRangeSelect feature is supported on the device
- false - The CVREFRefVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFRefVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_ReferenceVoltageSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFRefVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFValueSelect Function

Identifies whether the CVREFValueSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFValueSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFValueSelect feature is supported on the device
- false - The CVREFValueSelect feature is not supported on the device

Description

This function identifies whether the CVREFValueSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_ValueSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFValueSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFVoltageRangeSelect Function

Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFVoltageRangeSelect feature is supported on the device
- false - The CVREFVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_SourceVoltageSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFWideRangeControl Function

Identifies whether the CVREFWideRangeControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFWideRangeControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFWideRangeControl feature is supported on the device
- false - The CVREFWideRangeControl feature is not supported on the device

Description

This function identifies whether the CVREFWideRangeControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_WideRangeEnable
- PLIB_CMP_CVREF_WideRangeDisable
- PLIB_CMP_CVREF_WideRangeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFWideRangeControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsEnableControl Function

Identifies whether the ComparatorEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The ComparatorEnableControl feature is supported on the device
- false - The ComparatorEnableControl feature is not supported on the device

Description

This function identifies whether the ComparatorEnableControl feature is available on the CMP module. When this function returns true, these

functions are supported on the device:

- PLIB_CMP_Enable
- PLIB_CMP_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsInterruptEventSelect Function

Identifies whether the InterruptEventSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInterruptEventSelect(CMP_MODULE_ID index);
```

Returns

- true - The InterruptEventSelect feature is supported on the device
- false - The InterruptEventSelect feature is not supported on the device

Description

This function identifies whether the InterruptEventSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_InterruptEventSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInterruptEventSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsInvertingInputSelect Function

Identifies whether the InvertingInputSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInvertingInputSelect(CMP_MODULE_ID index);
```

Returns

- true - The InvertingInputSelect feature is supported on the device

- false - The InvertingInputSelect feature is not supported on the device

Description

This function identifies whether the InvertingInputSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_InvertingInputChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInvertingInputSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsInvertOutputControl Function

Identifies whether the InvertOutputSelectControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInvertOutputControl(CMP_MODULE_ID index);
```

Returns

- true - The InvertOutputSelectControl feature is supported on the device
- false - The InvertOutputSelectControl feature is not supported on the device

Description

This function identifies whether the InvertOutputSelectControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputInvertEnable
- PLIB_CMP_OutputInvertDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInvertOutputControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsNonInvertingInputSelect Function

Identifies whether the NonInvertingInputSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsNonInvertingInputSelect(CMP_MODULE_ID index);
```

Returns

- true - The NonInvertingInputSelect feature is supported on the device
- false - The NonInvertingInputSelect feature is not supported on the device

Description

This function identifies whether the NonInvertingInputSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_NonInvertingInputChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsNonInvertingInputSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputEnableControl Function

Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The ComparatorOutputEnableControl feature is supported on the device
- false - The ComparatorOutputEnableControl feature is not supported on the device

Description

This function identifies whether the ComparatorOutputEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputEnable
- PLIB_CMP_OutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputLevelControl Function

Identifies whether the OutputLevelSelectControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputLevelControl(CMP_MODULE_ID index);
```

Returns

- true - The OutputLevelSelectControl feature is supported on the device
- false - The OutputLevelSelectControl feature is not supported on the device

Description

This function identifies whether the OutputLevelSelectControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputLogicHigh
- PLIB_CMP_OutputLogicLow

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputLevelControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsStopInIdle(CMP_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_StopInIdleModeEnable
- PLIB_CMP_StopInIdleModeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsStopInIdle(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputStatusGet Function

Identifies whether the OutputStatusGet feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputStatusGet(CMP_MODULE_ID index);
```

Returns

- true - The OutputStatusGet feature is supported on the device
- false - The OutputStatusGet feature is not supported on the device

Description

This function identifies whether the OutputStatusGet feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputStatusGet(CMP_MODULE_ID index)

d) Data Types and Constants

CMP_CLOCK_DIVIDE Enumeration

Defines Comparator Filter Clock Divide.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_FILTER_CLOCK_DIVIDE_1_1,
  CMP_FILTER_CLOCK_DIVIDE_1_2,
  CMP_FILTER_CLOCK_DIVIDE_1_4,
  CMP_FILTER_CLOCK_DIVIDE_1_8,
  CMP_FILTER_CLOCK_DIVIDE_1_16,
  CMP_FILTER_CLOCK_DIVIDE_1_32,
  CMP_FILTER_CLOCK_DIVIDE_1_64,
  CMP_FILTER_CLOCK_DIVIDE_1_128
} CMP_CLOCK_DIVIDE;
```

Members

Members	Description
CMP_FILTER_CLOCK_DIVIDE_1_1	Comparator Filter Clock Division is 1:1
CMP_FILTER_CLOCK_DIVIDE_1_2	Comparator Filter Clock Division is 1:2
CMP_FILTER_CLOCK_DIVIDE_1_4	Comparator Filter Clock Division is 1:4
CMP_FILTER_CLOCK_DIVIDE_1_8	Comparator Filter Clock Division is 1:8
CMP_FILTER_CLOCK_DIVIDE_1_16	Comparator Filter Clock Division is 1:16

CMP_FILTER_CLOCK_DIVIDE_1_32	Comparator Filter Clock Division is 1:32
CMP_FILTER_CLOCK_DIVIDE_1_64	Comparator Filter Clock Division is 1:64
CMP_FILTER_CLOCK_DIVIDE_1_128	Comparator Filter Clock Division is 1:128

Description

Comparator Filter Clock Divide Select

This macro defines the Comparator filter clock divide.

CMP_CVREF_BANDGAP_SELECT Enumeration

Lists the band gap selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_BANDGAP_1_2V,
  CMP_CVREF_BANDGAP_0_6V,
  CMP_CVREF_BANDGAP_VREFPLUS
} CMP_CVREF_BANDGAP_SELECT;
```

Members

Members	Description
CMP_CVREF_BANDGAP_1_2V	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_BANDGAP_0_6V	Select the Band Gap Reference Source as 0.6 V
CMP_CVREF_BANDGAP_VREFPLUS	Select the Band Gap Reference Source as VREF

Description

CVREF band gap select enumeration

This enumeration lists the possible band gap selection options.

Remarks

None.

CMP_CVREF_REFERENCE_SELECT Enumeration

Lists the reference selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_RESISTOR_LADDER_VOLTAGE,
  CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE
} CMP_CVREF_REFERENCE_SELECT;
```

Members

Members	Description
CMP_CVREF_RESISTOR_LADDER_VOLTAGE	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE	Select the Band Gap Reference Source as 0.6 V

Description

CVREF reference select enumeration

This enumeration lists the possible reference selection options.

Remarks

None.

CMP_CVREF_VALUE Enumeration

Lists the voltage reference value selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_VALUE_0,
  CMP_CVREF_VALUE_1,
  CMP_CVREF_VALUE_2,
  CMP_CVREF_VALUE_3,
  CMP_CVREF_VALUE_4,
  CMP_CVREF_VALUE_5,
  CMP_CVREF_VALUE_6,
  CMP_CVREF_VALUE_7,
  CMP_CVREF_VALUE_8,
  CMP_CVREF_VALUE_9,
  CMP_CVREF_VALUE_10,
  CMP_CVREF_VALUE_11,
  CMP_CVREF_VALUE_12,
  CMP_CVREF_VALUE_13,
  CMP_CVREF_VALUE_14,
  CMP_CVREF_VALUE_15
} CMP_CVREF_VALUE;
```

Members

Members	Description
CMP_CVREF_VALUE_0	Voltage reference value 0
CMP_CVREF_VALUE_1	Voltage reference value 1
CMP_CVREF_VALUE_2	Voltage reference value 2
CMP_CVREF_VALUE_3	Voltage reference value 3
CMP_CVREF_VALUE_4	Voltage reference value 4
CMP_CVREF_VALUE_5	Voltage reference value 5
CMP_CVREF_VALUE_6	Voltage reference value 6
CMP_CVREF_VALUE_7	Voltage reference value 7
CMP_CVREF_VALUE_8	Voltage reference value 8
CMP_CVREF_VALUE_9	Voltage reference value 9
CMP_CVREF_VALUE_10	Voltage reference value 10
CMP_CVREF_VALUE_11	Voltage reference value 11
CMP_CVREF_VALUE_12	Voltage reference value 12
CMP_CVREF_VALUE_13	Voltage reference value 13
CMP_CVREF_VALUE_14	Voltage reference value 14
CMP_CVREF_VALUE_15	Voltage reference value 15

Description

CVREF voltage reference value selection enumeration

This enumeration lists the possible Voltage reference value selection options.

Remarks

None.

CMP_CVREF_VOLTAGE_SOURCE Enumeration

Lists the Voltage source options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_VOLTAGE_SOURCE_INTERNAL,
  CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL
} CMP_CVREF_VOLTAGE_SOURCE;
```

Members

Members	Description
CMP_CVREF_VOLTAGE_SOURCE_INTERNAL	Select VDD/VSS source
CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL	Select external voltage source

Description

CVREF Voltage source selection enumeration

This enumeration lists the possible Voltage source selection options.

Remarks

None.

CMP_FILTER_CLOCK Enumeration

Defines Comparator filter input clock

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_FILTER_CLOCK_FP,
  CMP_FILTER_CLOCK_FOSC,
  CMP_FILTER_CLOCK_SYNCO1,
  CMP_FILTER_CLOCK_T2CLK,
  CMP_FILTER_CLOCK_T3CLK,
  CMP_FILTER_CLOCK_T4CLK,
  CMP_FILTER_CLOCK_T5CLK
} CMP_FILTER_CLOCK;
```

Members

Members	Description
CMP_FILTER_CLOCK_FP	FP will be the Comparator filter input clock
CMP_FILTER_CLOCK_FOSC	FOSC will be the Comparator filter input clock
CMP_FILTER_CLOCK_SYNCO1	SYNCO1 will be the Comparator filter input clock
CMP_FILTER_CLOCK_T2CLK	T2CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T3CLK	T3CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T4CLK	T4CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T5CLK	T5CLK will be the Comparator filter input clock

Description

Comparator Filter Input Clock Select

This macro defines the Comparator filter input clock

CMP_INTERRUPT_EVENT Enumeration

Defines Comparator interrupt events.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_INTERRUPT_GENERATION_DISABLED
} CMP_INTERRUPT_EVENT;
```

Members

Members	Description
CMP_INTERRUPT_GENERATION_DISABLED	Select VDD/VSS source

Description

Comparator interrupt event Select

This macro defines the Comparator interrupt events.

CMP_INVERTING_INPUT Enumeration

Defines the list of Comparator inverting Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_INVERTING_INPUT_EXTERNAL_PIN_B,
  CMP_INVERTING_INPUT_EXTERNAL_PIN_C,
  CMP_INVERTING_INPUT_EXTERNAL_PIN_D,
  CMP_INVERTING_INPUT_EXTERNAL_NEGATIVE_PIN,
  CMP_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN,
  CMP_INVERTING_INPUT_OTHER_MODULE_EXTERNAL_POSITIVE_PIN,
  CMP_INVERTING_INPUT_IVREF,
  CMP_INPUT_C2IN_NEGATIVE,
  CMP_INPUT_C2IN_POSITIVE,
  CMP_INPUT_C1IN_POSITIVE,
  CMP_INPUT_IVREF
} CMP_INVERTING_INPUT;
```

Members

Members	Description
CMP_INVERTING_INPUT_EXTERNAL_PIN_B	Select external voltage source at pin CxINB as inverting input. If using CMP_ID_1 , this pin is C1INB
CMP_INVERTING_INPUT_EXTERNAL_PIN_C	Select external voltage source at pin CxINC as inverting input
CMP_INVERTING_INPUT_EXTERNAL_PIN_D	Select external voltage source at pin CxIND as inverting input
CMP_INVERTING_INPUT_EXTERNAL_NEGATIVE_PIN	Select external voltage source at pin CxIN- as inverting input
CMP_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CxIN+ as inverting input. If using CMP_ID_1 , this pin is C1IN+
CMP_INVERTING_INPUT_OTHER_MODULE_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CyIN+ as inverting input. If using CMP_ID_1 , this pin is C2IN+. If using CMP_ID_2 , this pin is C1IN+
CMP_INVERTING_INPUT_IVREF	Select internal voltage ,VDD/VSS source as inverting input
CMP_INPUT_C2IN_NEGATIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_C2IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_C1IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_IVREF	The member is obsolete and maintained here only for the backward compatibility.

Description

Comparator inverting input channel select

This macro defines the super set of Comparator inverting Inputs. Not all options will be available on all microcontrollers. Refer to the processor header for the specific controller in use to determine which options are supported.

CMP_MASK_A Enumeration

Defines Comparator Mask A Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_A_PWM1L,
  CMP_MASK_A_PWM1H,
  CMP_MASK_A_PWM2L,
  CMP_MASK_A_PWM2H,
  CMP_MASK_A_PWM3L,
  CMP_MASK_A_PWM3H,
  CMP_MASK_A_PTGO18,
  CMP_MASK_A_PTGO19,
  CMP_MASK_A_FLT2,
  CMP_MASK_A_FLT4
} CMP_MASK_A;
```

Members

Members	Description
CMP_MASK_A_PWM1L	PWM1L will be the Comparator Mask A input
CMP_MASK_A_PWM1H	PWM1H will be the Comparator Mask A input
CMP_MASK_A_PWM2L	PWM2L will be the Comparator Mask A input
CMP_MASK_A_PWM2H	PWM2H will be the Comparator Mask A input
CMP_MASK_A_PWM3L	PWM3L will be the Comparator Mask A input
CMP_MASK_A_PWM3H	PWM3H will be the Comparator Mask A input
CMP_MASK_A_PTGO18	PTGO18 will be the Comparator Mask A input
CMP_MASK_A_PTGO19	PTGO19 will be the Comparator Mask A input
CMP_MASK_A_FLT2	FLT2 will be the Comparator Mask A input
CMP_MASK_A_FLT4	FLT4 will be the Comparator Mask A input

Description

Comparator Mask A Input Select

This macro defines the Comparator Mask A Input.

CMP_MASK_B Enumeration

Defines Comparator Mask B Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_B_PWM1L,
  CMP_MASK_B_PWM1H,
  CMP_MASK_B_PWM2L,
  CMP_MASK_B_PWM2H,
  CMP_MASK_B_PWM3L,
  CMP_MASK_B_PWM3H,
  CMP_MASK_B_PTGO18,
  CMP_MASK_B_PTGO19,
  CMP_MASK_B_FLT2,
  CMP_MASK_B_FLT4
} CMP_MASK_B;
```

Members

Members	Description
CMP_MASK_B_PWM1L	PWM1L will be the Comparator Mask B input
CMP_MASK_B_PWM1H	PWM1H will be the Comparator Mask B input
CMP_MASK_B_PWM2L	PWM2L will be the Comparator Mask B input
CMP_MASK_B_PWM2H	PWM2H will be the Comparator Mask B input
CMP_MASK_B_PWM3L	PWM3L will be the Comparator Mask B input

CMP_MASK_B_PWM3H	PWM3H will be the Comparator Mask B input
CMP_MASK_B_PTGO18	PTGO18 will be the Comparator Mask B input
CMP_MASK_B_PTGO19	PTGO19 will be the Comparator Mask B input
CMP_MASK_B_FLT2	FLT2 will be the Comparator Mask B input
CMP_MASK_B_FLT4	FLT4 will be the Comparator Mask B input

Description

Comparator Mask B Input Select

This macro defines the Comparator Mask B Input.

CMP_MASK_C Enumeration

Defines Comparator Mask C Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_C_PWM1L,
  CMP_MASK_C_PWM1H,
  CMP_MASK_C_PWM2L,
  CMP_MASK_C_PWM2H,
  CMP_MASK_C_PWM3L,
  CMP_MASK_C_PWM3H,
  CMP_MASK_C_PTGO18,
  CMP_MASK_C_PTGO19,
  CMP_MASK_C_FLT2,
  CMP_MASK_C_FLT4
} CMP_MASK_C;
```

Members

Members	Description
CMP_MASK_C_PWM1L	PWM1L will be the Comparator mask C input
CMP_MASK_C_PWM1H	PWM1H will be the Comparator mask C input
CMP_MASK_C_PWM2L	PWM2L will be the Comparator mask C input
CMP_MASK_C_PWM2H	PWM2H will be the Comparator mask C input
CMP_MASK_C_PWM3L	PWM3L will be the Comparator mask C input
CMP_MASK_C_PWM3H	PWM3H will be the Comparator mask C input
CMP_MASK_C_PTGO18	PTGO18 will be the Comparator mask C input
CMP_MASK_C_PTGO19	PTGO19 will be the Comparator mask C input
CMP_MASK_C_FLT2	FLT2 will be the Comparator mask C input
CMP_MASK_C_FLT4	FLT4 will be the Comparator mask C input

Description

Comparator Mask C Input Select

This macro defines the Comparator Mask C Input.

CMP_MODULE_ID Enumeration

Identifies the Comparator modules supported

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_ID_1,
  CMP_ID_2,
  CMP_ID_3
} CMP_MODULE_ID;
```

Members

Members	Description
CMP_ID_1	Comparator Module 1 ID
CMP_ID_2	Comparator Module 2 ID
CMP_ID_3	Comparator Module 3 ID

Description

Comparator Module ID

This enumeration identifies the Comparator modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

CMP_NON_INVERTING_INPUT Enumeration

Defines the list of Comparator non-inverting Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_NON_INVERTING_INPUT_EXTERNAL_PIN_A,
  CMP_NON_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN,
  CMP_NON_INVERTING_INPUT_CVREF,
  CMP_INPUT_C2IN_POSITIVE,
  CMP_INPUT_INTERNAL_CVREF
} CMP_NON_INVERTING_INPUT;
```

Members

Members	Description
CMP_NON_INVERTING_INPUT_EXTERNAL_PIN_A	Select external voltage source at pin CxINA as non-inverting input
CMP_NON_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CxIN+ as non-inverting input
CMP_NON_INVERTING_INPUT_CVREF	Select internal voltage source CVREF as non-inverting input
CMP_INPUT_C2IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_INTERNAL_CVREF	The member is obsolete and maintained here only for the backward compatibility.

Description

Comparator Non inverting input channel select

This macro defines the super set of Comparator non-inverting Inputs. Not all options will be available on all microcontrollers. Refer to the processor header for the specific controller in use to determine which options are supported.

CMP_SPEED_POWER Enumeration

Defines the Speed/Power of the Comparator

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_SPEED_POWER_LOWER,
  CMP_SPEED_POWER_HIGHER
} CMP_SPEED_POWER;
```

Members

Members	Description
CMP_SPEED_POWER_LOWER	Comparator low-power, low-speed
CMP_SPEED_POWER_HIGHER	Comparator normal power, higher speed

Description

Comparator Speed/Power Select

This macro defines the Speed/Power of the Comparator.

Files

Files

Name	Description
plib_cmp.h	Comparator Peripheral Library Interface Header for Comparator module definitions.
help_plib_cmp.h	

Description

This section lists the source and header files used by the library.

plib_cmp.h

Comparator Peripheral Library Interface Header for Comparator module definitions.

Functions

	Name	Description
	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value, CVref.
	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.
	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
	PLIB_CMP_Disable	Disables the Comparator module.
	PLIB_CMP_Enable	Enables the Comparator module.
	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect	Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the CMP module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the CMP module.

	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputStatusGet	Identifies whether the OutputStatusGet feature exists on the CMP module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CMP module.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.
	PLIB_CMP_OutputInvertDisable	Comparator output is non-inverted.
	PLIB_CMP_OutputInvertEnable	Comparator output is inverted.
	PLIB_CMP_OutputLogicHigh	Comparator output bit will give a 'logic high' on satisfying the input condition.
	PLIB_CMP_OutputLogicLow	Comparator will be set to give 'logic low' on satisfying the input condition.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.
	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

Description

Comparator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Comparator Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Comparator peripheral.

File Name

plib_cmp.h

Company

Microchip Technology Inc.

help_plib_cmp.h

Enumerations

	Name	Description
	CMP_CLOCK_DIVIDE	Defines Comparator Filter Clock Divide.
	CMP_CVREF_BANDGAP_SELECT	Lists the band gap selection options.
	CMP_CVREF_REFERENCE_SELECT	Lists the reference selection options.
	CMP_CVREF_VALUE	Lists the voltage reference value selection options.
	CMP_CVREF_VOLTAGE_SOURCE	Lists the Voltage source options.
	CMP_FILTER_CLOCK	Defines Comparator filter input clock
	CMP_INTERRUPT_EVENT	Defines Comparator interrupt events.
	CMP_INVERTING_INPUT	Defines the list of Comparator inverting Input.
	CMP_MASK_A	Defines Comparator Mask A Input.
	CMP_MASK_B	Defines Comparator Mask B Input.
	CMP_MASK_C	Defines Comparator Mask C Input.
	CMP_MODULE_ID	Identifies the Comparator modules supported
	CMP_NON_INVERTING_INPUT	Defines the list of Comparator non-inverting Input.
	CMP_SPEED_POWER	Defines the Speed/Power of the Comparator

CTMU Peripheral Library

This section describes the Charge Time Measurement Unit (CTMU) Library.

Introduction

Charge Time Measurement Unit (CTMU) for Microchip Microcontrollers

This library provides a low-level abstraction of Charge Time Measurement Unit (CTMU) features. It provides a convenient C language interface, allowing the easy construction of CTMU code that can function across the full range of Microchip devices with CTMU modules.

Description

The CTMU is a flexible analog module that has a configurable current source with a digital configuration circuit built around it. Using its high precision current source the CTMU can support:

- Pulse timing measurements - pulse to pulse, high pulse width, and low pulse width - using an on-chip ADC
- Pulse regeneration with delays
- Temperature measurement (on devices with a temperature diode) using an on-chip ADC
- Capacitance measurement - capacitive touch, and capacitive sensor support - using an on-chip ADC

The CTMU module has the following features:

- On-chip precision current source with four ranges and trimmable output
- Current source controllable by software or external triggers
- Up to 32 input channels supported (ADC inputs)
- Up to 16 sources for triggering
- Level sensitive triggering with edge triggering available on some devices
- Control of triggering polarity and direction
- Two trigger channels with control of trigger sequence
- Time measurement resolution of 1 nanosecond
- Provides ADC capture strobe to automatically initiate ADC voltage measurements

Using the Library

This topic describes the basic architecture of the CTMU Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_can.h`

The interface to the CTMU Peripheral library is defined in the `plib_can.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the CTMU Peripheral library must include `peripheral.h`.

Library File:

The CTMU Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

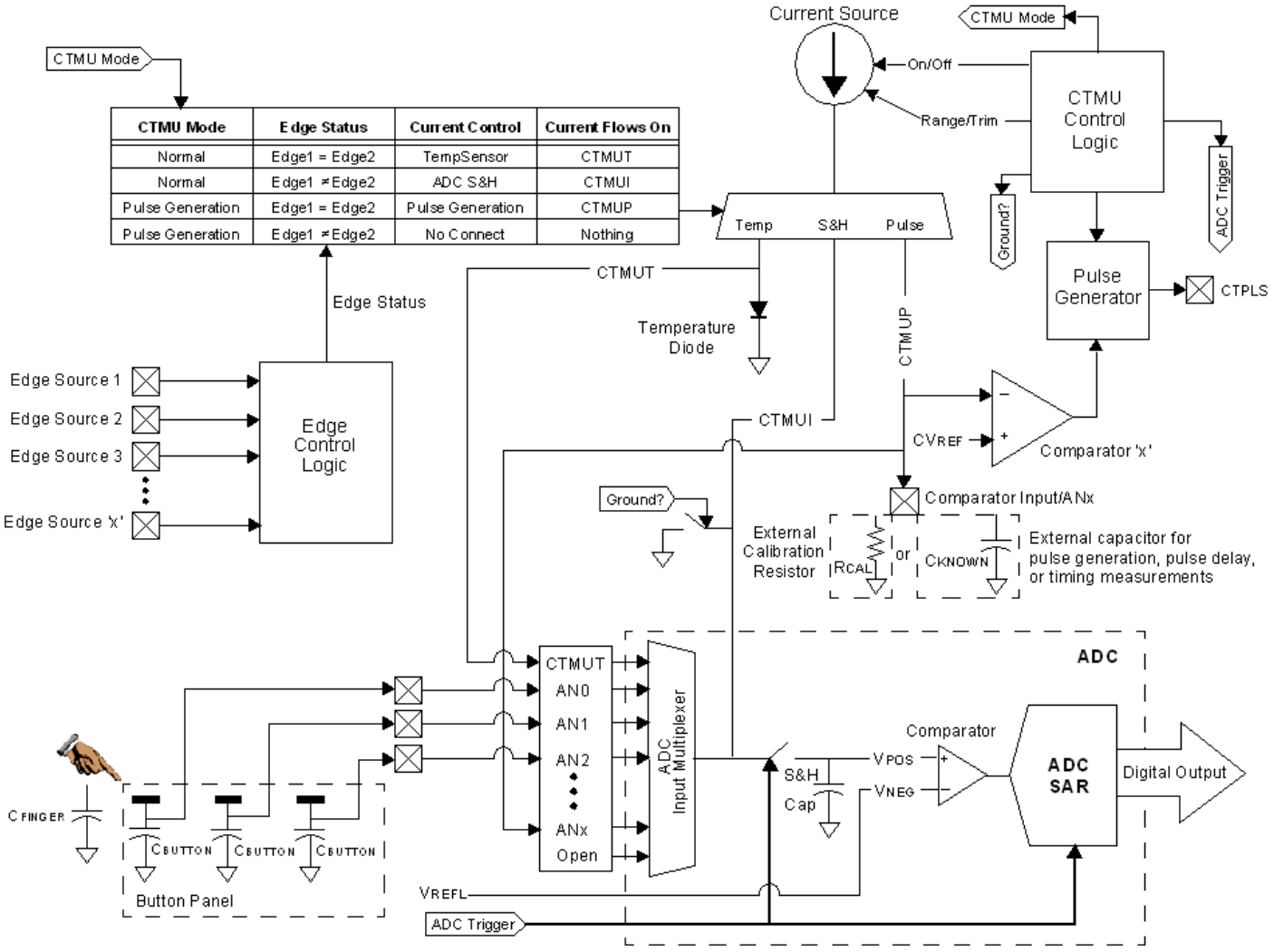
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the CTMU module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure shows a block diagram of the CTMU module, focusing on CTMU function rather than the bits in the interface:



The CTMU is built around a high precision current source (i.e., charge pump) and is closely integrated with the device's on-chip Analog-to-Digital Converter (ADC) module and an on-chip Comparator module. The ADC converts the voltage carried by the Sample and Hold (S&H) capacitor into a digital number representing ADC counts. These counts can be converted into a voltage and the voltage used by the embedded application. The Comparator module associated with the CTMU can compare the voltage on an external capacitor with a known voltage and output when the voltages match. See the Example Applications section, for details of these applications.

In the previous block diagram, an analog multiplexer (MUX) controls where the charge pump sends current. This multiplexer is controlled by the CTMU's mode (Normal versus Pulse Generation) and edge status, as shown by the look-up table. In Normal mode, current is switched between the on-chip temperature sensor (**CTMUT** path) and the ADC's S&H capacitor (**CTMUI** path). In the Pulse Generation mode, current is switched on and off over the **CTMUP** path, with the "on" current going to the negative input of the associated comparator and its special input pin, labeled "Comparator Input/ANx" in the diagram.

External triggers can be used to start and stop the charge pump using up to 16 edge sources. The charge pump is also controlled by manipulating two edge status bits in software.

The block labeled Edge Control Logic determines the edge status based on the selected edge sources and edge control settings. Basically, current is off if the edges are equal, and on if the edges are unequal. In other words, Current (on/off) = Edge 1 XOR Edge 2. Sources can trigger an edge in the control logic based on levels or transitions and can trigger going high-low or low-high. The order of edge events can be controlled, with Edge 1 always occurring before Edge 2, or edge events can occur in any order.

The *Ground?* control signal created by the CTMU Control Logic block enables the grounding of the **CTMUI** path, connecting the charge pump and the ADC's S&H capacitor. This is done before starting any capacitance or timing measurement to make sure there is no residual charge on the ADC's Sample and Hold capacitor and on any external capacitor, such as a button. To discharge the ADC's Sample and Hold capacitor the ADC must be sampling the input voltage (i.e., the switch before the S&H capacitor is closed). This is normally done by manually starting ADC sampling using the "Sampling" bit in one the ADC's control registers.

In Normal mode, current flows from the charge pump to the ADC via the **CTMUI** path. Current flows to external capacitors (such as button capacitors) through the ADC's analog input multiplexer.

The ADC input multiplexer has a special input signal, named "Open", which does not connect anything to the ADC's input. This minimizes stray capacitance and provides the smallest internal capacitance for very fast time measurements.

In Pulse Generation mode, current flows directly to a special comparator/analog input pin via the **CTMUP** path, bypassing the ADC's input

multiplexer and its approximately 2500 ohm series resistance. This extra series resistance is why the charge pump should only be calibrated in Pulse Generation mode, as shown in the block diagram. Except in the lowest current settings, this series resistance causes an error that is too large in the overall calibration resistance value to provide useable charge pump current measurements.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CTMU module.

Library Interface Section	Description
CTMU Control Functions	These functions can be used to enable, disable, and set CTMU features.
Edge Control and Status Functions	These functions can be used to configure CTMU edge control and status.
Feature Existence Functions	These functions can be used to determine whether or not a particular feature is available on the device.

How the Library Works

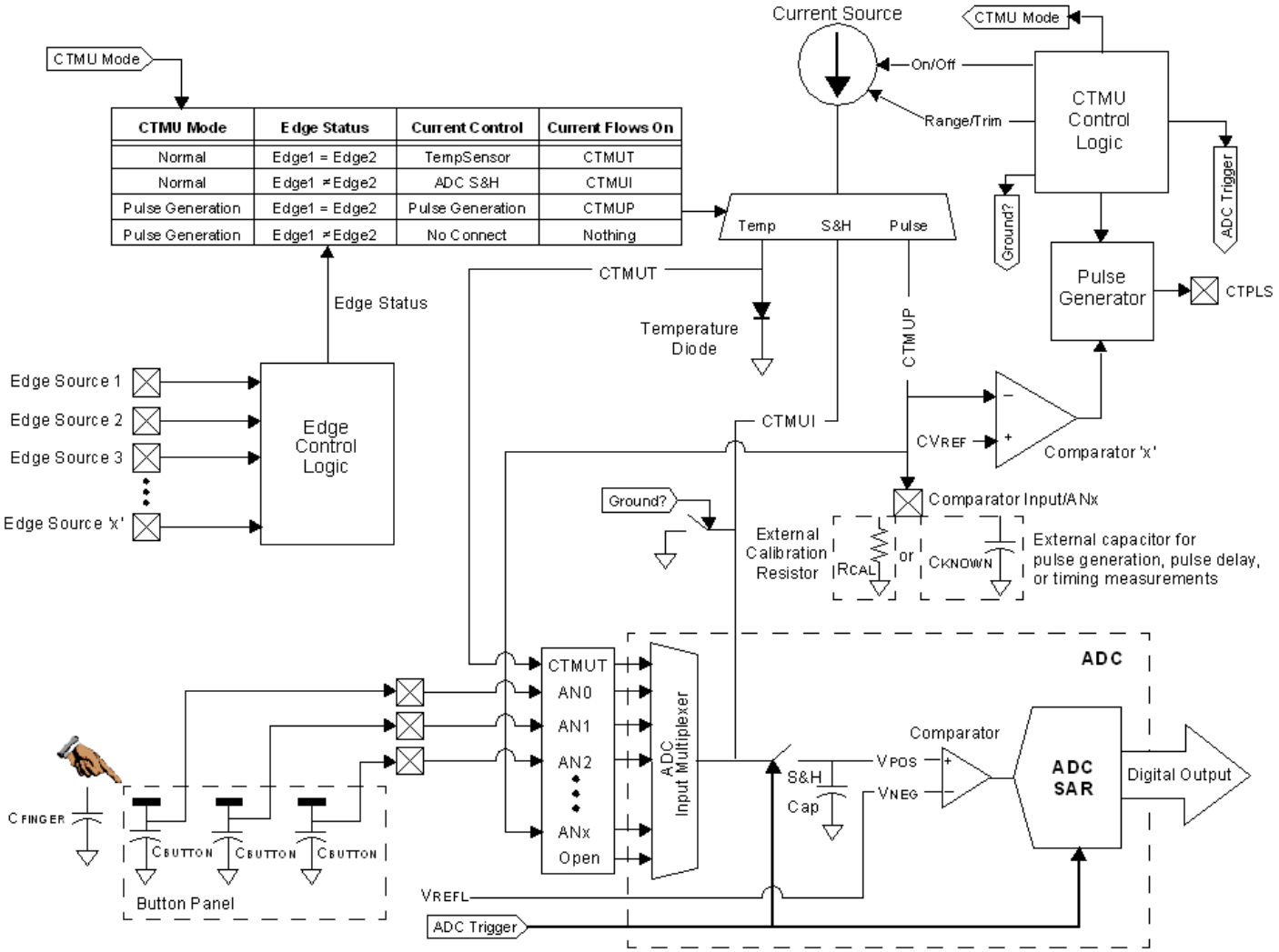
This section describes how the CTMU Peripheral Library works.

Description

The CTMU is a simple analog module, built around a high precision charge pump.

The focus of the CTMU peripheral library is controlling the output of the charge pump and interfacing with the CTMU's associated on-chip ADC.

The block diagram of the CTMU module is repeated here for reference. Next, we will discuss library primitives by functional group, starting with simple things such as turning the module on or off, and then proceeding to more complicated topics such as edge control.



CTMU On/Off and Reset

These features are controlled by these functions:

```
void PLIB_CTMU_Enable ( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_Disable ( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_PORDefaultsRestore ( CTMU_MODULE_ID ctmu_id )
```

The parameter ctmu_id identifies which CTMU module is referenced. Since all currently available devices only have a single CTMU, this parameter is primarily in support of future expansion.

Operation in Idle Mode

The CTMU can stop operating in Idle mode or operate when the chip is in Idle mode depending on:

```
void PLIB_CTMU_StopInIdleEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_StopInIdleDisable( CTMU_MODULE_ID ctmu_id )
```

Current Source

Charge pump output is controlled by a current range, and then a trim off of the nominal current of each of four current ranges.

These functions control current range and current trim:

```
void PLIB_CTMU_CurrentTrimSet( CTMU_MODULE_ID ctmu_id, signed short current_trim )
void PLIB_CTMU_CurrentRangeSet( CTMU_MODULE_ID ctmu_id, CTMU_CURRENT_RANGE current_range )
```

Current is shut off by grounding the charge pump or manipulating the status of two trigger edges, as shown previously. Edge status can be directly controlled in software or determined by two input signals.

The charge pump is grounded or not grounded using:

```
void PLIB_CTMU_CurrentSourceGrounded( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_CurrentSourceNotGrounded( CTMU_MODULE_ID ctmu_id )
```

Edge Control

The CTMU can use from four to 16 different sources to provide on and off triggers for the charge pump.

How a source changes edge status is completely controlled. Edges can trigger on source signals going high-to-low or low-to-high. On some devices, source signal transitions can also trigger an edge. On these devices triggers can occur on for both high-low transitions and low-high transitions.

The CTMU can require an edge sequence, with Edge 1 triggering before Edge 2 triggers, or can respond to either Edge 1 or Edge 2 in any order.

After edge sources have caused the charge pump to start and then stop, edge status must be reset by software before the CTMU can respond to edge sources again.

All CTMU edge control functions start with "PLIB_CTMU_Edge". Each type of control function is discussed in the following section.

Control of the CTMU charge pump by the edges is enabled/disabled by:

```
 void PLIB_CTMU_EdgeEnable(CTMU_MODULE_ID ctmu_id)
 void PLIB_CTMU_EdgeDisable(CTMU_MODULE_ID ctmu_id)
```

Even with edges disabled (the default setting after POR) the charge pump can be controlled in software by setting and clearing the module's edge status bits using:

```
 void PLIB_CTMU_EdgeStatusGotEdgeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
 void PLIB_CTMU_EdgeStatusNoEdgeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

Since there are two edges and the charge pump is turned on when the edge status is unequal, there are two ways of turning the charge pump on/off in software. The first way is to leave one edge set to its POR default (off) and to manipulate the other status. Typically, Edge 1 is modified and Edge 2 is untouched, as shown in this code example:

```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
// TO DO: Charge Delay
PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Stop charging button
```

The other way of turning the charge pump on/off in software mimics what happens when using external sources for edge signals. One edge is set to turn the charge pump on and then the other edge is set to turn the charge pump off:

```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
// TO DO: Charge Delay
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge2 ); // Stop charging button
```

To reset edge status for the next measurement you must clear both edges at the same time:

```
 void PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_MODULE_ID ctmu_id)
```

(To use two calls to PLIB_CTMU_EdgeStatusSetNoEdge would cause the charge pump to pulse between the first call and the second call because in between the status would not be equal, thus enabling the charge pump again.)

Edges can fire based on edges (i.e., transitions) or levels (POR default):

```
 void PLIB_CTMU_EdgeModeEdgeSensitiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
 void PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

You can also determine whether a low-to-high change triggers an edge:

```
 void PLIB_CTMU_EdgePolarityPositiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

or whether a high-to-low change triggers an edge:
```
void PLIB_CTMU_EdgePolarityNegativeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```
If Edge 1 turns on the charge pump and Edge 2 turns it off, you must call:
```
void PLIB_CTMU_EdgeSequenceEnable(CTMU_MODULE_ID ctmu_id)
```
in setting up the CTMU. This feature can be disabled to reinstate the POR default behavior using:
```
void PLIB_CTMU_EdgeSequenceDisable(CTMU_MODULE_ID ctmu_id)
```
With edge sequencing disabled, either edge can be used to control the charge pump.

Finally, the choice of signals for both edges is controlled by:
```
void PLIB_CTMU_EdgeSourceSelect(CTMU_MODULE_ID ctmu_id,
CTMU_EDGE_NUM edge_number,
CTMU_EDGE_SOURCE edge_source)
```

Current Output Control

As shown in the previous block diagram, where the charge pump current goes is controlled by he CTMU mode as well as the edge status.

Whether the CTMU is in capacitance/time or time pulse generation mode is controlled by:
```
void PLIB_CTMU_TimePulseGenEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_TimePulseGenDisable( CTMU_MODULE_ID ctmu_id )
```
As shown in the previous block diagram, when the CTMU is in capacitance/time (normal) mode the charge pump runs when Edge 1 Status != Edge 2 Status and stops when Edge 1 Status = Edge 2 Status. (Actually what happens is the charge pump switches from the ADC S&H capacitor to the on-chip temperature diode (if available), but the effect is the same, charge stops flowing.) In pulse generation mode, the charge pump switches to a "no connect" when Edge 1 Status = Edge 2 Status and is connected to CTMUP when Edge 1 Status != Edge 2 Status.

The CTMU charge pump can be manually turned on by the call:
```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, PLIB_CTMU_Edge1 );
```
and turned off by the call:
```
PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, PLIB_CTMU_Edge1 );
```
assuming that Edge 2 status is "NoEdge".

Interface with the ADC

Absolute and relative capacitance measurements, as well as pulse timing measurements, require using the associated on-chip ADC to measure the resulting voltage across an ADC input pin. ADC conversion can be triggered by a special ADC trigger strobe signal between the CTMU and ADC modules. The ADC trigger strobe fires when the charge pump changes from running to stopped. Temperature measurements by the ADC involve switching the ADC to a special (internal) input pin attached to the voltage diode.

The ADC trigger strobe is controlled by:
```
void PLIB_CTMU_ADCTriggerEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_ADCTriggerDisable( CTMU_MODULE_ID ctmu_id )
```

Power On Reset (POR) Status

The CTMU after a power on reset (POR) has its features configured as follows:
```
PLIB_CTMU_Disable(CTMU_ID_1);              // Turn CTMU off
PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_ID_1,CTMU_Edge1);
PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_ID_1,CTMU_Edge2);
PLIB_CTMU_EdgePolarityNegativeSet(CTMU_ID_1,CTMU_Edge1);
PLIB_CTMU_EdgePolarityNegativeSet(CTMU_ID_1,CTMU_Edge2);
PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_0);
PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);
PLIB_CTMU_StopInIdleDisable(CTMU_ID_1);   // Continue CTMU operation in Idle mode
PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1); // Capacitance/time measurement
PLIB_CTMU_EdgeDisable(CTMU_ID_1);         // Edges are blocked
PLIB_CTMU_EdgeSequenceDisable(CTMU_ID_1); // Edges can fire in any order
PLIB_CTMU_CurrentSourceNotGrounded(CTMU_ID_1);
PLIB_CTMU_ADCTriggerDisable(CTMU_ID_1);
PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,0);
PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_1000xBase);
All of these functions can be accomplished by a single call to
PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);
```

Device-to-Device Variations

Some devices do not have an on-chip temperature diode.

Some devices only support edge triggering based on source signal levels rather than transitions. For these devices the function PLIB_CTMU_EdgeModeSetEdgeSensitive does not apply.

The number of possible edge sources varies from device to device. Some have only four possible sources for Edge 1 and Edge 2. Other devices support up to 16 possible sources.

Refer to the **"Charge Time Measurement Unit (CTMU)**" chapter in the specific device data sheet for more information.

CTMU Setup

This section provides the CTMU setup sequence.

Description

The following sequence is a general guideline used to initialize the CTMU module:

1. Reset the CTMU back to POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the operating mode (Normal or Pulse Generation). (If you are using the Comparator Input/ANx pin, you must use the Pulse Generation Mode; otherwise, use the Normal mode).
5. Clear the Edge Status bits.
6. Optional setups:

- Clear the CTMU interrupt flag.
- Configure the ADC to trigger on the CTMU's strobe.
- Configure button/key circuit pins to be ADC inputs.
- Configure the ADC input multiplexer to point to the first button/key pin.
- Turn on the ADC module.

7. Turn on the CTMU module and wait 50 microseconds for the charge pump to stabilize.
8. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect S&H capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_RestorePORDefaults(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,10); // increase by 20%

// 4. Select the operating mode (Normal or Pulse Generation)
    PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1);

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusSetAllNoEdges(CTMU_ID_1);

// 6. Optional setups
//  Clear CTMU interrupt flag
//  Configure ADC to trigger on CTMU strobe
//  Configure button/key circuit pins to be ADC inputs
//  Configure ADC input multiplexer to point to first button/key pin
//  Turn on the ADC

// 7. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 8. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

General Setup

The following sequence is a general guideline used to initialize the CTMU module:

1. Reset CTMU back to POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the operating mode (Normal or Pulse Generation). (If you are using the Comparator Input/AN*x* pin then you must use the Pulse Generation Mode, otherwise, use the Normal mode).
5. Clear the Edge Status bits.
6. Optional setups:

- Clear CTMU interrupt flag.
- Configure ADC to trigger on CTMU strobe.
- Configure button/key circuit pins to be ADC inputs.
- Configure ADC input multiplexer to point to first button/key pin.
- Turn on the ADC module.

7. Turn on the CTMU module, wait 50 microseconds for charge pump to stabilize.
8. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect S&H capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,10); // increase by 20%

// 4. Select the operating mode (Normal or Pulse Generation)
    PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1);

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);

// 6. Optional setups
//  Clear CTMU interrupt flag
//  Configure ADC to trigger on CTMU strobe
//  Configure button/key circuit pins to be ADC inputs
//  Configure ADC input multiplexer to point to first button/key pin
//  Turn on the ADC

// 7. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 8. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

Measuring Time Between Pulses

This section describes the process for measuring time between pulses.

Description

To measure the time between rising edges of two input signals, first calibrate the CTMU charge pump, using a calibration resistor on the Comparator/ANx input pin. Then replace the resistor with a calibrated capacitor of the correct size for the expected time delay and desired charge pump settings (charge range and trim). Do the following to set up the CTMU for these measurements:

1. Reset the CTMU back to its POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the Pulse Generation generation.
5. Clear the Edge Status bits.
6. Select the input source for each edge.
7. Configure each edge to trigger on a rising edge.
8. Enable edge sequencing so Edge 1 occurs before Edge 2.
9. Enable edges to control the charge pump.
10. Turn on the CTMU module then wait 50 us for charge pump to stabilize.
11. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect sample and hold capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);
```

```
// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,+10); // increase by 20% = +10 * 2%

// 4. Select the operating mode (Pulse Generation)
    PLIB_CTMU_TimePulseGenEnable(CTMU_ID_1); // Use Comparator Input/ANx pin

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);

// 6. Configure the edge input sources for Edge 1 and Edge 2.
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_1);

// 7. Configure the input polarities:
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge1); // Trigger on Rising Edge
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge2); // Trigger on Rising Edge

// 8. Enable edge sequencing so Edge1 (rising) occurs before Edge 2 (rising)
    PLIB_CTMU_EdgeSequenceEnable(CTMU_ID_1);

// 9. Enable edges to control charge pump
    PLIB_CTMU_EdgeEnable(CTMU_ID_1);

// 10. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 11. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

The time between the rising edge of each pulse is given by:

```
Time = ( Capacitance * (VDD * ADC_Count / MaxADCCount) ) / Current
```

Measuring Pulse Width

This section describes the process for measuring pulse width.

Description

Measuring the pulse width of a single input is similar to measuring the time between two pulses. The only difference is that the same signal is used for both edges and Edge 2 is triggered on a falling edge rather than a rising edge.

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,+10); // increase by 20% = +10 * 2%

// 4. Select the operating mode (Pulse Generation)
    PLIB_CTMU_TimePulseGenEnable(CTMU_ID_1); // Use Comparator Input/ANx pin

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusSetAllNoEdges(CTMU_ID_1);

// 6. Use the same input source for both edges.
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_0);

// 7. Configure the input polarities:
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge1); // Trigger on Rising Edge
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge2); // Trigger on falling Edge

// 8. Enable edge sequencing so Edge 1 (rising) occurs before Edge 2 (falling)
    PLIB_CTMU_EdgeSequenceEnable(CTMU_ID_1);
```

```
// 9. Enable edges to control charge pump
    PLIB_CTMU_EdgeEnable(CTMU_ID_1);

// 10. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 11. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

Again, the time between the rising edge of the pulse and the falling edge is given by:

Time = (Capacitance * (VDD * ADC_Count / MaxADCCount)) / Current

Capacitive Touch Measurement

This section describes how to measure the capacitance of a button.

Description

Assuming that the CTMU has been configured as previously described, the following code example shows how to measure the capacitance of a button.

```
#define MAX_ADC_CHANNELS  13
// Mapping of port (A,B,C) and pin (0-15) for each ADC Channel
   unsigned short int PortPinADC[MAX_ADC_CHANNELS] =
                                      { 0xA0, 0xA1, 0xB0, 0xB1,
                                        0xB2, 0xB3, 0xC0, 0xC1,
                                        0xC2, 0xBF, 0xBE, 0xBD,
                                        0xC3 };
    unsigned short int Vpos;  // ADC voltage measurement
    unsigned short int iChan, // ADC channel assigned to each button
                       PortPinChan, // Entry in PortPinADC coding matrix for each button
                       iPort, // Port for each button (0xA,0xB,0xC for ports A,B,C)
                       iPin;  // Pin for each ADC channel (0-15)

    iChan = ScanChannels[button_set_number]; // button_set_number = button to be scanned
    PortPinChan = PortPinADC[iChan];
    iPort   = (PortPinChan>>4) & 0x0F;
    iPin    =  PortPinChan & 0x0F;

    // TO DO: Switch ADC to sensor pin given by iChan

    switch ( iPort )  // Make sure pin is setup for input
    {
        case 0xA:
            // TO DO: Tri-state iPin for Port A
            break;
        case 0xB:
            // TO DO: Tri-state iPin for Port B
            break;
        case 0xC:
            // TO DO: Tri-state iPin for Port C
            break;
    }

    // TO DO: Start ADC sampling to connect sample and hold cap to charge pump
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);    // Ground charge pump
    // TO DO: Wait for discharge of any residual charge from ADC S&H cap and button cap

    PLIB_CTMU_CurrentSourceNotGrounded(CTMU_ID_1); // Unground charge pump
    // TO DO: Disable interrupts to prevent ISRs from corrupting charge timing
    PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
    // TO DO: Wait for charge of button capacitor
    PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, CTMU_Edge1 );  // Stop charging button

 // Start A/D conversion
    // TO DO: Stop ADC sampling to initiate ADC conversion
    // TO DO: Enable interrupts, since timing is no longer critical
```

```
    // TO DO: Wait until ADC conversion complete
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);  // Ground charge pump

//Read new Vpos
    // TO DO: Read ADC results buffer for Vpos
```

Having measured VPOS, the software can now compare it to a limit. If VPOS is less than limit, the button is being pressed by the user.

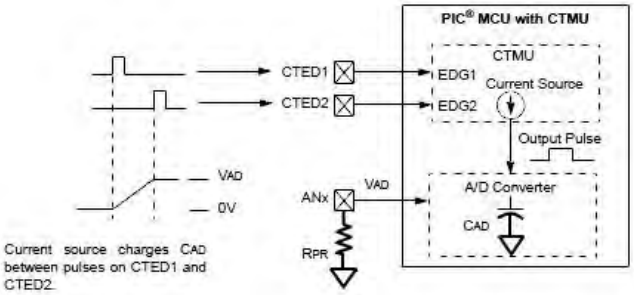
Example Applications

Examples of CTMU applications are discussed.

Description

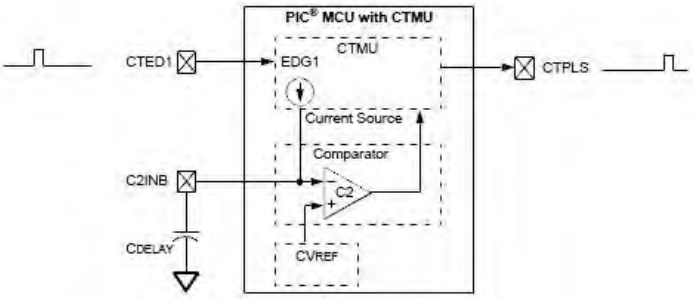
In most applications, the CTMU module provides the current to a circuit and the ADC module measures the resulting voltage. Depending on the setup of the CTMU and the ADC modules, this voltage can represent many things, including:

- On-chip temperature using the built-in temperature diode
- Finger press or swipe on a button panel by measuring relative capacitance
- True or absolute capacitance by measuring capacitance after calibrating the charge pump
- Pulse timing by turning the charge pump on/off with the pulse and then measuring the charge collected by a known capacitor



The CTMU External Edge Input pins, CTED1 and CTED2, are used for connecting two pulse signals to the CTMU. A calibrated capacitor, CKNOWN, is connected to an ADC analog input pin, ANx. The voltage captured by the ADC measures the time difference between the rising edge of the signal on CTED1, which turns on the CTMU's current source, and the rising edge of the signal on CTED2, which turns off the current source.

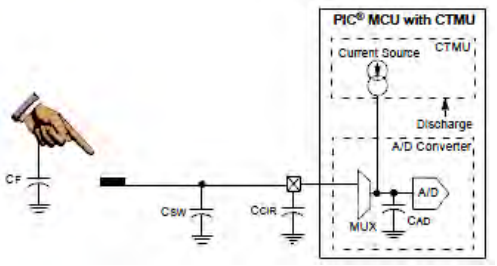
The CTMU and Comparator can be used, without the ADC's help, to delay an input pulse by a variable amount of time, using a known delay capacitor and the charge pumps settings.



In this example, an analog pulse on CTED1 is delayed by a fixed time using a special comparator associated with the CTMU (Comparator 2). Using a calibrated capacitor attached to a special comparator input pin, C2INB, a fixed delay can be added to the analog pulse based on the current and trim settings of the CTMU's charge pump. The time delay is based on the time required to bring the charge voltage on CDELAY up to CVREF. Once there, the comparator's output triggers the start of the output pulse.

Capacitive Touch

Capacitive touch is easy to implement with a CTMU module and its associated on-chip ADC. Since button assertion is a *change* in capacitance, no calibration is required. The software must merely detect the difference between an unpressed button and a button that is pressed. When the user's finger presses a capacitive touch button, the additional capacitance of the finger causes the ADC to see a lower voltage. So software interprets a button voltage drop as a button press by the user.



Configuring the Library

The library is configured for the supported CTMU module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) CTMU Control Functions

	Name	Description
	PLIB_CTMU_CurrentRangeSet	Selects the current source range.
	PLIB_CTMU_Disable	Disables the CTMU module.
	PLIB_CTMU_Enable	Enables the CTMU module.
	PLIB_CTMU_StopInIdleDisable	Sets the CTMU module to continue running when the device is in Idle mode.
	PLIB_CTMU_StopInIdleEnable	Sets the CTMU module to not operate when the device is in Idle mode.
	PLIB_CTMU_AutomaticADCTriggerDisable	Disables the module to automatically trigger an analog-to-digital conversion.
	PLIB_CTMU_AutomaticADCTriggerEnable	Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.
	PLIB_CTMU_CurrentDischargeDisable	Disables the Current discharge by not connecting the current source output to ground.
	PLIB_CTMU_CurrentDischargeEnable	Enables the Current discharge by connecting the current source output to ground.
	PLIB_CTMU_CurrentTrimSet	Trims current source off of the nominal value.
	PLIB_CTMU_TimePulseGenerationDisable	Disables generation of time pulses using Comparator 2.
	PLIB_CTMU_TimePulseGenerationEnable	Enables the generation of time pulses.

b) Edge Control and Status Functions

	Name	Description
	PLIB_CTMU_EdgeDisable	Disables the edges of the CTMU.
	PLIB_CTMU_EdgeEnable	Enables the edges of the CTMU.
	PLIB_CTMU_EdgeSequenceDisable	Disables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSequenceEnable	Enables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeIsSet	Gets the status of a CTMU edge.
	PLIB_CTMU_EdgePolaritySet	Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.
	PLIB_CTMU_EdgeSensitivitySet	Sets CTMU sensitivity for the selected edge.
	PLIB_CTMU_EdgeSet	Sets the status of a CTMU edge.
	PLIB_CTMU_EdgeTriggerSourceSelect	Assigns a trigger source for an edge.

c) Feature Existence Functions

	Name	Description
	PLIB_CTMU_ExistsAutomaticADCTrigger	Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentDischarge	Identifies whether the CurrentDischarge feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentRange	Identifies whether the CurrentRange feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentTrim	Identifies whether the CurrentTrim feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeEnable	Identifies whether the EdgeEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgePolarity	Identifies whether the EdgePolarity feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeSensitivity	Identifies whether the EdgeSensitivity feature exists on the CTMU module,
	PLIB_CTMU_ExistsEdgeSequencing	Identifies whether the EdgeSequencing feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeStatus	Identifies whether the EdgeStatus feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeTriggerSource	Identifies whether the EdgeTriggerSource feature exists on the CTMU module.

	PLIB_CTMU_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CTMU module.
	PLIB_CTMU_ExistsTimePulseGeneration	Identifies whether the TimePulseGeneration feature exists on the CTMU module.

Description

This section describes the Application Programming Interface (API) functions of the CTMU Peripheral Library.

Refer to each section for a detailed description.

a) CTMU Control Functions

PLIB_CTMU_CurrentRangeSet Function

Selects the current source range.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentRangeSet(CTMU_MODULE_ID index, CTMU_CURRENT_RANGE currentRange);
```

Returns

None.

Description

This function selects the current source range.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Select current range
PLIB_CTMU_CurrentRangeSet ( CTMU_ID_0, CTMU_CURRENT_RANGE_BASE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module
currentRange	Charge pump current range selected, one of the possible enumeration values from CTMU_CURRENT_RANGE enum

Function

void PLIB_CTMU_CurrentRangeSet (CTMU_MODULE_ID index, CTMU_CURRENT_RANGE currentRange)

PLIB_CTMU_Disable Function

Disables the CTMU module.

File

plib_ctmu.h

C

```
void PLIB_CTMU_Disable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the CTMU module. Typically, this function is called before reconfiguring the CTMU module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Turn off (disable) CTMU
PLIB_CTMU_Disable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_Disable (CTMU_MODULE_ID index)

PLIB_CTMU_Enable Function

Enables the CTMU module.

File

plib_ctmu.h

C

```
void PLIB_CTMU_Enable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the CTMU module. This function is called after configuring the CTMU module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Turn on (enable) CTMU Module
PLIB_CTMU_Enable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_Enable (CTMU_MODULE_ID index)

PLIB_CTMU_StopInIdleDisable Function

Sets the CTMU module to continue running when the device is in Idle mode.

File

plib_ctmu.h

C

```
void PLIB_CTMU_StopInIdleDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function sets the CTMU module to continue running when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CTMU to run when CPU is in Idle Mode
PLIB_CTMU_StopInIdleDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_StopInIdleDisable (CTMU_MODULE_ID index)

PLIB_CTMU_StopInIdleEnable Function

Sets the CTMU module to not operate when the device is in Idle mode.

File

plib_ctmu.h

C

```
void PLIB_CTMU_StopInIdleEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function sets the CTMU module to not operate when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Do not run in Idle mode
PLIB_CTMU_StopInIdleEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_StopInIdleEnable (CTMU_MODULE_ID index)

PLIB_CTMU_AutomaticADCTriggerDisable Function

Disables the module to automatically trigger an analog-to-digital conversion.

File

plib_ctmu.h

C

```
void PLIB_CTMU_AutomaticADCTriggerDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the module to automatically trigger an analog-to-digital conversion.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsAutomaticADCTrigger in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable ADC trigger from CTMU
PLIB_CTMU_AutomaticADCTriggerDisable ( CTMU_ID_0 );

//To Do: Tell ADC to use CTMU trigger
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_AutomaticADCTriggerDisable (CTMU_MODULE_ID index)

PLIB_CTMU_AutomaticADCTriggerEnable Function

Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.

File

plib_ctmu.h

C

```
void PLIB_CTMU_AutomaticADCTriggerEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.

The ADC trigger is asserted whenever the CTMU current source switches from ON to OFF. The ADC can then end sampling and start converting the measured voltage into bits.

Remarks

The ADC must be configured to use the CTMU trigger signal generated.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsAutomaticADCTrigger in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable ADC trigger from CTMU
PLIB_CTMU_AutomaticADCTriggerEnable ( CTMU_ID_0 );

//To Do: Tell ADC to use CTMU trigger
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_AutomaticADCTriggerEnable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentDischargeDisable Function

Disables the Current discharge by not connecting the current source output to ground.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentDischargeDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the Current discharge by not connecting the current source output to ground.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Measure button voltage
// TO DO: Start ADC sampling to connect S&H capacitor to charge pump
PLIB_CTMU_CurrentDischargeDisable ( CTMU_ID_0 );    // Ground charge pump
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_CurrentDischargeDisable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentDischargeEnable Function

Enables the Current discharge by connecting the current source output to ground.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentDischargeEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the Current discharge by connecting the current source output to ground.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Measure button voltage
// TO DO: Start ADC sampling to connect S&H capacitor to charge pump
PLIB_CTMU_CurrentDischargeEnable ( CTMU_ID_0 );    // Ground charge pump
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_CurrentDischargeEnable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentTrimSet Function

Trims current source off of the nominal value.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentTrimSet(CTMU_MODULE_ID index, int16_t currentTrim);
```

Description

This function trims current source off of the nominal value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentTrim in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set current trim, adjust from nominal by -10 x 2% = -20%
PLIB_CTMU_CurrentTrimSet ( CTMU_ID_0, -10 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module
currentTrim	Current trim index, from -31 to 31

Function

void PLIB_CTMU_CurrentTrimSet (CTMU_MODULE_ID index, int16_t currentTrim)

PLIB_CTMU_TimePulseGenerationDisable Function

Disables generation of time pulses using Comparator 2.

File

plib_ctmu.h

C

```
void PLIB_CTMU_TimePulseGenerationDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables generation of time pulses using Comparator 2.

Current source is made available for other measurements, including capacitance, time, or temperature (if temperature sensor is available).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsTimePulseGeneration in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Return to button relative capacitance measurements
PLIB_CTMU_TimePulseGenerationDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_TimePulseGenerationDisable (CTMU_MODULE_ID index)

PLIB_CTMU_TimePulseGenerationEnable Function

Enables the generation of time pulses.

File

plib_ctmu.h

C

```
void PLIB_CTMU_TimePulseGenerationEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables generation of time pulses using Comparator 2.

Current source will not be available for capacitance, time, or temperature measurements.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsTimePulseGeneration in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Send current to ADC S&H without going through ADC input MUX.
PLIB_CTMU_TimePulseGenerationEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_TimePulseGenerationEnable (CTMU_MODULE_ID index)

b) Edge Control and Status Functions

PLIB_CTMU_EdgeDisable Function

Disables the edges of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the edges of the CTMU (makes them blocked).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable edges, now can only turn CTMU on/off by modifying edge status
PLIB_CTMU_EdgeDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeDisable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeEnable Function

Enables the edges of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the edges of the CTMU (makes it not blocked).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable edges so that input signals can turn the CTMU charge pump on and off
PLIB_CTMU_EdgeEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeEnable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeSequenceDisable Function

Disables the edge sequence of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSequenceDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the edge sequence of the CTMU.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSequencing in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Trigger on Edge1 or Edge2, which ever one occurs first
PLIB_CTMU_EdgeSequenceDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeSequenceDisable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeSequenceEnable Function

Enables the edge sequence of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSequenceEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the edge sequence of the CTMU. Edge 1 must occur before Edge 2 can occur.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSequencing in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Trigger on Edge1 then Edge2, i.e., don't trigger on Edge 2
//until Edge 1 has occurred
PLIB_CTMU_EdgeSequenceEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeSequenceEnable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeIsSet Function

Gets the status of a CTMU edge.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_EdgeIsSet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber);
```

Returns

None.

Description

This function gets the status of the selected CTMU edge (given by edgeNumber).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool edgeStatus;

edgeStatus = PLIB_CTMU_EdgeIsSet ( CTMU_ID_0, CTMU_EDGE1 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.

Function

bool PLIB_CTMU_EdgeIsSet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber)

PLIB_CTMU_EdgePolaritySet Function

Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgePolaritySet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber, CTMU_EDGE_POLARITY
edgePolarity);
```

Returns

None.

Description

This function sets a CTMU edge to fire on an edge/level with the selected polarity of signal.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgePolarity in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Example to set both edges to trigger on positive level/edge.
PLIB_CTMU_EdgePolaritySet ( CTMU_ID_0, CTMU_EDGE1, CTMU_EDGE_X_POSITIVE_EDGE );
PLIB_CTMU_EdgePolaritySet ( CTMU_ID_0, CTMU_EDGE2, CTMU_EDGE_X_POSITIVE_EDGE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the polarity to be set, one of the possible elements of CTMU_EDGE_SELECT enum.
edgeSense	CTMU Edge polarity, one of the possible elements of the CTMU_EDGE_POLARITY enum.

Function

void PLIB_CTMU_EdgePolaritySet (CTMU_MODULE_ID index,CTMU_EDGE_SELECT edgeNumber,

CTMU_EDGE_POLARITY edgePolarity);

PLIB_CTMU_EdgeSensitivitySet Function

Sets CTMU sensitivity for the selected edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSensitivitySet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber, CTMU_EDGE_SENSITIVITY
edgeSense);
```

Returns

None.

Description

This function can be used to set the sensitivity of a particular edge to level-sensitive or edge sensitive.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSensitivity in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Example to set both edges to trigger on edges rather than levels
PLIB_CTMU_EdgeSensitivitySet ( CTMU_ID_0, CTMU_EDGE1, CTMU_EDGE_X_EDGE_SENSITIVE );
PLIB_CTMU_EdgeSensitivitySet ( CTMU_ID_0, CTMU_EDGE2, CTMU_EDGE_X_EDGE_SENSITIVE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU edge for which the sensitivity to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.
edgeSense	CTMU Edge sensitivity, one of the possible elements of the CTMU_EDGE_SENSITIVITY enum.

Function

void PLIB_CTMU_EdgeSensitivitySet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber,
CTMU_EDGE_SENSITIVITY edgeSense)

PLIB_CTMU_EdgeSet Function

Sets the status of a CTMU edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber);
```

Returns

None.

Description

This function sets the status of the selected CTMU edge (given by edge_number).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CTMU_EdgeSet ( CTMU_ID_0, CTMU_EDGE1 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.

Function

void PLIB_CTMU_EdgeSet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber)

PLIB_CTMU_EdgeTriggerSourceSelect Function

Assigns a trigger source for an edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeTriggerSourceSelect(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber,
CTMU_TRIGGER_SOURCES triggerSource);
```

Returns

None.

Description

This function will enable the CTMU edge (given by edge_number) to trigger on the signal provided by the trigger source selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeTriggerSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Set source 0 as the trigger source for both the edges
PLIB_CTMU_EdgeTriggerSourceSelect ( CTMU_ID_0, CTMU_EDGE1, CTMU_TRIGGERSOURCE_0 );
PLIB_CTMU_EdgeTriggerSourceSelect ( CTMU_ID_0, CTMU_EDGE2, CTMU_TRIGGERSOURCE_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.
triggerSource	CTMU Edge trigger source, one of the possible elements of the CTMU_TRIGGER_SOURCES enum.

Function

void PLIB_CTMU_EdgeTriggerSourceSelect (CTMU_MODULE_ID index,

CTMU_EDGE_SELECT edgeNumber,

CTMU_TRIGGER_SOURCES triggerSource)

c) Feature Existence Functions

PLIB_CTMU_ExistsAutomaticADCTrigger Function

Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsAutomaticADCTrigger(CTMU_MODULE_ID index);
```

Returns

- true - The AutomaticADCTrigger feature is supported on the device
- false - The AutomaticADCTrigger feature is not supported on the device

Description

This function identifies whether the AutomaticADCTrigger feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_AutomaticADCTriggerDisable
- PLIB_CTMU_AutomaticADCTriggerEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsAutomaticADCTrigger(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentDischarge Function

Identifies whether the CurrentDischarge feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentDischarge(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentDischarge feature is supported on the device
- false - The CurrentDischarge feature is not supported on the device

Description

This function identifies whether the CurrentDischarge feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_CurrentDischargeEnable
- PLIB_CTMU_CurrentDischargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentDischarge(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentRange Function

Identifies whether the CurrentRange feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentRange(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentRange feature is supported on the device
- false - The CurrentRange feature is not supported on the device

Description

This function identifies whether the CurrentRange feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_CurrentRangeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentRange(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentTrim Function

Identifies whether the CurrentTrim feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentTrim(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentTrim feature is supported on the device
- false - The CurrentTrim feature is not supported on the device

Description

This function identifies whether the CurrentTrim feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_CurrentTrimSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentTrim(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeEnable Function

Identifies whether the EdgeEnable feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeEnable(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeEnable feature is supported on the device
- false - The EdgeEnable feature is not supported on the device

Description

This function identifies whether the EdgeEnable feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeDisable
- PLIB_CTMU_EdgeEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeEnable(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgePolarity Function

Identifies whether the EdgePolarity feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgePolarity(CTMU_MODULE_ID index);
```

Returns

- true - The EdgePolarity feature is supported on the device
- false - The EdgePolarity feature is not supported on the device

Description

This function identifies whether the EdgePolarity feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgePolaritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgePolarity(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeSensitivity Function

Identifies whether the EdgeSensitivity feature exists on the CTMU module,

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeSensitivity(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeSensitivity feature is supported on the device
- false - The EdgeSensitivity feature is not supported on the device

Description

This function identifies whether the EdgeSensitivity feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgeSensitivitySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeSensitivity(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeSequencing Function

Identifies whether the EdgeSequencing feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeSequencing(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeSequencing feature is supported on the device
- false - The EdgeSequencing feature is not supported on the device

Description

This function identifies whether the EdgeSequencing feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeSequenceDisable
- PLIB_CTMU_EdgeSequenceEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeSequencing(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeStatus Function

Identifies whether the EdgeStatus feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeStatus(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeStatus feature is supported on the device
- false - The EdgeStatus feature is not supported on the device

Description

This function identifies whether the EdgeStatus feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeIsSet
- PLIB_CTMU_EdgeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeStatus(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeTriggerSource Function

Identifies whether the EdgeTriggerSource feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeTriggerSource(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeTriggerSource feature is supported on the device
- false - The EdgeTriggerSource feature is not supported on the device

Description

This function identifies whether the EdgeTriggerSource feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgeTriggerSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeTriggerSource(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsModuleEnable Function

Identifies whether the ModuleEnable feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsModuleEnable(CTMU_MODULE_ID index);
```

Returns

- true - The ModuleEnable feature is supported on the device
- false - The ModuleEnable feature is not supported on the device

Description

This function identifies whether the ModuleEnable feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_Disable
- PLIB_CTMU_Enable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsModuleEnable(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsStopInIdle(CTMU_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_StopInIdleDisable
- PLIB_CTMU_StopInIdleEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsStopInIdle(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsTimePulseGeneration Function

Identifies whether the TimePulseGeneration feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsTimePulseGeneration(CTMU_MODULE_ID index);
```

Returns

- true - The TimePulseGeneration feature is supported on the device
- false - The TimePulseGeneration feature is not supported on the device

Description

This function identifies whether the TimePulseGeneration feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_TimePulseGenerationDisable
- PLIB_CTMU_TimePulseGenerationEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsTimePulseGeneration(CTMU_MODULE_ID index)

Files

Files

Name	Description
plib_ctmu.h	This file contains the interface definition for the CTMU Peripheral Library.

Description

This section lists the source and header files used by the library.

plib_ctmu.h

This file contains the interface definition for the CTMU Peripheral Library.

Functions

	Name	Description
	PLIB_CTMU_AutomaticADCTriggerDisable	Disables the module to automatically trigger an analog-to-digital conversion.
	PLIB_CTMU_AutomaticADCTriggerEnable	Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.

	PLIB_CTMU_CurrentDischargeDisable	Disables the Current discharge by not connecting the current source output to ground.
	PLIB_CTMU_CurrentDischargeEnable	Enables the Current discharge by connecting the current source output to ground.
	PLIB_CTMU_CurrentRangeSet	Selects the current source range.
	PLIB_CTMU_CurrentTrimSet	Trims current source off of the nominal value.
	PLIB_CTMU_Disable	Disables the CTMU module.
	PLIB_CTMU_EdgeDisable	Disables the edges of the CTMU.
	PLIB_CTMU_EdgeEnable	Enables the edges of the CTMU.
	PLIB_CTMU_EdgeIsSet	Gets the status of a CTMU edge.
	PLIB_CTMU_EdgePolaritySet	Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.
	PLIB_CTMU_EdgeSensitivitySet	Sets CTMU sensitivity for the selected edge.
	PLIB_CTMU_EdgeSequenceDisable	Disables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSequenceEnable	Enables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSet	Sets the status of a CTMU edge.
	PLIB_CTMU_EdgeTriggerSourceSelect	Assigns a trigger source for an edge.
	PLIB_CTMU_Enable	Enables the CTMU module.
	PLIB_CTMU_ExistsAutomaticADCTrigger	Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentDischarge	Identifies whether the CurrentDischarge feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentRange	Identifies whether the CurrentRange feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentTrim	Identifies whether the CurrentTrim feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeEnable	Identifies whether the EdgeEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgePolarity	Identifies whether the EdgePolarity feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeSensitivity	Identifies whether the EdgeSensitivity feature exists on the CTMU module,
	PLIB_CTMU_ExistsEdgeSequencing	Identifies whether the EdgeSequencing feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeStatus	Identifies whether the EdgeStatus feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeTriggerSource	Identifies whether the EdgeTriggerSource feature exists on the CTMU module.
	PLIB_CTMU_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CTMU module.
	PLIB_CTMU_ExistsTimePulseGeneration	Identifies whether the TimePulseGeneration feature exists on the CTMU module.
	PLIB_CTMU_StopInIdleDisable	Sets the CTMU module to continue running when the device is in Idle mode.
	PLIB_CTMU_StopInIdleEnable	Sets the CTMU module to not operate when the device is in Idle mode.
	PLIB_CTMU_TimePulseGenerationDisable	Disables generation of time pulses using Comparator 2.
	PLIB_CTMU_TimePulseGenerationEnable	Enables the generation of time pulses.

Description

Charge Time Measurement Unit (CTMU) Peripheral Library Header

This library provides a low-level abstraction of the Charge Time Measurement Unit(CTMU) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_ctmu.h

Company

Microchip Technology Inc.

Deadman Timer Peripheral Library

This section describes the Deadman Timer (DMT) Peripheral Library.

Introduction

This library provides a low-level abstraction of the DMT Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Deadman Timer (DMT) is to reset the processor in the event of a software malfunction.

Using the Library

This topic describes the basic architecture of the DMT Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_dmt.h`

The interface to the DMT Peripheral Library is defined in the `plib_dmt.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the DMT Peripheral Library must include `peripheral.h`.

Library File:

The DMT Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the Deadman Timer module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

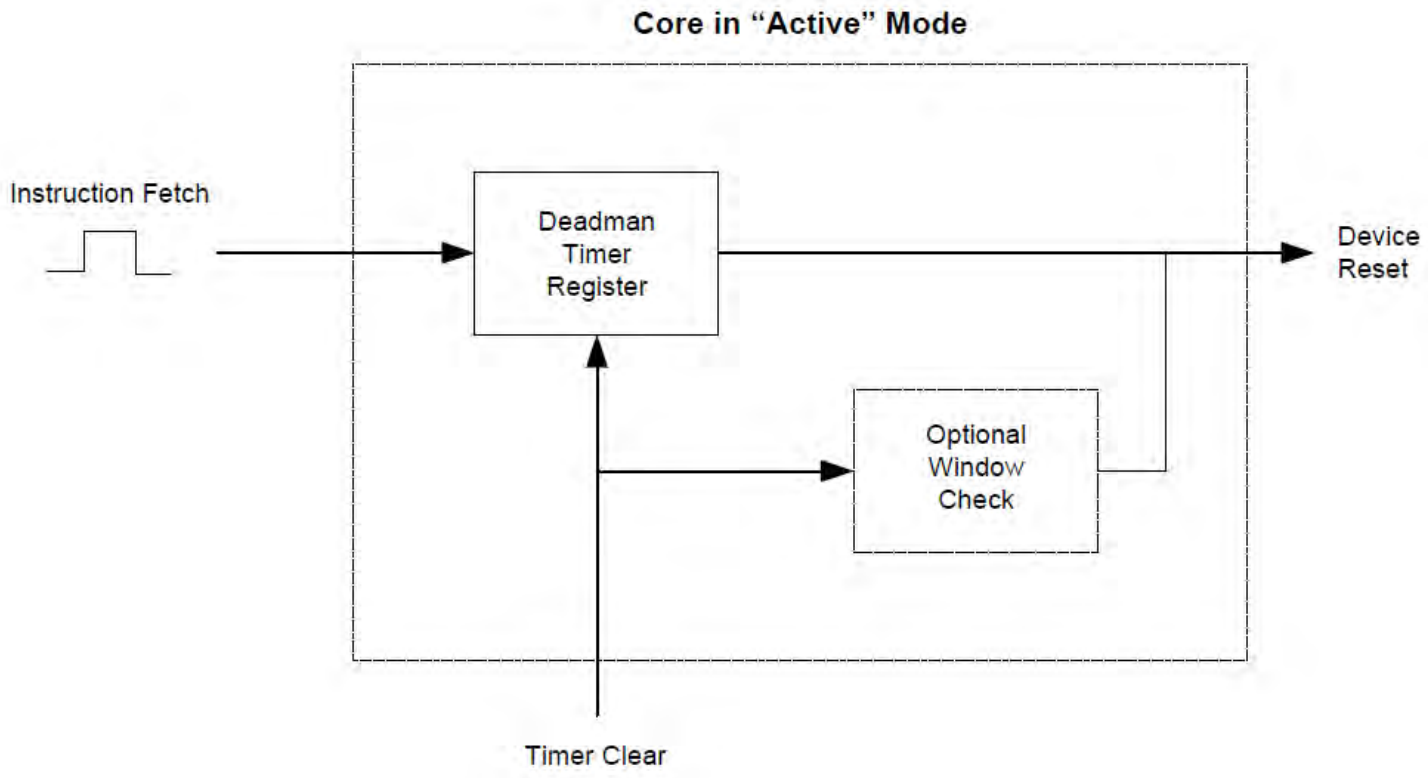
The Deadman Timer is a free-running instruction fetch timer, which is clocked whenever an instruction fetch occurs.

The Deadman Timer is active in the normal mode only. Since the Deadman Timer is only incremented by instruction fetches, the count value will not change when the core is inactive. So, the Deadman Timer remains inactive in Sleep and Idle modes.

The Deadman Timer module uses the Deadman register as a timer. If there is no reset signal from the software and if the counter overflows, this results in the device Reset.

For Windowed mode, resetting the counter when the count is not in the specified window will also lead to the device Reset.

Deadman Timer Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Deadman Timer module.

Library Interface Section	Description
General Configuration Functions	Includes module enable and disable functions.
General Status Functions	Status routines for the DMT.
Feature Existence Functions	Functions that determine existence of certain features.

How the Library Works

This section provides information on how the Deadman Timer Peripheral Library works.

Configuring the Library

The library is configured for the supported Deadman Timer module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DMT_ClearStep1	Resets the DMT module.
	PLIB_DMT_ClearStep2	Resets the DMT module.
	PLIB_DMT_Disable	Disables the DMT module.
	PLIB_DMT_Enable	Enables the DMT module.

b) General Status Functions

	Name	Description
	PLIB_DMT_BAD1Get	Returns BAD1 flag from the DMT Status Register.
	PLIB_DMT_BAD2Get	Returns BAD2 flag from the DMT Status Register.
	PLIB_DMT_CounterGet	Returns the DMT counter value.
	PLIB_DMT_IsEnabled	Returns the Deadman Timer on/off(enable/disable) status.
	PLIB_DMT_PostscalerIntervalGet	Returns the DMT postscaler interval value.
	PLIB_DMT_PostscalerValueGet	Returns the DMT postscaler value.
	PLIB_DMT_WindowIsOpen	Returns Window is Open flag from the DMT Status Register.
	PLIB_DMT_EventOccurred	Returns Event flag from the DMT Status Register.

c) Feature Existence Functions

	Name	Description
	PLIB_DMT_ExistsCounter	Identifies whether the Counter feature exists on the DMT module.
	PLIB_DMT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerInterval	Identifies whether the Postscaler Interval feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerValue	Identifies whether the Postscaler Value feature exists on the DMT module.
	PLIB_DMT_ExistsStatus	Identifies whether the Status feature exists on the DMT module.
	PLIB_DMT_ExistsStep1	Identifies whether the STEP1 Clear feature exists on the DMT module.
	PLIB_DMT_ExistsStep2	Identifies whether the STEP2 Clear feature exists on the DMT module.

d) Data Types and Constants

	Name	Description
	DMT_MODULE_ID	Identifies the supported DMT modules.

Description

This section describes the Application Programming Interface (API) functions of the Deadman Timer Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DMT_ClearStep1 Function

Resets the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_ClearStep1(DMT_MODULE_ID index);
```

Returns

None.

Description

This function performs the STEP1 Clearing of the DMT module. The DMT module should be cleared in two steps, within the interval determined by the Count Window before the DMT forces an NMI or device reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code does not include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );

//Application loop
while(1)
{
    PLIB_DMT_ClearStep1 ( DMT_ID_0 );
    //user code
    PLIB_DMT_ClearStep2 ( DMT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_ClearStep1 (DMT_MODULE_ID index)

PLIB_DMT_ClearStep2 Function

Resets the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_ClearStep2(DMT_MODULE_ID index);
```

Returns

None.

Description

This function performs the STEP2 Clearing of the DMT module. The DMT module should be cleared in two steps, within the interval determined by the Count Window before the DMT forces an NMI or device reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );

//Application loop
while(1)
{
    PLIB_DMT_ClearStep1 ( DMT_ID_0 );
    //user code
    PLIB_DMT_ClearStep2 ( DMT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_ClearStep2 (DMT_MODULE_ID index)

PLIB_DMT_Disable Function

Disables the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_Disable(DMT_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMT module if it is enabled in software.

Remarks

This function will not disable the DMT module if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The DMT module must have been enabled through software.

Example

```
PLIB_DMT_Disable ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_Disable (DMT_MODULE_ID index)

PLIB_DMT_Enable Function

Enables the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_Enable(DMT_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMT module. If it is already enabled through the Configuration bits, it will keep it enabled.

Remarks

Calling this function is not necessary if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_Enable (DMT_MODULE_ID index)

b) General Status Functions

PLIB_DMT_BAD1Get Function

Returns BAD1 flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_BAD1Get(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the DMT BAD1 Flag. This flag is set when there is an incorrect write to STEP1, such as the wrong value, writing too early, or writing too late.

Remarks

The flag returned will indicated if a STEP1 write was not successful.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
if( PLIB_DMT_BAD1Get ( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_BAD1Get (DMT_MODULE_ID index)

PLIB_DMT_BAD2Get Function

Returns BAD2 flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_BAD2Get(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the DMT BAD2 Flag. This flag is set when there is an incorrect write to STEP2, such as the wrong value, writing too early, or writing too late.

Remarks

The flag returned will indicated if a STEP2 write was not successful.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
PLIB_DMT_ClearStep2();
if( PLIB_DMT_BAD2Get ( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_BAD2Get (DMT_MODULE_ID index)

PLIB_DMT_CounterGet Function

Returns the DMT counter value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_CounterGet(DMT_MODULE_ID index);
```

Returns

The counter value.

Description

This function returns the DMT counter value. The value is the number of instructions counted since the count was last cleared.

Remarks

The value returned will be right-aligned.

Refer the data sheet of the specific device to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsCounterValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_CounterGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_CounterGet (DMT_MODULE_ID index)

PLIB_DMT_IsEnabled Function

Returns the Deadman Timer on/off(enable/disable) status.

File

plib_dmt.h

C

```
bool PLIB_DMT_IsEnabled(DMT_MODULE_ID index);
```

Returns

true - If the Deadman Timer is on false - If the Deadman Timer is off

Description

Returns the 'true', if the Deadman Timer is already ON. Otherwise returns 'false'.

Remarks

This function returns 'true' if the device is enabled either though the Configuration bits or in the software.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_DMT_IsEnabled ( DMT_ID_0 ) )
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_IsEnabled (DMT_MODULE_ID index)

PLIB_DMT_PostscalerIntervalGet Function

Returns the DMT postscaler interval value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_PostscalerIntervalGet(DMT_MODULE_ID index);
```

Returns

The postscaler interval.

Description

This function returns the DMT postscaler interval. The value will correspond to a total number of instructions for DMT window, a value determined by configuration bits.

Remarks

The value returned will be right-aligned.

Refer the data sheet of the specific device to get the association of the configuration bits corresponding to this postscaler value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_PostscalerIntervalGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_PostscalerIntervalGet (DMT_MODULE_ID index)

PLIB_DMT_PostscalerValueGet Function

Returns the DMT postscaler value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_PostscalerValueGet(DMT_MODULE_ID index);
```

Returns

The postscaler value.

Description

This function returns the DMT postscaler value. The value will correspond to a total number of instructions for DMT timeout, a value determined by configuration bits.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the association of the configuration bits corresponding to this postscaler value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_PostscalerValueGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_PostscalerValueGet (DMT_MODULE_ID index)

PLIB_DMT_WindowIsOpen Function

Returns Window is Open flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_WindowIsOpen(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the flag indicating the DMT Window is open.

Remarks

The flag returned will indicated if the DMT window is open.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
if( PLIB_DMT_WindowIsOpen( DMT_ID_0 ))
{
   PLIB_DMT_ClearStep2();
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_WindowIsOpen(DMT_MODULE_ID index)

PLIB_DMT_EventOccurred Function

Returns Event flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_EventOccurred(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the flag indicating a DMT event has happened, such as a Window Open, or a Bad Flag is set.

Remarks

The flag returned will indicate if a DMT event has happened.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
if( PLIB_DMT_EventOccurred( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_EventOccurred(DMT_MODULE_ID index)

c) Feature Existence Functions

PLIB_DMT_ExistsCounter Function

Identifies whether the Counter feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsCounter(DMT_MODULE_ID index);
```

Returns

- true - The Counter feature is supported on the device
- false - The Counter feature is not supported on the device

Description

This function identifies whether the Counter feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_CounterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsCounter(DMT_MODULE_ID index)

PLIB_DMT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsEnableControl(DMT_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the DMT module. When this function returns true, these functions are supported on the device:

- PLIB_DMT_Enable
- PLIB_DMT_Disable
- PLIB_DMT_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsEnableControl(DMT_MODULE_ID index)

PLIB_DMT_ExistsPostscalerInterval Function

Identifies whether the Postscaler Interval feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsPostscalerInterval(DMT_MODULE_ID index);
```

Returns

- true - The Postscaler Interval feature is supported on the device
- false - The Postscaler Interval feature is not supported on the device

Description

This function identifies whether the Postscaler Interval feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_PostscalerIntervalGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsInterval(DMT_MODULE_ID index)

PLIB_DMT_ExistsPostscalerValue Function

Identifies whether the Postscaler Value feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsPostscalerValue(DMT_MODULE_ID index);
```

Returns

- true - The Postscaler Value feature is supported on the device
- false - The Postscaler Value feature is not supported on the device

Description

This function identifies whether the PostscalerValue feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_PostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsPostscalerValue(DMT_MODULE_ID index)

PLIB_DMT_ExistsStatus Function

Identifies whether the Status feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStatus(DMT_MODULE_ID index);
```

Returns

Existence of the Status feature:

- true - When Status feature is supported on the device
- false - When Status feature is not supported on the device

Description

This function identifies whether the Status feature is available on the DMT module. When this function returns true, these functions are supported on the device:

- PLIB_DMT_WindowIsOpen
- PLIB_DMT_EventOccurred
- PLIB_DMT_BAD1Get
- PLIB_DMT_BAD2Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStatus(DMT_MODULE_ID index)

PLIB_DMT_ExistsStep1 Function

Identifies whether the STEP1 Clear feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStep1(DMT_MODULE_ID index);
```

Returns

- true - The STEP1 Clear feature is supported on the device
- false - The STEP1 Clear feature is not supported on the device

Description

This function identifies whether the Step 1 Clear feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_ClearStep1

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStep1(DMT_MODULE_ID index)

PLIB_DMT_ExistsStep2 Function

Identifies whether the STEP2 Clear feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStep2(DMT_MODULE_ID index);
```

Returns

- true - The STEP2 Clear feature is supported on the device
- false - The STEP2 Clear feature is not supported on the device

Description

This function identifies whether the STEP2 Clear feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_ClearStep2

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStep2(DMT_MODULE_ID index)

d) Data Types and Constants

DMT_MODULE_ID Enumeration

Identifies the supported DMT modules.

File

plib_dmt_help.h

C

```
typedef enum {
  DMT_ID_0,
  DMT_NUMBER_OF_MODULES
} DMT_MODULE_ID;
```

Members

Members	Description
DMT_ID_0	DMT Module 0 ID
DMT_NUMBER_OF_MODULES	Number of available WDT modules.

Description

DMT Module ID

This enumeration identifies the available DMT modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_dmt.h	Deadman Timer (DMT) Peripheral Library interface header for Deadman Timer common definitions.
plib_dmt_help.h	

Description

This section lists the source and header files used by the library.

plib_dmt.h

Deadman Timer (DMT) Peripheral Library interface header for Deadman Timer common definitions.

Functions

	Name	Description
	PLIB_DMT_BAD1Get	Returns BAD1 flag from the DMT Status Register.
	PLIB_DMT_BAD2Get	Returns BAD2 flag from the DMT Status Register.
	PLIB_DMT_ClearStep1	Resets the DMT module.
	PLIB_DMT_ClearStep2	Resets the DMT module.
	PLIB_DMT_CounterGet	Returns the DMT counter value.
	PLIB_DMT_Disable	Disables the DMT module.
	PLIB_DMT_Enable	Enables the DMT module.
	PLIB_DMT_EventOccurred	Returns Event flag from the DMT Status Register.
	PLIB_DMT_ExistsCounter	Identifies whether the Counter feature exists on the DMT module.
	PLIB_DMT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerInterval	Identifies whether the Postscaler Interval feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerValue	Identifies whether the Postscaler Value feature exists on the DMT module.
	PLIB_DMT_ExistsStatus	Identifies whether the Status feature exists on the DMT module.
	PLIB_DMT_ExistsStep1	Identifies whether the STEP1 Clear feature exists on the DMT module.
	PLIB_DMT_ExistsStep2	Identifies whether the STEP2 Clear feature exists on the DMT module.
	PLIB_DMT_IsEnabled	Returns the Deadman Timer on/off(enable/disable) status.
	PLIB_DMT_PostscalerIntervalGet	Returns the DMT postscaler interval value.
	PLIB_DMT_PostscalerValueGet	Returns the DMT postscaler value.
	PLIB_DMT_WindowIsOpen	Returns Window is Open flag from the DMT Status Register.

Description

Deadman Timer (DMT) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Deadman Timer Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Deadman Timer peripheral.

File Name

plib_dmt.h

Company

Microchip Technology Inc.

plib_dmt_help.h

Enumerations

	Name	Description
	DMT_MODULE_ID	Identifies the supported DMT modules.

Section

Data Types & Constants

Device Control Peripheral Library

This section describes the Device Control Peripheral Library.

Introduction

Device Control Peripheral Library for Microchip microcontrollers.

Description

This library provides a low-level abstraction of the Device Control module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby abstracting hardware differences from one microcontroller variant to another.

Using the Library

This topic describes the basic architecture of the Device Control Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_devcon.h`

The interface to the Device Control Peripheral Library is defined in the `plib_devcon.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Device Control Peripheral Library must include `peripheral.h`.

Library File:

The Device Control Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Device Control module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The Device Control (DEVCON) peripheral library is a collection of functions that may be used by many different modules to perform tasks not specific to any given module. These functions include:

- System Locking and Unlocking
- JTAG, and Trace enable/disable
- Reading of device ID and version
- Other general device configuration settings

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Device Control module.

Library Interface Section	Description
System Functions	APIs that are used for general configuration of the module, such as enabling and disabling the Device Control Peripheral Library.
Data Types and Constants	These data types and constants are required while interacting and configuring the Device Control Peripheral Library.
Feature Existence Functions	These functions identify whether a particular feature exists on the Device Control Peripheral Library.

How the Library Works

Provides information on how the library works.

Description

System Locking an Unlocking

Many operations require the system to be unlocked prior to the operation being performed. This is to prevent catastrophic failure due to an inadvertent register write. These operations include but are not limited to:

- Software reset
- Clock, oscillator and peripheral bus frequency changes
- System bus access permissions
- PPS and PMD control register locking

The following functions provide simple APIs to perform system lock/unlock:

- PLIB_DEVCON_SystemLock, PLIB_DEVCON_SystemUnlock
- PLIB_DEVCON_DeviceRegistersLock, PLIB_DEVCON_DeviceRegistersUnlock (PPS/PMD)

JTAG and Trace

JTAG and Trace are enabled/disabled using the following functions:

- PLIB_DEVCON_JTAGPortEnable, PLIB_DEVCON_JTAGPortEnable
- PLIB_DEVCON_2WireJTAGEnableTDO, PLIB_DEVCON_2WireJTAGDisableTDO
- PLIB_DEVCON_TraceOutputEnable, PLIB_DEVCON_TraceOutputDisable

Device ID and Version Number

The device ID and version number are obtained using the following functions:

- PLIB_DEVCON_DeviceIdGet
- PLIB_DEVCON_DeviceVersionGet

General Device Configuration Settings

The following functions provide general device configuration settings:

- PLIB_DEVCON_AlternateClockEnable, PLIB_DEVCON_AlternateClockDisable
- PLIB_DEVCON_FlashErrCorrectionModeSet
- PLIB_DEVCON_IgnoreDebugFreezeEnable, PLIB_DEVCON_IgnoreDebugFreezeDisable

Configuring the Library

The library is configured for the supported Device Control module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) System Functions

	Name	Description
	PLIB_DEVCON_2WireJTAGDisableTDO	Disables 2-Wire JTAG protocol use of TDO.
	PLIB_DEVCON_2WireJTAGEnableTDO	Enables 2-Wire JTAG protocol to use TDO.
	PLIB_DEVCON_AlternateClockDisable	Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.
	PLIB_DEVCON_AlternateClockEnable	Selects the alternate clock source for Input Capture or Output Compare modules.
	PLIB_DEVCON_DeviceIdGet	Gets the device identifier.
	PLIB_DEVCON_DeviceRegistersLock	Locks device module registers, preventing modifications to the registers.
	PLIB_DEVCON_DeviceRegistersUnlock	Unlocks device module registers, allowing modifications to the registers.
	PLIB_DEVCON_DeviceVersionGet	Gets the device version.
	PLIB_DEVCON_FlashErrCorrectionModeSet	Sets Flash error correction operation.
	PLIB_DEVCON_IgnoreDebugFreezeDisable	Module stops when commanded by debugger.
	PLIB_DEVCON_IgnoreDebugFreezeEnable	Allows module to ignore FREEZE command from debugger and continue running.
	PLIB_DEVCON_JTAGPortDisable	Disables the JTAG port on the device.
	PLIB_DEVCON_JTAGPortEnable	Enables the JTAG port on the device.
	PLIB_DEVCON_SystemLock	Executes the system lock sequence.
	PLIB_DEVCON_SystemUnlock	Executes the system unlock sequence.

	PLIB_DEVCON_TraceOutputDisable	Disables trace outputs and the stop trace clock.
	PLIB_DEVCON_TraceOutputEnable	Enables trace outputs and the start trace clock.
	PLIB_DEVCON_USBPHYSleepModeSet	Selects USB PHY clocking during Sleep mode.
	PLIB_DEVCON_AnalogIOChargePumpDisable	Disables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_AnalogIOChargePumpEnable	Enables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_ExistsAnalogChargePumpControl	Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.
	PLIB_DEVCON_BootExtSelect	Routes SPI0 pins to the PIC32WK pads.
	PLIB_DEVCON_BootIPFSelect	Routes SPI0 pins to In-Package Flash.
	PLIB_DEVCON_HSUARTDisable	Disables High Speed UART.
	PLIB_DEVCON_HSUARTEnable	Enables High Speed UART.
	PLIB_DEVCON_OTPConfigLock	Locks or unlocks the configuration registers.
	PLIB_DEVCON_OTPConfigUnlock	unlocks the configuration registers.

b) MPLL Functions

	Name	Description
	PLIB_DEVCON_MPLLDisable	Disables the MPLL.
	PLIB_DEVCON_MPLLEnable	Enables the MPLL.
	PLIB_DEVCON_MPLLInputDivSet	Sets the MPLL Input Divider bits.
	PLIB_DEVCON_MPLLIsReady	Reads MPLL status.
	PLIB_DEVCON_MPLLMultiplierSet	Sets the MPLL Multiplier bits.
	PLIB_DEVCON_MPLLODiv1Set	Sets the MPLL output divider 1 bits.
	PLIB_DEVCON_MPLLODiv2Set	Sets the MPLL output divider 2 bits.
	PLIB_DEVCON_MPLLVrefSet	Sets the VREF control setting.
	PLIB_DEVCON_MPLLVregDisable	Disables the MPLL VREG.
	PLIB_DEVCON_MPLLVregEnable	Enables the MPLL VREG.
	PLIB_DEVCON_MPLLVregIsReady	Reads the MPLL VREG status.

c) Feature Existence Functions

	Name	Description
	PLIB_DEVCON_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceRegsLockUnlock	Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceVerAndId	Identifies whether the DeviceVerAndId feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsECCModes	Identifies whether the ECCModes feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsIgnoreDebugFreeze	Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGEnable	Identifies whether the JTAGEnable feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGUsesTDO	Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsSystemLockUnlock	Identifies whether the SysLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsTraceOutput	Identifies whether the TraceOutput feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet	Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsMPLL	Identifies whether the MPLL feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsBootSelection	Identifies whether the BootSelection feature exists on the DEVCON module
	PLIB_DEVCON_ExistsHSUARTControl	Identifies whether the HSUARTControl feature exists on the DEVCON module
	PLIB_DEVCON_ExistsOTPConfigLockUnlock	Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

d) Data Types and Constants

	Name	Description
	DEVCON_ALT_CLOCK_TARGET	Selects Input Capture or Output Compare modules.
	DEVCON_DEBUG_PERIPHERAL	Sets modules to ignore debugger's freeze command.
	DEVCON_ECC_CONFIG	Selects how ECC is applied to Flash memory.
	DEVCON_REGISTER_SET	Selects module registers for write lock or unlock.
	DEVCON_USB_SLEEP_MODE	Selects whether the USB clock operates in Sleep mode.
	DEVCON_MODULE_ID	Identifies the supported DEVCON modules.

	DEVCON_MPLL_OUTPUT_DIVIDER	Specifies the MPLL Output divider bits.
	DEVCON_MPLL_VREF_CONTROL	VREF control.

Description

This section describes the Application Programming Interface (API) functions of the Device Control Peripheral Library.
Refer to each section for a detailed description.

a) System Functions

PLIB_DEVCON_2WireJTAGDisableTDO Function

Disables 2-Wire JTAG protocol use of TDO.

File

plib_devcon.h

C

```
void PLIB_DEVCON_2WireJTAGDisableTDO(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables 2-Wire JTAG protocol use of TDO.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_2WireJTAGDisableTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_2WireJTAGEnableTDO Function

Enables 2-Wire JTAG protocol to use TDO.

File

plib_devcon.h

C

```
void PLIB_DEVCON_2WireJTAGEnableTDO(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables 2-Wire JTAG protocol to use TDO.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_2WireJTAGEnableTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_AlternateClockDisable Function

Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AlternateClockDisable(DEVCON_MODULE_ID index, DEVCON_ALT_CLOCK_TARGET targetAltClock);
```

Returns

None.

Description

This function disables the alternate clock source for the Input Capture or Output Compare modules. The primary clock source will be used instead.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_AlternateClockDisable(DEVCON_ID, DEVCON_ALT_CLOCK_IC || DEVCON_ALT_CLOCK_OC );
```

Parameters

Parameters	Description
index	Always DEVCON_ID
targetAltClock	DEVCON_ALT_CLOCK_IC or DEVCON_ALT_CLOCK_OC

Function

void PLIB_DEVCON_AlternateClockDisable(DEVCON_MODULE_ID index,
DEVCON_ALT_CLOCK_TARGET targetAltClock)

PLIB_DEVCON_AlternateClockEnable Function

Selects the alternate clock source for Input Capture or Output Compare modules.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AlternateClockEnable(DEVCON_MODULE_ID index, DEVCON_ALT_CLOCK_TARGET targetAltClock);
```

Returns

None.

Description

This function selects the alternate clock source for the Input Capture or Output Compare modules.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_AlternateClockEnable(DEVCON_ID, DEVCON_ALT_CLOCK_IC || DEVCON_ALT_CLOCK_OC );
```

Parameters

Parameters	Description
index	Always DEVCON_ID
targetAltClock	DEVCON_ALT_CLOCK_IC or DEVCON_ALT_CLOCK_OC

Function

void PLIB_DEVCON_AlternateClockEnable(DEVCON_MODULE_ID index,
DEVCON_ALT_CLOCK_TARGET targetAltClock)

PLIB_DEVCON_DeviceIdGet Function

Gets the device identifier.

File

plib_devcon.h

C

```
uint32_t PLIB_DEVCON_DeviceIdGet(DEVCON_MODULE_ID index);
```

Returns

The version of the device.

Description

This function returns the device identifier.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

uint32_t PLIB_DEVCON_DeviceIdGet(DEVCON_MODULE_ID index)

PLIB_DEVCON_DeviceRegistersLock Function

Locks device module registers, preventing modifications to the registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_DeviceRegistersLock(DEVCON_MODULE_ID index, DEVCON_REGISTER_SET registersToLock);
```

Returns

None.

Description

This function locks device module registers, preventing modifications to the registers.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID,  DEVCON_ALL_REGISTERS);
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID,DEVCON_PPS_REGISTERS);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
registersToLock	element from DEVCON_REGISTER_SET, which can be ORed together

Function

void PLIB_DEVCON_DeviceRegistersLock(DEVCON_MODULE_ID index,
DEVCON_REGISTER_SET registersToLock)

PLIB_DEVCON_DeviceRegistersUnlock Function

Unlocks device module registers, allowing modifications to the registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_MODULE_ID index, DEVCON_REGISTER_SET registersToLock);
```

Returns

None.

Description

This function unlocks device module registers, allowing modifications to the registers.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID,  DEVCON_ALL_REGISTERS);
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID,DEVCON_PPS_REGISTERS);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
registersToLock	element from DEVCON_REGISTER_SET, which can be ORed together

Function

void PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_MODULE_ID index,
DEVCON_REGISTER_SET registersToLock)

PLIB_DEVCON_DeviceVersionGet Function

Gets the device version.

File

plib_devcon.h

C

```
uint8_t PLIB_DEVCON_DeviceVersionGet(DEVCON_MODULE_ID index);
```

Returns

The version of the device.

Description

This functions returns the device version.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

uint8_t PLIB_DEVCON_DeviceVersionGet(DEVCON_MODULE_ID index)

PLIB_DEVCON_FlashErrCorrectionModeSet Function

Sets Flash error correction operation.

File

plib_devcon.h

C

```
void PLIB_DEVCON_FlashErrCorrectionModeSet(DEVCON_MODULE_ID index, DEVCON_ECC_CONFIG flashECC);
```

Returns

None.

Description

This function sets Flash error correction operation.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Once ECC has been locked, it cannot be unlocked except through a system reset.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Always DEVCON_ID

flashECC	DEVCON_ECC_DISABLED, DEVCON_ECC_DISABLED_LOCKED, DEVCON_DYN_ECC_ENABLED_LOCKED, or DEVCON_FLASH_ECC_ENABLED_LOCKED

Function

void PLIB_DEVCON_FlashErrCorrectionModeSet(DEVCON_MODULE_ID index,
DEVCON_ECC_CONFIG flashECC)

PLIB_DEVCON_IgnoreDebugFreezeDisable Function

Module stops when commanded by debugger.

File

plib_devcon.h

C

```
void PLIB_DEVCON_IgnoreDebugFreezeDisable(DEVCON_MODULE_ID index, DEVCON_DEBUG_PERIPHERAL myPeripheral);
```

Returns

None.

Description

This function stops the module when commanded by the debugger.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Peripherals can be ORed together.

Preconditions

None.

Example

```
PLIB_DEVCON_DebugIgnoreFreezeEnable(DEVCON_ID,DEVCON_DEBUG_ALL);
PLIB_DEVCON_DebugIgnoreFreezeDisable(DEVCON_ID,DEVCON_DEBUG_SPI1);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
myPeripheral	DEVCON_DEBUG_USB, DEVCON_DEBUG_UART1, DEVCON_DEBUG_UART2, DEVCON_DEBUG_SPI1, or DEVCON_DEBUG_ALL (for all modules)

Function

void PLIB_DEVCON_IgnoreDebugFreezeDisable(DEVCON_MODULE_ID index,
DEVCON_DEBUG_PERIPHERAL myPeripheral)

PLIB_DEVCON_IgnoreDebugFreezeEnable Function

Allows module to ignore FREEZE command from debugger and continue running.

File

plib_devcon.h

C

```
void PLIB_DEVCON_IgnoreDebugFreezeEnable(DEVCON_MODULE_ID index, DEVCON_DEBUG_PERIPHERAL myPeripheral);
```

Returns

None.

Description

This function allows the module to ignore the FREEZE command from the debugger and continue running.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Peripherals can be ORed together.

Preconditions

None.

Example

```
PLIB_DEVCON_DebugIgnoreFreezeEnable(DEVCON_ID,DEVCON_DEBUG_ALL);
PLIB_DEVCON_DebugIgnoreFreezeDisable(DEVCON_ID,DEVCON_DEBUG_SPI1);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
myPeripheral	DEVCON_DEBUG_USB, DEVCON_DEBUG_UART1, DEVCON_DEBUG_UART2, DEVCON_DEBUG_SPI1, or DEVCON_DEBUG_ALL (for all modules)

Function

void PLIB_DEVCON_IgnoreDebugFreezeEnable(DEVCON_MODULE_ID index,
DEVCON_DEBUG_PERIPHERAL myPeripheral)

PLIB_DEVCON_JTAGPortDisable Function

Disables the JTAG port on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_JTAGPortDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_JTAGPortDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_JTAGPortEnable Function

Enables the JTAG port on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_JTAGPortEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_JTAGPortEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_SystemLock Function

Executes the system lock sequence.

File

plib_devcon.h

C

```
void PLIB_DEVCON_SystemLock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function executes the system lock sequence.

Remarks

Should only be called after PLIB_DEVCON_SystemUnlock and the action that required the unlock have been performed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_SystemLock(DEVCON_MODULE_ID index)

PLIB_DEVCON_SystemUnlock Function

Executes the system unlock sequence.

File

plib_devcon.h

C

```
void PLIB_DEVCON_SystemUnlock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function executes the system unlock sequence.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_SystemUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_TraceOutputDisable Function

Disables trace outputs and the stop trace clock.

File

plib_devcon.h

C

```
void PLIB_DEVCON_TraceOutputDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables trace outputs and the stop trace clock.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_TraceOutputDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_TraceOutputEnable Function

Enables trace outputs and the start trace clock.

File

plib_devcon.h

C

```
void PLIB_DEVCON_TraceOutputEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables trace outputs and the start trace clock (trace probe must be present).

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_TraceOutputEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_USBPHYSleepModeSet Function

Selects USB PHY clocking during Sleep mode.

File

plib_devcon.h

C

```
void PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_MODULE_ID index, DEVCON_USB_SLEEP_MODE sleepOrRun);
```

Returns

None.

Description

This function selects USB PHY clocking during Sleep mode.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_ID,DEVCON_USB_NO_CLOCK_IN_SLEEP)
```

Parameters

Parameters	Description
index	Always DEVCON_ID
sleepOrRun	DEVCON_USB_NO_CLOCK_IN_SLEEP or DEVCON_USB_CLOCK_IN_SLEEP

Function

void PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_MODULE_ID index,
DEVCON_USB_SLEEP_MODE sleepOrRun)

PLIB_DEVCON_AnalogIOChargePumpDisable Function

Disables the I/O Analog Charge Pump on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AnalogIOChargePumpDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the I/O Analog Charge Pump on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_AnalogIOChargePumpDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_AnalogIOChargePumpEnable Function

Enables the I/O Analog Charge Pump on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AnalogIOChargePumpEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the I/O Analog Charge Pump on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_AnalogIOChargePumpEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsAnalogChargePumpControl Function

Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsAnalogChargePumpControl(DEVCON_MODULE_ID index);
```

Returns

- true - The I/O Analog Charge Pump feature feature is supported on the device
- false - The I/O Analog Charge Pump feature feature is not supported on the device

Description

This function identifies whether the I/O Analog Charge Pump feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_AnalogIOChargePumpEnable

- PLIB_DEVCON_AnalogIOChargePumpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsAnalogChargePumpControl(DEVCON_MODULE_ID index)

PLIB_DEVCON_BootExtSelect Function

Routes SPI0 pins to the PIC32WK pads.

File

plib_devcon.h

C

```
void PLIB_DEVCON_BootExtSelect(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function routes the SPI0 pins to the PIC32WK pads.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_BootExtSelect(DEVCON_MODULE_ID index)

PLIB_DEVCON_BootIPFSelect Function

Routes SPI0 pins to In-Package Flash.

File

plib_devcon.h

C

```
void PLIB_DEVCON_BootIPFSelect(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function routes the SPI0 pins to In-Package Flash.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_BootIPFSelect(DEVCON_MODULE_ID index)

PLIB_DEVCON_HSUARTDisable Function

Disables High Speed UART.

File

plib_devcon.h

C

```
void PLIB_DEVCON_HSUARTDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the high speed UART and UART 1 can be configured with PPS.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_HSUARTDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_HSUARTEnable Function

Enables High Speed UART.

File

plib_devcon.h

C

```
void PLIB_DEVCON_HSUARTEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the high speed UART and UART 1 will be using dedicated UART pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_HSUARTEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_OTPConfigLock Function

Locks or unlocks the configuration registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_OTPConfigLock(DEVCON_MODULE_ID index, DEVCON_CFGLOCK lockType);
```

Returns

None.

Description

This function locks or unlocks the configuration register as per the locktype specified.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
lockType	Enumerated config lock value

Function

void PLIB_DEVCON_OTPConfigLock(DEVCON_MODULE_ID index, DEVCON_CFGLOCK lockType)

PLIB_DEVCON_OTPConfigUnlock Function

unlocks the configuration registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_OTPConfigUnlock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the configuration register provided, the CFGLOCK field is not to a value to locked until the reset.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_OTPConfigUnlock(DEVCON_MODULE_ID index)

b) MPLL Functions

PLIB_DEVCON_MPLLDisable Function

Disables the MPLL.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the MPLL.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLEnable Function

Enables the MPLL.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the MPLL.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLInputDivSet Function

Sets the MPLL Input Divider bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLInputDivSet(DEVCON_MODULE_ID index, uint8_t value);
```

Returns

None.

Description

This function sets the MPLL Input Divider bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Multiplier value (between 1 & 63)

Function

PLIB_DEVCON_MPLLInputDivSet(DEVCON_MODULE_ID index, uint8_t value)

PLIB_DEVCON_MPLLIsReady Function

Reads MPLL status.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_MPLLIsReady(DEVCON_MODULE_ID index);
```

Returns

- true - MPLL clock is stable
- false - MPLL clock is not stable

Description

This function reads MPLL status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLIsReady(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLMultiplierSet Function

Sets the MPLL Multiplier bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLMultiplierSet(DEVCON_MODULE_ID index, uint8_t value);
```

Returns

None.

Description

This function sets the MPLL Multiplier bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Multiplier value (between 16 & 160)

Function

PLIB_DEVCON_MPLLMultiplierSet(DEVCON_MODULE_ID index, uint8_t value)

PLIB_DEVCON_MPLLODiv1Set Function

Sets the MPLL output divider 1 bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLODiv1Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits);
```

Returns

None.

Description

This function sets the MPLL output divider 1 bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
bits	Enumerated divider value

Function

PLIB_DEVCON_MPLLODiv1Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits)

PLIB_DEVCON_MPLLODiv2Set Function

Sets the MPLL output divider 2 bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLODiv2Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits);
```

Returns

None.

Description

This function sets the MPLL output divider 2 bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
bits	Enumerated divider value

Function

PLIB_DEVCON_MPLLODiv2Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits)

PLIB_DEVCON_MPLLVrefSet Function

Sets the VREF control setting.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVrefSet(DEVCON_MODULE_ID index, DEVCON_MPLL_VREF_CONTROL vref);
```

Returns

None.

Description

This function sets the VREF control setting.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Enumerated VREF Control setting

Function

PLIB_DEVCON_MPLLVrefSet(DEVCON_MODULE_ID index, DEVCON_MPLL_VREF_CONTROL vref)

PLIB_DEVCON_MPLLVregDisable Function

Disables the MPLL VREG.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVregDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the MPLL VREG.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLVregEnable Function

Enables the MPLL VREG.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVregEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the MPLL VREG.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLVregIsReady Function

Reads the MPLL VREG status.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_MPLLVregIsReady(DEVCON_MODULE_ID index);
```

Returns

- true - MPLL VREG is ready
- false - MPLL VREG is not ready

Description

This function reads the MPLL VREG status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregIsReady(DEVCON_MODULE_ID index)

c) Feature Existence Functions

PLIB_DEVCON_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsAlternateClock(DEVCON_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_AlternateClockEnable
- PLIB_DEVCON_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsAlternateClock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsDeviceRegsLockUnlock Function

Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsDeviceRegsLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The DeviceRegsLockUnlock feature is supported on the device
- false - The DeviceRegsLockUnlock feature is not supported on the device

Description

This function identifies whether the DeviceRegsLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DeviceRegistersLock
- PLIB_DEVCON_DeviceRegistersUnlock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsDeviceRegsLockUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsDeviceVerAndId Function

Identifies whether the DeviceVerAndId feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsDeviceVerAndId(DEVCON_MODULE_ID index);
```

Returns

- true - The DeviceVerAndId feature is supported on the device
- false - The DeviceVerAndId feature is not supported on the device

Description

This function identifies whether the DeviceVerAndId feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DeviceVersionGet
- PLIB_DEVCON_DeviceIdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsDeviceVerAndId(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsECCModes Function

Identifies whether the ECCModes feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsECCModes(DEVCON_MODULE_ID index);
```

Returns

- true - The ECCModes feature is supported on the device
- false - The ECCModes feature is not supported on the device

Description

This function identifies whether the ECCModes feature is available on the DEVCON module. When this function returns true, this function is supported on the device:

- PLIB_DEVCON_FlashErrCorrectionModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsECCModes(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsIgnoreDebugFreeze Function

Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsIgnoreDebugFreeze(DEVCON_MODULE_ID index);
```

Returns

- true - The IgnoreDebugFreeze feature is supported on the device
- false - The IgnoreDebugFreeze feature is not supported on the device

Description

This function identifies whether the IgnoreDebugFreeze feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DebugIgnoreFreezeEnable
- PLIB_DEVCON_IgnoreDebugFreezeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsIgnoreDebugFreeze(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsJTAGEnable Function

Identifies whether the JTAGEnable feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsJTAGEnable(DEVCON_MODULE_ID index);
```

Returns

- true - The JTAGEnable feature is supported on the device
- false - The JTAGEnable feature is not supported on the device

Description

This function identifies whether the JTAGEnable feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_JTAGPortEnable
- PLIB_DEVCON_JTAGPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsJTAGEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsJTAGUsesTDO Function

Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsJTAGUsesTDO(DEVCON_MODULE_ID index);
```

Returns

- true - The JTAGUsesTDO feature is supported on the device
- false - The JTAGUsesTDO feature is not supported on the device

Description

This function identifies whether the JTAGUsesTDO feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_2WireJTAGEnableTDO
- PLIB_DEVCON_2WireJTAGDisableTDO

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsJTAGUsesTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsSystemLockUnlock Function

Identifies whether the SysLockUnlock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsSystemLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The SysLockUnlock feature is supported on the device
- false - The SysLockUnlock feature is not supported on the device

Description

This function identifies whether the SysLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_SystemUnlock
- PLIB_DEVCON_SystemLock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsSystemLockUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsTraceOutput Function

Identifies whether the TraceOutput feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsTraceOutput(DEVCON_MODULE_ID index);
```

Returns

- true - The TraceOutput feature is supported on the device
- false - The TraceOutput feature is not supported on the device

Description

This function identifies whether the TraceOutput feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_TraceOutputEnable
- PLIB_DEVCON_TraceOutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsTraceOutput(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet Function

Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet(DEVCON_MODULE_ID index);
```

Returns

- true - The USB_PHY_SleepModeSet feature is supported on the device
- false - The USB_PHY_SleepModeSet feature is not supported on the device

Description

This function identifies whether the USB_PHY_SleepModeSet feature is available on the DEVCON module. When this function returns true, this function is supported on the device:

- PLIB_DEVCON_USBPHYSleepModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsMPLL Function

Identifies whether the MPLL feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsMPLL(DEVCON_MODULE_ID index);
```

Returns

- true - The MPLL feature is supported on the device
- false - The MPLL feature is not supported on the device

Description

This function identifies whether the MPLL feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_MPLLIsReady
- PLIB_DEVCON_MPLLEnable
- PLIB_DEVCON_MPLLDisable
- PLIB_DEVCON_MPLLODiv1Set
- PLIB_DEVCON_MPLLODiv2Set
- PLIB_DEVCON_MPLLVregIsReady
- PLIB_DEVCON_MPLLVregEnable
- PLIB_DEVCON_MPLLVregDisable
- PLIB_DEVCON_MPLLMultiplierSet
- PLIB_DEVCON_MPLLInputDivSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsMPLL(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsBootSelection Function

Identifies whether the BootSelection feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsBootSelection(DEVCON_MODULE_ID index);
```

Returns

- true - The BootSelection feature is supported on the device

- false - The BootSelection feature is not supported on the device

Description

This function identifies whether the BootSelection feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_BootIPFSelect
- PLIB_DEVCON_BootExtSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsBootSelection(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsHSUARTControl Function

Identifies whether the HSUARTControl feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsHSUARTControl(DEVCON_MODULE_ID index);
```

Returns

- true - The HSUARTControl feature is supported on the device
- false - The HSUARTControl feature is not supported on the device

Description

This function identifies whether the HSUARTControl feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_HSUARTEnable
- PLIB_DEVCON_HSUARTDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsHSUARTControl(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsOTPConfigLockUnlock Function

Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsOTPConfigLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The OTPConfigLockUnlock feature is supported on the device
- false - The OTPConfigLockUnlock feature is not supported on the device

Description

This function identifies whether the OTPConfigLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_OTPConfigLock
- PLIB_DEVCON_OTPConfigUnlock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsOTPConfigLockUnlock(DEVCON_MODULE_ID index)

d) Data Types and Constants

DEVCON_ALT_CLOCK_TARGET Enumeration

Selects Input Capture or Output Compare modules.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_ALT_CLOCK_IC,
  DEVCON_ALT_CLOCK_OC
} DEVCON_ALT_CLOCK_TARGET;
```

Members

Members	Description
DEVCON_ALT_CLOCK_IC	Input Capture
DEVCON_ALT_CLOCK_OC	Output Compare

Description

Alternate Clock Targets Enumeration

This enumeration selects the Input Capture or Output Compare module for enabling or disabling the alternate clock.

Remarks

None.

DEVCON_DEBUG_PERIPHERAL Enumeration

Sets modules to ignore debugger's freeze command.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_DEBUG_USB,
  DEVCON_DEBUG_UART1,
  DEVCON_DEBUG_UART2,
  DEVCON_DEBUG_SPI1,
  DEVCON_DEBUG_ALL
} DEVCON_DEBUG_PERIPHERAL;
```

Members

Members	Description
DEVCON_DEBUG_USB	USB module
DEVCON_DEBUG_UART1	UART 1
DEVCON_DEBUG_UART2	UART 2
DEVCON_DEBUG_SPI1	SPI 1
DEVCON_DEBUG_ALL	USB, UART 1, UART 2, and SPI 1

Description

Ignore Debugger Freeze for Peripheral Module(s) enumeration

This enumeration sets modules to ignore the debugger's freeze command.

Remarks

None.

DEVCON_ECC_CONFIG Enumeration

Selects how ECC is applied to Flash memory.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_ECC_DISABLED,
  DEVCON_ECC_DISABLED_LOCKED,
  DEVCON_DYN_ECC_ENABLED_LOCKED,
  DEVCON_FLASH_ECC_ENABLED_LOCKED
} DEVCON_ECC_CONFIG;
```

Members

Members	Description
DEVCON_ECC_DISABLED	ECC and dynamic ECC are disabled
DEVCON_ECC_DISABLED_LOCKED	ECC and dynamic ECC are disabled and the configuration locked
DEVCON_DYN_ECC_ENABLED_LOCKED	Dynamic Flash ECC is enabled and the configuration is locked
DEVCON_FLASH_ECC_ENABLED_LOCKED	Flash ECC is enabled, the configuration is locked, word Flash writes disabled

Description

Flash Error Correcting Code (ECC) Configuration enumeration

This enumeration selects how ECC is applied to Flash memory.

Remarks

None.

DEVCON_REGISTER_SET Enumeration

Selects module registers for write lock or unlock.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_PPS_REGISTERS,
  DEVCON_PMD_REGISTERS,
  DEVCON_PERMISSION_GROUP_REGISTERS,
  DEVCON_ALL_REGISTERS
} DEVCON_REGISTER_SET;
```

Members

Members	Description
DEVCON_PPS_REGISTERS	Peripheral Pin Select registers
DEVCON_PMD_REGISTERS	Peripheral Module Disable registers
DEVCON_PERMISSION_GROUP_REGISTERS	Permission Group registers
DEVCON_ALL_REGISTERS	All lockable registers

Description

Module Registers for Lock/Unlock enumeration.

This enumeration selects the module registers for write lock or unlock.

Remarks

Can be ORed together.

DEVCON_USB_SLEEP_MODE Enumeration

Selects whether the USB clock operates in Sleep mode.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_USB_NO_CLOCK_IN_SLEEP,
  DEVCON_USB_CLOCK_IN_SLEEP
} DEVCON_USB_SLEEP_MODE;
```

Members

Members	Description
DEVCON_USB_NO_CLOCK_IN_SLEEP	USB PHY clock is shut down when Sleep mode is active.
DEVCON_USB_CLOCK_IN_SLEEP	USP PHY clock continues to run when Sleep mode is active

Description

USB Clock In Sleep Mode Enumeration

This enumeration selects whether the USB clock operates in Sleep mode.

Remarks

None.

DEVCON_MODULE_ID Enumeration

Identifies the supported DEVCON modules.

File

help_plib_devcon.h

C

```
typedef enum {
  DEVCON_ID_0,
  DEVCON_NUMBER_OF_MODULES
} DEVCON_MODULE_ID;
```

Members

Members	Description
DEVCON_ID_0	DEVCON Module 0 ID
DEVCON_NUMBER_OF_MODULES	Number of available DEVCON modules.

Description

DEVCON Module ID

This enumeration identifies the DEVCON modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

DEVCON_MPLL_OUTPUT_DIVIDER Enumeration

Specifies the MPLL Output divider bits.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_MPLL_ODIV_1,
  DEVCON_MPLL_ODIV_2,
  DEVCON_MPLL_ODIV_3,
  DEVCON_MPLL_ODIV_4,
  DEVCON_MPLL_ODIV_5,
  DEVCON_MPLL_ODIV_6,
  DEVCON_MPLL_ODIV_7
} DEVCON_MPLL_OUTPUT_DIVIDER;
```

Description

MPLL Output Divider bits enumeration

This enumeration specifies the MPLL Output divider bits.

Remarks

None.

DEVCON_MPLL_VREF_CONTROL Enumeration

VREF control.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_MPLL_VREF_EXT,
  DEVCON_MPLL_VREF_VDD,
  DEVCON_MPLL_VREF_VSS,
  DEVCON_MPLL_VREF_INT
} DEVCON_MPLL_VREF_CONTROL;
```

Description

MPLL VREF Control enumeration

This enumeration provides VREF control.

Remarks

None.

Files

Files

Name	Description
plib_devcon.h	Defines the Device Control Peripheral Library Interface.
help_plib_devcon.h	This is file help_plib_devcon.h.

Description

This section lists the source and header files used by the library.

plib_devcon.h

Defines the Device Control Peripheral Library Interface.

Enumerations

	Name	Description
	DEVCON_ALT_CLOCK_TARGET	Selects Input Capture or Output Compare modules.
	DEVCON_DEBUG_PERIPHERAL	Sets modules to ignore debugger's freeze command.
	DEVCON_ECC_CONFIG	Selects how ECC is applied to Flash memory.
	DEVCON_MPLL_OUTPUT_DIVIDER	Specifies the MPLL Output divider bits.
	DEVCON_MPLL_VREF_CONTROL	VREF control.
	DEVCON_REGISTER_SET	Selects module registers for write lock or unlock.
	DEVCON_USB_SLEEP_MODE	Selects whether the USB clock operates in Sleep mode.

Functions

	Name	Description
	PLIB_DEVCON_2WireJTAGDisableTDO	Disables 2-Wire JTAG protocol use of TDO.
	PLIB_DEVCON_2WireJTAGEnableTDO	Enables 2-Wire JTAG protocol to use TDO.
	PLIB_DEVCON_AlternateClockDisable	Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.
	PLIB_DEVCON_AlternateClockEnable	Selects the alternate clock source for Input Capture or Output Compare modules.
	PLIB_DEVCON_AnalogIOChargePumpDisable	Disables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_AnalogIOChargePumpEnable	Enables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_BootExtSelect	Routes SPI0 pins to the PIC32WK pads.
	PLIB_DEVCON_BootIPFSelect	Routes SPI0 pins to In-Package Flash.
	PLIB_DEVCON_DeviceIdGet	Gets the device identifier.
	PLIB_DEVCON_DeviceRegistersLock	Locks device module registers, preventing modifications to the registers.
	PLIB_DEVCON_DeviceRegistersUnlock	Unlocks device module registers, allowing modifications to the registers.
	PLIB_DEVCON_DeviceVersionGet	Gets the device version.
	PLIB_DEVCON_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsAnalogChargePumpControl	Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsBootSelection	Identifies whether the BootSelection feature exists on the DEVCON module
	PLIB_DEVCON_ExistsDeviceRegsLockUnlock	Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceVerAndId	Identifies whether the DeviceVerAndId feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsECCModes	Identifies whether the ECCModes feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsHSUARTControl	Identifies whether the HSUARTControl feature exists on the DEVCON module
	PLIB_DEVCON_ExistsIgnoreDebugFreeze	Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGEnable	Identifies whether the JTAGEnable feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGUsesTDO	Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsMPLL	Identifies whether the MPLL feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsOTPConfigLockUnlock	Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

	PLIB_DEVCON_ExistsSystemLockUnlock	Identifies whether the SysLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsTraceOutput	Identifies whether the TraceOutput feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet	Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.
	PLIB_DEVCON_FlashErrCorrectionModeSet	Sets Flash error correction operation.
	PLIB_DEVCON_HSUARTDisable	Disables High Speed UART.
	PLIB_DEVCON_HSUARTEnable	Enables High Speed UART.
	PLIB_DEVCON_IgnoreDebugFreezeDisable	Module stops when commanded by debugger.
	PLIB_DEVCON_IgnoreDebugFreezeEnable	Allows module to ignore FREEZE command from debugger and continue running.
	PLIB_DEVCON_JTAGPortDisable	Disables the JTAG port on the device.
	PLIB_DEVCON_JTAGPortEnable	Enables the JTAG port on the device.
	PLIB_DEVCON_MPLLDisable	Disables the MPLL.
	PLIB_DEVCON_MPLLEnable	Enables the MPLL.
	PLIB_DEVCON_MPLLInputDivSet	Sets the MPLL Input Divider bits.
	PLIB_DEVCON_MPLLIsReady	Reads MPLL status.
	PLIB_DEVCON_MPLLMultiplierSet	Sets the MPLL Multiplier bits.
	PLIB_DEVCON_MPLLODiv1Set	Sets the MPLL output divider 1 bits.
	PLIB_DEVCON_MPLLODiv2Set	Sets the MPLL output divider 2 bits.
	PLIB_DEVCON_MPLLVrefSet	Sets the VREF control setting.
	PLIB_DEVCON_MPLLVregDisable	Disables the MPLL VREG.
	PLIB_DEVCON_MPLLVregEnable	Enables the MPLL VREG.
	PLIB_DEVCON_MPLLVregIsReady	Reads the MPLL VREG status.
	PLIB_DEVCON_OTPConfigLock	Locks or unlocks the configuration registers.
	PLIB_DEVCON_OTPConfigUnlock	unlocks the configuration registers.
	PLIB_DEVCON_SystemLock	Executes the system lock sequence.
	PLIB_DEVCON_SystemUnlock	Executes the system unlock sequence.
	PLIB_DEVCON_TraceOutputDisable	Disables trace outputs and the stop trace clock.
	PLIB_DEVCON_TraceOutputEnable	Enables trace outputs and the start trace clock.
	PLIB_DEVCON_USBPHYSleepModeSet	Selects USB PHY clocking during Sleep mode.

Description

Device Control (DEVCON) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Device Control Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Device Control module.

File Name

plib_devcon.h

Company

Microchip Technology Inc.

help_plib_devcon.h

Enumerations

	Name	Description
	DEVCON_MODULE_ID	Identifies the supported DEVCON modules.

Description

This is file help_plib_devcon.h.

DMA Peripheral Library Help

This section describes the Direct Memory Access (DMA) Peripheral Library.

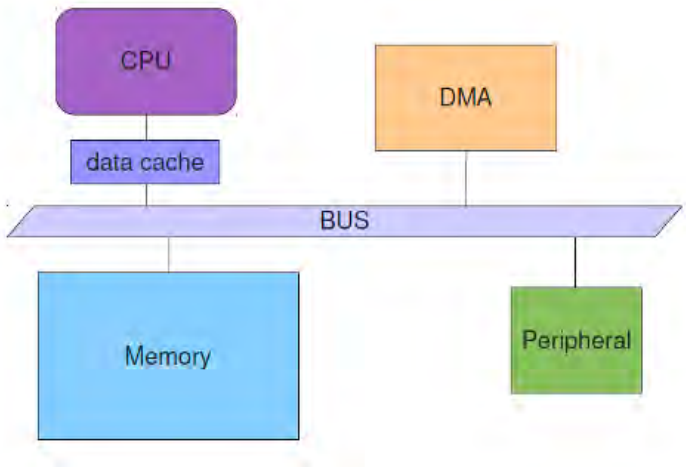
Introduction

Direct Memory Access (DMA) is a feature (sub-system) of the microcontroller that allows certain hardware sub-systems within the microcontroller to access the system memory independently of the processor (or the CPU).

Description

DMA also assists in accessing peripheral memory along with the system memory (data RAM). The DMA sub-system in the PIC family of microcontrollers is optimized for high-performance, real-time, embedded applications where determinism and system latency are priorities.

DMA Hardware Abstraction Model



Using the Library

This topic describes the basic architecture of the DMA Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_dma.h`

The interface to the DMA Peripheral Library is defined in the `plib_dma.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the DMA Peripheral Library must include `peripheral.h`.

Library File:

The DMA Peripheral Library (.a) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony?What is MPLAB Harmony? section for how the peripheral interacts with the framework.

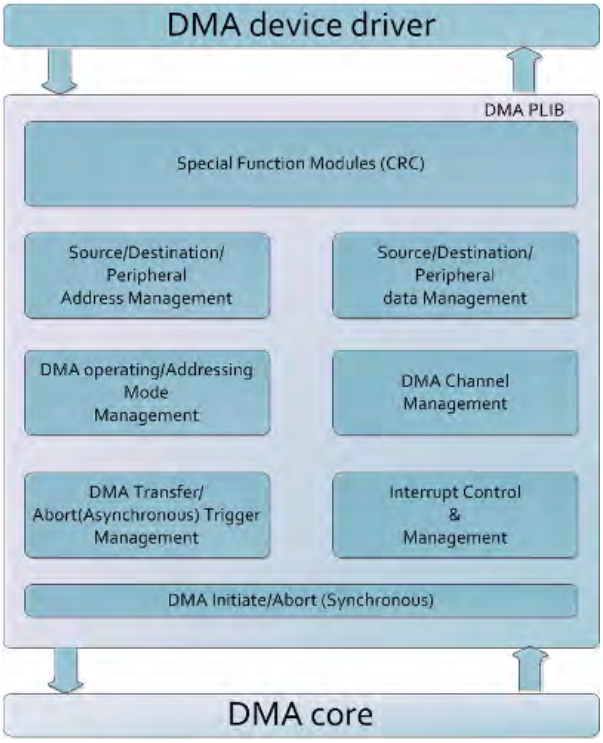
Hardware Abstraction Model

This library provides the low-level abstraction of the DMA module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The DMA Peripheral Library software is depicted using the following block diagram.

DMA Software Abstraction Block Diagram



The DMA Peripheral Library takes requests from application software or the DMA device driver and controls the DMA core (the DMA hardware) as per the inputs passed to the library.

The major components of the DMA Peripheral Library are:

- Source/Destination/Peripheral Address Management
- Source/Destination/Peripheral Data Management
- DMA Operating/Addressing Mode Management
- DMA Channel Management
- Interrupt Control and Management

The topic Library Overview explains in detail each block of the DMA Peripheral Library software architecture.

Note: The interface provided is a superset of all the functionality of the available DMA module on the device. Refer to the **"DMA"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each DMA module on your device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DMA module.

Library Interface Section	Description
Transfer/Abort (Synchronous)	Handles the synchronous DMA transfer start and End. This force request is controlled fully by the software.
Transfer/Abort (Asynchronous) Trigger Management	Each DMA transaction can be asynchronously initiated or aborted by an interrupt. This feature handles the required configuration.
Interrupt Control and Management	Each DMA channel can be configured to respond to events such as address error (upper address error, lower address error, etc.), transfer count half done, transfer count done, source data completely transferred, etc.
Operating/Addressing Mode Management	Before initiating a DMA transaction, the addressing and operating modes of the DMA module need to be configured (per channel basis). As an example, the DMA module can be configured for one shot mode with peripheral indirect addressing mode for a DMA transaction.
Channel Management	Each DMA channel can be individually configured for a DMA transaction. Priorities can be assigned, a channel can be enabled or disabled, channel chaining can be used, etc.
Source/Destination/Peripheral Address Management	For each DMA transaction to start, the DMA core needs to know the source and destination addresses. This feature handles this type of configuration.

Source/Destination/Peripheral Data Management	For each DMA transaction, the amount of data to be transferred, the source and destination size, the transaction type (byte/word), and transaction size can be configured by this feature.
Special Function Modules (CRC)	Certain high-end controllers support this CRC generator. This feature can be assigned to any of the available DMA channels. This feature handles the configuration of these aspects.
Status (Including Channel)	This feature specifies the application to access the status information of the DMA and each channel.
Feature Existence Functions	Lists the interface routines that determine whether or not the feature is supported by the device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine which modes are supported.

General Configuration

DMA General configuration includes enabling, disabling DMA module globally. It also involves DMA suspend control and behavior in Idle mode.

Enable/Disable DMA

Enabling and disabling the DMA are the basic routines that may be called during initialization and exit routines of the application.

DMA Enable/Disable Example:

```
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// Disable the DMA
PLIB_DMA_Disable ( DMA_ID_0 );
// Check the enable status of the DMA controller.
if(PLIB_DMA_IsEnabled( DMA_ID_0 ))
{
// application code
}
```

Transfer/Abort (Synchronous)

A DMA transaction can be initiated/aborted asynchronously by configuring an interrupt for a channel. Refer to Transfer/Abort (Asynchronous) Trigger Management for details.

In many instances, an application or a device driver needs to initiate a DMA transfer or abort an ongoing DMA transfer for reasons not necessarily known. To handle such cases, the DMA core supports a forceful DMA start and abort.

Note: The DMA abort is not supported by all devices. Refer to the specific device data sheet to determine availability of this feature.

When a DMA force start request is sent and the channel is already processing/pending to process an interrupt-based DMA transfer, the force transfer request will be ignored. In certain devices, a flag is raised to indicate a collision of DMA requests. Refer to Status (Including Channel) for details. In certain devices, an interrupt is issued to indicate that the active channel may not be able to keep up with the demands from the peripheral module being serviced. For more details refer to Interrupt Control and Management.

A DMA force transfer is initiated by the PLIB_DMA_StartTransferSet function.

A DMA force abort is initiated by the PLIB_DMA_AbortTransferSet function.

DMA Transfer/Abort (Synchronous) Configuration

A DMA transfer can be initiated or aborted forcefully by the software.

Example:

```
// Initiate a forced transfer on channel-3
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
// Abort a transfer on channel-3
PLIB_DMA_AbortTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

Transfer/Abort (Asynchronous) Trigger Management

DMA Start IRQ

Each channel can be configured to initiate a DMA transaction based on the interrupt from a source. This can be enabled/disable per channel basis. To initiate a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the start IRQ for this channel. Which source can act as the start IRQ is device-dependent. In certain devices, any of the available IRQs can be configured as the starting IRQ, while in other devices, there is restriction on which interrupt source can act as the start IRQ. Refer to the specific device data sheet to determine feature availability.

The starting IRQ is enabled by the PLIB_DMA_ChannelXTriggerEnable function. A DMA channel can be configured to start a DMA transfer based on the event on an IRQ using the PLIB_DMA_ChannelXStartIRQSet function.

DMA Abort IRQ

Each channel can be configured to abort a DMA transaction based on the interrupt from a source. This can be enabled/disabled per channel basis. To abort a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the abort IRQ for this channel.

The abort IRQ is enabled by the PLIB_DMA_ChannelXTriggerEnable function. A DMA channel can be configured to abort a DMA transfer based on the event on an IRQ using the ChannelXAbortIRQSet function.

DMA Pattern Matching Termination

Pattern Match Termination mode allows the user to end a transfer if a byte of data written during a transaction matches a specific pattern. This feature is useful in applications where a variable data size is required and eases the setup of the DMA channel. The UART module is a good example of where this can be effectively used. For details about the pattern data configuration refer to Channel Management**.**

DMA Pattern match mode is enabled using the PLIB_DMA_ChannelXTriggerEnable function**.** It can also be disabled using the PLIB_DMA_ChannelXTriggerDisable function**.**

DMA Transfer/Abort (Asynchronous) Configuration

A DMA transfer can also be triggered/aborted asynchronously by using an interrupt event. The interrupt event required to initiate/abort a DMA transfer for a channel needs to be set. Prior to this, the start IRQ and abort IRQ both need to be enabled.

Example:

```
// Enable the start IRQ. An interrupt event can trigger a DMA transfer.
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Enable the abort IRQ. An interrupt event can abort a DMA transfer on channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
// Set the start IRQ to UART-2 RX Interrupt
PLIB_DMA_ChannelXStartIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_RECEIVE );
// Set the abort IRQ to UART-2 error Interrupt
PLIB_DMA_ChannelXAbortIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_ERROR );
```

In some devices, abort IRQ trigger is not supported. Refer to the specific device data sheet to determine availability.

DMA Transfer Pattern Match Abort Configuration

The DMA transfer can also be aborted on a pattern match trigger (supported on PIC32 devices). Refer to Transfer/Abort (Asynchronous) Trigger Management for more information on pattern match abort. To use this feature, it must first be enabled. The pattern data needs to be programmed for the required channel. Once this is done, when a DMA transfer is in progress on that particular channel, if the DMA controller comes across the data matching the programmed data the transfer is automatically aborted.

Example:

```
// Enable the pattern match abort for channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
// Set a pattern data to match
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, DMA_CHANNEL_3, 0xEE );
```

Interrupt Control and Management

The DMA module has the ability to generate interrupts reflecting the events that occur during the channel's data transfer.

The following are all of the possible interrupts generated during DMA operation. This topic provides information for configuring these interrupts.

Error Interrupts

This type of event is signalled when an address error has occurred during the channel transfer operation. An address error also can occur if a DMA operation accesses an address beyond configured address limits (low and high).

Abort Interrupts

This type of event is signalled when a DMA transfer is aborted because of a system event matching the abort IRQ configured for the current channel.

DMA Completion Interrupts (Block Completion Interrupts)

This event occurs when a channel transfers a block of configured data.

Call Complete Interrupts

This event occurs when a channel completes a cell sized data transfer.

DMA Midpoint Interrupts

This event occurs when the channel is at the midpoint of completing the data transfer.

Overrun Interrupts

When a DMA channel receives a trigger (software or hardware) while it is servicing another request, an overrun condition occurs. This indicates that the channel is being requested before its current transaction is finished. An overrun condition will not result in termination of the current transaction.

Source Address Pointer Activity Interrupts

This event occurs when the channel source pointer reached the end of the source or when it reaches the midpoint of the source depending on the way this interrupt is configured.

Destination Address Pointer Activity Interrupts

This event occurs when the channel destination pointer reached the end of the destination or when it reaches the midpoint of the destination depending on the way this interrupt is configured.

This feature helps in configuring the previously listed interrupts. Depending on the device selected, all or some of the interrupts can be individually enabled and disabled. In addition, this feature obtains the status of the interrupts previously mentioned.

All of the interrupts belonging to the DMA channel map to the corresponding channel interrupt vector. In general, each channel of the DMA has a dedicated vectored interrupt. Therefore, the ISR for each channel should internally check for the status of the previously mentioned interrupts to respond to the corresponding events.

DMA Interrupt Control and Management

Each DMA channel has a vectored interrupt. However, each DMA channel internally generates various interrupts, which will invoke the same channel interrupt based on various events. This feature helps to enable/disable them and is also used to obtain the interrupt status.

Example:

```
// Enable the channel-3 source done interrupt
PLIB_DMA_ChannelXINTSourceEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_SOURCE_DONE );
// Get the status of address error on channel-3
if(PLIB_DMA_ChannelXINTSourceFlagGet ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_ADDRESS_ERROR ))
{
// application code for error handling
}
// Disable the channel abort interrupt
PLIB_DMA_ChannelXINTSourceDisable (DMA_ID_0, DMA_CHANNEL_3, DMA_INT_TRANSFER_ABORT);
```

Operating/Addressing Mode Management

DMA Operating Modes

The various operating/transfer modes supported are:

- One-Shot mode
- Repeated One-Shot mode
- Continuous mode
- Repeated Continuous mode
- Ping-Pong mode
- Null Data Write mode

One-Shot Mode

In One-Shot mode, a single transfer is performed for each trigger event. The trigger event can be synchronous (refer to Transfer/Abort (Synchronous) or asynchronous (refer to Transfer/Abort (Asynchronous) Trigger Management). When a single one-shot transfer occurs and the DMA count is decremented to zero, this disables the channel and requires the channel to be re-enabled to perform the next transaction.

Repeated One-Shot Mode

In Repeated One-Shot mode, single transfers occurs repeatedly so long as triggers are being provided. When the DMA count reaches zero, the

channel is not disabled as in One-Shot mode, but the count is reloaded and transfers begin again.

Continuous Mode

In Continuous mode, a single trigger starts a sequence of back-to-back transfers. These continue with each transfer decrementing the DMA count until it reaches zero. At this point, and like one-shot mode, the channel is disabled.

Repeated Continuous Mode

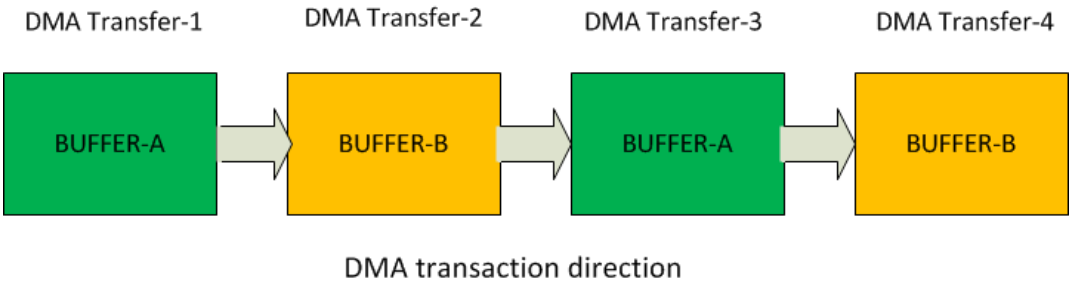
This mode can be seen as a combination of Continuous and Repeated One-Shot modes. Data transfers keep occurring so as long as triggers are provided and multiple transfers can occur with each trigger. When the DMA count reaches zero, the address and data registers are reloaded and the process is repeated.

Ping-Pong Mode

Certain devices support Ping-Pong mode, which allows the CPU to process one buffer while a second buffer operates with the DMA channel. The net result is that the CPU has the entire DMA block transfer time to process the buffer that is currently not being used by the DMA channel.

The following figure demonstrates the data flow for four DMA transactions using Ping-Pong mode.

Ping-Pong Mode



When the DMA is filling buffer-B, the CPU can process buffer-A and vice-versa.

Ping-Pong mode can be combined with One-Shot and Continuous modes (previously discussed) to create other combinations.

Null Data Write Mode

A DMA transfer operates only in one direction from the source address to the destination address. However, some communication protocols require symmetrical buffer accesses, meaning that for every read operation performed on a buffer, there must be an accompanying write operation. The Null Write mode is designed to satisfy this requirement. This mode works by transferring data from the source to the destination like any other DMA operation. Once this is done, the transferred data still with the DMA buffer is written back to the source.

DMA Addressing Modes

The addressing modes can be broadly classified into the following categories:

- Register Indirect with Post Increment Addressing mode
- Register Indirect without Post Increment Addressing mode
- Source/destination increment based on size
- Source/destination decrement based on size
- Peripheral Indirect Addressing mode

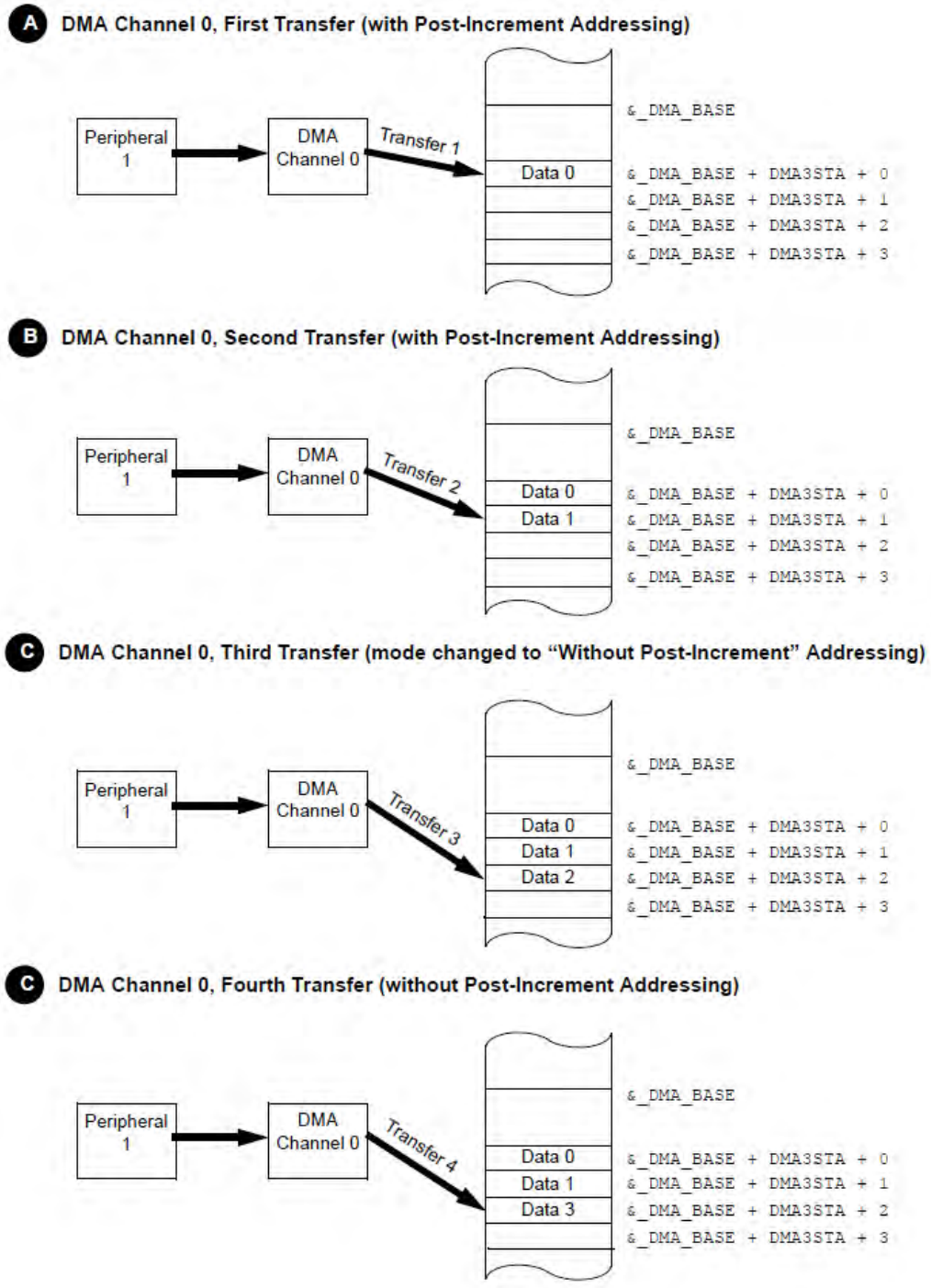
Register Indirect With Post Increment Addressing Mode

In this mode, the source data address is incremented by one location after each DMA transaction. The starting address is provided by a start address offset register.

Register Indirect Without Post Increment Addressing Mode

In this mode, the source data address is not updated after each DMA transaction. Therefore, the next DMA transfer is initiated from the same RAM address. The starting address is provided by a start address offset register.

Register Indirect Addressing With and Without Post-increment



Source/Destination Increment Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address increments by the size of data (word/byte) configured.

Source/Destination Decrement Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address decrements by the size of data (word/byte) configured.

Peripheral Indirect Addressing Mode

Peripheral Indirect Addressing (PIA) is a special auto-incrementing mode for the transfer of data to and from a multi-level peripheral buffer. This mode is only available for specific peripherals designed with its use in mind. Refer to the specific device data sheet to determine availability of this mode.

PIA-capable peripherals - When selected, the PIA-enabled peripheral generates a short Indirect Address (IA) (size defined by the peripheral) to the DMA channel. The IA is logically ORed with either the contents of the source address or the destination address to define a specific address for the peripheral inside the DMA address space.

DMA Operating/Transfer Modes:

Refer to Operating/Addressing Mode Management for descriptions of the available operating modes.

The following code demonstrates usage of Continuous mode.

Example:

```
// Enable the DMA
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// program the source and destination addresses
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// set the data transfer mode to one-shot
PLIB_DMA_ChannelXOperatingTransferModeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_MODE_CONTINUOUS );
// select the source addressing mode
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// select the destination addressing mode
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
// Enable the channel
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Initiate the transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

DMA Addressing Modes:

DMA addressing modes can be selected by using following code example.

DMA Addressing mode Example:

```
// set the source address to increment upon each transfer
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// Set the destination address to decrement upon each transfer
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE );
```

Channel Management

Each Channel in the DMA module is can be independently configured for a specific transaction with different properties.

The following aspects of the channel can be configured using this feature.

DMA Channel Enable/Disable

Each channel can be enabled or disabled to control the transactions on the channel. A DMA transfer can only occur when the channel is enabled. A DMA channel can be enabled using the PLIB_DMA_ChannelXEnable function. When a channel is disabled by calling the PLIB_DMA_ChannelXDisable function, no data transfers can occur on the channel.

DMA Channel Priority

The DMA module has a natural priority associated with each of the channels. Channel 0 has the highest priority. Each DMA channel can be assigned priority. This priority scheme changes depending on the device selected. There are three ways in which priorities are assigned in PIC microcontrollers. In the first method, two bits are allocated for each channel for priority. Therefore, priorities 0 to 3 can be assigned to channels. The second is called round-robin. In this method, the module starts by giving Channel 0 preference in any data transfer conflicts. For each successive transfer conflict, the next higher channel receives preference, continuing as a cycle through all the channels. The third method is called fixed scheme, which always gives priority to the lowest requesting channel number. The channel priority can be set using the PLIB_DMA_ChannelPrioritySelect function or the PLIB_DMA_ChannelXPrioritySelect function depending on the device selected.

DMA Channel Chaining

Channel chaining is an enhancement to the DMA channel operation. A channel (slave channel) can be chained to an adjacent channel (master channel). The slave channel will be enabled when a block transfer of the master channel completes. At this point, any event on the slave channel

will initiate a DMA transfer. If the channel has an event pending, a DMA transfer will begin immediately. The channel chaining can be enabled or disabled using the PLIB_DMA_ChannelXChainEnable function and the PLIB_DMA_ChannelXChainDisable function, respectively. The channel chaining higher/lower channel number can be configured using the PLIB_DMA_ChannelXChainToHigher function and the PLIB_DMA_ChannelXChainToLower function.

DMA Channel Transfer Direction

Each DMA transfer will have a direction. It can be memory to peripheral, peripheral to memory, or memory to memory. This can be set using the PLIB_DMA_ChannelXTransferDirectionSelect function.

DMA Channel Pattern Data

On supported devices, a DMA transfer can be aborted by a pattern match (refer to Transfer/Abort (Asynchronous) Trigger Management). The pattern data is per channel basis. It can be set using the PLIB_DMA_ChannelXPatternDataSet function.

Channel Auto Enable

This feature can be used to keep the channel active even if a block transfer or pattern match occurs. This mode is useful for applications that do repeated pattern matching. To enable this feature use the PLIB_DMA_ChannelXAutoEnable function. To disable this feature use the PLIB_DMA_ChannelXAutoDisable function.

DMA Channel Configuration

The following example configures DMA Channel 3 by setting its priority to two, enabling the auto-enable feature (refer to Channel Management section), enabling the channel chain feature and chaining Channel 3 and Channel 2, and finally enabling Channel 3.

Example:

```
// Disable the channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect  ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable the channel chain feature
PLIB_DMA_ChannelXChainEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// chain channel 3 to channel 2
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
```

In some devices it is required to select the DMA transfer direction before initiating DMA transfer. This can be done using the following code.

Example:

```
// Disable channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Set the data transfer direction from memory to peripheral
PLIB_DMA_ChannelXTransferDirectionSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
// we can also check the current channels transfer direction
// check the data transfer direction
if ( DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY == PLIB_DMA_ChannelXTransferDirectionGet ( DMA_ID_0,
DMA_CHANNEL_3 ) )
{

// application code

}
```

Source/Destination/Peripheral Address Management

This topic describes configuring the source/destination/peripheral addresses involved in the DMA transfer.

Source/Destination Start Address

Before initiating a DMA transfer, the channel should be programmed with the source and the destination addresses. In supported devices, this can be done using the PLIB_DMA_ChannelXSourceStartAddressSet Function and the PLIB_DMA_ChannelXDestinationStartAddressSet Function, respectively. At any point, the programmed addresses can be read using the PLIB_DMA_ChannelXSourceStartAddressGet Function and the PLIB_DMA_ChannelXDestinationStartAddressGet Function.

Source/Destination Start Address Offset

In some devices, a dedicated region of RAM is allocated for DMA transfers typically called DMA RAM. For any DMA transfer, the source or destination address needs to be defined as an offset from the starting of the DMA RAM for such devices. Use the

PLIB_DMA_ChannelXStartAddressOffsetSet Function to set the offset addresses of buffers A and B. Buffers A and B are used in Ping-Pong mode. To learn more, refer to Operating/Addressing Mode Management.

Peripheral Address

When DMA is used for a peripheral to memory or a memory to peripheral transfer, the peripheral address within a register must be set. This register needs to be programmed with the peripheral location where data transfers takes place. For example this can be the I2C buffer Special Function Register (SFR) location or the UART Receive register, etc. To set the peripheral address use the PLIB_DMA_ChannelXPeripheralAddressSet Function.

DMA Source/Destination/Peripheral Address Control

The first step in initiating a DMA transfer is to set the source and destination addresses.

The following example sets the source address and the destination address for Channel 3.

Example:

```
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
```

On certain devices, the source starting address needs to be given as an offset from the start of the DMA RAM address (in such devices, only memory to peripheral transfers and vice-versa are possible). In such devices, the source start address is set using the PLIB_DMA_ChannelXStartAddressOffsetSet function, as shown in the following code.

Example:

```
// Set the offset-A register to 0x1000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000, DMA_ADDRESS_OFFSET_PRIMARY);
// If ping-pong mode is enabled we will make use of buffer-B, and in that case
// we need to set an address for that also
// Set the offset-B to 0x2000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000, DMA_ADDRESS_OFFSET_SECONDARY);
```

The following example sets the peripheral address when the DMA needs to be initiated between memory and peripheral.

DMA Peripheral Address Set Example:

```
// Set the peripheral address to ADC1 buffer for channel-3 transfers.
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0, DMA_CHANNEL_3, (volatile uint16_t)(&ADC1BUF0));
```

There are also equivalent functions available within this library that read the source/destination/peripheral address set.

Source/Destination/Peripheral Data Management

This topic describes the data configuration.

Source/Destination Transfer Size Configuration

In supported devices, each channel can be programmed to read a particular sized data from a source, and write in a particular size to a destination. Both the source and destination sizes are configurable. Use the PLIB_DMA_ChannelXSourceSizeSet Function and the PLIB_DMA_ChannelXDestinationSizeSet Function for read size and write size configuration, respectively. For example, the source size can be 200 bytes and the destination size can be 100 bytes. However, it is logical to have both set to the same value. In some devices, the amount of data that needs to be transferred is independent of the source and destination, and is called transfer count. In such devices use the PLIB_DMA_ChannelXTransferCountSet Function to set the total amount of data (block transfer data).

Cell Size Configuration

In supported devices, when DMA transfer is initiated, the amount of data configured for a cell transfer will be delivered to the destination per trigger. This can be configured using the PLIB_DMA_ChannelXCellSizeSet Function.

Channel Data Size

The DMA controller can handle both byte and word (16-bit) transactions. Each DMA channel is individually configurable for the data size using the PLIB_DMA_ChannelXDataSizeSelect Function.

Source/Destination Pointer Read

In supported devices, the application can read the byte in source/destination to which the DMA core is currently pointing. This can be done using the PLIB_DMA_ChannelXSourcePointerGet Function and the PLIB_DMA_ChannelXDestinationPointerGet Function for source and destination, respectively.

DMA Source/Destination/Peripheral Data Control

After setting the source/destination address the amount of data that needs to be transferred is set in a DMA operation.

However, prior to that the data transaction type (byte/word) needs to be configured on certain devices.

Example:
```
// Set channel-3 transactions as byte transactions
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_8 );
// Set channel-3 transactions as 16-bit transactions
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_16 );
// Get the current data size configured
if( DMA_CHANNEL_BYTE == PLIB_DMA_ChannelXDataSizeGet ( DMA_ID_0, DMA_CHANNEL_3 ))
{
// application code
}
```

Example:
```
// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
```

Some high-end microcontrollers (especially PIC32 devices) use the method of cell transfer and block transfer. A cell transfer is the amount of data a DMA channel transfers per an event. In such microcontrollers the application can configure the sizes of the source and destination (per channel basis). A block transfer is defined as the number of bytes transferred when a channel is enabled. A block transfer is comprised of several cell transfers. For example, if the source and destination size is 100 and the cell size is 10, it will take 10 cell transfers to complete the block transfer, which is 100 bytes (larger of source or destination sizes).

DMA cell transfer configuration Example:
```
// Set the source size to be 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the destination size to be 100
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the cell size to be 10
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 10 );
```

Special Function Modules (CRC)

Selected high-end devices support a special feature called the CRC generator.

Depending on the device, the is a highly configurable, 16-bit or 32-bit CRC generator. The CRC can be assigned to any available DMA channel using PLIB_DMA_CRCChannelSelect Function. The CRC is enabled using the PLIB_DMA_CRCEnable Function.

Optionally, the data from the source can be subjected to byte reordering using the PLIB_DMA_CRCByteOrderSelect Function. The data is then optionally passed to the Linear Shift Feedback Register (LFSR) CRC or the IP header checksum blocks using the PLIB_DMA_CRCTypeSet Function.

The CRC feature is attached to a channel in one of two possible modes, Background or Append.

Background Mode

The CRC is calculated in the background, with normal DMA behavior maintained. This can be selected by disabling the Append mode.

Append Mode

Data read from the source is not written to the destination, but the CRC data is accumulated in the CRC data register. The accumulated CRC is written to the location specified by the destination address when a block transfer completes. This mode is enabled by the PLIB_DMA_CRCAppendModeEnable Function.

The following code shows DMA CRC calculation in Append mode.

Example:
```
// CRC of the Flash block
uint32_t blockCRC;
bool error = false;
// disable the channel 0 interrupts using the Interrupts Peripheral Library
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect  ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the channel chain feature
PLIB_DMA_ChannelXChainDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the start IRQ.
PLIB_DMA_ChannelXTriggerDisable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Seed the CRC generator
PLIB_DMA_CRCDataWrite ( DMA_ID_0, 0xFFFF );
// use the standard CCITT CRC 16 polynomial : X^16+X^12+X^5+1
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, 0x1021 );
```

```
// set polynomial length to 16
PLIB_DMA_CRCPolynomialLengthSet (DMA_ID_0, 16);
// Enable append mode
PLIB_DMA_CRCAppendModeEnable ( DMA_ID_0 );
// Attach CRC to DMA channel-3
PLIB_DMA_CRCChannelSelect (DMA_ID_0, DMA_CHANNEL_3);
// Enable the CRC
PLIB_DMA_CRCEnable ( DMA_ID_0 );
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// set source and destination size to 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set cell size to 100
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
//Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Force initiate a transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

Status (including Channel)

This module gives application access to various DMA and channel status information.

Configuring the Library

The library is configured for the supported DMA module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DMA_BusyActiveReset	Resets the BUSY bit of the DMA controller.
	PLIB_DMA_BusyActiveSet	Sets the BUSY bit of the DMA controller.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.
	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.

b) Transfer/Abort (Synchronous) Functions

	Name	Description
	PLIB_DMA_AbortTransferSet	Aborts transfer on the specified channel.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.

c) Transfer/Abort (Asynchronous) Trigger Management Functions

	Name	Description
	PLIB_DMA_ChannelXAbortIRQSet	Sets the IRQ to abort the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).
	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.
	PLIB_DMA_ChannelXTriggerSourceNumberGet	Gets the source number for the DMA channel interrupts.

d) Interrupt Control and Management Functions

	Name	Description
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.

	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.

e) Operating/Addressing Mode Configuration Functions

	Name	Description
	PLIB_DMA_ChannelXAddressModeGet	Gets the channel address mode.
	PLIB_DMA_ChannelXAddressModeSelect	Sets the channel address mode.
	PLIB_DMA_ChannelXDestinationAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.

f) Source/Destination/Peripheral Address Control Interface Functions

	Name	Description
	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

g) Source/Destination/Peripheral Data Management Functions

	Name	Description
	PLIB_DMA_ChannelXCellProgressPointerGet	Returns the number of bytes transferred since the last event.
	PLIB_DMA_ChannelXCellSizeGet	Reads the cell size (in bytes) configured for the specified channel.
	PLIB_DMA_ChannelXCellSizeSet	Writes the specified cell size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDataSizeGet	Returns the current data size for the specified channel.
	PLIB_DMA_ChannelXDataSizeSelect	Selects the data size for the specified channel.
	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.
	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.

	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.

h) Channel Configuration Functions

	Name	Description
	PLIB_DMA_ChannelPrioritySelect	Sets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelPriorityGet	Gets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelXAutoDisable	Channel is disabled after a block transfer is complete.
	PLIB_DMA_ChannelXAutoEnable	Channel is continuously enabled.
	PLIB_DMA_ChannelXBusyActiveSet	Sets the Busy bit to active.
	PLIB_DMA_ChannelXBusyInActiveSet	Sets the Busy bit to inactive.
	PLIB_DMA_ChannelXChainDisable	Disables the channel chaining for the specified DMA channel.
	PLIB_DMA_ChannelXChainEnable	Channel chain feature is enabled.
	PLIB_DMA_ChannelXChainToHigher	Chains the specified channel to a channel higher in natural priority.
	PLIB_DMA_ChannelXChainToLower	Chains the specified channel to a channel lower in natural priority.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.

i) CRC Module Interface Functions

	Name	Description
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModeIsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.
	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.

j) Global Status Functions

	Name	Description
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

k) Channel Status Functions

	Name	Description
	PLIB_DMA_ChannelXAutoIsEnabled	Returns the channel automatic enable status.
	PLIB_DMA_ChannelXBufferedDataIsWritten	Returns the buffered data write status for the specified channel.
	PLIB_DMA_ChannelXBusyIsBusy	Returns the busy status of the specified channel.
	PLIB_DMA_ChannelXChainIsEnabled	Returns the chain status of the specified channel.
	PLIB_DMA_ChannelXCollisionStatus	Returns the status of the specified collision type for the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeIsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelBitsGet	Returns the DMA channel bits.

l) Feature Existence Functions

	Name	Description
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnoreByte	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.

	PLIB_DMA_ExistsChannelXPatternLength	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPriority	Identifies whether the ChannelXPriority feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourcePointer	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceSize	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceStartAddress	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXStartIRQ	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXTrigger	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	PLIB_DMA_ExistsCRC	Identifies whether the CRC feature exists on the DMA module.
	PLIB_DMA_ExistsCRCAppendMode	Identifies whether the CRCAppendMode feature exists on the DMA module.
	PLIB_DMA_ExistsCRCBitOrder	Identifies whether the CRCBitOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCByteOrder	Identifies whether the CRCByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCChannel	Identifies whether the CRCChannel feature exists on the DMA module.
	PLIB_DMA_ExistsCRCData	Identifies whether the CRCData feature exists on the DMA module.
	PLIB_DMA_ExistsCRCPolynomialLength	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	PLIB_DMA_ExistsCRCType	Identifies whether the CRCType feature exists on the DMA module.
	PLIB_DMA_ExistsCRCWriteByteOrder	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCXOREnable	Identifies whether the CRCXOREnable feature exists on the DMA module.
	PLIB_DMA_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMA module.
	PLIB_DMA_ExistsLastBusAccess	Identifies whether the LastBusAccess feature exists on the DMA module.
	PLIB_DMA_ExistsRecentAddress	Identifies whether the RecentAddress feature exists on the DMA module.
	PLIB_DMA_ExistsStartTransfer	Identifies whether the StartTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the DMA module.
	PLIB_DMA_ExistsSuspend	Identifies whether the Suspend feature exists on the DMA module.

m) Data Types and Constants

	Name	Description
	DMA_ADDRESS_OFFSET_TYPE	Lists the possible DMA address offsets.
	DMA_CHANNEL	Lists the possible DMA channels available.
	DMA_CHANNEL_ADDRESSING_MODE	Lists the possible channel addressing modes.
	DMA_CHANNEL_COLLISION	Lists the possible channel collision types.
	DMA_CHANNEL_DATA_SIZE	Lists the possible data sizes for the DMA channel.
	DMA_CHANNEL_PRIORITY	Lists the possible channel priorities.
	DMA_CHANNEL_TRANSFER_DIRECTION	Lists the possible data transfer directions.
	DMA_CHANNEL_TRIGGER_TYPE	Lists the possible DMA channel triggers.
	DMA_CRC_BIT_ORDER	Lists the possible CRC calculation bit orders
	DMA_CRC_BYTE_ORDER	Lists the possible byte orders.
	DMA_CRC_TYPE	Lists the possible checksums.
	DMA_DESTINATION_ADDRESSING_MODE	Lists the possible destination addressing modes.
	DMA_INT_TYPE	Lists the possible Interrupt types for a particular channel-X
	DMA_MODULE_ID	Possible instances of the DMA module.
	DMA_PATTERN_LENGTH	Gives the DMA pattern length
	DMA_PING_PONG_MODE	Lists the possible ping-pong modes.
	DMA_SOURCE_ADDRESSING_MODE	Lists the possible source addressing modes.
	DMA_TRANSFER_MODE	Lists the possible DMA transfer/operating modes.
	DMA_TRIGGER_SOURCE	Lists the possible DMA trigger sources.
	DMA_CHANNEL_INT_SOURCE	Lists the Interrupt Source number for the available DMA channels.

Description

This section describes the Application Programming Interface (API) functions of the DMA Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DMA_BusyActiveReset Function

Resets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
void PLIB_DMA_BusyActiveReset(DMA_MODULE_ID index);
```

Returns

None.

Description

This function resets the BUSY bit of the DMA controller. The DMA module is disabled and is not actively transferring data.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_BusyActiveReset( DMA_ID_0 );
```

Function

void PLIB_DMA_BusyActiveReset (DMA_MODULE_ID index)

PLIB_DMA_BusyActiveSet Function

Sets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
void PLIB_DMA_BusyActiveSet(DMA_MODULE_ID index);
```

Returns

None.

Description

This function sets the BUSY bit of the DMA controller. The DMA module is active.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_BusyActiveSet( DMA_ID_0 );
```

Function

void PLIB_DMA_BusyActiveSet (DMA_MODULE_ID index)

PLIB_DMA_Disable Function

DMA module is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_Disable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_Disable( DMA_ID_0 );
```

Function

void PLIB_DMA_Disable (DMA_MODULE_ID index)

PLIB_DMA_Enable Function

DMA module is enabled.

File

plib_dma.h

C

```
void PLIB_DMA_Enable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_Enable( DMA_ID_0 );
```

Function

void PLIB_DMA_Enable (DMA_MODULE_ID index)

PLIB_DMA_StopInIdleDisable Function

DMA transfers continue during Idle mode.

File

plib_dma.h

C

```
void PLIB_DMA_StopInIdleDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function causes DMA transfers to continue during Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStopInIdle function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StopInIdleDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_StopInIdleDisable (DMA_MODULE_ID index)

PLIB_DMA_StopInIdleEnable Function

DMA transfers are halted during Idle mode.

File

plib_dma.h

C

```
void PLIB_DMA_StopInIdleEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function halts DMA transfers during Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStopInIdle function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StopInIdleEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_StopInIdleEnable (DMA_MODULE_ID index)

PLIB_DMA_SuspendDisable Function

DMA suspend is disabled and the DMA module operates normally.

File

plib_dma.h

C

```
void PLIB_DMA_SuspendDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA suspend. The DMA module continues to operate normally.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_SuspendDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_SuspendDisable (DMA_MODULE_ID index)

PLIB_DMA_SuspendEnable Function

DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.

File

plib_dma.h

C

```
void PLIB_DMA_SuspendEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function suspends the DMA transfers to allow uninterrupted access by the CPU to the data bus.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_SuspendEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_SuspendEnable (DMA_MODULE_ID index)

b) Transfer/Abort (Synchronous) Functions

PLIB_DMA_AbortTransferSet Function

Aborts transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_AbortTransferSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function aborts transfer on the specified channel. This is a forced abort controlled via software.

Remarks

This function implements an operation of the AbortTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsAbortTransfer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_AbortTransferSet (    DMA_ID_0,
                               DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_AbortTransferSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_StartTransferSet Function

Initiates transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_StartTransferSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function initiates transfer on the specified channel. This is a forced transfer controlled via software.

Remarks

This function implements an operation of the StartTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStartTransfer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StartTransferSet (    DMA_ID_0,
                               DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_StartTransferSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

c) Transfer/Abort (Asynchronous) Trigger Management Functions

PLIB_DMA_ChannelXAbortIRQSet Function

Sets the IRQ to abort the DMA transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAbortIRQSet(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_TRIGGER_SOURCE IRQ);
```

Returns

None.

Description

This function sets the IRQ to abort the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Remarks

This function implements an operation of the ChannelXAbortIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAbortIRQ function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_TIMER_CORE;
PLIB_DMA_ChannelXAbortIRQSet (  DMA_ID_0,
                                DMA_CHANNEL_4,
                                irq );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Function

void PLIB_DMA_ChannelXAbortIRQSet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRIGGER_SOURCE IRQ)

PLIB_DMA_ChannelXStartIRQSet Function

Sets the IRQ to initiate the DMA transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXStartIRQSet(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_TRIGGER_SOURCE IRQnum);
```

Returns

None.

Description

This function sets the IRQ to initiate the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Remarks

This function implements an operation of the ChannelXStartIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXStartIRQ function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_OUTPUT_COMPARE_1;
PLIB_DMA_ChannelXStartIRQSet (  DMA_ID_0,
                                DMA_CHANNEL_4,
                                irq );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Function

void PLIB_DMA_ChannelXStartIRQSet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRIGGER_SOURCE IRQnum)

PLIB_DMA_ChannelXTriggerDisable Function

Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTriggerDisable(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

None.

Description

This function disables the DMA transfer abort via a matching interrupt (specified by the IRQ). The interrupt number IRQ is ignored and does not terminate a transfer.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerDisable (   DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

void PLIB_DMA_ChannelXTriggerDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerEnable Function

Enables the specified DMA channel trigger.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTriggerEnable(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

None.

Description

This function enables the specified DMA channel trigger.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerEnable (    DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

void PLIB_DMA_ChannelXTriggerEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerIsEnabled Function

Returns the enable status of the specified DMA transfer/abort trigger.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXTriggerIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

- true - The specified trigger is enabled
- false - The specified trigger is disabled

Description

This function returns the enable status of the specified DMA transfer/abort trigger.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool startTriggerstatus;
startTriggerstatus = PLIB_DMA_ChannelXTriggerIsEnabled (
                                        DMA_ID_0,
                                        DMA_CHANNEL_4,
                                        DMA_CHANNEL_TRIGGER_TRANSFER_START );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

bool PLIB_DMA_ChannelXTriggerIsEnabled(DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerSourceNumberGet Function

Gets the source number for the DMA channel interrupts.

File

plib_dma.h

C

```
DMA_CHANNEL_INT_SOURCE PLIB_DMA_ChannelXTriggerSourceNumberGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function returns the interrupt source number for the specified DMA channel.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerSourceNumberGet (   DMA_ID_0,
                                            DMA_CHANNEL_4);
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXTriggerSourceNumberGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

d) Interrupt Control and Management Functions

PLIB_DMA_ChannelXINTSourceDisable Function

Disables the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceDisable(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function disables the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceDisable ( DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceDisable (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceEnable Function

Enables the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceEnable(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function enables the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceEnable (  DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceEnable (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagClear Function

Clears the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceFlagClear(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function clears the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagClear (   DMA_ID_0,
                                        spiDMAChannel,
                                        DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceFlagClear (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagGet Function

Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXINTSourceFlagGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

- true - The interrupt flag is set
- false - The interrupt flag is not set

Description

This function returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
bool AddressErrorINTStatus;
AddressErrorINTStatus = PLIB_DMA_ChannelXINTSourceFlagGet (
                                                    DMA_ID_0,
                                                    spiDMAChannel,
                                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

bool PLIB_DMA_ChannelXINTSourceFlagGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagSet Function

Sets the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceFlagSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function sets the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagSet ( DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceFlagSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceIsEnabled Function

Returns the enable status of the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXINTSourceIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

- true - The interrupt is enabled
- false - The interrupt is not enabled

Description

This function returns the enable status of the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
bool dmaINTSourceEnableStatus;
dmaINTSourceEnableStatus = PLIB_DMA_ChannelXINTSourceIsEnabled (
                                                    DMA_ID_0,
                                                    spiDMAChannel,
                                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

bool PLIB_DMA_ChannelXINTSourceIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

e) Operating/Addressing Mode Configuration Functions

PLIB_DMA_ChannelXAddressModeGet Function

Gets the channel address mode.

File

plib_dma.h

C

```
DMA_CHANNEL_ADDRESSING_MODE PLIB_DMA_ChannelXAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelAddressMode - One of the possible source addressing modes listed by DMA_CHANNEL_ADDRESSING_MODE

Description

This function gets the channel address mode.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_ADDRESSING_MODE channelAddressMode;
channelAddressMode = PLIB_DMA_ChannelXAddressModeGet (  DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_ADDRESSING_MODE PLIB_DMA_ChannelXAddressModeGet (
DMA_MODULE_ID index,

DMA_CHANNEL channel)

PLIB_DMA_ChannelXAddressModeSelect Function

Sets the channel address mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_CHANNEL_ADDRESSING_MODE channelAddressMode);
```

Returns

None.

Description

This function sets the channel address mode.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAddressModeSelect (
                    DMA_ID_0,
                    DMA_CHANNEL_4,
                    DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelAddressMode	One of the possible channel addressing modes listed by DMA_CHANNEL_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXAddressModeSelect (

DMA_MODULE_ID index,

DMA_CHANNEL channel,

DMA_CHANNEL_ADDRESSING_MODE channelAddressMode)

PLIB_DMA_ChannelXDestinationAddressModeGet Function

Gets the source address mode of the specified channel.

File

plib_dma.h

C

```
DMA_DESTINATION_ADDRESSING_MODE PLIB_DMA_ChannelXDestinationAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL
channel);
```

Returns

- destinationAddressMode - One of the possible source addressing modes listed by DMA_DESTINATION_ADDRESSING_MODE

Description

This function gets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_SOURCE_ADDRESSING_MODE destinationAddressMode;
destinationAddressMode = PLIB_DMA_ChannelXDestinationAddressModeGet (
                                                        DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_DESTINATION_ADDRESSING_MODE PLIB_DMA_ChannelXDestinationAddressModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDestinationAddressModeSelect Function

Sets the source address mode of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_DESTINATION_ADDRESSING_MODE destinationAddressMode);
```

Returns

None.

Description

This function Sets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDestinationAddressModeSelect (
                        DMA_ID_0,
                DMA_CHANNEL_4,
                        DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
destinationAddressMode	One of the possible source addressing modes listed by DMA_DESTINATION_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXDestinationAddressModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_DESTINATION_ADDRESSING_MODE destinationAddressMode)

PLIB_DMA_ChannelXNullWriteModeDisable Function

Disables the Null Write mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXNullWriteModeDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the Null Write mode. No dummy write is initiated.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXNullWriteModeDisable ( DMA_ID_0,
                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXNullWriteModeDisable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXNullWriteModeEnable Function

Enables the Null Write mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXNullWriteModeEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the Null Write mode. A dummy write is initiated to the source address for every write to the destination address.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXNullWriteModeEnable (  DMA_ID_0,
                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXNullWriteModeEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXOperatingTransferModeGet Function

Returns the current transfer/operating mode for the specified channel.

File

plib_dma.h

C

```
DMA_TRANSFER_MODE PLIB_DMA_ChannelXOperatingTransferModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channeltransferMode - One of the possible operating/transfer modes listed by DMA_TRANSFER_MODE

Description

This function returns the current transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_TRANSFER_MODE channeltransferMode;
channeltransferMode = PLIB_DMA_ChannelXOperatingTransferModeGet (
                                                        DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_TRANSFER_MODE PLIB_DMA_ChannelXOperatingTransferModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXOperatingTransferModeSelect Function

Selects the transfer/operating mode for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXOperatingTransferModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_TRANSFER_MODE channeltransferMode);
```

Returns

None.

Description

This function selects the transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_TRANSFER_MODE channeltransferMode = DMA_MODE_REPEATED_CONTINUOUS;
PLIB_DMA_ChannelXOperatingTransferModeSelect (  DMA_ID_0,
                                                DMA_CHANNEL_4,
                                                channeltransferMode );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channeltransferMode	One of the possible operating/transfer modes listed by DMA_TRANSFER_MODE

Function

void PLIB_DMA_ChannelXOperatingTransferModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRANSFER_MODE channeltransferMode)

PLIB_DMA_ChannelXReloadDisable Function

Disables reloading of the address and count registers.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXReloadDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables reloading of the address and count registers. The source, destination address, and the DMA count registers are not reloaded to their previous values upon the start of the next operation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXReloadDisable (    DMA_ID_0,
                                    DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXReloadDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXReloadEnable Function

Enables reloading of the address and count registers.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXReloadEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables reloading of the address and count registers. The source, destination address, and the DMA count registers are reloaded to their previous values upon the start of the next operation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXReloadEnable ( DMA_ID_0,
                                DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXReloadEnable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXSourceAddressModeGet Function

Gets the source address mode of the specified channel.

File

plib_dma.h

C

```
DMA_SOURCE_ADDRESSING_MODE PLIB_DMA_ChannelXSourceAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- DMA_SOURCE_ADDRESSING_MODE - One of the possible source addressing modes listed by DMA_SOURCE_ADDRESSING_MODE

Description

This function gets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode;
sourceAddressMode = PLIB_DMA_ChannelXSourceAddressModeGet ( DMA_ID_0,
                                                            DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_SOURCE_ADDRESSING_MODE PLIB_DMA_ChannelXSourceAddressModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXSourceAddressModeSelect Function

Sets the source address mode of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode);
```

Returns

None.

Description

This function sets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXSourceAddressModeSelect (
                            DMA_ID_0,
                            DMA_CHANNEL_4,
                            DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
sourceAddressMode	One of the possible source addressing modes listed by DMA_SOURCE_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXSourceAddressModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode)

PLIB_DMA_ChannelXReloadIsEnabled Function

Returns the address and count registers reload enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXReloadIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The address and count registers reload is enabled
- false - The address and count registers reload is disabled

Description

This function returns the address and count registers reload enable status.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chAddressCountReloadStatus;
chAddressCountReloadStatus = PLIB_DMA_ChannelXReloadIsEnabled (
                                                              DMA_ID_0,
                                                              DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXReloadIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

f) Source/Destination/Peripheral Address Control Interface Functions

PLIB_DMA_ChannelXDestinationStartAddressGet Function

Reads the destination start address configured for the specified channel.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_ChannelXDestinationStartAddressGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint32_t - The destination start address configured for this channel

Description

This function reads the destination start address configured for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t DestinationStartAddress;
DestinationStartAddress = PLIB_DMA_ChannelXDestinationStartAddressGet (
                                                              DMA_ID_0,
                                                              spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint32_t PLIB_DMA_ChannelXDestinationStartAddressGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationStartAddressSet Function

Writes the specified destination start address into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationStartAddressSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint32_t
destinationStartAddress);
```

Returns

None.

Description

This function writes the specified destination start address into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t destinationStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXDestinationStartAddressSet(    DMA_ID_0,
                                                spiDMAChannel,
                                                destinationStartAddress );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationStartAddress	The destination start address

Function

void PLIB_DMA_ChannelXDestinationStartAddressSet (
DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint32_t destinationStartAddress)

PLIB_DMA_ChannelXPeripheralAddressGet Function

Gets the peripheral address configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXPeripheralAddressGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint16_t - The peripheral address configured for the specified channel

Description

This function gets the peripheral address configured for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t peripheraladdress;
peripheraladdress = PLIB_DMA_ChannelXPeripheralAddressGet ( DMA_ID_0,
                                                            DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXPeripheralAddressGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPeripheralAddressSet Function

Sets the peripheral address for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPeripheralAddressSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t
peripheraladdress);
```

Returns

None.

Description

This function sets the peripheral address for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t peripheraladdress = 0x100;
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0,
                                        DMA_CHANNEL_4,
                                        peripheraladdress );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
peripheraladdress	The peripheral address for the specified channel

Function

void PLIB_DMA_ChannelXPeripheralAddressSet(DMA_MODULE_ID index,

DMA_CHANNEL channel ,
uint16_t peripheraladdress)

PLIB_DMA_ChannelXSourceStartAddressGet Function

Reads the source start address configured for the specified channel.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_ChannelXSourceStartAddressGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint32_t - The source start address configured for this channel

Description

This function reads the source start address configured for the specified channel.

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t SourceStartAddress;
SourceStartAddress = PLIB_DMA_ChannelXSourceStartAddressGet(DMA_ID_0,
                                                            spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint32_t PLIB_DMA_ChannelXSourceStartAddressGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceStartAddressSet Function

Writes the specified source start address into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceStartAddressSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint32_t
sourceStartAddress);
```

Returns

None.

Description

This function writes the specified Source start address into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t sourceStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXSourceStartAddressSet (    DMA_ID_0,
                                            spiDMAChannel,
                                            sourceStartAddress );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceStartAddress	The source start address

Function

void PLIB_DMA_ChannelXSourceStartAddressSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint32_t sourceStartAddress)

PLIB_DMA_ChannelXStartAddressOffsetGet Function

Gets the primary/secondary start address (DPSRAM) offset.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXStartAddressOffsetGet(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_ADDRESS_OFFSET_TYPE offset);
```

Returns

- uint16_t - The primary/secondary DPSRAM start address offset

Description

This function gets the primary/secondary start address (DPSRAM) offset.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t addressOffsetA;
addressOffsetA = PLIB_DMA_ChannelXStartAddressOffsetGet (
                                                DMA_ID_0,
                                                DMA_CHANNEL_4,
                                                address,
                                                DMA_ADDRESS_OFFSET_PRIMARY );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
offset	The type of the address offset (primary/secondary)

Function

uint16_t PLIB_DMA_ChannelXStartAddressOffsetGet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_ADDRESS_OFFSET_TYPE offset)

PLIB_DMA_ChannelXStartAddressOffsetSet Function

Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXStartAddressOffsetSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t address,
DMA_ADDRESS_OFFSET_TYPE offset);
```

Returns

None.

Description

This function sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t address = 0x100;
PLIB_DMA_ChannelXStartAddressOffsetSet (    DMA_ID_0,
                                            DMA_CHANNEL_4,
                                            address,
                                            DMA_ADDRESS_OFFSET_PRIMARY );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
address	The primary/secondary DPSRAM start address offset
offset	The type of the address offset (primary/secondary)

Function

void PLIB_DMA_ChannelXStartAddressOffsetSet(DMA_MODULE_ID index,
DMA_CHANNEL channel ,
uint16_t address,
DMA_ADDRESS_OFFSET_TYPE offset)

g) Source/Destination/Peripheral Data Management Functions

PLIB_DMA_ChannelXCellProgressPointerGet Function

Returns the number of bytes transferred since the last event.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXCellProgressPointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The number of bytes transferred since the last event.

The cell progress pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function returns the number of bytes transferred since the last event.

Remarks

This function implements an operation of the ChannelXCellProgressPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellProgressPointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t CellProgress;
CellProgress = PLIB_DMA_ChannelXCellProgressPointerGet (   DMA_ID_0,
                                                           spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXCellProgressPointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXCellSizeGet Function

Reads the cell size (in bytes) configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXCellSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The cell size configured (in bytes) for this channel

The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the cell size (in bytes) configured for the specified channel.

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize;
cellSize = PLIB_DMA_ChannelXCellSizeGet (   DMA_ID_0,
                                            spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXCellSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXCellSizeSet Function

Writes the specified cell size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXCellSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t CellSize);
```

Returns

None.

Description

This function writes the specified cell size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize = 0x10;
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, spiDMAChannel, cellSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
cellSize	The cell size in bytes. The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXCellSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t cellSize)

PLIB_DMA_ChannelXDataSizeGet Function

Returns the current data size for the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_DATA_SIZE PLIB_DMA_ChannelXDataSizeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelDataSize - One of the possible data sizes listed by DMA_CHANNEL_DATA_SIZE

Description

This function returns the current data size for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize;
channelDataSize = PLIB_DMA_ChannelXDataSizeGet( DMA_ID_0,
                                                DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_DATA_SIZE PLIB_DMA_ChannelXDataSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDataSizeSelect Function

Selects the data size for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDataSizeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_DATA_SIZE
channelDataSize);
```

Returns

None.

Description

This function selects the data size for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize = DMA_CHANNEL_DATA_8;
PLIB_DMA_ChannelXDataSizeSelect (   DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    channelDataSize );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelDataSize	One of the possible data sizes listed by DMA_CHANNEL_DATA_SIZE

Function

void PLIB_DMA_ChannelXDataSizeSelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_DATA_SIZE channelDataSize)

PLIB_DMA_ChannelXDestinationPointerGet Function

Reads the current byte of the destination being pointed to for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXDestinationPointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The destination byte being pointed to for this channel.

The destination pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the current byte of the destination being pointed to for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationPointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t destinationbyte;
destinationbyte = PLIB_DMA_ChannelXDestinationPointerGet ( DMA_ID_0,
                                                           spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXDestinationPointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationSizeGet Function

Reads the destination size configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXDestinationSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The destination size configured (in bytes) for this channel.

The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the destination size configured for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t DestinationSize;
DestinationSize = PLIB_DMA_ChannelXDestinationSizeGet ( DMA_ID_0,
                                                        spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXDestinationSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationSizeSet Function

Writes the specified destination size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t
destinationSize);
```

Returns

None.

Description

This function writes the specified destination size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t destinationSize = 0xA00;
PLIB_DMA_ChannelXDestinationSizeSet( DMA_ID_0, spiDMAChannel, destinationSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationSize	The destination size. The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXDestinationSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t destinationSize)

PLIB_DMA_ChannelXPatternDataGet Function

Returns the pattern matching (for DMA abort) data programmed for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXPatternDataGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The pattern matching data programmed for the current channel.

The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function returns pattern matching (for DMA abort) data programmed for the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t patternData;
patternData = PLIB_DMA_ChannelXPatternDataGet ( DMA_ID_0, spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXPatternDataGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXPatternDataSet Function

Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternDataSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t patternData);
```

Returns

None.

Description

This function writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
```

```
uint16_t patternData = '0';
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, spiDMAChannel, patternData );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
patternData	The pattern matching DATA programmed for the current channel (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXPatternDataSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t patternData)

PLIB_DMA_ChannelXSourcePointerGet Function

Reads the current byte of the source being pointed to for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXSourcePointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The source byte being pointed to for this channel.

The source pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the current byte of the source being pointed to for the specified channel.

Remarks

This function implements an operation of the ChannelXSourcePointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourcePointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourcebyte;
sourcebyte = PLIB_DMA_ChannelXSourcePointerGet ( DMA_ID_0, spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXSourcePointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceSizeGet Function

Reads the source size configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXSourceSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The Source size configured (in bytes) for this channel.

The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the source size configured for the specified channel.

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize;
sourceSize = PLIB_DMA_ChannelXSourceSizeGet (    DMA_ID_0,
                                                 spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXSourceSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceSizeSet Function

Writes the specified source size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t sourceSize);
```

Returns

None.

Description

This function writes the specified source size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize = 0xA00;
PLIB_DMA_ChannelXSourceSizeSet (    DMA_ID_0,
                                    spiDMAChannel,
                                    sourceSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceSize	The source size. The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Function

void PLIB_DMA_ChannelXSourceSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t sourceSize)

PLIB_DMA_ChannelXTransferCountGet Function

Gets the DMA data transfer count that is programmed for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXTransferCountGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint16_t - The DMA transfer count for the channel

Description

This function gets the DMA data transfer count that is programmed for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t transferCount;
transferCount = PLIB_DMA_ChannelXTransferCountGet ( DMA_ID_0,
                                                    DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXTransferCountGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXTransferCountSet Function

Sets the DMA data transfer count for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTransferCountSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t transferCount);
```

Returns

None.

Description

This function sets the DMA data transfer count for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t transferCount = 0x10;
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    transferCount );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
transferCount	The DMA transfer count for the channel

Function

void PLIB_DMA_ChannelXTransferCountSet(DMA_MODULE_ID index,
DMA_CHANNEL channel ,
uint16_t transferCount)

h) Channel Configuration Functions

PLIB_DMA_ChannelPrioritySelect Function

Sets the priority scheme of the DMA channels.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelPrioritySelect(DMA_MODULE_ID index, DMA_CHANNEL_PRIORITY channelPriority);
```

Returns

None.

Description

This function sets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_ROUND_ROBIN;
PLIB_DMA_ChannelPrioritySelect( DMA_ID_0, channelPriority );
```

Parameters

Parameters	Description
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Function

void PLIB_DMA_ChannelPrioritySelect (DMA_MODULE_ID index,
DMA_CHANNEL_PRIORITY channelPriority)

PLIB_DMA_ChannelPriorityGet Function

Gets the priority scheme of the DMA channels.

File

plib_dma.h

C

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelPriorityGet(DMA_MODULE_ID index);
```

Returns

- channelPriority - One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Description

This function gets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_PRIORITY channelPriority;
channelPriority = PLIB_DMA_ChannelPriorityGet( DMA_ID_0 );
```

Function

DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelPriorityGet (DMA_MODULE_ID index)

PLIB_DMA_ChannelXAutoDisable Function

Channel is disabled after a block transfer is complete.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAutoDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables a channel after a block transfer is complete.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAutoDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXAutoDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXAutoEnable Function

Channel is continuously enabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAutoEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the channel continuously. The channel is not automatically disabled after a block transfer is complete.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAutoEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXAutoEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyActiveSet Function

Sets the Busy bit to active.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXBusyActiveSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function sets the Busy bit to active, indicating the channel is active or has been enabled.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXBusyActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXBusyActiveSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyInActiveSet Function

Sets the Busy bit to inactive.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXBusyInActiveSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function sets the Busy bit to inactive, indicating the channel is inactive or has been disabled.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXBusyInActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXBusyInActiveSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainDisable Function

Disables the channel chaining for the specified DMA channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the channel chaining for the specified DMA channel.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainDisable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainEnable Function

Channel chain feature is enabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the channel chain feature.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainEnable (DMA_MODULE_ID index,

DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainToHigher Function

Chains the specified channel to a channel higher in natural priority.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainToHigher(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will chain the specified channel to a channel higher in natural priority. CH5 will be enabled by a CH4 transfer complete.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChain function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainToHigher ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainToHigher (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainToLower Function

Chains the specified channel to a channel lower in natural priority.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainToLower(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will chain the specified channel to a channel lower in natural priority. CH3 will be enabled by a CH4 transfer complete.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChain function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainToLower(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDisable Function

Disable the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will disable the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXEnable Function

Enable the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will enable the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPriorityGet Function

Gets the priority of the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelXPriorityGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelPriority - One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Description

This function gets the priority of the specified channel.

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;
DMA_CHANNEL_PRIORITY channelPriority;
channelPriority = PLIB_DMA_ChannelXPriorityGet( DMA_ID_0, channel );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelXPriorityGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPrioritySelect Function

Sets the priority of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPrioritySelect(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_PRIORITY
channelPriority);
```

Returns

None.

Description

This function sets the priority of the specified channel.

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_PRIORITY_3;
PLIB_DMA_ChannelXPrioritySelect( DMA_ID_0, channel, channelPriority );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Function

void PLIB_DMA_ChannelXPrioritySelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_PRIORITY channelPriority)

PLIB_DMA_ChannelXTransferDirectionGet Function

Returns the data transfer direction of the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_TRANSFER_DIRECTION PLIB_DMA_ChannelXTransferDirectionGet(DMA_MODULE_ID index, DMA_CHANNEL
channel);
```

Returns

- DMA_CHANNEL_TRANSFER_DIRECTION - The transfer direction indicated by the DMA_CHANNEL_TRANSFER_DIRECTION

Description

This function returns the data transfer direction of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection;
chTransferDirection = PLIB_DMA_ChannelXTransferDirectionGet (
                                                              DMA_ID_0,
                                                              DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_TRANSFER_DIRECTION PLIB_DMA_ChannelXTransferDirectionGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXTransferDirectionSelect Function

Selects the data transfer direction of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTransferDirectionSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection);
```

Returns

None.

Description

This function selects the data transfer direction of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTransferDirectionSelect (
                              DMA_ID_0,
                              DMA_CHANNEL_4,
                              DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
chTransferDirection	The transfer direction indicated by DMA_CHANNEL_TRANSFER_DIRECTION

Function

void PLIB_DMA_ChannelXTransferDirectionSelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection)

PLIB_DMA_ChannelXDisabledDisablesEvents Function

Channel start/abort events will be ignored even if the channel is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisabledDisablesEvents(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will allow the channel start/abort events to be ignored even if the channel is disabled.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDisabled function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisabledDisablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisabledDisablesEvents (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDisabledEnablesEvents Function

Channel start/abort events will be registered even if the channel is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisabledEnablesEvents(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will allow the channel register start/abort events even if the channel is disabled.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDisabled function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisabledEnablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisabledEnablesEvents (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPatternIgnoreByteDisable Function

Disables the pattern match ignore byte.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreByteDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteDisable(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXPatternIgnoreByteDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreByteEnable Function

Enables the pattern match ignore byte.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreByteEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to

automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteEnable(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXPatternIgnoreByteEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled Function

Returns the state of the pattern match ignore byte.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The pattern match ignore byte is enabled
- false - The pattern match ignore byte is disabled

Description

This function returns the state (enabled or disabled) of the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool patternIsEnabled;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
patternIsEnabled = PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreGet Function

Returns the pattern match ignore value.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_ChannelXPatternIgnoreGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint8_t - Pattern match ignore value

Description

This function returns the value of the pattern match ignore.

Remarks

This function implements an operation of the ChannelXPatternIgnore feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnore function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t patternMatch;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
patternMatch = PLIB_DMA_ChannelXPatternIgnoreGet ( DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint8_t PLIB_DMA_ChannelXPatternIgnoreGet(DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreSet Function

Sets the pattern match ignore value.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint8_t pattern);
```

Returns

None.

Description

This function sets the value of the pattern match ignore.

Remarks

This function implements an operation of the ChannelXPatternIgnore feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnore function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t patternMatch = 0x8;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreSet ( DMA_ID_0, dmaChannel,patternMatch);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
pattern	Pattern match ignore value

Function

void PLIB_DMA_ChannelXPatternIgnoreSet (DMA_MODULE_ID index,
DMA_CHANNEL channel, uint8_t pattern);

PLIB_DMA_ChannelXPatternLengthGet Function

Returns the pattern match length.

File

plib_dma.h

C

```
DMA_PATTERN_LENGTH PLIB_DMA_ChannelXPatternLengthGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- patternLen - Length of pattern match (either 1 or 2)

Description

This function returns the length of the byte matching the CHPIGN bits during a pattern match that may be ignored during the pattern match determination when the CHPIGNEN bit is set.

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = PLIB_DMA_ChannelXPatternLengthGet(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_PATTERN_LENGTH PLIB_DMA_ChannelXPatternLengthGet(DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel);

PLIB_DMA_ChannelXPatternLengthSet Function

Sets the pattern match length.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternLengthSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_PATTERN_LENGTH patternLen);
```

Returns

None.

Description

This function sets the length of the pattern match ignore to 1 or 2 bytes.

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = DMA_PATTERN_MATCH_LENGTH_1BYTE;
PLIB_DMA_ChannelXPatternLengthSet(DMA_ID_0, dmaChannel, patternLen);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
patternLen	Length of pattern match (either 1 or 2)

Function

void PLIB_DMA_ChannelXPatternLengthSet(DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,DMA_PATTERN_LENGTH patternLen)

i) CRC Module Interface Functions

PLIB_DMA_CRCAppendModeDisable Function

Disables the CRC append mode.

File

plib_dma.h

C

```
void PLIB_DMA_CRCAppendModeDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the CRC append mode. The DMA transfers data from the source through the CRC obeying WBO (DMA_MODULE_ID index, write byte order) as it writes the data to the destination.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCAppendModeDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCAppendModeDisable (DMA_MODULE_ID index)

PLIB_DMA_CRCAppendModeEnable Function

Enables the CRC append mode.

File

plib_dma.h

C

```
void PLIB_DMA_CRCAppendModeEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the CRC append mode. The DMA transfers data from the source into the CRC, but not to the destination. When a block transfer completes, the DMA writes the calculated CRC value to the location specified by the CHxDSA register.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCAppendModeEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCAppendModeEnable (DMA_MODULE_ID index)

PLIB_DMA_CRCAppendModeIsEnabled Function

Gets the enable status of the CRC append mode.

File

plib_dma.h

C

```
bool PLIB_DMA_CRCAppendModeIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - CRC append mode is enabled
- false - CRC append mode is disabled

Description

This function gets the enable status of the CRC append mode.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool DMAcrcAppendMode;
DMAcrcAppendMode = PLIB_DMA_CRCAppendModeIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_CRCAppendModeIsEnabled (DMA_MODULE_ID index)

PLIB_DMA_CRCBitOrderSelect Function

Selects the bit order for checksum calculation.

File

plib_dma.h

C

```
void PLIB_DMA_CRCBitOrderSelect(DMA_MODULE_ID index, DMA_CRC_BIT_ORDER bitOrder);
```

Returns

None.

Description

This function selects the bit order for checksum calculation.

Remarks

This function implements an operation of the CRCBitOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCBitOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCBitOrderSelect ( DMA_ID_0,DMA_CRC_BIT_ORDER_LSB );
```

Parameters

Parameters	Description
bitOrder	Specifies the bit order for CRC calculation

Function

void PLIB_DMA_CRCBitOrderSelect (DMA_MODULE_ID index,
DMA_CRC_BIT_ORDER bitOrder)

PLIB_DMA_CRCByteOrderGet Function

Gets the current byte order selected by the DMA module CRC feature.

File

plib_dma.h

C

```
DMA_CRC_BYTE_ORDER PLIB_DMA_CRCByteOrderGet(DMA_MODULE_ID index);
```

Returns

- byteOrder - One of the possible byte orders specified by DMA_CRC_BYTE_ORDER

Description

This function gets the current byte order selected by the DMA module CRC feature.

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

The WBO bit must be set to use this function.

Example

```
DMA_CRC_BYTE_ORDER byteOrder;
```

```
byteOrder = PLIB_DMA_CRCByteOrderGet ( DMA_ID_0 );
```

Function

DMA_CRC_BYTE_ORDER PLIB_DMA_CRCByteOrderGet (DMA_MODULE_ID index)

PLIB_DMA_CRCByteOrderSelect Function

Selects the byte order.

File

plib_dma.h

C

```
void PLIB_DMA_CRCByteOrderSelect(DMA_MODULE_ID index, DMA_CRC_BYTE_ORDER byteOrder);
```

Returns

None.

Description

This function selects the byte order.

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

The WBO bit must be set to use this function.

Example

```
PLIB_DMA_CRCByteOrderSelect (   DMA_ID_0,
                                DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY );
```

Parameters

Parameters	Description
byteOrder	One of the possible byte orders specified by DMA_CRC_BYTE_ORDER

Function

void PLIB_DMA_CRCByteOrderSelect (DMA_MODULE_ID index,
DMA_CRC_BYTE_ORDER byteOrder)

PLIB_DMA_CRCChannelGet Function

Returns the current DMA channel to which the CRC is assigned.

File

plib_dma.h

C

```
DMA_CHANNEL PLIB_DMA_CRCChannelGet(DMA_MODULE_ID index);
```

Returns

crcChannel - One of the possible DMA channels listed by DMA_CHANNEL

Description

This function returns the current DMA channel to which the CRC is assigned.

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCChannel function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL crcChannel;
crcChannel = PLIB_DMA_CRCChannelGet( DMA_ID_0 );
```

Function

DMA_CHANNEL PLIB_DMA_CRCChannelGet (DMA_MODULE_ID index)

PLIB_DMA_CRCChannelSelect Function

Assigns the CRC to the specified DMA channel.

File

plib_dma.h

C

```
void PLIB_DMA_CRCChannelSelect(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function assigns the CRC feature to the specified channel.

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCChannel function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCChannelSelect(  DMA_ID_0,
                            DMA_CHANNEL_5 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_CRCChannelSelect(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_CRCDataRead Function

Reads the contents of the DMA CRC data register.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_CRCDataRead(DMA_MODULE_ID index);
```

Returns

- uint32_t - 32-bit CRC data

The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the contents of the DMA CRC data register.

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCdata;
DMACRCdata = PLIB_DMA_CRCDataRead ( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_CRCDataRead (DMA_MODULE_ID index)

PLIB_DMA_CRCDataWrite Function

Writes the contents of the DMA CRC data register with the specified data.

File

plib_dma.h

C

```
void PLIB_DMA_CRCDataWrite(DMA_MODULE_ID index, uint32_t DMACRCdata);
```

Returns

None.

Description

This function writes the contents of the DMA CRC data register.

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCdata = 0x0E0E0E;
PLIB_DMA_CRCDataWrite ( DMA_ID_0, DMACRCdata );
```

Function

void PLIB_DMA_CRCDataWrite (DMA_MODULE_ID index,
uint32_t DMACRCdata)

PLIB_DMA_CRCDisable Function

Disables the DMA module CRC feature.

File

plib_dma.h

C

```
void PLIB_DMA_CRCDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA module CRC feature. The channel transfers proceed normally.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCDisable (DMA_MODULE_ID index)

PLIB_DMA_CRCEnable Function

Enables the DMA module CRC feature.

File

plib_dma.h

C

```
void PLIB_DMA_CRCEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA module CRC feature. The channel transfers are routed through the CRC.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCEnable (DMA_MODULE_ID index)

PLIB_DMA_CRCPolynomialLengthGet Function

Gets the current polynomial length.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_CRCPolynomialLengthGet(DMA_MODULE_ID index);
```

Returns

- uint8_t - Polynomial length

Description

This function gets the current polynomial length.

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCPolynomialLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t polyLength;
polyLength = PLIB_DMA_CRCPolynomialLengthGet( DMA_ID_0 );
```

Function

uint8_t PLIB_DMA_CRCPolynomialLengthGet (DMA_MODULE_ID index)

PLIB_DMA_CRCPolynomialLengthSet Function

Selects the polynomial length.

File

plib_dma.h

C

```
void PLIB_DMA_CRCPolynomialLengthSet(DMA_MODULE_ID index, uint8_t polyLength);
```

Returns

None.

Description

This function Selects the polynomial length.

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCPolynomialLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t polyLength = 0x2;
PLIB_DMA_CRCPolynomialLengthSet( DMA_ID_0, polyLength );
```

Parameters

Parameters	Description
polyLength	Polynomial length

Function

void PLIB_DMA_CRCPolynomialLengthSet (DMA_MODULE_ID index,

uint8_t polyLength)

PLIB_DMA_CRCTypeGet Function

Gets the current DMA module CRC feature type.

File

plib_dma.h

C

```
DMA_CRC_TYPE PLIB_DMA_CRCTypeGet(DMA_MODULE_ID index);
```

Returns

- CRCType - One of the possible CRC checksums listed by DMA_CRC_TYPE.

Description

This function gets the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCType function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CRC_TYPE CRCType;
CRCType = PLIB_DMA_CRCTypeGet( DMA_ID_0 );
```

Function

DMA_CRC_TYPE PLIB_DMA_CRCTypeGet (DMA_MODULE_ID index)

PLIB_DMA_CRCTypeSet Function

Selects the DMA module CRC feature type.

File

plib_dma.h

C

```
void PLIB_DMA_CRCTypeSet(DMA_MODULE_ID index, DMA_CRC_TYPE CRCType);
```

Returns

None.

Description

This function selects the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCType function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCTypeSet(DMA_ID_0,
                    DMA_CRC_IP_HEADER );
```

Parameters

Parameters	Description
CRCType	One of the possible CRC checksums listed by DMA_CRC_TYPE

Function

void PLIB_DMA_CRCTypeSet (DMA_MODULE_ID index,
DMA_CRC_TYPE CRCType)

PLIB_DMA_CRCWriteByteOrderAlter Function

The source data is written to the destination reordered as defined by the BYTO<1:0> bits.

File

plib_dma.h

C

```
void PLIB_DMA_CRCWriteByteOrderAlter(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables byte order alteration as specified by the BYTO<1:0> bits. The source data is written to the destination reordered as defined by the BYTO<1:0> bits.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCWriteByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCWriteByteOrderAlter ( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCWriteByteOrderAlter (DMA_MODULE_ID index)

PLIB_DMA_CRCWriteByteOrderMaintain Function

The source data is written to the destination unaltered.

File

plib_dma.h

C

```
void PLIB_DMA_CRCWriteByteOrderMaintain(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables byte order alteration. The source data is written to the destination unaltered.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCWriteByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCWriteByteOrderMaintain ( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCWriteByteOrderMaintain (DMA_MODULE_ID index)

PLIB_DMA_CRCXOREnableGet Function

Reads the CRC XOR register.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_CRCXOREnableGet(DMA_MODULE_ID index);
```

Returns

- uint32_t - 32-bit CRC XOR enable mask data

The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the CRC XOR register.

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCXOREnable function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCXORdata;
DMACRCXORdata = PLIB_DMA_CRCXOREnableGet ( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_CRCXOREnableGet (DMA_MODULE_ID index)

PLIB_DMA_CRCXOREnableSet Function

Writes to the CRC XOR enable register as per the specified enable mask.

File

plib_dma.h

C

```
void PLIB_DMA_CRCXOREnableSet(DMA_MODULE_ID index, uint32_t DMACRCXOREnableMask);
```

Returns

None.

Description

This function writes to the CRC XOR enable register as per the specified enable mask. Each enabled bit will be taken as input to the shift register.

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCXOREnable function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCXOREnableMask = 0x05EFFFFF;
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, DMACRCXOREnableMask );
```

Function

void PLIB_DMA_CRCXOREnableSet (DMA_MODULE_ID index,

uint32_t DMACRCXOREnableMask)

j) Global Status Functions

PLIB_DMA_CRCIsEnabled Function

Gets the enable status of the CRC feature.

File

plib_dma.h

C

```
bool PLIB_DMA_CRCIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The CRC feature is enabled
- false - The CRC feature is disabled

Description

This function gets the enable status of the CRC feature.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool DMAcrcStatus;
DMAcrcStatus = PLIB_DMA_CRCIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_CRCIsEnabled (DMA_MODULE_ID index)

PLIB_DMA_IsBusy Function

Gets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
bool PLIB_DMA_IsBusy(DMA_MODULE_ID index);
```

Returns

- true - DMA module is active
- false - DMA module is disabled and is not actively transferring data

Description

This function gets the BUSY bit of the DMA controller.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaBusyStatus;
dmaBusyStatus = PLIB_DMA_IsBusy( DMA_ID_0 );
```

Function

bool PLIB_DMA_IsBusy (DMA_MODULE_ID index)

PLIB_DMA_IsEnabled Function

Returns the DMA module enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_IsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The DMA is enabled
- false - The DMA is disabled

Description

This function returns the DMA module enable status.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_IsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_IsEnabled (DMA_MODULE_ID index)

PLIB_DMA_LastBusAccessIsRead Function

Returns true if the last DMA bus access was a read.

File

plib_dma.h

C

```
bool PLIB_DMA_LastBusAccessIsRead(DMA_MODULE_ID index);
```

Returns

- true - The last bus access was a read
- false - The last bus access was not a read

Description

This function returns true if the last DMA bus access was a read.

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsLastBusAccess function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaLastBusAccessType;
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsRead( DMA_ID_0 );
```

Function

bool PLIB_DMA_LastBusAccessIsRead (DMA_MODULE_ID index)

PLIB_DMA_LastBusAccessIsWrite Function

Returns true if the last DMA bus access was a write.

File

plib_dma.h

C

```
bool PLIB_DMA_LastBusAccessIsWrite(DMA_MODULE_ID index);
```

Returns

- true - The last bus access was a write operation
- false - The last bus access was not a write operation

Description

This function returns true if the last DMA bus access was a write operation.

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsLastBusAccess function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaLastBusAccessType;
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsWrite( DMA_ID_0 );
```

Function

bool PLIB_DMA_LastBusAccessIsWrite (DMA_MODULE_ID index)

PLIB_DMA_RecentAddressAccessed Function

Returns the address of the most recent DMA access.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_RecentAddressAccessed(DMA_MODULE_ID index);
```

Returns

- uint32_t - The most recent address accessed by the DMA

Description

This function returns the address of the most recent DMA access.

Remarks

This function implements an operation of the RecentAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsRecentAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t dmaLastAddressAccessed;
dmaLastAddressAccessed = PLIB_DMA_RecentAddressAccessed( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_RecentAddressAccessed (DMA_MODULE_ID index)

PLIB_DMA_SuspendIsEnabled Function

Returns the DMA suspend status.

File

plib_dma.h

C

```
bool PLIB_DMA_SuspendIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The DMA transfers are suspended
- false - The DMA operates normally

Description

This function returns the DMA suspend status.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaSuspendStatus;
dmaSuspendStatus = PLIB_DMA_SuspendIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_SuspendIsEnabled (DMA_MODULE_ID index)

k) Channel Status Functions

PLIB_DMA_ChannelXAutoIsEnabled Function

Returns the channel automatic enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXAutoIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - Channel automatic enable is on
- false - Channel automatic enable is off

Description

This function returns the channel automatic enable status.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific

device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool ChAutoEnableStatus;
ChAutoEnableStatus = PLIB_DMA_ChannelXAutoIsEnabled(DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXAutoIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBufferedDataIsWritten Function

Returns the buffered data write status for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXBufferedDataIsWritten(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The content of the DMA buffer has not been written to the location specified in the destination/source address or in Null Write mode
- false - The content of the DMA buffer has been written to the location specified in the destination/source address or in Null Write mode

Description

This function returns the buffered data write status for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chBuffWriteStatus;
chBuffWriteStatus = PLIB_DMA_ChannelXBufferedDataIsWritten( DMA_ID_0,
                                                            DMA_CHANNEL_3 );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXBufferedDataIsWritten (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyIsBusy Function

Returns the busy status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXBusyIsBusy(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The channel is active or has been enabled
- false - The channel is inactive or has been disabled

Description

This function returns the busy status of the specified channel.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool chBusyStatus;
chBusyStatus = PLIB_DMA_ChannelXBusyIsBusy ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXBusyIsBusy (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainIsEnabled Function

Returns the chain status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXChainIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The channel chain is on for this channel
- false - The channel chain is off for this channel

Description

This function returns the chain status of the specified channel.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool ChchainStatus;
ChchainStatus = PLIB_DMA_ChannelXChainIsEnabled( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXChainIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXCollisionStatus Function

Returns the status of the specified collision type for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXCollisionStatus(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_COLLISION
collisonType);
```

Returns

- true - A collision specified by collisonType was detected
- false - No collision of type collisonType was detected

Description

This function returns the status of the specified collision type for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool memWriteCollisionStatus;
memWriteCollisionStatus = PLIB_DMA_ChannelXMemoryWriteCollisionStatus(
                                        DMA_ID_0,
                                        DMA_CHANNEL_3,
                                        DMA_CHANNEL_COLLISION_MEMORY );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
collisonType	Collision type listed by DMA_CHANNEL_COLLISION

Function

bool PLIB_DMA_ChannelXCollisionStatus (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_COLLISION collisonType)

PLIB_DMA_ChannelXEventIsDetected Function

Returns the event status on the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXEventIsDetected(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - An event was detected
- false - No events were detected

Description

This function returns the event status on the specified channel.

Remarks

This function implements an operation of the ChannelXEvent feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_DMA_ExistsChannelXEvent function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool channeleventStatus;
channeleventStatus = PLIB_DMA_ChannelXEventIsDetected(  DMA_ID_0,
                                                        DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXEventIsDetected (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXIsEnabled Function

Return the enable status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The specified DMA channel is enabled
- false - The specified DMA channel is disabled

Description

This function will return the enable status of the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool chEnableStatus;
chEnableStatus = PLIB_DMA_ChannelXIsEnabled ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXNullWriteModeIsEnabled Function

Returns the enable status of the Null Write mode for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXNullWriteModeIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - Null write mode is enabled for this channel
- false - Null write mode is disabled for this channel

Description

This function returns the enable status of the Null Write mode for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chNullWriteStatus;
chNullWriteStatus = PLIB_DMA_ChannelXNullWriteModeIsEnabled (
                                                              DMA_ID_0,
                                                              DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXNullWriteModeIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPingPongModeGet Function

Returns the Ping-Pong mode status for the specified channel.

File

plib_dma.h

C

```
DMA_PING_PONG_MODE PLIB_DMA_ChannelXPingPongModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

mode - One of the possible Ping-Pong modes

Description

This function returns the Ping-Pong mode status for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_PING_PONG_MODE chPingPongStatus;
chPingPongStatus = PLIB_DMA_ChannelXPingPongModeGet(DMA_ID_0,
                                                    DMA_CHANNEL_3 );
if (DMA_PING_PONG_SECONDARY == chPingPongStatus)
    {
        \Application
    }
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

DMA_PING_PONG_MODE PLIB_DMA_ChannelXPingPongModeGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelBitsGet Function

Returns the DMA channel bits.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_ChannelBitsGet(DMA_MODULE_ID index);
```

Returns

- uint8_t - DMA channel bits

Description

This function returns the channel bits.

Remarks

This function implements an operation of the ChannelBits feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelBits function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t dmaChBits;
dmaChBits = PLIB_DMA_ChannelBitsGet( DMA_ID_0 );
```

Function

uint8_t PLIB_DMA_ChannelBitsGet (DMA_MODULE_ID index)

I) Feature Existence Functions

PLIB_DMA_ExistsAbortTransfer Function

Identifies whether the AbortTransfer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsAbortTransfer(DMA_MODULE_ID index);
```

Returns

- true - The AbortTransfer feature is supported on the device
- false - The AbortTransfer feature is not supported on the device

Description

This function identifies whether the AbortTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_AbortTransferSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsAbortTransfer(DMA_MODULE_ID index)

PLIB_DMA_ExistsBusy Function

Identifies whether the Busy feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsBusy(DMA_MODULE_ID index);
```

Returns

- true - The Busy feature is supported on the device
- false - The Busy feature is not supported on the device

Description

This function identifies whether the Busy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_BusyActiveSet
- PLIB_DMA_BusyActiveReset
- PLIB_DMA_IsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsBusy(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelBits Function

Identifies whether the ChannelBits feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelBits(DMA_MODULE_ID index);
```

Returns

- true - The ChannelBits feature is supported on the device
- false - The ChannelBits feature is not supported on the device

Description

This function identifies whether the ChannelBits feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelBitsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelBits(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelX Function

Identifies whether the ChannelX feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelX(DMA_MODULE_ID index);
```

Returns

- true - The ChannelX feature is supported on the device
- false - The ChannelX feature is not supported on the device

Description

This function identifies whether the ChannelX feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXEnable
- PLIB_DMA_ChannelXIsEnabled
- PLIB_DMA_ChannelXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelX(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXAbortIRQ Function

Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXAbortIRQ(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXAbortIRQ feature is supported on the device
- false - The ChannelXAbortIRQ feature is not supported on the device

Description

This function identifies whether the ChannelXAbortIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXAbortIRQSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXAbortIRQ(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXAuto Function

Identifies whether the ChannelXAuto feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXAuto(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXAuto feature is supported on the device
- false - The ChannelXAuto feature is not supported on the device

Description

This function identifies whether the ChannelXAuto feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXAutoEnable
- PLIB_DMA_ChannelXAutoDisable
- PLIB_DMA_ChannelXAutoIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXAuto(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXBusy Function

Identifies whether the ChannelXBusy feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXBusy(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXBusy feature is supported on the device
- false - The ChannelXBusy feature is not supported on the device

Description

This function identifies whether the ChannelXBusy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXBusyActiveSet
- PLIB_DMA_ChannelXBusyInActiveSet
- PLIB_DMA_ChannelXBusyIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXBusy(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXCellProgressPointer Function

Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXCellProgressPointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXCellProgressPointer feature is supported on the device
- false - The ChannelXCellProgressPointer feature is not supported on the device

Description

This function identifies whether the ChannelXCellProgressPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXCellProgressPointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXCellProgressPointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXCellSize Function

Identifies whether the ChannelXCellSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXCellSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXCellSize feature is supported on the device
- false - The ChannelXCellSize feature is not supported on the device

Description

This function identifies whether the ChannelXCellSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXCellSizeGet
- PLIB_DMA_ChannelXCellSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXCellSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXChain Function

Identifies whether the ChannelXChain feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXChain(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXChain feature is supported on the device
- false - The ChannelXChain feature is not supported on the device

Description

This function identifies whether the ChannelXChain feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXChainToLower
- PLIB_DMA_ChannelXChainToHigher

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXChain(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXChainEnbl Function

Identifies whether the ChannelXChainEnbl feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXChainEnbl(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXChainEnbl feature is supported on the device
- false - The ChannelXChainEnbl feature is not supported on the device

Description

This function identifies whether the ChannelXChainEnbl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXChainEnable
- PLIB_DMA_ChannelXChainDisable
- PLIB_DMA_ChannelXChainIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXChainEnbl(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationPointer Function

Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationPointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationPointer feature is supported on the device
- false - The ChannelXDestinationPointer feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXDestinationPointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationPointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationSize Function

Identifies whether the ChannelXDestinationSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationSize feature is supported on the device
- false - The ChannelXDestinationSize feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDestinationSizeGet
- PLIB_DMA_ChannelXDestinationSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationStartAddress Function

Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationStartAddress(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationStartAddress feature is supported on the device
- false - The ChannelXDestinationStartAddress feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDestinationStartAddressGet
- PLIB_DMA_ChannelXDestinationStartAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationStartAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDisabled Function

Identifies whether the ChannelXDisabled feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDisabled(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDisabled feature is supported on the device
- false - The ChannelXDisabled feature is not supported on the device

Description

This function identifies whether the ChannelXDisabled feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDisabledEnablesEvents
- PLIB_DMA_ChannelXDisabledDisablesEvents

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDisabled(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXEvent Function

Identifies whether the ChannelXEvent feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXEvent(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXEvent feature is supported on the device
- false - The ChannelXEvent feature is not supported on the device

Description

This function identifies whether the ChannelXEvent feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXEventIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXEvent(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXINTSource Function

Identifies whether the ChannelXINTSource feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXINTSource(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXINTSource feature is supported on the device
- false - The ChannelXINTSource feature is not supported on the device

Description

This function identifies whether the ChannelXINTSource feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXINTSourceEnable
- PLIB_DMA_ChannelXINTSourceDisable
- PLIB_DMA_ChannelXINTSourceIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXINTSource(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXINTSourceFlag Function

Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXINTSourceFlag(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXINTSourceFlag feature is supported on the device
- false - The ChannelXINTSourceFlag feature is not supported on the device

Description

This function identifies whether the ChannelXINTSourceFlag feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXINTSourceFlagGet
- PLIB_DMA_ChannelXINTSourceFlagSet
- PLIB_DMA_ChannelXINTSourceFlagClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXINTSourceFlag(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternData Function

Identifies whether the ChannelXPatternData feature exists on the DMA module

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternData(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternData feature is supported on the device
- false - The ChannelXPatternData feature is not supported on the device

Description

This function identifies whether the ChannelXPatternData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternDataGet
- PLIB_DMA_ChannelXPatternDataSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternData(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternIgnore Function

Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternIgnore(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternIgnore feature is supported on the device
- false - The ChannelXPatternIgnore feature is not supported on the device

Description

This function identifies whether the ChannelXPatternIgnore feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternIgnoreSet
- PLIB_DMA_ChannelXPatternIgnoreGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternIgnore(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternIgnoreByte Function

Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternIgnoreByte(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternIgnoreByte feature is supported on the device
- false - The ChannelXPatternIgnoreByte feature is not supported on the device

Description

This function identifies whether the ChannelXPatternIgnoreByte feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternIgnoreByteEnable
- PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled
- PLIB_DMA_ChannelXPatternIgnoreByteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternIgnoreByte(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternLength Function

Identifies whether the ChannelXPatternLength feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternLength(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternLength feature is supported on the device
- false - The ChannelXPatternLength feature is not supported on the device

Description

This function identifies whether the ChannelXPatternLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternLengthSet
- PLIB_DMA_ChannelXPatternLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternLength(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPriority Function

Identifies whether the ChannelXPriority feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPriority(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPriority feature is supported on the device
- false - The ChannelXPriority feature is not supported on the device

Description

This function identifies whether the ChannelXPriority feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPrioritySelect
- PLIB_DMA_ChannelXPriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPriority(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourcePointer Function

Identifies whether the ChannelXSourcePointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourcePointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourcePointer feature is supported on the device
- false - The ChannelXSourcePointer feature is not supported on the device

Description

This function identifies whether the ChannelXSourcePointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXSourcePointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourcePointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourceSize Function

Identifies whether the ChannelXSourceSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourceSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourceSize feature is supported on the device
- false - The ChannelXSourceSize feature is not supported on the device

Description

This function identifies whether the ChannelXSourceSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXSourceSizeGet
- PLIB_DMA_ChannelXSourceSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourceSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourceStartAddress Function

Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourceStartAddress(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourceStartAddress feature is supported on the device
- false - The ChannelXSourceStartAddress feature is not supported on the device

Description

This function identifies whether the ChannelXSourceStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXSourceStartAddressGet
- PLIB_DMA_ChannelXSourceStartAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourceStartAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXStartIRQ Function

Identifies whether the ChannelXStartIRQ feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXStartIRQ(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXStartIRQ feature is supported on the device
- false - The ChannelXStartIRQ feature is not supported on the device

Description

This function identifies whether the ChannelXStartIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXStartIRQSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXStartIRQ(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXTrigger Function

Identifies whether the ChannelXTrigger feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXTrigger(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXTrigger feature is supported on the device
- false - The ChannelXTrigger feature is not supported on the device

Description

This function identifies whether the ChannelXTrigger feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXTriggerEnable
- PLIB_DMA_ChannelXTriggerIsEnabled
- PLIB_DMA_ChannelXTriggerDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXTrigger(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRC Function

Identifies whether the CRC feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRC(DMA_MODULE_ID index);
```

Returns

- true - The CRC feature is supported on the device
- false - The CRC feature is not supported on the device

Description

This function identifies whether the CRC feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCEnable
- PLIB_DMA_CRCDisable
- PLIB_DMA_CRCIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRC(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCAppendMode Function

Identifies whether the CRCAppendMode feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCAppendMode(DMA_MODULE_ID index);
```

Returns

- true - The CRCAppendMode feature is supported on the device
- false - The CRCAppendMode feature is not supported on the device

Description

This function identifies whether the CRCAppendMode feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCAppendModeEnable
- PLIB_DMA_CRCAppendModeDisable
- PLIB_DMA_CRCAppendModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCAppendMode(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCBitOrder Function

Identifies whether the CRCBitOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCBitOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCBitOrder feature is supported on the device
- false - The CRCBitOrder feature is not supported on the device

Description

This function identifies whether the CRCBitOrder feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_CRCBitOrderSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCBitOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCByteOrder Function

Identifies whether the CRCByteOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCByteOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCByteOrder feature is supported on the device
- false - The CRCByteOrder feature is not supported on the device

Description

This function identifies whether the CRCByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCByteOrderSelect
- PLIB_DMA_CRCByteOrderGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCByteOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCChannel Function

Identifies whether the CRCChannel feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCChannel(DMA_MODULE_ID index);
```

Returns

- true - The CRCChannel feature is supported on the device
- false - The CRCChannel feature is not supported on the device

Description

This function identifies whether the CRCChannel feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCChannelSelect
- PLIB_DMA_CRCChannelGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCChannel(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCData Function

Identifies whether the CRCData feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCData(DMA_MODULE_ID index);
```

Returns

- true - The CRCData feature is supported on the device
- false - The CRCData feature is not supported on the device

Description

This function identifies whether the CRCData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCDataRead
- PLIB_DMA_CRCDataWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCData(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCPolynomialLength Function

Identifies whether the CRCPolynomialLength feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCPolynomialLength(DMA_MODULE_ID index);
```

Returns

- true - The CRCPolynomialLength feature is supported on the device
- false - The CRCPolynomialLength feature is not supported on the device

Description

This function identifies whether the CRCPolynomialLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCPolynomialLengthSet
- PLIB_DMA_CRCPolynomialLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCPolynomialLength(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCType Function

Identifies whether the CRCType feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCType(DMA_MODULE_ID index);
```

Returns

- true - The CRCType feature is supported on the device
- false - The CRCType feature is not supported on the device

Description

This function identifies whether the CRCType feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCTypeGet
- PLIB_DMA_CRCTypeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCType(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCWriteByteOrder Function

Identifies whether the CRCWriteByteOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCWriteByteOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCWriteByteOrder feature is supported on the device
- false - The CRCWriteByteOrder feature is not supported on the device

Description

This function identifies whether the CRCWriteByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCWriteByteOrderAlter
- PLIB_DMA_CRCWriteByteOrderMaintain

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCWriteByteOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCXOREnable Function

Identifies whether the CRCXOREnable feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCXOREnable(DMA_MODULE_ID index);
```

Returns

- true - The CRCXOREnable feature is supported on the device
- false - The CRCXOREnable feature is not supported on the device

Description

This function identifies whether the CRCXOREnable feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCXOREnableSet
- PLIB_DMA_CRCXOREnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCXOREnable(DMA_MODULE_ID index)

PLIB_DMA_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsEnableControl(DMA_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_Enable
- PLIB_DMA_Disable
- PLIB_DMA_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsEnableControl(DMA_MODULE_ID index)

PLIB_DMA_ExistsLastBusAccess Function

Identifies whether the LastBusAccess feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsLastBusAccess(DMA_MODULE_ID index);
```

Returns

- true - The LastBusAccess feature is supported on the device
- false - The LastBusAccess feature is not supported on the device

Description

This function identifies whether the LastBusAccess feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_LastBusAccessIsRead
- PLIB_DMA_LastBusAccessIsWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsLastBusAccess(DMA_MODULE_ID index)

PLIB_DMA_ExistsRecentAddress Function

Identifies whether the RecentAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsRecentAddress(DMA_MODULE_ID index);
```

Returns

- true - The RecentAddress feature is supported on the device
- false - The RecentAddress feature is not supported on the device

Description

This function identifies whether the RecentAddress feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_RecentAddressAccessed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsRecentAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsStartTransfer Function

Identifies whether the StartTransfer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsStartTransfer(DMA_MODULE_ID index);
```

Returns

- true - The StartTransfer feature is supported on the device
- false - The StartTransfer feature is not supported on the device

Description

This function identifies whether the StartTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_StartTransferSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsStartTransfer(DMA_MODULE_ID index)

PLIB_DMA_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsStopInIdle(DMA_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_StopInIdleEnable
- PLIB_DMA_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsStopInIdle(DMA_MODULE_ID index)

PLIB_DMA_ExistsSuspend Function

Identifies whether the Suspend feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsSuspend(DMA_MODULE_ID index);
```

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Description

This function identifies whether the Suspend feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_SuspendEnable
- PLIB_DMA_SuspendDisable
- PLIB_DMA_SuspendIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsSuspend(DMA_MODULE_ID index)

m) Data Types and Constants

DMA_ADDRESS_OFFSET_TYPE Enumeration

Lists the possible DMA address offsets.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESS_OFFSET_PRIMARY,
  DMA_ADDRESS_OFFSET_SECONDARY
} DMA_ADDRESS_OFFSET_TYPE;
```

Members

Members	Description
DMA_ADDRESS_OFFSET_PRIMARY	address offset-A
DMA_ADDRESS_OFFSET_SECONDARY	address offset-B

Description

DMA address offsets

This enumeration lists all the possible DMA address offsets.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL Enumeration

Lists the possible DMA channels available.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_0,
  DMA_CHANNEL_1,
  DMA_CHANNEL_2,
  DMA_CHANNEL_3,
  DMA_CHANNEL_4,
  DMA_CHANNEL_5,
  DMA_CHANNEL_6,
  DMA_CHANNEL_7
} DMA_CHANNEL;
```

Members

Members	Description
DMA_CHANNEL_0	DMA Channel 0
DMA_CHANNEL_1	DMA Channel 1

DMA_CHANNEL_2	DMA Channel 2
DMA_CHANNEL_3	DMA Channel 3
DMA_CHANNEL_4	DMA Channel 4
DMA_CHANNEL_5	DMA Channel 5
DMA_CHANNEL_6	DMA Channel 6
DMA_CHANNEL_7	DMA Channel 7

Description

DMA channel

This enumeration lists all the possible DMA channels available.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_ADDRESSING_MODE Enumeration

Lists the possible channel addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT,
  DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT,
  DMA_ADDRESSING_PERIPHERAL_INDIRECT
} DMA_CHANNEL_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT	register indirect with post increment
DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT	register indirect without post increment
DMA_ADDRESSING_PERIPHERAL_INDIRECT	peripheral indirect addressing

Description

DMA channel addressing modes

This enumeration lists all the possible channel addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_COLLISION Enumeration

Lists the possible channel collision types.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_COLLISION_MEMORY,
  DMA_CHANNEL_COLLISION_PERIPHERAL,
  DMA_CHANNEL_COLLISION_TRANSFER_REQUEST
} DMA_CHANNEL_COLLISION;
```

Members

Members	Description
DMA_CHANNEL_COLLISION_MEMORY	Memory Write Collision
DMA_CHANNEL_COLLISION_PERIPHERAL	Peripheral Write collision
DMA_CHANNEL_COLLISION_TRANSFER_REQUEST	Transfer request collision

Description

DMA channel collision types

This enumeration lists all the possible channel collision types.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

DMA_CHANNEL_DATA_SIZE Enumeration

Lists the possible data sizes for the DMA channel.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_DATA_8,
  DMA_CHANNEL_DATA_16
} DMA_CHANNEL_DATA_SIZE;
```

Members

Members	Description
DMA_CHANNEL_DATA_8	8 bit data size
DMA_CHANNEL_DATA_16	16 bit data size

Description

DMA channel data size

This enumeration lists all the possible data sizes for the DMA channels.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_PRIORITY Enumeration

Lists the possible channel priorities.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_PRIORITY_0,
  DMA_CHANNEL_PRIORITY_1,
  DMA_CHANNEL_PRIORITY_2,
  DMA_CHANNEL_PRIORITY_3,
  DMA_CHANNEL_ROUND_ROBIN,
  DMA_CHANNEL_FIXED_PRIORITY
} DMA_CHANNEL_PRIORITY;
```

Members

Members	Description
DMA_CHANNEL_PRIORITY_0	priority 0
DMA_CHANNEL_PRIORITY_1	priority 1
DMA_CHANNEL_PRIORITY_2	priority 2
DMA_CHANNEL_PRIORITY_3	priority 3
DMA_CHANNEL_ROUND_ROBIN	round-robin scheme
DMA_CHANNEL_FIXED_PRIORITY	fixed priority scheme

Description

DMA channel priority types

This enumeration lists all the possible channel priorities.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_TRANSFER_DIRECTION Enumeration

Lists the possible data transfer directions.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL,
  DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY,
  DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY,
  DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL
} DMA_CHANNEL_TRANSFER_DIRECTION;
```

Members

Members	Description
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL	DMA transfer happens from the memory to peripheral
DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY	DMA transfer happens from the peripheral to memory
DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY	DMA transfer happens from the memory to memory
DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL	DMA transfer happens from the peripheral to peripheral

Description

DMA channel data transfer direction

This enumeration lists all the possible data transfer directions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_TRIGGER_TYPE Enumeration

Lists the possible DMA channel triggers.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_TRIGGER_TRANSFER_START,
  DMA_CHANNEL_TRIGGER_TRANSFER_ABORT,
  DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT
} DMA_CHANNEL_TRIGGER_TYPE;
```

Members

Members	Description
DMA_CHANNEL_TRIGGER_TRANSFER_START	Trigger for DMA transfer start
DMA_CHANNEL_TRIGGER_TRANSFER_ABORT	Trigger for DMA transfer abort
DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT	Trigger for DMA transfer abort via pattern match

Description

DMA trigger types

This enumeration lists all the possible DMA channel triggers.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_BIT_ORDER Enumeration

Lists the possible CRC calculation bit orders

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_BIT_ORDER_LSB,
  DMA_CRC_BIT_ORDER_MSB
} DMA_CRC_BIT_ORDER;
```

Members

Members	Description
DMA_CRC_BIT_ORDER_LSB	CRC is calculated least significant bit first
DMA_CRC_BIT_ORDER_MSB	CRC is calculated most significant bit first

Description

DMA CRC calculation bit order

This enumeration lists all the possible CRC calculation bit orders

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_BYTE_ORDER Enumeration

Lists the possible byte orders.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_BYTEORDER_NO_SWAPPING,
  DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY,
  DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY,
  DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY
} DMA_CRC_BYTE_ORDER;
```

Members

Members	Description
DMA_CRC_BYTEORDER_NO_SWAPPING	No swapping, Source byte order
DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY	Endian byte swap on word boundaries (reverse source byte order)
DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY	Swap half-words on word boundaries (reverse source half-word order with source byte order per half-word)
DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY	Endian byte swap on half-word boundaries (reverse source half-word order with reverse source byte order per half-word)

Description

CRC Byte order type

This enumeration lists all the possible byte orders handled by CRC module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_TYPE Enumeration

Lists the possible checksums.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_IP_HEADER,
  DMA_CRC_LFSR
} DMA_CRC_TYPE;
```

Members

Members	Description
DMA_CRC_IP_HEADER	CRC module will calculate IP header checksum
DMA_CRC_LFSR	CRC module will calculate Linear Feedback Shift Registers checksum

Description

CRC type

This enumeration lists all the possible checksums handled by CRC module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_DESTINATION_ADDRESSING_MODE Enumeration

Lists the possible destination addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_DESTINATION_UNCHANGED,
  DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT
} DMA_DESTINATION_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_DESTINATION_UNCHANGED	DMADST remains unchanged after a transfer completion
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE	DMADST is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE	DMADST is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT	DMADST is used in peripheral indirect addressing and remains unchanged

Description

DMA destination addressing modes

This enumeration lists all the possible destination addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_INT_TYPE Enumeration

Lists the possible Interrupt types for a particular channel-X

File

help_plib_dma.h

C

```
typedef enum {
  DMA_INT_ADDRESS_ERROR,
```

```
  DMA_INT_TRANSFER_ABORT,
  DMA_INT_CELL_TRANSFER_COMPLETE,
  DMA_INT_BLOCK_TRANSFER_COMPLETE,
  DMA_INT_DESTINATION_HALF_FULL,
  DMA_INT_DESTINATION_DONE,
  DMA_INT_SOURCE_HALF_EMPTY,
  DMA_INT_SOURCE_DONE,
  DMA_INT_COUNT_HALF_DONE,
  DMA_INT_CHANNEL_OVERRUN,
  DMA_INT_COMPLETE,
  DMA_INT_LOW_ADDRESS_LIMIT,
  DMA_INT_HIGH_ADDRESS_LIMIT
} DMA_INT_TYPE;
```

Members

Members	Description
DMA_INT_ADDRESS_ERROR	Channel address error interrupt
DMA_INT_TRANSFER_ABORT	channel transfer abort interrupt
DMA_INT_CELL_TRANSFER_COMPLETE	channel cell transfer complete
DMA_INT_BLOCK_TRANSFER_COMPLETE	Channel block transfer complete
DMA_INT_DESTINATION_HALF_FULL	Channel destination half full interrupt
DMA_INT_DESTINATION_DONE	Channel destination done interrupt
DMA_INT_SOURCE_HALF_EMPTY	Channel source half empty interrupt
DMA_INT_SOURCE_DONE	Channel source done interrupt
DMA_INT_COUNT_HALF_DONE	DMA count has reached its halfway
DMA_INT_CHANNEL_OVERRUN	channel overrun. channel is triggered while it is still completing the operation based on the previous trigger
DMA_INT_COMPLETE	DMA complete operation interrupt
DMA_INT_LOW_ADDRESS_LIMIT	DMA low address limit interrupt
DMA_INT_HIGH_ADDRESS_LIMIT	DMA high address limit interrupt

Description

DMA channel X interrupt types

This enumeration lists all the possible interrupt types for a channel. Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts).

Remarks

Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts). This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_MODULE_ID Enumeration

Possible instances of the DMA module.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ID_0
} DMA_MODULE_ID;
```

Members

Members	Description
DMA_ID_0	first instance of the DMA

Description

DMA module ID

This data type defines the possible instances of the DMA module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_PATTERN_LENGTH Enumeration

Gives the DMA pattern length

File

help_plib_dma.h

C

```
typedef enum {
  DMA_PATTERN_MATCH_LENGTH_1BYTE,
  DMA_PATTERN_MATCH_LENGTH_2BYTES
} DMA_PATTERN_LENGTH;
```

Members

Members	Description
DMA_PATTERN_MATCH_LENGTH_1BYTE	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 1 BYTE
DMA_PATTERN_MATCH_LENGTH_2BYTES	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 2 BYTES

Description

DMA pattern length

This enumeration gives the length of the pattern to be matched with CHPIGN<7:0> in DCHxCON<31:24.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_PING_PONG_MODE Enumeration

Lists the possible ping-pong modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_PING_PONG_PRIMARY,
  DMA_PING_PONG_SECONDARY
} DMA_PING_PONG_MODE;
```

Members

Members	Description
DMA_PING_PONG_PRIMARY	DMAxSTA is selected in the ping-pong mode
DMA_PING_PONG_SECONDARY	DMAxSTB is selected in the ping-pong mode

Description

DMA PING-PONG modes

This enumeration lists all the possible ping-pong modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_SOURCE_ADDRESSING_MODE Enumeration

Lists the possible source addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_SOURCE_UNCHANGED,
  DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE,
```

```
  DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT
} DMA_SOURCE_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_SOURCE_UNCHANGED	DMASRC remains unchanged after a transfer completion
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE	DMASRC is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE	DMASRC is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT	DMASRC is used in peripheral indirect addressing and remains unchanged

Description

DMA source addressing modes

This enumeration lists all the possible source addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_TRANSFER_MODE Enumeration

Lists the possible DMA transfer/operating modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_MODE_ONE_SHOT,
  DMA_MODE_REPEATED_ONE_SHOT,
  DMA_MODE_CONTINUOUS,
  DMA_MODE_REPEATED_CONTINUOUS,
  DMA_MODE_ONE_SHOT_PING_PONG_ENABLED,
  DMA_MODE_ONE_SHOT_PING_PONG_DISABLED,
  DMA_MODE_CONTINUOUS_PING_PONG_DISABLED,
  DMA_MODE_CONTINUOUS_PING_PONG_ENABLED
} DMA_TRANSFER_MODE;
```

Members

Members	Description
DMA_MODE_ONE_SHOT	one-shot transfer
DMA_MODE_REPEATED_ONE_SHOT	repeated one-shot transfer
DMA_MODE_CONTINUOUS	continuous transfer
DMA_MODE_REPEATED_CONTINUOUS	repeated continuous transfer
DMA_MODE_ONE_SHOT_PING_PONG_ENABLED	one-shot transfer with ping-pong buffers
DMA_MODE_ONE_SHOT_PING_PONG_DISABLED	one-shot transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_DISABLED	continuous transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_ENABLED	continuous transfer with ping-pong buffers

Description

DMA transfer/operating mode

This enumeration lists the possible DMA transfer/operating modes.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

DMA_TRIGGER_SOURCE Enumeration

Lists the possible DMA trigger sources.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_TRIGGER_TIMER_CORE,
  DMA_TRIGGER_SOFTWARE_0,
  DMA_TRIGGER_SOFTWARE_1,
  DMA_TRIGGER_EXTERNAL_0,
  DMA_TRIGGER_EXTERNAL_1,
  DMA_TRIGGER_EXTERNAL_2,
  DMA_TRIGGER_EXTERNAL_3,
  DMA_TRIGGER_EXTERNAL_4,
  DMA_TRIGGER_TIMER_1,
  DMA_TRIGGER_TIMER_2,
  DMA_TRIGGER_TIMER_3,
  DMA_TRIGGER_TIMER_4,
  DMA_TRIGGER_TIMER_5,
  DMA_TRIGGER_INPUT_CAPTURE_1,
  DMA_TRIGGER_INPUT_CAPTURE_2,
  DMA_TRIGGER_INPUT_CAPTURE_3,
  DMA_TRIGGER_INPUT_CAPTURE_4,
  DMA_TRIGGER_INPUT_CAPTURE_5,
  DMA_TRIGGER_INPUT_CAPTURE_1_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_2_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_3_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_4_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_5_ERROR,
  DMA_TRIGGER_OUTPUT_COMPARE_1,
  DMA_TRIGGER_OUTPUT_COMPARE_2,
  DMA_TRIGGER_OUTPUT_COMPARE_3,
  DMA_TRIGGER_OUTPUT_COMPARE_4,
  DMA_TRIGGER_OUTPUT_COMPARE_5,
  DMA_TRIGGER_SPI_1_ERROR,
  DMA_TRIGGER_SPI_1_RECEIVE,
  DMA_TRIGGER_SPI_1_TRANSMIT,
  DMA_TRIGGER_SPI_1A_ERROR,
  DMA_TRIGGER_SPI_1A_RECEIVE,
  DMA_TRIGGER_SPI_1A_TRANSMIT,
  DMA_TRIGGER_SPI_2_ERROR,
  DMA_TRIGGER_SPI_2_RECEIVE,
  DMA_TRIGGER_SPI_2_TRANSMIT,
  DMA_TRIGGER_SPI_2A_ERROR,
  DMA_TRIGGER_SPI_2A_RECEIVE,
  DMA_TRIGGER_SPI_2A_TRANSMIT,
  DMA_TRIGGER_SPI_3_ERROR,
  DMA_TRIGGER_SPI_3_RECEIVE,
  DMA_TRIGGER_SPI_3_TRANSMIT,
  DMA_TRIGGER_SPI_3A_ERROR,
  DMA_TRIGGER_SPI_3A_RECEIVE,
  DMA_TRIGGER_SPI_3A_TRANSMIT,
  DMA_TRIGGER_SPI_4_ERROR,
  DMA_TRIGGER_SPI_4_RECEIVE,
  DMA_TRIGGER_SPI_4_TRANSMIT,
  DMA_TRIGGER_I2C_1_ERROR,
  DMA_TRIGGER_I2C_1_SLAVE,
  DMA_TRIGGER_I2C_1_MASTER,
  DMA_TRIGGER_I2C_1A_ERROR,
  DMA_TRIGGER_I2C_1A_SLAVE,
  DMA_TRIGGER_I2C_1A_MASTER,
  DMA_TRIGGER_I2C_2_ERROR,
  DMA_TRIGGER_I2C_2_SLAVE,
  DMA_TRIGGER_I2C_2_MASTER,
  DMA_TRIGGER_I2C_2A_ERROR,
  DMA_TRIGGER_I2C_2A_SLAVE,
  DMA_TRIGGER_I2C_2A_MASTER,
  DMA_TRIGGER_I2C_3_ERROR,
  DMA_TRIGGER_I2C_3_SLAVE,
  DMA_TRIGGER_I2C_3_MASTER,
  DMA_TRIGGER_I2C_3A_ERROR,
  DMA_TRIGGER_I2C_3A_SLAVE,
  DMA_TRIGGER_I2C_3A_MASTER,
  DMA_TRIGGER_I2C_4_ERROR,
  DMA_TRIGGER_I2C_4_SLAVE,
  DMA_TRIGGER_I2C_4_MASTER,
  DMA_TRIGGER_I2C_5_ERROR,
  DMA_TRIGGER_I2C_5_SLAVE,
```

```
    DMA_TRIGGER_I2C_5_MASTER,
    DMA_TRIGGER_USART_1_ERROR,
    DMA_TRIGGER_USART_1_RECEIVE,
    DMA_TRIGGER_USART_1_TRANSMIT,
    DMA_TRIGGER_USART_1A_ERROR,
    DMA_TRIGGER_USART_1A_RECEIVE,
    DMA_TRIGGER_USART_1A_TRANSMIT,
    DMA_TRIGGER_USART_1B_ERROR,
    DMA_TRIGGER_USART_1B_RECEIVE,
    DMA_TRIGGER_USART_1B_TRANSMIT,
    DMA_TRIGGER_USART_2_ERROR,
    DMA_TRIGGER_USART_2_RECEIVE,
    DMA_TRIGGER_USART_2_TRANSMIT,
    DMA_TRIGGER_USART_2A_ERROR,
    DMA_TRIGGER_USART_2A_RECEIVE,
    DMA_TRIGGER_USART_2A_TRANSMIT,
    DMA_TRIGGER_USART_2B_ERROR,
    DMA_TRIGGER_USART_2B_RECEIVE,
    DMA_TRIGGER_USART_2B_TRANSMIT,
    DMA_TRIGGER_USART_3_ERROR,
    DMA_TRIGGER_USART_3_RECEIVE,
    DMA_TRIGGER_USART_3_TRANSMIT,
    DMA_TRIGGER_USART_3A_ERROR,
    DMA_TRIGGER_USART_3A_RECEIVE,
    DMA_TRIGGER_USART_3A_TRANSMIT,
    DMA_TRIGGER_USART_3B_ERROR,
    DMA_TRIGGER_USART_3B_RECEIVE,
    DMA_TRIGGER_USART_3B_TRANSMIT,
    DMA_TRIGGER_USART_4_ERROR,
    DMA_TRIGGER_USART_4_RECEIVE,
    DMA_TRIGGER_USART_4_TRANSMIT,
    DMA_TRIGGER_USART_5_ERROR,
    DMA_TRIGGER_USART_5_RECEIVE,
    DMA_TRIGGER_USART_5_TRANSMIT,
    DMA_TRIGGER_USART_6_ERROR,
    DMA_TRIGGER_USART_6_RECEIVE,
    DMA_TRIGGER_USART_6_TRANSMIT,
    DMA_TRIGGER_CHANGE_NOTICE,
    DMA_TRIGGER_CHANGE_NOTICE_A,
    DMA_TRIGGER_CHANGE_NOTICE_B,
    DMA_TRIGGER_CHANGE_NOTICE_C,
    DMA_TRIGGER_CHANGE_NOTICE_D,
    DMA_TRIGGER_CHANGE_NOTICE_E,
    DMA_TRIGGER_CHANGE_NOTICE_F,
    DMA_TRIGGER_CHANGE_NOTICE_G,
    DMA_TRIGGER_DMA_0,
    DMA_TRIGGER_DMA_1,
    DMA_TRIGGER_DMA_2,
    DMA_TRIGGER_DMA_3,
    DMA_TRIGGER_DMA_4,
    DMA_TRIGGER_DMA_5,
    DMA_TRIGGER_DMA_6,
    DMA_TRIGGER_DMA_7,
    DMA_TRIGGER_COMPARATOR_1,
    DMA_TRIGGER_COMPARATOR_2,
    DMA_TRIGGER_COMPARATOR_3,
    DMA_TRIGGER_ADC_1,
    DMA_TRIGGER_PARALLEL_PORT,
    DMA_TRIGGER_CAN_1,
    DMA_TRIGGER_CAN_2,
    DMA_TRIGGER_CLOCK_MONITOR,
    DMA_TRIGGER_RTCC,
    DMA_TRIGGER_FLASH_CONTROL,
    DMA_TRIGGER_USB_1,
    DMA_TRIGGER_ETH_1,
    DMA_TRIGGER_PARALLEL_PORT_ERROR,
    DMA_TRIGGER_CTMU
} DMA_TRIGGER_SOURCE;
```

Members

Members	Description
DMA_TRIGGER_TIMER_CORE	Core timer
DMA_TRIGGER_SOFTWARE_0	Software 0

DMA_TRIGGER_SOFTWARE_1	Software 1
DMA_TRIGGER_EXTERNAL_0	External 0
DMA_TRIGGER_EXTERNAL_1	External 1
DMA_TRIGGER_EXTERNAL_2	External 2
DMA_TRIGGER_EXTERNAL_3	External 3
DMA_TRIGGER_EXTERNAL_4	External 4
DMA_TRIGGER_TIMER_1	Timer1
DMA_TRIGGER_TIMER_2	Timer2
DMA_TRIGGER_TIMER_3	Timer3
DMA_TRIGGER_TIMER_4	Timer4
DMA_TRIGGER_TIMER_5	Timer5
DMA_TRIGGER_INPUT_CAPTURE_1	Input Capture 1
DMA_TRIGGER_INPUT_CAPTURE_2	Input Capture 2
DMA_TRIGGER_INPUT_CAPTURE_3	Input Capture 3
DMA_TRIGGER_INPUT_CAPTURE_4	Input Capture 4
DMA_TRIGGER_INPUT_CAPTURE_5	Input Capture 5
DMA_TRIGGER_INPUT_CAPTURE_1_ERROR	Input Capture 1 error
DMA_TRIGGER_INPUT_CAPTURE_2_ERROR	Input Capture 2 error
DMA_TRIGGER_INPUT_CAPTURE_3_ERROR	Input Capture 3 error
DMA_TRIGGER_INPUT_CAPTURE_4_ERROR	Input Capture 4 error
DMA_TRIGGER_INPUT_CAPTURE_5_ERROR	Input Capture 5 error
DMA_TRIGGER_OUTPUT_COMPARE_1	Output Compare 1
DMA_TRIGGER_OUTPUT_COMPARE_2	Output Compare 2
DMA_TRIGGER_OUTPUT_COMPARE_3	Output Compare 3
DMA_TRIGGER_OUTPUT_COMPARE_4	Output Compare 4
DMA_TRIGGER_OUTPUT_COMPARE_5	Output Compare 5
DMA_TRIGGER_SPI_1_ERROR	SPI1 error
DMA_TRIGGER_SPI_1_RECEIVE	SPI1 receive
DMA_TRIGGER_SPI_1_TRANSMIT	SPI1 transmit
DMA_TRIGGER_SPI_1A_ERROR	SPI1A error
DMA_TRIGGER_SPI_1A_RECEIVE	SPI1A receive
DMA_TRIGGER_SPI_1A_TRANSMIT	SPI1A transmit
DMA_TRIGGER_SPI_2_ERROR	SPI2 error
DMA_TRIGGER_SPI_2_RECEIVE	SPI2 receive
DMA_TRIGGER_SPI_2_TRANSMIT	SPI2 transmit
DMA_TRIGGER_SPI_2A_ERROR	SPI2A receive
DMA_TRIGGER_SPI_2A_RECEIVE	SPI2A receive
DMA_TRIGGER_SPI_2A_TRANSMIT	SPI2A transmit
DMA_TRIGGER_SPI_3_ERROR	SPI3 error
DMA_TRIGGER_SPI_3_RECEIVE	SPI3 receive
DMA_TRIGGER_SPI_3_TRANSMIT	SPI3 transmit
DMA_TRIGGER_SPI_3A_ERROR	SPI3A receive
DMA_TRIGGER_SPI_3A_RECEIVE	SPI3A receive
DMA_TRIGGER_SPI_3A_TRANSMIT	SPI3A receive
DMA_TRIGGER_SPI_4_ERROR	SPI4 error
DMA_TRIGGER_SPI_4_RECEIVE	SPI4 receive
DMA_TRIGGER_SPI_4_TRANSMIT	SPI4 transmit
DMA_TRIGGER_I2C_1_ERROR	I2C1 error
DMA_TRIGGER_I2C_1_SLAVE	I2C1 slave
DMA_TRIGGER_I2C_1_MASTER	I2C1 master
DMA_TRIGGER_I2C_1A_ERROR	I2C1A error
DMA_TRIGGER_I2C_1A_SLAVE	I2C1A slave
DMA_TRIGGER_I2C_1A_MASTER	I2C1A master
DMA_TRIGGER_I2C_2_ERROR	I2C2 error
DMA_TRIGGER_I2C_2_SLAVE	I2C2 slave

DMA_TRIGGER_I2C_2_MASTER	I2C2 master
DMA_TRIGGER_I2C_2A_ERROR	I2C2A error
DMA_TRIGGER_I2C_2A_SLAVE	I2C2A slave
DMA_TRIGGER_I2C_2A_MASTER	I2C2A master
DMA_TRIGGER_I2C_3_ERROR	I2C3 error
DMA_TRIGGER_I2C_3_SLAVE	I2C3 slave
DMA_TRIGGER_I2C_3_MASTER	I2C3 master
DMA_TRIGGER_I2C_3A_ERROR	I2C3A error
DMA_TRIGGER_I2C_3A_SLAVE	I2C3A slave
DMA_TRIGGER_I2C_3A_MASTER	I2C3A master
DMA_TRIGGER_I2C_4_ERROR	I2C4 error
DMA_TRIGGER_I2C_4_SLAVE	I2C4 slave
DMA_TRIGGER_I2C_4_MASTER	I2C4 master
DMA_TRIGGER_I2C_5_ERROR	I2C5 error
DMA_TRIGGER_I2C_5_SLAVE	I2C5 slave
DMA_TRIGGER_I2C_5_MASTER	I2C5 master
DMA_TRIGGER_USART_1_ERROR	USART1 error
DMA_TRIGGER_USART_1_RECEIVE	USART1 receive
DMA_TRIGGER_USART_1_TRANSMIT	USART1 transmit
DMA_TRIGGER_USART_1A_ERROR	USART1A error
DMA_TRIGGER_USART_1A_RECEIVE	USART1A receive
DMA_TRIGGER_USART_1A_TRANSMIT	USART1A transmit
DMA_TRIGGER_USART_1B_ERROR	USART1B error
DMA_TRIGGER_USART_1B_RECEIVE	USART1B receive
DMA_TRIGGER_USART_1B_TRANSMIT	USART1B transmit
DMA_TRIGGER_USART_2_ERROR	USART2 error
DMA_TRIGGER_USART_2_RECEIVE	USART2 receive
DMA_TRIGGER_USART_2_TRANSMIT	USART2 transmit
DMA_TRIGGER_USART_2A_ERROR	USART2A error
DMA_TRIGGER_USART_2A_RECEIVE	USART2A receive
DMA_TRIGGER_USART_2A_TRANSMIT	USART2A transmit
DMA_TRIGGER_USART_2B_ERROR	USART2B error
DMA_TRIGGER_USART_2B_RECEIVE	USART2B receive
DMA_TRIGGER_USART_2B_TRANSMIT	USART2B transmit
DMA_TRIGGER_USART_3_ERROR	USART3 error
DMA_TRIGGER_USART_3_RECEIVE	USART3 receive
DMA_TRIGGER_USART_3_TRANSMIT	USART3 transmit
DMA_TRIGGER_USART_3A_ERROR	USART3A error
DMA_TRIGGER_USART_3A_RECEIVE	USART3A receive
DMA_TRIGGER_USART_3A_TRANSMIT	USART3A transmit
DMA_TRIGGER_USART_3B_ERROR	USART3B error
DMA_TRIGGER_USART_3B_RECEIVE	USART3B receive
DMA_TRIGGER_USART_3B_TRANSMIT	USART3B transmit
DMA_TRIGGER_USART_4_ERROR	USART4 error
DMA_TRIGGER_USART_4_RECEIVE	USART3B receive
DMA_TRIGGER_USART_4_TRANSMIT	USART4 transmit
DMA_TRIGGER_USART_5_ERROR	USART5 error
DMA_TRIGGER_USART_5_RECEIVE	USART5 receive
DMA_TRIGGER_USART_5_TRANSMIT	USART5 transmit
DMA_TRIGGER_USART_6_ERROR	USART6 error
DMA_TRIGGER_USART_6_RECEIVE	USART6 receive
DMA_TRIGGER_USART_6_TRANSMIT	USART6 transmit
DMA_TRIGGER_CHANGE_NOTICE	Change notice
DMA_TRIGGER_CHANGE_NOTICE_A	Change notice A
DMA_TRIGGER_CHANGE_NOTICE_B	Change notice B

DMA_TRIGGER_CHANGE_NOTICE_C	Change notice C
DMA_TRIGGER_CHANGE_NOTICE_D	Change notice D
DMA_TRIGGER_CHANGE_NOTICE_E	Change notice E
DMA_TRIGGER_CHANGE_NOTICE_F	Change notice F
DMA_TRIGGER_CHANGE_NOTICE_G	Change notice G
DMA_TRIGGER_DMA_0	DMA Channel 0
DMA_TRIGGER_DMA_1	DMA Channel 1
DMA_TRIGGER_DMA_2	DMA Channel 2
DMA_TRIGGER_DMA_3	DMA Channel 3
DMA_TRIGGER_DMA_4	DMA Channel 4
DMA_TRIGGER_DMA_5	DMA Channel 5
DMA_TRIGGER_DMA_6	DMA Channel 6
DMA_TRIGGER_DMA_7	DMA Channel 7
DMA_TRIGGER_COMPARATOR_1	Comparator 1
DMA_TRIGGER_COMPARATOR_2	DMA Channel 2
DMA_TRIGGER_COMPARATOR_3	DMA Channel 3
DMA_TRIGGER_ADC_1	ADC1
DMA_TRIGGER_PARALLEL_PORT	Parallel port
DMA_TRIGGER_CAN_1	CAN1
DMA_TRIGGER_CAN_2	CAN2
DMA_TRIGGER_CLOCK_MONITOR	Clock monitor
DMA_TRIGGER_RTCC	RTCC
DMA_TRIGGER_FLASH_CONTROL	Flash control
DMA_TRIGGER_USB_1	USB1
DMA_TRIGGER_ETH_1	ETH1
DMA_TRIGGER_PARALLEL_PORT_ERROR	Parallel port error
DMA_TRIGGER_CTMU	CTMU

Description

DMA trigger sources

This enumeration lists all the possible DMA trigger sources.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_INT_SOURCE Enumeration

Lists the Interrupt Source number for the available DMA channels.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_0_INT_SOURCE,
  DMA_CHANNEL_1_INT_SOURCE,
  DMA_CHANNEL_2_INT_SOURCE,
  DMA_CHANNEL_3_INT_SOURCE,
  DMA_CHANNEL_4_INT_SOURCE,
  DMA_CHANNEL_5_INT_SOURCE,
  DMA_CHANNEL_6_INT_SOURCE,
  DMA_CHANNEL_7_INT_SOURCE
} DMA_CHANNEL_INT_SOURCE;
```

Members

Members	Description
DMA_CHANNEL_0_INT_SOURCE	DMA Channel 0 Interrupt Source
DMA_CHANNEL_1_INT_SOURCE	DMA Channel 1 Interrupt Source
DMA_CHANNEL_2_INT_SOURCE	DMA Channel 2 Interrupt Source

DMA_CHANNEL_3_INT_SOURCE	DMA Channel 3 Interrupt Source
DMA_CHANNEL_4_INT_SOURCE	DMA Channel 4 Interrupt Source
DMA_CHANNEL_5_INT_SOURCE	DMA Channel 5 Interrupt Source
DMA_CHANNEL_6_INT_SOURCE	DMA Channel 6 Interrupt Source
DMA_CHANNEL_7_INT_SOURCE	DMA Channel 7 Interrupt Source

Description

DMA channel Interrupt source number

This enumeration lists the Interrupt Source number for all of the available DMA channels.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_dma.h	Defines the DMA Peripheral Library interface functions.
help_plib_dma.h	Defines the DMA Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_dma.h

Defines the DMA Peripheral Library interface functions.

Functions

	Name	Description
	PLIB_DMA_AbortTransferSet	Aborts transfer on the specified channel.
	PLIB_DMA_BusyActiveReset	Resets the BUSY bit of the DMA controller.
	PLIB_DMA_BusyActiveSet	Sets the BUSY bit of the DMA controller.
	PLIB_DMA_ChannelBitsGet	Returns the DMA channel bits.
	PLIB_DMA_ChannelPriorityGet	Gets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelPrioritySelect	Sets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelXAbortIRQSet	Sets the IRQ to abort the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXAddressModeGet	Gets the channel address mode.
	PLIB_DMA_ChannelXAddressModeSelect	Sets the channel address mode.
	PLIB_DMA_ChannelXAutoDisable	Channel is disabled after a block transfer is complete.
	PLIB_DMA_ChannelXAutoEnable	Channel is continuously enabled.
	PLIB_DMA_ChannelXAutoIsEnabled	Returns the channel automatic enable status.
	PLIB_DMA_ChannelXBufferedDataIsWritten	Returns the buffered data write status for the specified channel.
	PLIB_DMA_ChannelXBusyActiveSet	Sets the Busy bit to active.
	PLIB_DMA_ChannelXBusyInActiveSet	Sets the Busy bit to inactive.
	PLIB_DMA_ChannelXBusyIsBusy	Returns the busy status of the specified channel.
	PLIB_DMA_ChannelXCellProgressPointerGet	Returns the number of bytes transferred since the last event.
	PLIB_DMA_ChannelXCellSizeGet	Reads the cell size (in bytes) configured for the specified channel.
	PLIB_DMA_ChannelXCellSizeSet	Writes the specified cell size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXChainDisable	Disables the channel chaining for the specified DMA channel.
	PLIB_DMA_ChannelXChainEnable	Channel chain feature is enabled.
	PLIB_DMA_ChannelXChainIsEnabled	Returns the chain status of the specified channel.
	PLIB_DMA_ChannelXChainToHigher	Chains the specified channel to a channel higher in natural priority.
	PLIB_DMA_ChannelXChainToLower	Chains the specified channel to a channel lower in natural priority.
	PLIB_DMA_ChannelXCollisionStatus	Returns the status of the specified collision type for the specified channel.

	PLIB_DMA_ChannelXDataSizeGet	Returns the current data size for the specified channel.
	PLIB_DMA_ChannelXDataSizeSelect	Selects the data size for the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeIsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.
	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.

	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.
	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.
	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).
	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.
	PLIB_DMA_ChannelXTriggerSourceNumberGet	Gets the source number for the DMA channel interrupts.
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModeIsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.
	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.

	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnoreByte	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternLength	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPriority	Identifies whether the ChannelXPriority feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourcePointer	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceSize	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceStartAddress	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXStartIRQ	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXTrigger	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	PLIB_DMA_ExistsCRC	Identifies whether the CRC feature exists on the DMA module.
	PLIB_DMA_ExistsCRCAppendMode	Identifies whether the CRCAppendMode feature exists on the DMA module.
	PLIB_DMA_ExistsCRCBitOrder	Identifies whether the CRCBitOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCByteOrder	Identifies whether the CRCByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCChannel	Identifies whether the CRCChannel feature exists on the DMA module.
	PLIB_DMA_ExistsCRCData	Identifies whether the CRCData feature exists on the DMA module.
	PLIB_DMA_ExistsCRCPolynomialLength	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	PLIB_DMA_ExistsCRCType	Identifies whether the CRCType feature exists on the DMA module.
	PLIB_DMA_ExistsCRCWriteByteOrder	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCXOREnable	Identifies whether the CRCXOREnable feature exists on the DMA module.
	PLIB_DMA_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMA module.
	PLIB_DMA_ExistsLastBusAccess	Identifies whether the LastBusAccess feature exists on the DMA module.
	PLIB_DMA_ExistsRecentAddress	Identifies whether the RecentAddress feature exists on the DMA module.
	PLIB_DMA_ExistsStartTransfer	Identifies whether the StartTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the DMA module.
	PLIB_DMA_ExistsSuspend	Identifies whether the Suspend feature exists on the DMA module.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.
	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

Description

DMA Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Direct Memory Access (DMA) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the DMA module.

File Name

plib_dma.h

Company

Microchip Technology Inc.

help_plib_dma.h

Defines the DMA Peripheral Library data types.

Enumerations

	Name	Description
	DMA_ADDRESS_OFFSET_TYPE	Lists the possible DMA address offsets.
	DMA_CHANNEL	Lists the possible DMA channels available.
	DMA_CHANNEL_ADDRESSING_MODE	Lists the possible channel addressing modes.
	DMA_CHANNEL_COLLISION	Lists the possible channel collision types.
	DMA_CHANNEL_DATA_SIZE	Lists the possible data sizes for the DMA channel.
	DMA_CHANNEL_INT_SOURCE	Lists the Interrupt Source number for the available DMA channels.
	DMA_CHANNEL_PRIORITY	Lists the possible channel priorities.
	DMA_CHANNEL_TRANSFER_DIRECTION	Lists the possible data transfer directions.
	DMA_CHANNEL_TRIGGER_TYPE	Lists the possible DMA channel triggers.
	DMA_CRC_BIT_ORDER	Lists the possible CRC calculation bit orders
	DMA_CRC_BYTE_ORDER	Lists the possible byte orders.
	DMA_CRC_TYPE	Lists the possible checksums.
	DMA_DESTINATION_ADDRESSING_MODE	Lists the possible destination addressing modes.
	DMA_INT_TYPE	Lists the possible Interrupt types for a particular channel-X
	DMA_MODULE_ID	Possible instances of the DMA module.
	DMA_PATTERN_LENGTH	Gives the DMA pattern length
	DMA_PING_PONG_MODE	Lists the possible ping-pong modes.
	DMA_SOURCE_ADDRESSING_MODE	Lists the possible source addressing modes.
	DMA_TRANSFER_MODE	Lists the possible DMA transfer/operating modes.
	DMA_TRIGGER_SOURCE	Lists the possible DMA trigger sources.

Description

DMA Peripheral Library data types.

This header file contains data types and constants that make up the Interface to the direct memory access(DMA) peripheral library (PLIB) for Microchip microcontrollers. The definitions in this file are for DMA module.

File Name

help_plib_dma.h

Company

Microchip Technology Inc.

Dual Data Rate (DDR) SDRAM Peripheral Library

This topic describes the Dual Data Rate (DDR) SDRAM Peripheral Library.

Introduction

This library provides a low-level abstraction of the DDR SDRAM Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The DDR SDRAM controller and associated physical interface (PHY) provide access to external SDRAM from the CPU and other targets within the device. The specific targets may vary between devices, but typically include graphics controller(s), DMA, and other peripherals. External DRAM expands the volatile memory available for the device, and is useful for operating systems, graphics controllers/accelerators, and other operations requiring large amounts of memory.

The SDRAM and controller must be initialized before the SDRAM can be accessed. The initialization is specific to the timing constraints and addressing of the particular SDRAM in use, as well as the clocks to the controller and PHY. This library provides functions for initializing the controller, PHY, and external SDRAM.

Using the Library

This topic describes the basic architecture of the DDR SDRAM Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ddr.h`

The interface to the DDR SDRAM Peripheral Library is defined in the `plib_ddr.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the DDR SDRAM Peripheral Library must include `peripheral.h`.

Library File:

The DDR SDRAM Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the Understanding MPLAB Harmony section for how the library interacts with the framework.

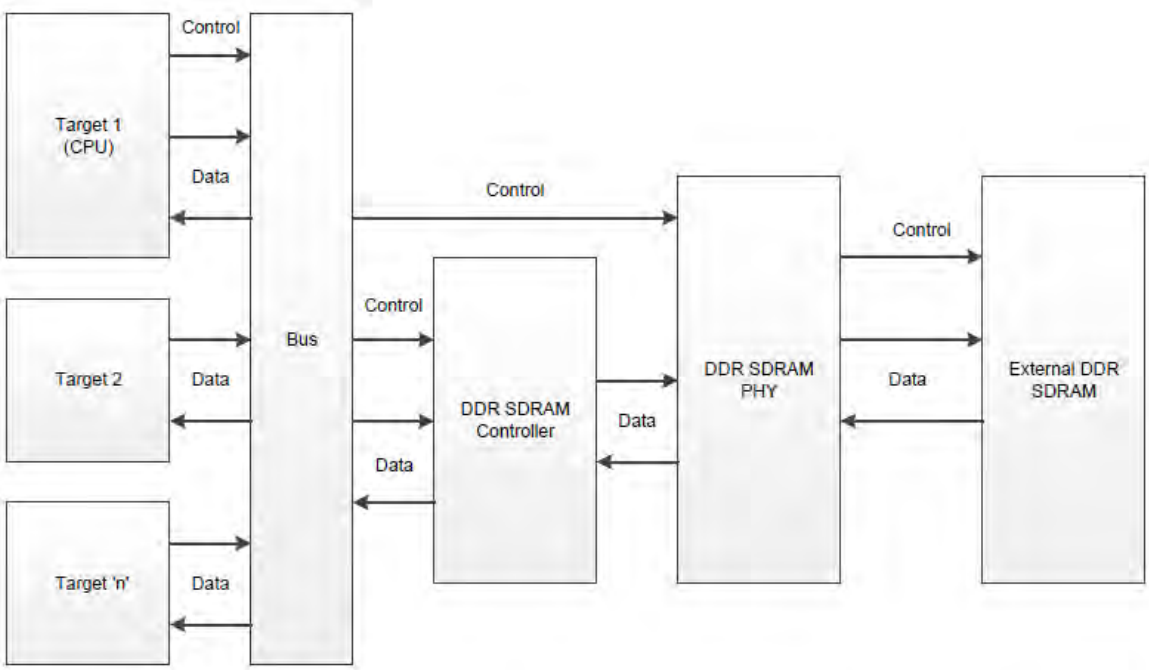
Abstraction Model

This library provides the low-level abstraction of the Dual Data Rate (DDR) SDRAM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The DDR SDRAM controller and PHY must be initialized prior to accessing SDRAM. The SDRAM itself must also be initialized. This library provides interface routines to perform these initializations.

DDR SDRAM Peripheral Library Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various subsections, each of the sub-sections addresses one of the blocks or the overall operation of the DDR SDRAM module.

Library Interface Section	Description
General Configuration Functions	The SDRAM controller must be configured for the system and device in use. General configuration options include DRAM type, operation mode and On-Die-Termination (ODT) settings.
Target Arbitration Functions	Arbitration parameters for each target can be programmed individually. The arbitration parameters determine the relative bandwidth assigned to each target. Adjusting target arbitration parameters can improve overall system performance by maximizing bandwidth utilization among targets.
SDRAM Addressing Functions	DDR SDRAM is addressed in terms of banks, rows and columns. The CPU address space is translated to SDRAM addresses by the controller. The addressing parameters are determined by the size and geometry of the SDRAM, and must be initialized in the controller to access the SDRAM correctly.
SDRAM Timing Functions	To operate correctly, timing delays must be inserted for various SDRAM operations. The timing delays are device-specific, and are specified in the SDRAM data sheet. These delays are used to initialize the controller for correct operation.
SDRAM Initialization Functions	The SDRAM must be initialized prior to use. Initialization consists of writing system-specific values to registers within the SDRAM itself, using the Host Command Interface. The registers and initialization sequence are standardized across SDRAM devices.
PHY Initialization Functions	Like the SDRAM controller, the PHY must be initialized prior to use. PHY parameters include ODT and drive-strength settings, DLL settings, and Self-Calibration-Logic (SCL) settings.
Feature Existence Functions	Interface routines that can be used to determine whether or not the feature is supported by the device.

How the Library Works

This section provides information on how the DDR SDRAM Peripheral Library works.

Description

General Configuration

General configuration of the SDRAM controller includes the following:

- On-Die-Termination (ODT) settings for data reads and writes
- Endianness
- DDR type (DDR2 or DDR3)
- Rate mode (full or half)

Target Arbitration

Initializing the arbitration parameters for each target is a two-step procedure. First, the minimum number of uninterrupted bursts (without interference from another target) is programmed, using the PLIB_DDR_MinLimit function. Then the number of bursts within a specific time-out period is set using the PLIB_DDR_ReqPeriod and PLIB_DDR_MinCommand functions. The relative time-outs and number of bursts between targets determine the total bandwidth allocation.

For details, refer to the **"DDR SDRAM"** chapter in the specific device data sheet and **Section 55. "DDR2 SDRAM Controller"** (DS60001321) in the *"PIC32 Family Reference Manual"*.

SDRAM Addressing

The address used by the CPU to access the SDRAM must be converted into a format understood by the SDRAM. The SDRAM uses bank, row, column and chip select bits to access data within the memory. The SDRAM addressing functions use shift and mask operations to translate the CPU address into bank, row, column and chip-select. There are four addressing functions, one each for bank, row, column and chip select. Each function takes mask and shift arguments. The mask parameter determines the number of bits that make up the address component, and the shift parameter determines where in the 32-bit CPU address the component is located. Refer to the DDR SDRAM Family Reference Manual for more details and examples.

PHY Initialization

The PHY controls the physical interface from the PIC32 to SDRAM. ODT is provided by the pads, and can be enabled or disabled using the PHY initialization functions. Setting ODT correctly for the platform can improve signal integrity by minimizing signal reflection. Enabling/disabling ODT and setting the impedance is done using the following functions:

- PLIB_DDR_PHY_OdtEnable
- PLIB_DDR_PHY_OdtDisable

Similarly, the pad drive strength can be set using the PLIB_DDR_PHY_DriveStrengthSet function.

Fine tuning of the ODT impedance and drive strength can be performed using the calibration functions PLIB_DDR_PHY_OdtCal and PLIB_DDR_PHY_DrvStrgthCal.

The SDRAM PHY contains Self-Calibration-Logic (SCL) that is run automatically during initialization. Various SCL parameters can be set using the following functions:

- PLIB_DDR_PHY_SCLEnable - enables SCL for a specific Chip Select
- PLIB_DDR_PHY_ReadCASLatencySet - sets the read CAS latency during SCL
- PLIB_DDR_PHY_WriteCASLatencySet - sets the write CAS latency during SCL
- PLIB_DDR_PHY_OdtCSEnable - enables ODT on Chip Select during SCL
- PLIB_DDR_PHY_SCLDelay - sets the SCL delay

SDRAM Timing

Several timing parameters must be initialized for the SDRAM interface to operate correctly. These include:

- Refresh interval and delay
- Read and write delays
- Precharge delays
- Addressing delays
- Bank activate delays

Timing parameters for specific SDRAMs are provided in the SDRAM data sheet. These are passed to the SDRAM timing initialization functions along with the controller clock period. The initialization functions then calculate the delays and set the controller timing registers to the correct values. Refer to **Section 55. "DDR2 SDRAM Controller"** (DS60001321)in the *"PIC32 Family Reference Manual"* for details.

SDRAM Initialization

The SDRAM is initialized by writing a series of commands to internal registers of the SDRAM itself. The register command interface is encoded on the control and address lines to the SDRAM. The SDRAM controller includes a series of registers that are initialized and transmitted to the SDRAM using the SDRAM initialization functions.

The sequence is as follows:

1. Write the initialization words to the host command registers using the PLIB_DDR_CmdDataWrite function.
2. Set the number of commands with the PLIB_DDR_NumHostCmdsSet function.
3. Write the commands to the SDRAM using PLIB_DDR_CmdDataSend.
4. Wait for the SDRAM to acknowledge valid initialization using PLIB_DDR_CmdDataIsComplete.
5. Enable the SDRAM for normal operation with the PLIB_DDR_ControllerEnable function.

The initialization command sequence is specified in the data sheet of the SDRAM device. An example initialization sequence is provided in

Section 55. "DDR2 SDRAM Controller" (DS60001321) of the *"PIC32 Family Reference Manual"*.

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DDR_BigEndianSet	Sets the DDR data endianness to big.
	PLIB_DDR_FullRateSet	Sets the DDR controller to Full-rate mode.
	PLIB_DDR_HalfRateSet	Sets the DDR controller to Half-rate mode.
	PLIB_DDR_LittleEndianSet	Sets the DDR data endianness to little.
	PLIB_DDR_MaxPendingRefSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_OdtReadDisable	Selects which Chip Select to disable ODT for data reads.
	PLIB_DDR_OdtReadEnable	Selects which Chip Select to enable ODT for data reads.
	PLIB_DDR_OdtWriteDisable	Selects which Chip Select to disable ODT for data writes.
	PLIB_DDR_OdtWriteEnable	Selects which Chip Select to enable ODT for data writes.
	PLIB_DDR_AutoPchrgDisable	Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPowerDownDisable	Prevents the DDR controller from automatically entering Power-down mode.
	PLIB_DDR_AutoPchrgPowerDownDisable	Prevents the DDR controller from automatically entering Precharge Power-down mode.
	PLIB_DDR_AutoSelfRefreshDisable	Prevents the DDR controller from automatically entering Self-refresh mode.
	PLIB_DDR_AutoPchrgPowerDownEnable	Enables the DDR controller to automatically enter Precharge Power-down mode.
	PLIB_DDR_AutoPowerDownEnable	Enables the DDR controller to automatically enter Power-down mode.
	PLIB_DDR_AutoSelfRefreshEnable	Enables the DDR controller to automatically enter Self-refresh mode.
	PLIB_DDR_AutoPchrgEnable	Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_TfawDelaySet	Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

b) Target Arbitration Functions

	Name	Description
	PLIB_DDR_MinCommand	Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.
	PLIB_DDR_MinLimit	Sets the minimum number of bursts for a DDR target.
	PLIB_DDR_ReqPeriod	Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.

c) SDRAM Initialization Functions

	Name	Description
	PLIB_DDR_CmdDataIsComplete	Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.
	PLIB_DDR_CmdDataSend	Transmits the data in the command registers to the SDRAM.
	PLIB_DDR_CmdDataValid	Indicates to the controller that the data in the command registers is valid.
	PLIB_DDR_ControllerEnable	Enables the controller for normal operations after SDRAM is initialized.
	PLIB_DDR_MaxCmdBrstCntSet	Sets the maximum number of commands that can be written to the controller in Burst mode.
	PLIB_DDR_NumHostCmdsSet	Sets the number of commands to be transmitted to the SDRAM.
	PLIB_DDR_CmdDataWrite	Writes an SDRAM command word to a command register in the controller.

d) SDRAM Addressing Functions

	Name	Description
	PLIB_DDR_BankAddressSet	Initializes the DDR controller memory configuration registers for bank address.
	PLIB_DDR_ChipSelectAddressSet	Initializes the DDR controller memory configuration registers for Chip Select address.
	PLIB_DDR_ColumnAddressSet	Initializes the DDR controller memory configuration registers for the column address.
	PLIB_DDR_RowAddressSet	Initializes the DDR controller memory configuration registers for row address.

e) SDRAM Timing Functions

	Name	Description
	PLIB_DDR_OdtReadParamSet	Sets the ODT parameters for data reads.
	PLIB_DDR_OdtWriteParamSet	Sets the ODT parameters for data writes.
	PLIB_DDR_DataDelaySet	Initializes the DDR controller with the data delays needed for the specific DDR in use.
	PLIB_DDR_PowerDownDelaySet	Initializes the DDR controller with the power down delays needed for the specific DDR in use.
	PLIB_DDR_SelfRefreshDelaySet	Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.
	PLIB_DDR_PrechargAllBanksSet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_RefreshTimingSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_WriteWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_PrechargeToRASDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToCASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASBankDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_ReadToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_WriteToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_WriteReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

f) PHY Initialization Functions

	Name	Description
	PLIB_DDR_PHY_DDRTypeSet	Sets the DRAM type for the PHY.
	PLIB_DDR_PHY_DllMasterDelayStartSet	Sets the start value of the digital DLL master delay line.
	PLIB_DDR_PHY_DllRecalibDisable	Disables periodic recalibration of the internal digital DLL.
	PLIB_DDR_PHY_DllRecalibEnable	Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.
	PLIB_DDR_PHY_DriveStrengthSet	Disables On Die Termination.
	PLIB_DDR_PHY_DrvStrgthCal	Calibrates the pad NFET and PFET output impedance.
	PLIB_DDR_PHY_ExternalDllEnable	Enables the use of an external DLL.
	PLIB_DDR_PHY_ExtraClockDisable	Does not enable the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_ExtraClockEnable	Enables the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_InternalDllEnable	Enables the use of the internal digital DLL.
	PLIB_DDR_PHY_OdtCal	Calibrates the pull-up and pull-down ODT impedance.
	PLIB_DDR_PHY_OdtCSDisable	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtCSEnable	Enables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtDisable	Disables On Die Termination.
	PLIB_DDR_PHY_OdtEnable	Enables On Die Termination and sets the value.
	PLIB_DDR_PHY_PadReceiveEnable	Enables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PreambleDlySet	Sets the length of the preamble for data writes.
	PLIB_DDR_PHY_ReadCASLatencySet	Sets the read CAS latency while running SCL test.
	PLIB_DDR_PHY_SCLDelay	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_SCLEnable	Enables SCL on the Chip Select 'cs'.

	PLIB_DDR_PHY_SCLTestBurstModeSet	Sets the burst mode of the DRAM while running SCL test.
	PLIB_DDR_PHY_WriteCASLatencySet	Sets the write CAS latency while running SCL test.
	PLIB_DDR_PHY_PadReceiveDisable	Disables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_SCLCapClkDelaySet	Sets capture clock delay during SCL.
	PLIB_DDR_PHY_SCLDDRClkDelaySet	Sets DDR clock delay during SCL.
	PLIB_DDR_PHY_HalfRateSet	Sets the PHY to half rate mode.
	PLIB_DDR_PHY_SCLStart	Runs PHY Self Calibration.
	PLIB_DDR_PHY_SCLStatus	Checks status of PHY Self Calibration.
	PLIB_DDR_PHY_AddCtlDriveStrengthSet	Sets the drive strength for the PHY address and control pads.
	PLIB_DDR_PHY_DataDriveStrengthSet	Sets the drive strength for the PHY data pads.
	PLIB_DDR_PHY_WriteCmdDelayDisable	Disables write command delay.
	PLIB_DDR_PHY_WriteCmdDelayEnable	Enables write command delay. The extra delay is needed for devices with even write latency (WL).

g) Feature Existence Functions

	Name	Description
	PLIB_DDR_ExistsAddressMapping	Identifies whether the AddressMapping feature exists on the DDR module.
	PLIB_DDR_ExistsArbitrationControl	Identifies whether the ArbitrationControl feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPowerDown	Identifies whether the AutoPowerDown feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPrecharge	Identifies whether the AutoPrecharge feature exists on the DDR module.
	PLIB_DDR_ExistsAutoSelfRefresh	Identifies whether the AutoSelfRefresh feature exists on the DDR module.
	PLIB_DDR_ExistsDDRCommands	Identifies whether the DDRCommands feature exists on the DDR module.
	PLIB_DDR_ExistsDDRControllerConfig	Identifies whether the DDRControllerConfig feature exists on the DDR module.
	PLIB_DDR_ExistsODTConfig	Identifies whether the ODTConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_DLLCalibration	Identifies whether the PHY_DLLCalibration feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_PadConfig	Identifies whether the PHY_PadConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_SCLConfig	Identifies whether the PHY_SCLConfig feature exists on the DDR module
	PLIB_DDR_ExistsRefreshConfig	Identifies whether the RefreshConfig feature exists on the DDR module.
	PLIB_DDR_ExistsTimingDelays	Identifies whether the TimingDelays feature exists on the DDR module

h) Data Types and Constants

	Name	Description
	DDR_CMD_IDLE_NOP	
	DDR_PHY_DDR_TYPE	Defines the DDR type.
	DDR_CMD_LOAD_MODE	This is macro DDR_CMD_LOAD_MODE.
	DDR_PHY_DRIVE_STRENGTH	Defines the PHY Pad drive strength value.
	DDR_CMD_PRECHARGE_ALL	This is macro DDR_CMD_PRECHARGE_ALL.
	DDR_PHY_ODT	Defines the ODT termination value.
	DDR_CMD_REFRESH	This is macro DDR_CMD_REFRESH.
	DDR_PHY_PREAMBLE_DLY	Defines the PHY preamble delay value.
	DDR_PHY_SCL_BURST_MODE	Defines the burst mode for SCL.
	DDR_PHY_SCL_DELAY	Defines the SCL delay.
	DDR_HOST_CMD_REG	Defines the DDR host command register.
	DDR_MODULE_ID	Enumeration: DDR_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	DDR_TARGET	Defines the different targets to which the DDR controller is connected.

Description

This section describes the Application Programming Interface (API) functions of the DDR SDRAM Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DDR_BigEndianSet Function

Sets the DDR data endianness to big.

File

plib_ddr.h

C

```
void PLIB_DDR_BigEndianSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR data endianness to big.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_BigEndianSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_BigEndianSet(DDR_MODULE_ID index);

PLIB_DDR_FullRateSet Function

Sets the DDR controller to Full-rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_FullRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR controller to Full-rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_FullRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_FullRateSet(DDR_MODULE_ID index);

PLIB_DDR_HalfRateSet Function

Sets the DDR controller to Half-rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_HalfRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR controller to Half-rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_HalfRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_HalfRateSet(DDR_MODULE_ID index);

PLIB_DDR_LittleEndianSet Function

Sets the DDR data endianness to little.

File

plib_ddr.h

C

```
void PLIB_DDR_LittleEndianSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR data endianness to little.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_LittleEndianSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_LittleEndianSet(DDR_MODULE_ID index);

PLIB_DDR_MaxPendingRefSet Function

Initializes the DDR controller refresh configuration.

File

plib_ddr.h

C

```
void PLIB_DDR_MaxPendingRefSet(DDR_MODULE_ID index, uint8_t maxRefs);
```

Returns

None.

Description

This function initializes the DDR controller refresh configuration and programs the number of refreshes that may be pending at one time. The following register field is programmed with this function:

- MAXREFS<4:0> - DDRREFCFG<24:26>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define MAX_PEND_REF 7

PLIB_DDR_MaxPendingRefSet(DDR_ID_0, MAX_PEND_REF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
maxRefs	Maximum number of refreshes that may be pending at one time

Function

void PLIB_DDR_MaxPendingRefSet(DDR_MODULE_ID index, uint8_t maxRefs);

PLIB_DDR_OdtReadDisable Function

Selects which Chip Select to disable ODT for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadDisable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to disable ODT for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadDisable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtReadDisable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtReadEnable Function

Selects which Chip Select to enable ODT for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadEnable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to enable ODT for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtReadEnable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtWriteDisable Function

Selects which Chip Select to disable ODT for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteDisable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to disable ODT for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteDisable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtWriteDisable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtWriteEnable Function

Selects which Chip Select to enable ODT for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteEnable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to enable ODT for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtWriteEnable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_AutoPchrgDisable Function

Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from issuing a auto-precharge command to close the bank at the end of every user command.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPowerDownDisable Function

Prevents the DDR controller from automatically entering Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPowerDownDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPowerDownDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPowerDownDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgPowerDownDisable Function

Prevents the DDR controller from automatically entering Precharge Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgPowerDownDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Precharge Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgPowerDownDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgPowerDownDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoSelfRefreshDisable Function

Prevents the DDR controller from automatically entering Self-refresh mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoSelfRefreshDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Self-refresh mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoSelfRefreshDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoSelfRefreshDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgPowerDownEnable Function

Enables the DDR controller to automatically enter Precharge Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgPowerDownEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Precharge Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgPowerDownEnable(DDR_ID_0);
```

I

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgPowerDownEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoPowerDownEnable Function

Enables the DDR controller to automatically enter Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPowerDownEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPowerDownEnable(DDR_ID_0, pdd);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPowerDownEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoSelfRefreshEnable Function

Enables the DDR controller to automatically enter Self-refresh mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoSelfRefreshEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Self-refresh mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoSelfRefreshEnable(DDR_ID_0, srd);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoSelfRefreshEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgEnable Function

Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to issue a auto-precharge command to close the bank at the end of every user command.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgEnable(DDR_MODULE_ID index);

PLIB_DDR_TfawDelaySet Function

Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_TfawDelaySet(DDR_MODULE_ID index, uint32_t tFaw, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the four-bank activation window for the DDR in use. The following register field is programmed with this function:

- FAWTDLY<5:0> - DDRDLYCFG3<21:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tFAW            35000   // psec

PLIB_DDR_TfawDelaySet(DDR_ID_0, tFAW, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tFaw	Four bank activation window (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_TfawDelaySet(DDR_MODULE_ID index, uint32_t tFaw, uint32_t ctrlClkPer);

b) Target Arbitration Functions

PLIB_DDR_MinCommand Function

Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.

File

plib_ddr.h

C

```
void PLIB_DDR_MinCommand(DDR_MODULE_ID index, uint8_t minCmd, DDR_TARGET target);
```

Returns

None.

Description

This function sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks. Used with PLIB_DDR_ReqPeriod to determine the total bandwidth allocated to a target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect() prior to using this function.

Example

```
PLIB_DDR_MinCommand(DDR_ID_0, 83, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minCmd	Number of bursts
target	Target for minCmd setting

Function

void PLIB_DDR_MinCommand(DDR_MODULE_ID index, uint8_t minCmd, DDR_TARGET target)

PLIB_DDR_MinLimit Function

Sets the minimum number of bursts for a DDR target.

File

plib_ddr.h

C

```
void PLIB_DDR_MinLimit(DDR_MODULE_ID index, uint8_t minLim, DDR_TARGET target);
```

Returns

None.

Description

This function sets the minimum number of bursts (two cycles per burst) that a target must have uninterrupted access to without interference from another target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect prior to using this function.

Example

```
PLIB_DDR_MinLimit(DDR_ID_0, 1, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minLim	Number of bursts
target	Target for minLim setting

Function

void PLIB_DDR_MinLimit(DDR_MODULE_ID index, uint8_t minLim, DDR_TARGET target)

PLIB_DDR_ReqPeriod Function

Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.

File

plib_ddr.h

C

```
void PLIB_DDR_ReqPeriod(DDR_MODULE_ID index, uint8_t reqPer, DDR_TARGET target);
```

Returns

None.

Description

This function sets the timeout for MINCMD number of bursts to be serviced for a DDR target. Used with PLIB_DDR_MinCommand to determine the total bandwidth allocated to a target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect prior to using this function.

Example

```
PLIB_DDR_ReqPeriod(DDR_ID_0, 167, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
reqPer	Number of clocks (multiplied by 4)
target	Target for reqPer setting

Function

void PLIB_DDR_ReqPeriod(DDR_MODULE_ID index, uint8_t reqPer, DDR_TARGET target)

c) SDRAM Initialization Functions

PLIB_DDR_CmdDataIsComplete Function

Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.

File

plib_ddr.h

C

```
bool PLIB_DDR_CmdDataIsComplete(DDR_MODULE_ID index);
```

Returns

None.

Description

This function returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
while (PLIB_DDR_CmdDataIsComplete(DDR_ID_0));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_DDR_CmdDataIsComplete(DDR_MODULE_ID index);

PLIB_DDR_CmdDataSend Function

Transmits the data in the command registers to the SDRAM.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataSend(DDR_MODULE_ID index);
```

Returns

None.

Description

This function transmits the data in the command registers to the SDRAM.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_CmdDataSend(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_CmdDataSend(DDR_MODULE_ID index);

PLIB_DDR_CmdDataValid Function

Indicates to the controller that the data in the command registers is valid.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataValid(DDR_MODULE_ID index);
```

Returns

None.

Description

This function indicates to the controller that the data in the command registers is valid.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_CmdDataValid(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_CmdDataValid(DDR_MODULE_ID index);

PLIB_DDR_ControllerEnable Function

Enables the controller for normal operations after SDRAM is initialized.

File

plib_ddr.h

C

```
void PLIB_DDR_ControllerEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the controller for normal operations after SDRAM is initialized.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ControllerEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_ControllerEnable(DDR_MODULE_ID index);

PLIB_DDR_MaxCmdBrstCntSet Function

Sets the maximum number of commands that can be written to the controller in Burst mode.

File

plib_ddr.h

C

```
void PLIB_DDR_MaxCmdBrstCntSet(DDR_MODULE_ID index, int8_t maxCmds);
```

Returns

None.

Description

This function sets the maximum number of commands that can be written to the controller in Burst mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_MaxCmdBrstCntSet(DDR_ID_0, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
maxCmds	Maximum number of commands that can be written to the controller in burst mode

Function

void PLIB_DDR_MaxCmdBrstCntSet(DDR_MODULE_ID index, int8_t maxCmds);

PLIB_DDR_NumHostCmdsSet Function

Sets the number of commands to be transmitted to the SDRAM.

File

plib_ddr.h

C

```
void PLIB_DDR_NumHostCmdsSet(DDR_MODULE_ID index, int8_t numCmds);
```

Returns

None.

Description

This function sets the number of commands to be transmitted to the SDRAM.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_NumHostCmdsSet(DDR_ID_0, 0x0B);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
numCmds	Number of commands to be transmitted to the SDRAM

Function

void PLIB_DDR_NumHostCmdsSet(DDR_MODULE_ID index, int8_t numCmds);

PLIB_DDR_CmdDataWrite Function

Writes an SDRAM command word to a command register in the controller.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataWrite(DDR_MODULE_ID index, DDR_HOST_CMD_REG cmdReg, uint32_t cmdData);
```

Returns

None.

Description

This function writes an SDRAM command word to a command register in the controller.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define DDR_CMD_IDLE_NOP 0x00FFFFFF
PLIB_DDR_CmdDataWrite(DDR_ID_0, DDR_HOST_CMD_REG_100, DDR_CMD_IDLE_NOP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cmdData	32-bit command data word
cmdReg	Command register

Function

void PLIB_DDR_CmdDataWrite(DDR_MODULE_ID index, DDR_HOST_CMD_REG cmdReg, uint32_t cmdData);

d) SDRAM Addressing Functions

PLIB_DDR_BankAddressSet Function

Initializes the DDR controller memory configuration registers for bank address.

File

plib_ddr.h

C

```
void PLIB_DDR_BankAddressSet(DDR_MODULE_ID index, uint32_t bnkShft, uint32_t bnkMsk);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the bank address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_BankAddressSet(DDR_ID_0, 9, 0x03);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bnkShft	Number of bits to shift the bank address
bnkMsk	Bank address bit mask

Function

void PLIB_DDR_BankAddressSet(DDR_MODULE_ID index, uint32_t bnkShft, uint32_t bnkMsk);

- BNKADDR<4:0> - DDRMEMCFG0<12:8>
- BNKADDRMSK<2:0> - DDRMEMCFG4<2:0>

PLIB_DDR_ChipSelectAddressSet Function

Initializes the DDR controller memory configuration registers for Chip Select address.

File

plib_ddr.h

C

```
void PLIB_DDR_ChipSelectAddressSet(DDR_MODULE_ID index, uint32_t csShft, uint32_t csMsk);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the Chip Select address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ChipSelectAddressSet(DDR_ID_0, 18, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csShft	Number of bits to shift the Chip Select address
csMsk	Chip select address bit mask

Function

void PLIB_DDR_ChipSelectAddressSet(DDR_MODULE_ID index, uint32_t csShft, uint32_t csMsk);

- CSADDR<4:0> - DDRMEMCFG0<20:16>
- CSADDRMSK<2:0> - DDRMEMCFG4<8:6>

PLIB_DDR_ColumnAddressSet Function

Initializes the DDR controller memory configuration registers for the column address.

File

plib_ddr.h

C

```
void PLIB_DDR_ColumnAddressSet(DDR_MODULE_ID index, uint32_t colShft, uint32_t colMskLo, uint32_t colMskHi);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the column address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ColumnAddressSet(DDR_ID_0, 0, 0x1FF, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
colShft	Number of bits to shift the column address
colMaskLo	Column address bit mask low
colMaskHi	Column address bit mask high

Function

void PLIB_DDR_ColumnAddressSet(DDR_MODULE_ID index, uint32_t colShft, uint32_t colMskLo, uint32_t colMskHi);

- CLHADDR<4:0> - DDRMEMCFG0<28:24>
- CLADDRHMSK<12:0> - DDRMEMCFG2<12:0>
- CLADDRLMSK<12:0> - DDRMEMCFG3<12:0>

PLIB_DDR_RowAddressSet Function

Initializes the DDR controller memory configuration registers for row address.

File

plib_ddr.h

C

```
void PLIB_DDR_RowAddressSet(DDR_MODULE_ID index, uint32_t rowShft, uint32_t rowMsk);
```

Returns

None.

Description

this function initializes the DDR controller memory configuration registers for the row address. The following register fields are programmed with this function:

- RWADDR<4:0> - DDRMEMCFG0<4:0>
- RWADDRMASK<12:0> - DDRMEMCFG1<12:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_RowAddressSet(DDR_ID_0, 12, 0x1FFF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rowShft	Number of bits to shift the row address
rowMask	Row address bit mask

Function

void PLIB_DDR_RowAddressSet(DDR_MODULE_ID index, uint32_t rowShft, uint32_t rowMsk);

e) SDRAM Timing Functions

PLIB_DDR_OdtReadParamSet Function

Sets the ODT parameters for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadParamSet(DDR_MODULE_ID index, uint8_t rLen, uint8_t rDly);
```

Returns

None.

Description

This function sets the ODT parameters for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadParamSet(DDR_ID_0, 2, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLen	The number of clocks the ODT is turned on for reads
rDly	The number of clocks after a read command before turning on ODT to the DDR

Function

void PLIB_DDR_OdtReadParamSet(DDR_MODULE_ID index, uint8_t rLen, uint8_t rDly);

PLIB_DDR_OdtWriteParamSet Function

Sets the ODT parameters for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteParamSet(DDR_MODULE_ID index, uint8_t wLen, uint8_t wDly);
```

Returns

None.

Description

This function sets the ODT parameters for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteParamSet(DDR_ID_0, 3, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
wLen	The number of clocks the ODT is turned on for writes
wDly	The number of clocks after a write command before turning on ODT to the DDR

Function

void PLIB_DDR_OdtWriteParamSet(DDR_MODULE_ID index, uint8_t wLen, uint8_t wDly);

PLIB_DDR_DataDelaySet Function

Initializes the DDR controller with the data delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_DataDelaySet(DDR_MODULE_ID index, uint8_t rLat, uint8_t wLat);
```

Returns

None.

Description

This function initializes the DDR controller with the data delays for the DDR in use. The following register fields are programmed with this function:

- RBENDDLY<3:0> - DDRDLYCFG2<31:28>
- NXTDATRQDLY<3:0> - DDRXFERCFG<3:0>
- NXTDATAVDLY<4:0> - DDRXFERCFG<7:4>, DDRDLYCFG1<28>
- RDATENDLY<3:0> - DDRXFERCFG<19:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_DataDelaySet(DDR_ID_0, 5, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLat	Read (CAS) latency in cycles (RL)
wLat	Write latency in cycles (WL)

Function

void PLIB_DDR_DataDelaySet(DDR_MODULE_ID index, uint8_t rLat, uint8_t wLat);

PLIB_DDR_PowerDownDelaySet Function

Initializes the DDR controller with the power down delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PowerDownDelaySet(DDR_MODULE_ID index, uint32_t pwrDnDly, uint32_t tCKE, uint32_t tXP);
```

Returns

None.

Description

This function initializes the DDR controller with the power down delays for the DDR in use. The folPowerDownDelaySetlowing register fields are programmed with this function:

- PWRDNDLY<7:0> - DDRPWRCFG<11:4>
- PWRDNMINDLY<3:0> - DDR2DLYCFG1<19:16>
- PWRDNEXDLY<5:0> - DDR2DLYCFG1<25:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define CLK_PERIOD          2500 // psec
#define CTRL_CLK_PERIOD     (CLK_PERIOD * 2)
#define PWR_DN_DLY          8
#define tCKE                3
#define tXP                 2

PLIB_DDR_PowerDownDelaySet(DDR_ID_0, PWR_DN_DLY, tCKE, tXP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pwrDnDly	Power down delay (entry) in cycles
tCKE	Power down minimum delay in cycles (tCKE)
tXP	Power down exit delay in cycles (tXP)

Function

void PLIB_DDR_PowerDownDelaySet(DDR_MODULE_ID index, uint32_t pwrDnDly,
uint32_t tCKE, uint32_t tXP);

PLIB_DDR_SelfRefreshDelaySet Function

Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_SelfRefreshDelaySet(DDR_MODULE_ID index, uint32_t slfRefDly, uint32_t tCKE, uint32_t tDLLK);
```

Returns

None.

Description

This function initializes the DDR controller with the self-refresh delays for the specific DDR in use. The following register fields are programmed

with this function:

- SLFREFDLY<9:0> - DDRPWRCFG<21:12>
- SLFREFMINDLY<7:0> - DDRDLYCFG1<7:0>
- SLFREFEXDLY<8:0> - DDRDLYCFG1<15:8>, DDR2DLYCFG1<30>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define SELF_REF_DLY        17
#define tCKE                3
PLIB_DDR_SelfRefreshDelaySet(DDR_ID_0, SELF_REF_DLY, tCKE, 85);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
slfRefDly	Self refresh delay in number of clock cycles divided by 1024 (e.g., 1 = 1024 clocks, 2 = 2048 clocks, etc.)
tCKE	Power down minimum delay in cycles (tCKE)
tDLLK	DLL lock delay time in cycles (tDLLK)

Function

void PLIB_DDR_SelfRefreshDelaySet(DDR_MODULE_ID index, uint32_t slfRefDly,

uint32_t tCKE, uint32_t tDLLK);

PLIB_DDR_PrechargAllBanksSet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PrechargAllBanksSet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- PCHRGALLDLY<3:0> - DDRDLYCFG2<3:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRP             12500   // psec

PLIB_DDR_PrechargAllBanksSet(DDR_ID_0, tRP, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

tRP	Precharge to active delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_PrechargAllBanksSet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);

PLIB_DDR_ReadWriteDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen, uint8_t wLat, uint8_t rLat);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- R2WDLY<3:0> - DDRDLYCFG0<27:24>
- RMWDLY<3:0> - DDRDLYCFG0<31:28>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
#define WL  4
#define RL  5
PLIB_DDR_ReadWriteDelaySet(DDR_ID_0 BL, WL, RL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
rLat	Read latency in cycles (RL)

Function

void PLIB_DDR_ReadWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen,

uint8_t wLat, uint8_t rLat);

PLIB_DDR_RefreshTimingSet Function

Initializes the DDR controller refresh configuration.

File

plib_ddr.h

C

```
void PLIB_DDR_RefreshTimingSet(DDR_MODULE_ID index, uint32_t tRFC, uint32_t tRFI, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller refresh configuration and programs the refresh interval and delay between refreshes. The following register fields are programmed with this function:

- REFCNT<15:0> - DDRREFCFG<15:0>
- REFDLY<5:0> - DDRREFCFG<23:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)

#define tRFC            75000   // psec
#define tRFI            7800000 // psec

PLIB_DDR_RefreshConfig(DDR_ID_0, tRFC, tRFI, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRFC	Refresh Cycle Time in clock cycles
tRFI	Refresh Interval in clock cycles
ctrlClkPer	DDR Controller clock period

Function

void PLIB_DDR_RefreshTimingSet(DDR_MODULE_ID index, uint32_t tRFC, uint32_t tRFI,
uint32_t ctrlClkPer);

PLIB_DDR_WriteWriteDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- W2WDLY<3:0> - DDRDLYCFG0<19:16>
- W2WCSDLY<3:0> - DDRDLYCFG0<23:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
PLIB_DDR_WriteWriteDelaySet(DDR_ID_0, BL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)

Function

void PLIB_DDR_WriteWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen);

PLIB_DDR_PrechargeToRASDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PrechargeToRASDelaySet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- PCHRG2RASDLY<3:0> - DDRDLYCFG2<24:27>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRP             12500   // psec

PLIB_DDR_PrechargeToRASDelaySet(DDR_ID_0, tRP, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRP	Precharge to active delay (psec)
ctrlClkPer	DDR Controller clock period

Function

void PLIB_DDR_PrechargeToRASDelaySet(DDR_MODULE_ID index, uint32_t tRP,
uint32_t ctrlClkPer);

PLIB_DDR_RASToCASDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToCASDelaySet(DDR_MODULE_ID index, uint32_t tRCD, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2CASDLY<3:0> - DDRDLYCFG2<23:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRCD            12500   // psec

PLIB_DDR_RASToCASDelaySet(DDR_ID_0, tRCD, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRCD	RAS to CAS delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToCASDelaySet(DDR_MODULE_ID index, uint32_t tRCD,

uint32_t ctrlClkPer);

PLIB_DDR_RASToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRAS, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- RAS2PCHRGDLY<4:0> - DDRDLYCFG3<4:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRAS            45000   // psec

PLIB_DDR_RASToPrechargeDelaySet(DDR_ID_0, tRAS, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRAS	Active to Precharge delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRAS, uint32_t ctrlClkPer);

PLIB_DDR_RASToRASBankDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToRASBankDelaySet(DDR_MODULE_ID index, uint32_t tRC, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2RASSBNKDLY<5:0> - DDRDLYCFG3<13:8>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRC             57500   // psec

PLIB_DDR_RASToRASBankDelaySet(DDR_ID_0, tRC, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRC	Row cycle time (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToRASBankDelaySet(DDR_MODULE_ID index, uint32_t tRC, uint32_t ctrlClkPer);

PLIB_DDR_RASToRASDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToRASDelaySet(DDR_MODULE_ID index, uint32_t tRRD, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2RASDLY<3:0> - DDRDLYCFG2<19:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRRD            7500    // psec

PLIB_DDR_RASToRASDelaySet(DDR_ID_0, tRRD, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRRD	RAS to RAS delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToRASDelaySet(DDR_MODULE_ID index, uint32_t tRRD, uint32_t ctrlClkPer);

PLIB_DDR_ReadToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRTP, uint8_t bLen, uint32_t
ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- R2PCHRGDLY<3:0> - DDRDLYCFG2<11:8>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRTP            7500    // psec
#define BL              2

PLIB_DDR_ReadToPrechargeDelaySet(DDR_ID_0, tRTP, BL, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRTP	Read to precharge delay (psec)
bLen	Burst length in cycles (BL)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_ReadToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRTP,
uint8_t bLen, uint32_t ctrlClkPer);

PLIB_DDR_WriteToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tWR, uint8_t bLen, uint8_t wLat,
uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- W2PCHRGDLY<4:0> - DDRDLYCFG2<15:12>, DDRDLYCFG1<26>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRTP            7500    // psec
#define BL              2
#define wLat            4

PLIB_DDR_WriteToPrechargeDelaySet(DDR_ID_0, tRTP, BL, wLat, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tWR	Write recovery delay (psec)
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_WriteToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tWR,
uint8_t bLen, uint8_t wLat, uint32_t ctrlClkPer);

PLIB_DDR_ReadReadDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadReadDelaySet(DDR_MODULE_ID index, uint8_t bLen);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- R2RDLY<3:0> - DDRDLYCFG0<11:8>
- R2RCSDLY<3:0> - DDRDLYCFG0<15:12>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
PLIB_DDR_ReadReadDelaySet(DDR_ID_0, BL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)

Function

void PLIB_DDR_ReadReadDelaySet(DDR_MODULE_ID index, uint8_t bLen);

PLIB_DDR_WriteReadDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteReadDelaySet(DDR_MODULE_ID index, uint32_t tWTR, uint8_t bLen, uint8_t wLat, uint32_t
ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- W2RDLY<4:0> - DDRDLYCFG0<3:0>, DDRDLYCFG1<27>
- W2RCSDLY<4:0> - DDRDLYCFG0<7:4>, DDRDLYCFG1<28>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define CLK_PERIOD          2500 // psec
```

```
#define CTRL_CLK_PERIOD     (CLK_PERIOD * 2)
#define tWTR                7500

PLIB_DDR_WriteReadDelaySet(DDR_ID_0, tWTR, 2, 4, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tWTR	Write to read latency (in psecs)
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
ctrlClkPer	Control clock period (psec)

Function

void PLIB_DDR_WriteReadDelaySet(DDR_MODULE_ID index, uint32_t tWTR,
uint8_t bLen, uint8_t wLat, uint32_t ctrlClkPer);

f) PHY Initialization Functions

PLIB_DDR_PHY_DDRTypeSet Function

Sets the DRAM type for the PHY.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DDRTypeSet(DDR_MODULE_ID index, DDR_PHY_DDR_TYPE ddrType);
```

Returns

None.

Description

This function sets the DRAM type for the PHY.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DDRTypeSet(DDR_ID_0, DDR_PHY_DDR_TYPE_DDR2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
ddrType	DDR type (DDR2 or DDR3)

Function

void PLIB_DDR_PHY_DDRTypeSet(DDR_MODULE_ID index, DDR_PHY_DDR_TYPE ddrType);

PLIB_DDR_PHY_DllMasterDelayStartSet Function

Sets the start value of the digital DLL master delay line.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_MODULE_ID index, uint8_t dlyStart);
```

Returns

None.

Description

This function sets the start value of the digital DLL master delay line.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_ID_0, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dlyStart	

Function

void PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_MODULE_ID index, uint8_t dlyStart);

PLIB_DDR_PHY_DllRecalibDisable Function

Disables periodic recalibration of the internal digital DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllRecalibDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

this function disables periodic recalibration of the internal digital DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllRecalibDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_DllRecalibDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_DllRecalibEnable Function

Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllRecalibEnable(DDR_MODULE_ID index, uint32_t recalibCnt);
```

Returns

None.

Description

This function enables periodic recalibration of the internal digital DLL, and sets the recalibration period.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllRecalibEnable(DDR_ID_0, 100);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
recalibCnt	Period of DLL recalibration in units of (PHY clock * 256)

Function

void PLIB_DDR_PHY_DllRecalibEnable(DDR_MODULE_ID index, uint32_t recalibCnt);

PLIB_DDR_PHY_DriveStrengthSet Function

Disables On Die Termination.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

This function disables On Die Termination (ODT).

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_DriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_DrvStrgthCal Function

Calibrates the pad NFET and PFET output impedance.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DrvStrgthCal(DDR_MODULE_ID index, uint8_t nFet, uint8_t pFet);
```

Returns

None.

Description

This function calibrates the pad NFET and PFET output impedance.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DrvStrgthCal(DDR_ID_0, 0xF, 0xF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nFet	NFET impedance calibration value
pFet	PFET impedance calibration value

Function

void PLIB_DDR_PHY_DrvStrgthCal(DDR_MODULE_ID index, uint8_t nFet, uint8_t pFet);

PLIB_DDR_PHY_ExternalDllEnable Function

Enables the use of an external DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExternalDllEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the use of an external DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExternalDllEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExternalDllEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_ExtraClockDisable Function

Does not enable the drive pad for an extra clock cycle after a write burst.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExtraClockDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function does not enable the drive pad for an extra clock cycle after a write burst.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExtraClockDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExtraClockDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_ExtraClockEnable Function

Enables the drive pad for an extra clock cycle after a write burst.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExtraClockEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the drive pad for an extra clock cycle after a write burst.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExtraClockEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExtraClockEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_InternalDllEnable Function

Enables the use of the internal digital DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_InternalDllEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the use of the internal digital DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_InternalDllEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_InternalDllEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtCal Function

Calibrates the pull-up and pull-down ODT impedance.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCal(DDR_MODULE_ID index, uint8_t puCal, uint8_t pdCal);
```

Returns

None.

Description

This function calibrates the pull-up and pull-down ODT impedance.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCal(DDR_ID_0, 2, 2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
puCal	Pull-up calibration value
pdCal	Pull-down calibration value

Function

void PLIB_DDR_PHY_OdtCal(DDR_MODULE_ID index, uint8_t puCal, uint8_t pdCal);

PLIB_DDR_PHY_OdtCSDisable Function

Disables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCSDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCSDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtCSDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtCSEnable Function

Enables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCSEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCSEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtCSEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtDisable Function

Disables On Die Termination.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables On Die Termination.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtEnable Function

Enables On Die Termination and sets the value.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtEnable(DDR_MODULE_ID index, DDR_PHY_ODT odtVal);
```

Returns

None.

Description

This function enables On Die Termination (ODT) and sets the value.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtEnable(DDR_ID_0, DDR_PHY_ODT_75_OHM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtVal	The ODT value

Function

void PLIB_DDR_PHY_OdtEnable(DDR_MODULE_ID index, DDR_PHY_ODT odtVal);

PLIB_DDR_PHY_PadReceiveEnable Function

Enables pad receivers on bidirectional I/O.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PadReceiveEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables pad receivers on bidirectional I/O.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PadReceiveEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_PadReceiveEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_PreambleDlySet Function

Sets the length of the preamble for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PreambleDlySet(DDR_MODULE_ID index, DDR_PHY_PREAMBLE_DLY preDly);
```

Returns

None.

Description

This function sets the length of the preamble for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PreambleDlySet(DDR_ID_0, DDR_PHY_PREAMBLE_DLY_1_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
preDly	Preamble delay

Function

void PLIB_DDR_PHY_PreambleDlySet(DDR_MODULE_ID index, DDR_PHY_PREAMBLE_DLY preDly);

PLIB_DDR_PHY_ReadCASLatencySet Function

Sets the read CAS latency while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ReadCASLatencySet(DDR_MODULE_ID index, uint8_t rLat);
```

Returns

None.

Description

This function sets the read CAS latency while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ReadCASLatencySet(DDR_ID_0, 5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLat	Read CAS latency (RL)

Function

void PLIB_DDR_PHY_ReadCASLatencySet(DDR_MODULE_ID index, uint8_t rLat);

PLIB_DDR_PHY_SCLDelay Function

Disables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLDelay(DDR_MODULE_ID index, DDR_PHY_SCL_DELAY sclDly);
```

Returns

None.

Description

This function disables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLDelay(DDR_ID_0, DDR_PHY_SCL_DELAY_SINGLE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
sclDly	SCL delay

Function

void PLIB_DDR_PHY_SCLDelay(DDR_MODULE_ID index, DDR_PHY_SCL_DELAY sclDly);

PLIB_DDR_PHY_SCLEnable Function

Enables SCL on the Chip Select 'cs'.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLEnable(DDR_MODULE_ID index, uint8_t cs);
```

Returns

None.

Description

This function enables SCL on the Chip Select 'cs'.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cs	Chip select

Function

void PLIB_DDR_PHY_SCLEnable(DDR_MODULE_ID index, uint8_t cs);

PLIB_DDR_PHY_SCLTestBurstModeSet Function

Sets the burst mode of the DRAM while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_MODULE_ID index, DDR_PHY_SCL_BURST_MODE brstMode);
```

Returns

None.

Description

This function sets the burst mode of the DRAM while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_ID_0, DDR_PHY_SCL_BURST_MODE_8);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
brstMode	Burst mode of the DRAM

Function

void PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_MODULE_ID index, DDR_PHY_SCL_BURST_MODE brstMode);

PLIB_DDR_PHY_WriteCASLatencySet Function

Sets the write CAS latency while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCASLatencySet(DDR_MODULE_ID index, uint8_t wLat);
```

Returns

None.

Description

This function sets the write CAS latency while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCASLatencySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
wLat	Write CAS latency (WL)

Function

void PLIB_DDR_PHY_WriteCASLatencySet(DDR_MODULE_ID index, uint8_t wLat);

PLIB_DDR_PHY_PadReceiveDisable Function

Disables pad receivers on bidirectional I/O.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PadReceiveDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables pad receivers on bidirectional I/O.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PadReceiveDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_PadReceiveDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLCapClkDelaySet Function

Sets capture clock delay during SCL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_MODULE_ID index, uint8_t capDly);
```

Returns

None.

Description

This function sets capture clock delay during SCL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
capDly	Capture Clock delay

Function

void PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_MODULE_ID index, uint8_t capDly);

PLIB_DDR_PHY_SCLDDRClkDelaySet Function

Sets DDR clock delay during SCL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_MODULE_ID index, uint8_t ddrDly);
```

Returns

None.

Description

This function sets DDR clock delay during SCL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
ddrDly	DDR Clock delay

Function

void PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_MODULE_ID index, uint8_t ddrDly);

PLIB_DDR_PHY_HalfRateSet Function

Sets the PHY to half rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_HalfRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the PHY to half rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_HalfRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_HalfRateSet(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLStart Function

Runs PHY Self Calibration.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLStart(DDR_MODULE_ID index);
```

Returns

None.

Description

This function runs PHY Self Calibration.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLStart(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_SCLStart(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLStatus Function

Checks status of PHY Self Calibration.

File

plib_ddr.h

C

```
bool PLIB_DDR_PHY_SCLStatus(DDR_MODULE_ID index);
```

Returns

- true - The PHY SCL has passed
- false - The PHY SCL has not passed

Description

This function checks status of PHY Self Calibration.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLStatus(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_DDR_PHY_SCLStatus(DDR_MODULE_ID index);

PLIB_DDR_PHY_AddCtlDriveStrengthSet Function

Sets the drive strength for the PHY address and control pads.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

Sets the drive strength for the PHY address and control pads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_DataDriveStrengthSet Function

Sets the drive strength for the PHY data pads.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DataDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

Sets the drive strength for the PHY data pads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DataDriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_DataDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_WriteCmdDelayDisable Function

Disables write command delay.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

Disables write command delay.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_WriteCmdDelayEnable Function

Enables write command delay. The extra delay is needed for devices with even write latency (WL).

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

Enables write command delay. The extra delay is needed for devices with even write latency (WL).

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_MODULE_ID index);

g) Feature Existence Functions

PLIB_DDR_ExistsAddressMapping Function

Identifies whether the AddressMapping feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAddressMapping(DDR_MODULE_ID index);
```

Returns

- true - The AddressMapping feature is supported on the device

- false - The AddressMapping feature is not supported on the device

Description

This function identifies whether the AddressMapping feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_RowAddressSet
- PLIB_DDR_ColumnAddressSet
- PLIB_DDR_BankAddressSet
- PLIB_DDR_ChipSelectAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAddressMapping(DDR_MODULE_ID index)

PLIB_DDR_ExistsArbitrationControl Function

Identifies whether the ArbitrationControl feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsArbitrationControl(DDR_MODULE_ID index);
```

Returns

- true - The ArbitrationControl feature is supported on the device
- false - The ArbitrationControl feature is not supported on the device

Description

This function identifies whether the ArbitrationControl feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_TargetSelect
- PLIB_DDR_MinLimit
- PLIB_DDR_ReqPeriod
- PLIB_DDR_MinCommand

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsArbitrationControl(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoPowerDown Function

Identifies whether the AutoPowerDown feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoPowerDown(DDR_MODULE_ID index);
```

Returns

- true - The AutoPowerDown feature is supported on the device
- false - The AutoPowerDown feature is not supported on the device

Description

This function identifies whether the AutoPowerDown feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoPowerDownEnable
- PLIB_DDR_AutoPowerDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoPowerDown(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoPrecharge Function

Identifies whether the AutoPrecharge feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoPrecharge(DDR_MODULE_ID index);
```

Returns

- true - The AutoPrecharge feature is supported on the device
- false - The AutoPrecharge feature is not supported on the device

Description

This function identifies whether the AutoPrecharge feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoPchrgEnable
- PLIB_DDR_AutoPchrgDisable
- PLIB_DDR_AutoPchrgPowerDownEnable
- PLIB_DDR_AutoPchrgPowerDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoPrecharge(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoSelfRefresh Function

Identifies whether the AutoSelfRefresh feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoSelfRefresh(DDR_MODULE_ID index);
```

Returns

- true - The AutoSelfRefresh feature is supported on the device
- false - The AutoSelfRefresh feature is not supported on the device

Description

This function identifies whether the AutoSelfRefresh feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoSelfRefreshEnable
- PLIB_DDR_AutoSelfRefreshDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoSelfRefresh(DDR_MODULE_ID index)

PLIB_DDR_ExistsDDRCommands Function

Identifies whether the DDRCommands feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsDDRCommands(DDR_MODULE_ID index);
```

Returns

- true - The DDRCommands feature is supported on the device
- false - The DDRCommands feature is not supported on the device

Description

This function identifies whether the DDRCommands feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_MaxCmdBrstCntSet
- PLIB_DDR_NumHostCmdsSet
- PLIB_DDR_CmdDataWrite
- PLIB_DDR_CmdDataValid
- PLIB_DDR_CmdDataSend
- PLIB_DDR_CmdDataIsComplete
- PLIB_DDR_ControllerEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsDDRCommands(DDR_MODULE_ID index)

PLIB_DDR_ExistsDDRControllerConfig Function

Identifies whether the DDRControllerConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsDDRControllerConfig(DDR_MODULE_ID index);
```

Returns

- true - The DDRControllerConfig feature is supported on the device
- false - The DDRControllerConfig feature is not supported on the device

Description

This function identifies whether the DDRControllerConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_BigEndianSet
- PLIB_DDR_LittleEndianSet
- PLIB_DDR_FullRateSet
- PLIB_DDR_HalfRateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsDDRControllerConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsODTConfig Function

Identifies whether the ODTConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsODTConfig(DDR_MODULE_ID index);
```

Returns

- true - The ODTConfig feature is supported on the device

- false - The ODTConfig feature is not supported on the device

Description

This function identifies whether the ODTConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_OdtReadEnable
- PLIB_DDR_OdtReadDisable
- PLIB_DDR_OdtWriteEnable
- PLIB_DDR_OdtWriteDisable
- PLIB_DDR_OdtWriteParamSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsODTConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_DLLCalibration Function

Identifies whether the PHY_DLLCalibration feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_DLLCalibration(DDR_MODULE_ID index);
```

Returns

- true - The PHY_DLLCalibration feature is supported on the device
- false - The PHY_DLLCalibration feature is not supported on the device

Description

This function identifies whether the PHY_DLLCalibration feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_DllRecalibEnable
- PLIB_DDR_PHY_DllRecalibDisable
- PLIB_DDR_PHY_DllMasterDelayStartSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_DLLCalibration(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_PadConfig Function

Identifies whether the PHY_PadConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_PadConfig(DDR_MODULE_ID index);
```

Returns

- true - The PHY_PadConfig feature is supported on the device
- false - The PHY_PadConfig feature is not supported on the device

Description

This function identifies whether the PHY_PadConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_OdtEnable
- PLIB_DDR_PHY_OdtDisable
- PLIB_DDR_PHY_DriveStrengthSet
- PLIB_DDR_PHY_OdtCal
- PLIB_DDR_PHY_DrvStrgthCal
- PLIB_DDR_PHY_ExtraClockEnable
- PLIB_DDR_PHY_ExtraClockDisable
- PLIB_DDR_PHY_InternalDllEnable
- PLIB_DDR_PHY_ExternalDllEnable
- PLIB_DDR_PHY_PadReceiveEnable
- PLIB_DDR_PHY_PreambleDlySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_PadConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_SCLConfig Function

Identifies whether the PHY_SCLConfig feature exists on the DDR module

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_SCLConfig(DDR_MODULE_ID index);
```

Returns

- true - The PHY_SCLConfig feature is supported on the device
- false - The PHY_SCLConfig feature is not supported on the device

Description

This function identifies whether the PHY_SCLConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_SCLTestBurstModeSet
- PLIB_DDR_PHY_DDRTypeSet
- PLIB_DDR_PHY_ReadCASLatencySet
- PLIB_DDR_PHY_WriteCASLatencySet
- PLIB_DDR_PHY_OdtCSEnable

- PLIB_DDR_PHY_OdtCSDisable
- PLIB_DDR_PHY_SCLDelay
- PLIB_DDR_PHY_SCLEnable
- PLIB_DDR_PHY_SCLDDRClkDelaySet
- PLIB_DDR_PHY_SCLCapClkDelaySet
- PLIB_DDR_PHY_SCLStart
- PLIB_DDR_PHY_SCLStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_SCLConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsRefreshConfig Function

Identifies whether the RefreshConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsRefreshConfig(DDR_MODULE_ID index);
```

Returns

- true - The RefreshConfig feature is supported on the device
- false - The RefreshConfig feature is not supported on the device

Description

This function identifies whether the RefreshConfig feature is available on the DDR module. When this function returns true, this function is supported on the device:

- PLIB_DDR_RefreshConfig

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsRefreshConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsTimingDelays Function

Identifies whether the TimingDelays feature exists on the DDR module

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsTimingDelays(DDR_MODULE_ID index);
```

Returns

- true - The TimingDelays feature is supported on the device
- false - The TimingDelays feature is not supported on the device

Description

This function identifies whether the TimingDelays feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_ReadWriteDelaySet
- PLIB_DDR_SelfRefreshDelaySet
- PLIB_DDR_PowerDownDelaySet
- PLIB_DDR_PchrgDelaySet
- PLIB_DDR_AddressDelaySet
- PLIB_DDR_DataDelaySet
- PLIB_DDR_TfawDelaySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsTimingDelays(DDR_MODULE_ID index)

h) Data Types and Constants

DDR_CMD_IDLE_NOP Macro

File

plib_ddr.h

C

```
#define DDR_CMD_IDLE_NOP 0x00FFFFFF
```

Section

Constants and Data Types

DDR_PHY_DDR_TYPE Enumeration

Defines the DDR type.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_DDR_TYPE_DDR2,
  DDR_PHY_DDR_TYPE_DDR3
} DDR_PHY_DDR_TYPE;
```

Description

DDR Type

This data type defines the DDR type.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_LOAD_MODE Macro

File

plib_ddr.h

C

```
#define DDR_CMD_LOAD_MODE 0x00FFF001
```

Description

This is macro DDR_CMD_LOAD_MODE.

DDR_PHY_DRIVE_STRENGTH Enumeration

Defines the PHY Pad drive strength value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_DRIVE_STRENGTH_60,
  DDR_PHY_DRIVE_STRENGTH_FULL
} DDR_PHY_DRIVE_STRENGTH;
```

Description

PHY Pad Drive Strength Value

This data type defines the PHY Pad drive strength value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_PRECHARGE_ALL Macro

File

plib_ddr.h

C

```
#define DDR_CMD_PRECHARGE_ALL 0x00FFF401
```

Description

This is macro DDR_CMD_PRECHARGE_ALL.

DDR_PHY_ODT Enumeration

Defines the ODT termination value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_ODT_75_OHM,
  DDR_PHY_ODT_150_OHM
} DDR_PHY_ODT;
```

Description

On-Die-Termination Value

This data type defines the ODT termination value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_REFRESH Macro

File

plib_ddr.h

C

```
#define DDR_CMD_REFRESH 0x00FFF801
```

Description

This is macro DDR_CMD_REFRESH.

DDR_PHY_PREAMBLE_DLY Enumeration

Defines the PHY preamble delay value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_PREAMBLE_DLY_2_0,
  DDR_PHY_PREAMBLE_DLY_1_5,
  DDR_PHY_PREAMBLE_DLY_1_0
} DDR_PHY_PREAMBLE_DLY;
```

Description

PHY Preamble Delay

This data type defines the PHY preamble delay value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_PHY_SCL_BURST_MODE Enumeration

Defines the burst mode for SCL.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_SCL_BURST_MODE_4,
  DDR_PHY_SCL_BURST_MODE_8
} DDR_PHY_SCL_BURST_MODE;
```

Description

PHY Preamble Delay

This data type defines the burst mode for SCL.

Remarks

The actual definition of this enumeration is device-specific.

DDR_PHY_SCL_DELAY Enumeration

Defines the SCL delay.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_SCL_DELAY_SINGLE,
  DDR_PHY_SCL_DELAY_DOUBLE
} DDR_PHY_SCL_DELAY;
```

Description

DDR Type

This data type defines the SCL delay.

Remarks

The actual definition of this enumeration is device-specific.

DDR_HOST_CMD_REG Enumeration

Defines the DDR host command register.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_HOST_CMD_REG_100,
  DDR_HOST_CMD_REG_101,
  DDR_HOST_CMD_REG_102,
  DDR_HOST_CMD_REG_103,
  DDR_HOST_CMD_REG_104,
  DDR_HOST_CMD_REG_105,
  DDR_HOST_CMD_REG_106,
  DDR_HOST_CMD_REG_107,
  DDR_HOST_CMD_REG_108,
  DDR_HOST_CMD_REG_109,
  DDR_HOST_CMD_REG_110,
  DDR_HOST_CMD_REG_111,
  DDR_HOST_CMD_REG_112,
  DDR_HOST_CMD_REG_113,
  DDR_HOST_CMD_REG_114,
  DDR_HOST_CMD_REG_115,
  DDR_HOST_CMD_REG_200,
  DDR_HOST_CMD_REG_201,
  DDR_HOST_CMD_REG_202,
  DDR_HOST_CMD_REG_203,
  DDR_HOST_CMD_REG_204,
  DDR_HOST_CMD_REG_205,
  DDR_HOST_CMD_REG_206,
  DDR_HOST_CMD_REG_207,
  DDR_HOST_CMD_REG_208,
  DDR_HOST_CMD_REG_209,
  DDR_HOST_CMD_REG_210,
  DDR_HOST_CMD_REG_211,
  DDR_HOST_CMD_REG_212,
  DDR_HOST_CMD_REG_213,
  DDR_HOST_CMD_REG_214,
  DDR_HOST_CMD_REG_215
} DDR_HOST_CMD_REG;
```

Description

DDR Host Command Register

this data type defines the DDR host command register.

Remarks

The actual definition of this enumeration is device-specific.

DDR_MODULE_ID Enumeration

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_ID_1,
  DDR_NUMBER_OF_MODULES
} DDR_MODULE_ID;
```

Members

Members	Description
DDR_ID_1	DDR 1
DDR_NUMBER_OF_MODULES	Total number of DDR modules available

Description

Enumeration: DDR_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

DDR_TARGET Enumeration

Defines the different targets to which the DDR controller is connected.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_TARGET_CPU,
  DDR_TARGET_GC_IN,
  DDR_TARGET_GC_OUT,
  DDR_TARGET_GPU_IN,
  DDR_TARGET_GPU_OUT
} DDR_TARGET;
```

Description

Targets

This data type defines the different targets to which the DDR controller is connected.

Remarks

The actual definition of this enumeration is device-specific.

Files

Files

Name	Description
plib_ddr.h	Defines the DDR Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_ddr.h

Defines the DDR Peripheral Library Interface.

Functions

	Name	Description
	PLIB_DDR_AutoPchrgDisable	Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPchrgEnable	Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPchrgPowerDownDisable	Prevents the DDR controller from automatically entering Precharge Power-down mode.
	PLIB_DDR_AutoPchrgPowerDownEnable	Enables the DDR controller to automatically enter Precharge Power-down mode.
	PLIB_DDR_AutoPowerDownDisable	Prevents the DDR controller from automatically entering Power-down mode.
	PLIB_DDR_AutoPowerDownEnable	Enables the DDR controller to automatically enter Power-down mode.
	PLIB_DDR_AutoSelfRefreshDisable	Prevents the DDR controller from automatically entering Self-refresh mode.
	PLIB_DDR_AutoSelfRefreshEnable	Enables the DDR controller to automatically enter Self-refresh mode.
	PLIB_DDR_BankAddressSet	Initializes the DDR controller memory configuration registers for bank address.
	PLIB_DDR_BigEndianSet	Sets the DDR data endianness to big.
	PLIB_DDR_ChipSelectAddressSet	Initializes the DDR controller memory configuration registers for Chip Select address.
	PLIB_DDR_CmdDataIsComplete	Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.
	PLIB_DDR_CmdDataSend	Transmits the data in the command registers to the SDRAM.
	PLIB_DDR_CmdDataValid	Indicates to the controller that the data in the command registers is valid.
	PLIB_DDR_CmdDataWrite	Writes an SDRAM command word to a command register in the controller.
	PLIB_DDR_ColumnAddressSet	Initializes the DDR controller memory configuration registers for the column address.
	PLIB_DDR_ControllerEnable	Enables the controller for normal operations after SDRAM is initialized.
	PLIB_DDR_DataDelaySet	Initializes the DDR controller with the data delays needed for the specific DDR in use.
	PLIB_DDR_ExistsAddressMapping	Identifies whether the AddressMapping feature exists on the DDR module.
	PLIB_DDR_ExistsArbitrationControl	Identifies whether the ArbitrationControl feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPowerDown	Identifies whether the AutoPowerDown feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPrecharge	Identifies whether the AutoPrecharge feature exists on the DDR module.
	PLIB_DDR_ExistsAutoSelfRefresh	Identifies whether the AutoSelfRefresh feature exists on the DDR module.
	PLIB_DDR_ExistsDDRCommands	Identifies whether the DDRCommands feature exists on the DDR module.
	PLIB_DDR_ExistsDDRControllerConfig	Identifies whether the DDRControllerConfig feature exists on the DDR module.
	PLIB_DDR_ExistsODTConfig	Identifies whether the ODTConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_DLLCalibration	Identifies whether the PHY_DLLCalibration feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_PadConfig	Identifies whether the PHY_PadConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_SCLConfig	Identifies whether the PHY_SCLConfig feature exists on the DDR module
	PLIB_DDR_ExistsRefreshConfig	Identifies whether the RefreshConfig feature exists on the DDR module.
	PLIB_DDR_ExistsTimingDelays	Identifies whether the TimingDelays feature exists on the DDR module
	PLIB_DDR_FullRateSet	Sets the DDR controller to Full-rate mode.
	PLIB_DDR_HalfRateSet	Sets the DDR controller to Half-rate mode.
	PLIB_DDR_LittleEndianSet	Sets the DDR data endianness to little.
	PLIB_DDR_MaxCmdBrstCntSet	Sets the maximum number of commands that can be written to the controller in Burst mode.
	PLIB_DDR_MaxPendingRefSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_MinCommand	Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.
	PLIB_DDR_MinLimit	Sets the minimum number of bursts for a DDR target.
	PLIB_DDR_NumHostCmdsSet	Sets the number of commands to be transmitted to the SDRAM.
	PLIB_DDR_OdtReadDisable	Selects which Chip Select to disable ODT for data reads.
	PLIB_DDR_OdtReadEnable	Selects which Chip Select to enable ODT for data reads.
	PLIB_DDR_OdtReadParamSet	Sets the ODT parameters for data reads.
	PLIB_DDR_OdtWriteDisable	Selects which Chip Select to disable ODT for data writes.
	PLIB_DDR_OdtWriteEnable	Selects which Chip Select to enable ODT for data writes.
	PLIB_DDR_OdtWriteParamSet	Sets the ODT parameters for data writes.
	PLIB_DDR_PHY_AddCtlDriveStrengthSet	Sets the drive strength for the PHY address and control pads.
	PLIB_DDR_PHY_DataDriveStrengthSet	Sets the drive strength for the PHY data pads.
	PLIB_DDR_PHY_DDRTypeSet	Sets the DRAM type for the PHY.

	PLIB_DDR_PHY_DllMasterDelayStartSet	Sets the start value of the digital DLL master delay line.
	PLIB_DDR_PHY_DllRecalibDisable	Disables periodic recalibration of the internal digital DLL.
	PLIB_DDR_PHY_DllRecalibEnable	Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.
	PLIB_DDR_PHY_DriveStrengthSet	Disables On Die Termination.
	PLIB_DDR_PHY_DrvStrgthCal	Calibrates the pad NFET and PFET output impedance.
	PLIB_DDR_PHY_ExternalDllEnable	Enables the use of an external DLL.
	PLIB_DDR_PHY_ExtraClockDisable	Does not enable the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_ExtraClockEnable	Enables the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_HalfRateSet	Sets the PHY to half rate mode.
	PLIB_DDR_PHY_InternalDllEnable	Enables the use of the internal digital DLL.
	PLIB_DDR_PHY_OdtCal	Calibrates the pull-up and pull-down ODT impedance.
	PLIB_DDR_PHY_OdtCSDisable	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtCSEnable	Enables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtDisable	Disables On Die Termination.
	PLIB_DDR_PHY_OdtEnable	Enables On Die Termination and sets the value.
	PLIB_DDR_PHY_PadReceiveDisable	Disables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PadReceiveEnable	Enables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PreambleDlySet	Sets the length of the preamble for data writes.
	PLIB_DDR_PHY_ReadCASLatencySet	Sets the read CAS latency while running SCL test.
	PLIB_DDR_PHY_SCLCapClkDelaySet	Sets capture clock delay during SCL.
	PLIB_DDR_PHY_SCLDDRClkDelaySet	Sets DDR clock delay during SCL.
	PLIB_DDR_PHY_SCLDelay	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_SCLEnable	Enables SCL on the Chip Select 'cs'.
	PLIB_DDR_PHY_SCLStart	Runs PHY Self Calibration.
	PLIB_DDR_PHY_SCLStatus	Checks status of PHY Self Calibration.
	PLIB_DDR_PHY_SCLTestBurstModeSet	Sets the burst mode of the DRAM while running SCL test.
	PLIB_DDR_PHY_WriteCASLatencySet	Sets the write CAS latency while running SCL test.
	PLIB_DDR_PHY_WriteCmdDelayDisable	Disables write command delay.
	PLIB_DDR_PHY_WriteCmdDelayEnable	Enables write command delay. The extra delay is needed for devices with even write latency (WL).
	PLIB_DDR_PowerDownDelaySet	Initializes the DDR controller with the power down delays needed for the specific DDR in use.
	PLIB_DDR_PrechargAllBanksSet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_PrechargeToRASDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToCASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASBankDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_ReadReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_ReadToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_RefreshTimingSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_ReqPeriod	Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.
	PLIB_DDR_RowAddressSet	Initializes the DDR controller memory configuration registers for row address.
	PLIB_DDR_SelfRefreshDelaySet	Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.
	PLIB_DDR_TfawDelaySet	Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

	PLIB_DDR_WriteReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_WriteToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_WriteWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

Macros

	Name	Description
	DDR_CMD_IDLE_NOP	
	DDR_CMD_LOAD_MODE	This is macro DDR_CMD_LOAD_MODE.
	DDR_CMD_PRECHARGE_ALL	This is macro DDR_CMD_PRECHARGE_ALL.
	DDR_CMD_REFRESH	This is macro DDR_CMD_REFRESH.

Description

Dual Data Rate (DDR) SDRAM Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the DDR Peripheral Library for Microchip microcontrollers. The definitions in this file are for the DDR module.

File Name

plib_ddr.h

Company

Microchip Technology Inc.

EBI Peripheral Library

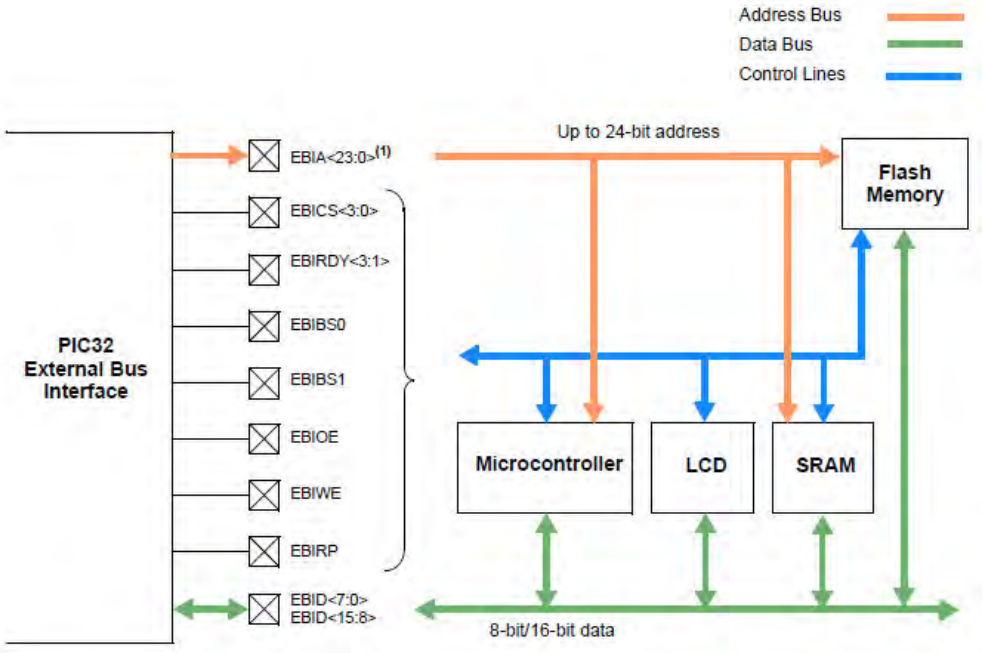
This section describes the External Bus Interface (EBI) Peripheral Library.

Introduction

This library provides a low-level abstraction of the EBI Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The External Bus Interface (EBI) allows for external physical memory to be mapped into the kernels virtual memory map. The External Bus Interface (EBI) module provides a high-speed, convenient way to interface external parallel memory devices to the PIC32 family device. With the EBI module, it is possible to connect asynchronous SRAM and NOR Flash devices, as well as non-memory devices such as camera sensors. The module also supports Low-Cost Controllerless (LCC) Graphics devices.



Note 1: No EBIA address pins are available on 64-pin devices and the EBIA<23:20> address pins are only available on 144-pin devices.

Using the Library

This topic describes the basic architecture of the EBI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ebi.h`

The interface to the EBI Peripheral library is defined in the plib_ebi.h header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the EBI Peripheral library must include `peripheral.h`.

Library File:

The EBI Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the EBI module.

Library Interface Section	Description
Get Functions	These functions can be used to return the status of a feature.

Set Functions	These functions can be used to configure features.
Feature Existence Functions	Lists the interface routines that describe whether or not the feature is supported by the device.

How the Library Works

Before using the EBI module, it must be enabled by the application developer(s). To work correctly, the device characteristics of the attached Memory/Flash must be set with the functions in this library. The Characteristics of the Memory/Flash will be found in the specific device data sheet.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Configuring the Library

The library is configured for the supported EBI module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Get Functions

	Name	Description
	PLIB_EBI_AddressHoldTimeGet	Returns the address hold time.
	PLIB_EBI_AddressSetupTimeGet	Returns the setup hold time.
	PLIB_EBI_BaseAddressGet	Returns the base address set for each Chip Select.
	PLIB_EBI_ControlEnableGet	Returns the status of the EBI enable bit.
	PLIB_EBI_DataTurnAroundTimeGet	Returns the data turn-around time set for the EBI bus.
	PLIB_EBI_FlashPowerDownModeGet	Returns the set power-down state.
	PLIB_EBI_FlashResetPinGet	Sets the control use of Flash Reset pin.
	PLIB_EBI_PageModeGet	Returns the Paging mode settings.
	PLIB_EBI_PageReadCycleTimeGet	Returns the cycle time for a page read.
	PLIB_EBI_ReadCycleTimeGet	Returns the read time in the number of clock cycles.
	PLIB_EBI_ReadyModeGet	Returns whether or not Ready mode was set.
	PLIB_EBI_StaticMemoryWidthRegisterGet	Returns the set width of the data bus.
	PLIB_EBI_WritePulseWidthGet	Returns the set hold time in clock cycles.
	PLIB_EBI_MemoryPageSizeGet	Returns the Paging mode settings.
	PLIB_EBI_FlashTimingGet	Returns the set Flash timing for external Flash.

b) Set Functions

	Name	Description
	PLIB_EBI_AddressPinEnableBitsSet	Sets the address pins used for EBI.
	PLIB_EBI_BaseAddressSet	Sets the base address for physical memory at each Chip Select pin.
	PLIB_EBI_ByteSelectPinSet	Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.
	PLIB_EBI_ChipSelectEnableSet	Sets the Chip Select pins for use with the EBI or GPIO.
	PLIB_EBI_DataEnableSet	Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.
	PLIB_EBI_FlashPowerDownModeSet	Sets the pin state for Flash devices on Power-down and Reset.
	PLIB_EBI_FlashResetPinSet	Sets the control use of the Flash Reset pin.
	PLIB_EBI_FlashTimingSet	Sets the timing for hold in reset for external Flash.
	PLIB_EBI_MemoryCharacteristicsSet	Sets the characteristics for memory or attached devices attached to the specified pin.
	PLIB_EBI_MemoryPagingSet	Sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	Sets the cycle time for page reading.
	PLIB_EBI_ReadyPin1ConfigSet	Sets the control use of ReadyPin1 and inverts the status.
	PLIB_EBI_ReadyPin2ConfigSet	Sets the control use of ReadyPin2 and inverts the status.
	PLIB_EBI_ReadyPin3ConfigSet	Sets the control use of ReadyPin3 and inverts the status.
	PLIB_EBI_ReadyPinSensSet	Sets the sensitivity of the Ready pin.
	PLIB_EBI_StaticMemoryWidthRegisterSet	Sets the data width of static memory.

	PLIB_EBI_WriteOutputControlSet	Sets the Write Enable and Output Enable control pins.
	PLIB_EBI_ControlEnableSet	Sets the EBI bus ON/OFF enable bit.
	PLIB_EBI_ReadyModeSet	Sets the use of Ready mode for each pin.

c) Feature Existence Functions

	Name	Description
	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module.
	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module.
	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module.
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module.
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module.
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module.
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module.
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module.
	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module.
	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module.
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module.
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module.
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module.

d) Data Types and Constants

	Name	Description
	EBI_ADDRESS_PORT	Selects the EBI address port.
	EBI_CS_TIMING	Selects the timing register set for Chip Select.
	EBI_MEMORY_SIZE	Selects the memory size for Chip Select.
	EBI_MEMORY_TYPE	Selects the memory type for Chip Select.
	EBI_MODULE_ID	Identifies the supported EBI modules.
	EBI_PAGE_SIZE	Selects the page size for the page mode device.
	EBI_STATIC_MEMORY_WIDTH	Selects the static memory width for the register EBISMTx.

Description

This section describes the Application Programming Interface (API) functions of the EBI Peripheral Library.

Refer to each section for a detailed description.

a) Get Functions

PLIB_EBI_AddressHoldTimeGet Function

Returns the address hold time.

File

plib_ebi.h

C

```
int PLIB_EBI_AddressHoldTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the Address and Data Hold time in the number of clocks.

Description

This function returns the address and data hold time.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_AddressHoldTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_AddressSetupTimeGet Function

Returns the setup hold time.

File

plib_ebi.h

C

```
int PLIB_EBI_AddressSetupTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the address setup time in the number of clocks.

Description

This function returns the setup hold time. A value of '0' is only valid in the case of SSRAM.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_AddressSetupTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_BaseAddressGet Function

Returns the base address set for each Chip Select.

File

plib_ebi.h

C

```
uint32_t PLIB_EBI_BaseAddressGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an unsigned integer that is the physical address of the attached device.

Description

This function returns the base address for each Chip Select pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select address pin address is being read

Function

PLIB_EBI_BaseAddressGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_ControlEnableGet Function

Returns the status of the EBI enable bit.

File

plib_ebi.h

C

```
bool PLIB_EBI_ControlEnableGet(EBI_MODULE_ID index);
```

Returns

Boolean:

- 1 = The EBI module is ON
- 0 = The EBI module is OFF

Description

This function returns the status of the EBI enable bit.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ControlEnableGet (EBI_MODULE_ID index)

PLIB_EBI_DataTurnAroundTimeGet Function

Returns the data turn-around time set for the EBI bus.

File

plib_ebi.h

C

```
int PLIB_EBI_DataTurnAroundTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles that is set in the turn-around register.

Description

This function returns the data turn-around time set for the EBI Bus. Returns the clock cycles (0-7) for static memory between read-to-write, write-to-read, and read-to-read when Chip Select changes.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_DataTurnAroundTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_FlashPowerDownModeGet Function

Returns the set power-down state.

File

plib_ebi.h

C

```
bool PLIB_EBI_FlashPowerDownModeGet(EBI_MODULE_ID index);
```

Returns

A bool, depending on the setting of PLIB_EBI_FlashPowerDownModeSet.

Description

This function returns the set power-down state.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_FlashPowerDownModeGet (EBI_MODULE_ID index)

PLIB_EBI_FlashResetPinGet Function

Sets the control use of Flash Reset pin.

File

plib_ebi.h

C

```
bool PLIB_EBI_FlashResetPinGet(EBI_MODULE_ID index);
```

Returns

Returns a Boolean.

Description

This function set the control use of the Flash Reset pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_FlashResetPinGet (EBI_MODULE_ID index)

PLIB_EBI_PageModeGet Function

Returns the Paging mode settings.

File

plib_ebi.h

C

```
bool PLIB_EBI_PageModeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

- 1 = Page Mode is used
- 0 = Page Mode is not used

Description

This function returns the state of the register PAGEMODE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_PageModeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_PageReadCycleTimeGet Function

Returns the cycle time for a page read.

File

plib_ebi.h

C

```
int PLIB_EBI_PageReadCycleTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles used to read a memory page.

Description

This function returns the cycle time for a page read. Read cycle time = TPRC + 1cycle. The value set and return are the same, the controller performs the read on the next clock, which is why there is a +1.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_PageReadCycleTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_ReadCycleTimeGet Function

Returns the read time in the number of clock cycles.

File

plib_ebi.h

C

```
int PLIB_EBI_ReadCycleTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the read cycle time in system clocks.

Description

This function returns the read time in number of clock cycles. Read Cycle time = TRC +1 clock cycle. The controller will always take one extra clock (the next clock) for a Read Cycle. The return will be the time set by the user.

Setting a Read Cycle time to a '0' will return a '0' when read, but the controller will add '1' to that value for the next available clock.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

ChipSelectNumber	Identifies which Chip Select number to which the device is attached

Function

PLIB_EBI_ReadCycleTimeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_ReadyModeGet Function

Returns whether or not Ready mode was set.

File

plib_ebi.h

C

```
bool PLIB_EBI_ReadyModeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

This function returns a bool.

- 1 = Ready input is used
- 0 = Ready input is not used

Description

This function returns the state of the register RDYMODE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_ReadyModeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_StaticMemoryWidthRegisterGet Function

Returns the set width of the data bus.

File

plib_ebi.h

C

```
EBI_STATIC_MEMORY_WIDTH PLIB_EBI_StaticMemoryWidthRegisterGet(EBI_MODULE_ID index, int RegisterNumber);
```

Returns

An enum type, which is the bus mode in use.

Description

This function returns the set width of the data bus.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_StaticMemoryWidthRegisterGet (EBI_MODULE_ID index,
int RegisterNumber)

PLIB_EBI_WritePulseWidthGet Function

Returns the set hold time in clock cycles.

File

plib_ebi.h

C

```
int PLIB_EBI_WritePulseWidthGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles to which the write pulse width is set.

Description

This function returns the set hold time in clock cycles. Write Pulse with = TWP + 1 clock cycle.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_WritePulseWidthGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_MemoryPageSizeGet Function

Returns the Paging mode settings.

File

plib_ebi.h

C

```
EBI_PAGE_SIZE PLIB_EBI_MemoryPageSizeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Size of the memory page.

Description

This function returns the state of the register PAGESIZE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_MemoryPageSizeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_FlashTimingGet Function

Returns the set Flash timing for external Flash.

File

plib_ebi.h

C

```
int PLIB_EBI_FlashTimingGet(EBI_MODULE_ID index);
```

Returns

Returns an integer, which is the number of clock cycles.

Description

This function returns the set number of clock cycles to hold the external Flash memory in reset.

Remarks

None.

Preconditions

MemoryType must be set to Flash.

Function

PLIB_EBI_FlashTimingGet (EBI_MODULE_ID index)

b) Set Functions

PLIB_EBI_AddressPinEnableBitsSet Function

Sets the address pins used for EBI.

File

plib_ebi.h

C

```
void PLIB_EBI_AddressPinEnableBitsSet(EBI_MODULE_ID index, EBI_ADDRESS_PORT AddressPort);
```

Returns

None.

Description

This function sets the address pins used for EBI.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
AddressPort	Identifies how many pins are used for addressing on the EBI bus

Function

PLIB_EBI_AddressPinEnableBitsSet (EBI_MODULE_ID index,
EBI_ADDRESS_PORT AddressPort)

PLIB_EBI_BaseAddressSet Function

Sets the base address for physical memory at each Chip Select pin.

File

plib_ebi.h

C

```
void PLIB_EBI_BaseAddressSet(EBI_MODULE_ID index, int ChipSelectNumber, uint32_t BaseAddress);
```

Returns

None.

Description

This function sets the base address for physical memory at each Chip Select pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select address pin is being assigned an address
BaseAddress	A physical address for memory

Function

PLIB_EBI_BaseAddressSet(EBI_MODULE_ID index, int ChipSelectNumber,
uint32_t BaseAddress)

PLIB_EBI_ByteSelectPinSet Function

Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.

File

plib_ebi.h

C

```
void PLIB_EBI_ByteSelectPinSet(EBI_MODULE_ID index, bool ByteSelect0, bool ByteSelect1);
```

Returns

None.

Description

This function sets the data Byte Select High <15:8> and Low <7:0> enable pins for use. If the system uses Byte Select High/Low pins for the data, the pins must be enabled for use in the EBI controller.

Remarks

None.

Preconditions

The EBIDEN0 and EBIDEN1 registers must first be enabled, which is done using the function PLIB_EBI_DataEnableSet.

Parameters

Parameters	Description
index	Identifier for the device instance
ByteSelect0	Identifies the Lower Byte Select Pin for enabling
ByteSelect1	Identifies the Upper Byte Select Pin for enabling

Function

PLIB_EBI_ByteSelectPinSet (EBI_MODULE_ID index, bool ByteSelect0, bool ByteSelect1)

PLIB_EBI_ChipSelectEnableSet Function

Sets the Chip Select pins for use with the EBI or GPIO.

File

plib_ebi.h

C

```
void PLIB_EBI_ChipSelectEnableSet(EBI_MODULE_ID index, bool ChipSelect0, bool ChipSelect1, bool
ChipSelect2, bool ChipSelect3);
```

Returns

None.

Description

This functions sets which Chip Select pins to use with the EBI or GPIO.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelect0	Identifies control of Chip Select 0 for enabling
ChipSelect1	Identifies control of Chip Select 1 for enabling
ChipSelect2	Identifies control of Chip Select 2 for enabling
ChipSelect3	Identifies control of Chip Select 3 for enabling

Function

PLIB_EBI_ChipSelectEnableSet (EBI_MODULE_ID index, bool ChipSelect0,
bool ChipSelect1, bool ChipSelect2, bool ChipSelect3)

PLIB_EBI_DataEnableSet Function

Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.

File

plib_ebi.h

C

```
void PLIB_EBI_DataEnableSet(EBI_MODULE_ID index, bool DataUpperByte, bool DataLowerByte);
```

Returns

None.

Description

This function sets the use of the Data Byte Select Pins, High and Low, for control with EBI or GPIO.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
DataUpperByte	Identifies control of Upper Data Byte for enabling
DataLowerByte	Identifies control of Lower Data Byte for enabling

Function

PLIB_EBI_DataEnableSet (EBI_MODULE_ID index, bool DataUpperByte, bool DataLowerByte)

PLIB_EBI_FlashPowerDownModeSet Function

Sets the pin state for Flash devices on Power-down and Reset.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashPowerDownModeSet(EBI_MODULE_ID index, bool FlashPowerDownMode);
```

Returns

None.

Description

This function sets the pin state for Flash devices upon a Power-down and Reset.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashPowerDownMode	A bool, which sets the power-down state for Flash memory

Function

PLIB_EBI_FlashPowerDownModeSet (EBI_MODULE_ID index, bool FlashPowerDownMode)

PLIB_EBI_FlashResetPinSet Function

Sets the control use of the Flash Reset pin.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashResetPinSet(EBI_MODULE_ID index, bool FlashResetPin);
```

Returns

None.

Description

This function sets the control use of the Flash Reset pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashReadPin	Flash Reset pin is used

Function

PLIB_EBI_FlashResetPinSet (EBI_MODULE_ID index, bool FlashReadPin)

PLIB_EBI_FlashTimingSet Function

Sets the timing for hold in reset for external Flash.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashTimingSet(EBI_MODULE_ID index, int FlashTiming);
```

Returns

None.

Description

This function sets the number of clock cycles to hold the external Flash memory in reset.

Remarks

None.

Preconditions

NOR Flash must be used with EBI instead of SRAM.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashTiming	An integer, which is the number of clock cycles

Function

PLIB_EBI_FlashTimingSet (EBI_MODULE_ID index, int FlashTiming)

PLIB_EBI_MemoryCharacteristicsSet Function

Sets the characteristics for memory or attached devices attached to the specified pin.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryCharacteristicsSet(EBI_MODULE_ID index, int ChipSelectNumber, EBI_MEMORY_TYPE
MemoryType, EBI_MEMORY_SIZE MemorySize, EBI_CS_TIMING TimingReg);
```

Returns

None.

Description

This function sets the characteristics for memory or attached devices attached to the specified pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select pin is used
MemoryType	Identifies which memory is used
MemorySize	An enum type, which sets the size of the attached memory device
TimingRegSet	Identifies the timing register

Function

PLIB_EBI_MemoryCharacteristicsSet(EBI_MODULE_ID index, int ChipSelectNumber,
EBI_MEMORY_TYPE MemoryType,
EBI_MEMORY_SIZE MemorySize,
EBI_CS_TIMING TimingReg)

PLIB_EBI_MemoryPagingSet Function

Sets the size of the memory page if paging is used.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryPagingSet(EBI_MODULE_ID index, int ChipSelectNumber, bool PageMode, EBI_PAGE_SIZE
MemoryPageSize);
```

Returns

- 1 = Device supports Page mode
- 0 = Device does not support Page mode

Description

This function sets the size of the memory page if paging is used.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Sets for that Chip Select pin <3:0>
PageMode	Enable or disable Page mode
MemoryPageSize	Size of memory pages

Function

PLIB_EBI_MemoryPagingSet (EBI_MODULE_ID index, int ChipSelectNumber,
bool PageMode, EBI_PAGE_SIZE MemoryPageSize)

PLIB_EBI_MemoryTimingConfigSet Function

Sets the cycle time for page reading.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryTimingConfigSet(EBI_MODULE_ID index, int CS_Timing_Reg, int PageReadTime, int
DataTurnAroundTime, int WritePulseWidth, int AddressHoldTime, int AddressSetupTime, int ReadCycleTime);
```

Returns

None.

Description

This function sets the cycle time for page reading.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
CS_Timing_Reg	Identifies which Chip Select Timing register to use EBISMT<0:2>
PageReadTime	The clock cycle needed for a Memory Page read
DataTurnAroundTime	Time between read-to-write, write-to-read, and read-to-read when Chip Select changes state.
WritePulseWidth	The clock cycles needed for a memory write
AddressHoldTime	The clock time needed for the address bus to hold
AddressSetupTime	The time needed for the address to settle
ReadCycleTime	_nt_

Function

PLIB_EBI_MemoryTimingConfigSet(EBI_MODULE_ID index, int CS_Timing_Reg,
int PageReadTime,
int DataTurnAroundTime,
int WritePulseWidth,
int AddressHoldTime,
int AddressSetupTime,
int ReadCycleTime)

PLIB_EBI_ReadyPin1ConfigSet Function

Sets the control use of ReadyPin1 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin1ConfigSet(EBI_MODULE_ID index, bool ReadyPin1Enable, bool ReadyPin1Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin1, and then inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin1Invert	Identifies the ON/OFF for invert
ReadyPin1Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin1ConfigSet (EBI_MODULE_ID index, bool ReadyPin1Enable,
bool ReadyPin1Invert)

PLIB_EBI_ReadyPin2ConfigSet Function

Sets the control use of ReadyPin2 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin2ConfigSet(EBI_MODULE_ID index, bool ReadyPin2Enable, bool ReadyPin2Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin2 and inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin2Invert	Identifies the ON/OFF for invert
ReadyPin2Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin2ConfigSet (EBI_MODULE_ID index, bool ReadyPin2Enable,
bool ReadyPin2Invert)

PLIB_EBI_ReadyPin3ConfigSet Function

Sets the control use of ReadyPin3 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin3ConfigSet(EBI_MODULE_ID index, bool ReadyPin3Enable, bool ReadyPin3Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin3, and then inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin3Invert	Identifies the ON/OFF for invert
ReadyPin3Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin3ConfigSet (EBI_MODULE_ID index, bool ReadyPin3Enable,
bool ReadyPin3Invert)

PLIB_EBI_ReadyPinSensSet Function

Sets the sensitivity of the Ready pin.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPinSensSet(EBI_MODULE_ID index, bool SensitivityControl);
```

Returns

None.

Description

This function sets the sensitivity of the Ready pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
SensitivityControl	Selection edge or level detection

Function

PLIB_EBI_ReadyPinSensSet (EBI_MODULE_ID index, bool SensitivityControl)

PLIB_EBI_StaticMemoryWidthRegisterSet Function

Sets the data width of static memory.

File

plib_ebi.h

C

```
void PLIB_EBI_StaticMemoryWidthRegisterSet(EBI_MODULE_ID index, int RegisterNumber, EBI_STATIC_MEMORY_WIDTH
StaticMemoryWidthRegister);
```

Returns

None.

Description

This function sets the data width of static memory.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
RegisterNumber	The ID for the register being set
StaticMemoryWidthRegister	Identifies a bus width of either 8 bits or 16 bits

Function

PLIB_EBI_StaticMemoryWidthRegisterSet (EBI_MODULE_ID index, int RegisterNumber,
EBI_STATIC_MEMORY_WIDTH StaticMemoryWidthRegister)

PLIB_EBI_WriteOutputControlSet Function

Sets the Write Enable and Output Enable control pins.

File

plib_ebi.h

C

```
void PLIB_EBI_WriteOutputControlSet(EBI_MODULE_ID index, bool WriteEnable, bool OutputEnable);
```

Returns

None.

Description

This function sets the Write Enable and Output Enable control pins.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
WriteEnable	Used for enabling Write Enable
OutputEnable	Used for enabling Output Enable

Function

PLIB_EBI_WriteOutputControlSet (EBI_MODULE_ID index, bool WriteEnable,
bool OutputEnable)

PLIB_EBI_ControlEnableSet Function

Sets the EBI bus ON/OFF enable bit.

File

plib_ebi.h

C

```
void PLIB_EBI_ControlEnableSet(EBI_MODULE_ID index, bool EnableBit);
```

Returns

None.

Description

This function sets the EBI bus ON/OFF enable bit.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
EnableBit	Identifies the enable bit

Function

PLIB_EBI_ControlEnableSet (EBI_MODULE_ID index, bool EnableBit)

PLIB_EBI_ReadyModeSet Function

Sets the use of Ready mode for each pin.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyModeSet(EBI_MODULE_ID index, bool ReadyPin0, bool ReadyPin1, bool ReadyPin2);
```

Returns

None.

Description

This function sets the use of Ready mode for each pin. The attached device will either pull the ready pin high or low.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance; ReadyPin0, ReadyPin1, ReadyPin2
ReadyPin(x)	Identifies the ready pin (1-3)

Function

PLIB_EBI_ReadyModeSet (EBI_MODULE_ID index, bool ReadyPin0, bool ReadyPin1,
bool ReadyPin2)

c) Feature Existence Functions

PLIB_EBI_ExistsAddressHoldTime Function

Identifies whether the AddressHoldTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressHoldTime(EBI_MODULE_ID index);
```

Returns

- true - The AddressHoldTime feature is supported on the device
- false - The AddressHoldTime feature is not supported on the device

Description

This function identifies whether the AddressHoldTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_AddressHoldTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressHoldTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsAddressPinEnableBits Function

Identifies whether the AddressPinEnableBits feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressPinEnableBits(EBI_MODULE_ID index);
```

Returns

- true - The AddressPinEnableBits feature is supported on the device
- false - The AddressPinEnableBits feature is not supported on the device

Description

This function identifies whether the AddressPinEnableBits feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_AddressPinEnableBitsSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressPinEnableBits(EBI_MODULE_ID index)

PLIB_EBI_ExistsAddressSetupTime Function

Identifies whether the AddressSetupTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressSetupTime(EBI_MODULE_ID index);
```

Returns

- true - The AddressSetupTime feature is supported on the device
- false - The AddressSetupTime feature is not supported on the device

Description

This function identifies whether the AddressSetupTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_AddressSetupTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressSetupTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsBaseAddress Function

Identifies whether the Base_Address feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsBaseAddress(EBI_MODULE_ID index);
```

Returns

- true - The Base_Address feature is supported on the device
- false - The Base_Address feature is not supported on the device

Description

This function identifies whether the Base_Address feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_BaseAddressSet
- PLIB_EBI_BaseAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsBaseAddress(EBI_MODULE_ID index)

PLIB_EBI_ExistsByteSelectPin Function

Identifies whether the ByteSelectPin feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsByteSelectPin(EBI_MODULE_ID index);
```

Returns

- true - The ByteSelectPin feature is supported on the device
- false - The ByteSelectPin feature is not supported on the device

Description

This function identifies whether the ByteSelectPin feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ByteSelectPinSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsByteSelectPin(EBI_MODULE_ID index)

PLIB_EBI_ExistsChipSelectEnable Function

Identifies whether the ChipSelectEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsChipSelectEnable(EBI_MODULE_ID index);
```

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ChipSelectEnable feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ChipSelectEnableSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsChipSelectEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsControlEnable Function

Identifies whether the ControlEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsControlEnable(EBI_MODULE_ID index);
```

Returns

- true - The ControlEnable feature is supported on the device
- false - The ControlEnable feature is not supported on the device

Description

This function identifies whether the ControlEnable feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_ControlEnableSet
- PLIB_EBI_ControlEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsControlEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsDataEnable Function

Identifies whether the DataEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsDataEnable(EBI_MODULE_ID index);
```

Returns

- true - The DataEnable feature is supported on the device
- false - The DataEnable feature is not supported on the device

Description

This function identifies whether the DataEnable feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_DataEnableSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsDataEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsDataTurnAroundTime Function

Identifies whether the DataTurnAroundTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsDataTurnAroundTime(EBI_MODULE_ID index);
```

Returns

- true - The DataTurnAroundTime feature is supported on the device
- false - The DataTurnAroundTime feature is not supported on the device

Description

This function identifies whether the DataTurnAroundTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_DataTurnAroundTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsDataTurnAroundTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashPowerDownMode Function

Identifies whether the FlashPowerDownMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashPowerDownMode(EBI_MODULE_ID index);
```

Returns

- true - The FlashPowerDownMode feature is supported on the device
- false - The FlashPowerDownMode feature is not supported on the device

Description

This function identifies whether the FlashPowerDownMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashPowerDownModeSet
- PLIB_EBI_FlashPowerDownModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashPowerDownMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashResetPin Function

Identifies whether the FlashResetPin feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashResetPin(EBI_MODULE_ID index);
```

Returns

- true - The FlashResetPin feature is supported on the device
- false - The FlashResetPin feature is not supported on the device

Description

This function identifies whether the FlashResetPin feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashResetPinSet
- PLIB_EBI_FlashResetPinGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashResetPin(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashTiming Function

Identifies whether the FlashTiming feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashTiming(EBI_MODULE_ID index);
```

Returns

- true - The FlashTiming feature is supported on the device
- false - The FlashTiming feature is not supported on the device

Description

This function identifies whether the FlashTiming feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashTimingSet
- PLIB_EBI_FlashTimingGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashTiming(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryCharacteristics Function

Identifies whether the MemoryCharacteristics feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryCharacteristics(EBI_MODULE_ID index);
```

Returns

- true - The MemoryCharacteristics feature is supported on the device

- false - The MemoryCharacteristics feature is not supported on the device

Description

This function identifies whether the MemoryCharacteristics feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryCharacteristicsSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryCharacteristics(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryPaging Function

Identifies whether the MemoryPaging feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryPaging(EBI_MODULE_ID index);
```

Returns

- true - The MemoryPaging feature is supported on the device
- false - The MemoryPaging feature is not supported on the device

Description

This function identifies whether the MemoryPaging feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryPagingSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryPaging(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryTimingConfig Function

Identifies whether the MemoryTimingConfig feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryTimingConfig(EBI_MODULE_ID index);
```

Returns

- true - The MemoryTimingConfig feature is supported on the device
- false - The MemoryTimingConfig feature is not supported on the device

Description

This function identifies whether the MemoryTimingConfig feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryTimingConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryTimingConfig(EBI_MODULE_ID index)

PLIB_EBI_ExistsPageMode Function

Identifies whether the PageMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsPageMode(EBI_MODULE_ID index);
```

Returns

- true - The PageMode feature is supported on the device
- false - The PageMode feature is not supported on the device

Description

This function identifies whether the PageMode feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_PageModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsPageMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsPageReadTime Function

Identifies whether the PageReadTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsPageReadTime(EBI_MODULE_ID index);
```

Returns

- true - The PageReadTime feature is supported on the device
- false - The PageReadTime feature is not supported on the device

Description

This function identifies whether the PageReadTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_PageReadCycleTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsPageReadTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadCycleTime Function

Identifies whether the ReadCycleTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadCycleTime(EBI_MODULE_ID index);
```

Returns

- true - The ReadCycleTime feature is supported on the device
- false - The ReadCycleTime feature is not supported on the device

Description

This function identifies whether the ReadCycleTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadCycleTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadCycleTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyMode Function

Identifies whether the ReadyMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyMode(EBI_MODULE_ID index);
```

Returns

- true - The ReadyMode feature is supported on the device
- false - The ReadyMode feature is not supported on the device

Description

This function identifies whether the ReadyMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_ReadyModeSet
- PLIB_EBI_ReadyModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin1Config Function

Identifies whether the ReadyPin1Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin1Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin1Config feature is supported on the device
- false - The ReadyPin1Config feature is not supported on the device

Description

This function identifies whether the ReadyPin1Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin1ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin1Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin2Config Function

Identifies whether the ReadyPin2Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin2Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin2Config feature is supported on the device
- false - The ReadyPin2Config feature is not supported on the device

Description

This function identifies whether the ReadyPin2Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin2ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin2Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin3Config Function

Identifies whether the ReadyPin3Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin3Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin3Config feature is supported on the device
- false - The ReadyPin3Config feature is not supported on the device

Description

This function identifies whether the ReadyPin3Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin3ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin3Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPinSens Function

Identifies whether the ReadyPinSens feature exists on the EBI module

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPinSens(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPinSens feature is supported on the device
- false - The ReadyPinSens feature is not supported on the device

Description

This function identifies whether the ReadyPinSens feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPinSensSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPinSens(EBI_MODULE_ID index)

PLIB_EBI_ExistsStaticMemoryWidthRegister Function

Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsStaticMemoryWidthRegister(EBI_MODULE_ID index);
```

Returns

- true - The StaticMemoryWidthRegister feature is supported on the device
- false - The StaticMemoryWidthRegister feature is not supported on the device

Description

This function identifies whether the StaticMemoryWidthRegister feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_StaticMemoryWidthRegisterSet
- PLIB_EBI_StaticMemoryWidthRegisterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsStaticMemoryWidthRegister(EBI_MODULE_ID index)

PLIB_EBI_ExistsWriteOutputControl Function

Identifies whether the WriteOutputControl feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsWriteOutputControl(EBI_MODULE_ID index);
```

Returns

- true - The WriteOutputControl feature is supported on the device
- false - The WriteOutputControl feature is not supported on the device

Description

This function identifies whether the WriteOutputControl feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_WriteOutputControlSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsWriteOutputControl(EBI_MODULE_ID index)

PLIB_EBI_ExistsWritePulseWidth Function

Identifies whether the WritePulseWidth feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsWritePulseWidth(EBI_MODULE_ID index);
```

Returns

- true - The WritePulseWidth feature is supported on the device
- false - The WritePulseWidth feature is not supported on the device

Description

This function identifies whether the WritePulseWidth feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_WritePulseWidthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsWritePulseWidth(EBI_MODULE_ID index)

d) Data Types and Constants

EBI_ADDRESS_PORT Enumeration

Selects the EBI address port.

File

plib_ebi_help.h

C

```
typedef enum {
  EBI_EBIADDR_PIN0,
  EBI_EBIADDR_PIN1,
  EBI_EBIADDR_PIN2,
  EBI_EBIADDR_PIN3,
  EBI_EBIADDR_PIN4,
  EBI_EBIADDR_PIN5,
  EBI_EBIADDR_PIN6,
  EBI_EBIADDR_PIN7,
  EBI_EBIADDR_PIN8,
  EBI_EBIADDR_PIN9,
  EBI_EBIADDR_PIN10,
  EBI_EBIADDR_PIN11,
  EBI_EBIADDR_PIN12,
  EBI_EBIADDR_PIN13,
  EBI_EBIADDR_PIN14,
  EBI_EBIADDR_PIN15,
  EBI_EBIADDR_PIN16,
  EBI_EBIADDR_PIN17,
  EBI_EBIADDR_PIN18,
  EBI_EBIADDR_PIN19,
  EBI_EBIADDR_PIN20,
  EBI_EBIADDR_PIN21,
  EBI_EBIADDR_PIN22,
  EBI_EBIADDR_PIN23,
  EBI_ADDR_PRESET8,
  EBI_ADDR_PRESET12,
  EBI_ADDR_PRESET16,
  EBI_ADDR_PRESET_ALL
} EBI_ADDRESS_PORT;
```

Members

Members	Description
EBI_EBIADDR_PIN0	PIN 0
EBI_EBIADDR_PIN1	PIN 1
EBI_EBIADDR_PIN2	PIN 2
EBI_EBIADDR_PIN3	PIN 3
EBI_EBIADDR_PIN4	PIN 4
EBI_EBIADDR_PIN5	PIN 5
EBI_EBIADDR_PIN6	PIN 6
EBI_EBIADDR_PIN7	PIN 7
EBI_EBIADDR_PIN8	PIN 8
EBI_EBIADDR_PIN9	PIN 9

EBI_EBIADDR_PIN10	PIN 10
EBI_EBIADDR_PIN11	PIN 11
EBI_EBIADDR_PIN12	PIN 12
EBI_EBIADDR_PIN13	PIN 13
EBI_EBIADDR_PIN14	PIN 14
EBI_EBIADDR_PIN15	PIN 15
EBI_EBIADDR_PIN16	PIN 16
EBI_EBIADDR_PIN17	PIN 17
EBI_EBIADDR_PIN18	PIN 18
EBI_EBIADDR_PIN19	PIN 19
EBI_EBIADDR_PIN20	PIN 20
EBI_EBIADDR_PIN21	PIN 21
EBI_EBIADDR_PIN22	PIN 22
EBI_EBIADDR_PIN23	PIN 23
EBI_ADDR_PRESET8	Preset 8
EBI_ADDR_PRESET12	Preset 12
EBI_ADDR_PRESET16	Preset 16
EBI_ADDR_PRESET_ALL	Preset All

Description

EBI Address Port

This enumeration selects the EBI address port.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_CS_TIMING Enumeration

Selects the timing register set for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  CS_TIMING_0,
  CS_TIMING_1,
  CS_TIMING_2
} EBI_CS_TIMING;
```

Members

Members	Description
CS_TIMING_0	Use EBISMT0
CS_TIMING_1	Use EBISMT1
CS_TIMING_2	Use EBISMT2

Description

Timing Register for Chip Select

This enumeration selects the timing register set for Chip Select

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MEMORY_SIZE Enumeration

Selects the memory size for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  CSNOTUSED,
  MEMORY_SIZE_64KB,
  MEMORY_SIZE_128KB,
  MEMORY_SIZE_256KB,
  MEMORY_SIZE_512KB,
  MEMORY_SIZE_1MB,
  MEMORY_SIZE_2MB,
  MEMORY_SIZE_4MB,
  MEMORY_SIZE_8MB,
  MEMORY_SIZE_16MB
} EBI_MEMORY_SIZE;
```

Members

Members	Description
CSNOTUSED	Chip Select Not Used
MEMORY_SIZE_64KB	64 KB (smaller memories alias within this range)
MEMORY_SIZE_128KB	128 KB
MEMORY_SIZE_256KB	256 KB
MEMORY_SIZE_512KB	512 KB
MEMORY_SIZE_1MB	1 MB
MEMORY_SIZE_2MB	2 MB
MEMORY_SIZE_4MB	4 MB
MEMORY_SIZE_8MB	8 MB
MEMORY_SIZE_16MB	16 MB

Description

Memory Size for Chip Select

This enumeration selects memory size for Chip Select.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MEMORY_TYPE Enumeration

Selects the memory type for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  SRAM,
  NORFLASH
} EBI_MEMORY_TYPE;
```

Members

Members	Description
SRAM	SRAM
NORFLASH	NOR-Flash

Description

Memory Type for Chip Select

This enumeration selects the memory type for Chip Select.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MODULE_ID Enumeration

Identifies the supported EBI modules.

File

plib_ebi_help.h

C

```
typedef enum {
  EBI_ID_0,
  EBI_NUMBER_OF_MODULES
} EBI_MODULE_ID;
```

Members

Members	Description
EBI_ID_0	EBI Module 0 ID
EBI_NUMBER_OF_MODULES	Number of available WDT modules.

Description

EBI Module ID

This enumeration identifies the available EBI modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_PAGE_SIZE Enumeration

Selects the page size for the page mode device.

File

plib_ebi_help.h

C

```
typedef enum {
  PAGE_WORD4,
  PAGE_WORD8,
  PAGE_WORD16,
  PAGE_WORD32
} EBI_PAGE_SIZE;
```

Members

Members	Description
PAGE_WORD4	4-word page
PAGE_WORD8	8-word page
PAGE_WORD16	16-word page
PAGE_WORD32	32-word page

Description

Page Size for page mode device

This enumeration selects the page size for the page mode device.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_STATIC_MEMORY_WIDTH Enumeration

Selects the static memory width for the register EBISMTx.

File

plib_ebi_help.h

C

```
typedef enum {
  MEMORY_WIDTH_8BIT,
  MEMORY_WIDTH_16BIT
} EBI_STATIC_MEMORY_WIDTH;
```

Members

Members	Description
MEMORY_WIDTH_8BIT	8 bits
MEMORY_WIDTH_16BIT	16 bits

Description

Memory Width

This enumeration selects the static memory width for the register EBISMTx.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_ebi.h	EBI Peripheral Library Interface Header.
plib_ebi_help.h	

Description

This section lists the source and header files used by the library.

plib_ebi.h

EBI Peripheral Library Interface Header.

Functions

	Name	Description
	PLIB_EBI_AddressHoldTimeGet	Returns the address hold time.
	PLIB_EBI_AddressPinEnableBitsSet	Sets the address pins used for EBI.
	PLIB_EBI_AddressSetupTimeGet	Returns the setup hold time.
	PLIB_EBI_BaseAddressGet	Returns the base address set for each Chip Select.
	PLIB_EBI_BaseAddressSet	Sets the base address for physical memory at each Chip Select pin.
	PLIB_EBI_ByteSelectPinSet	Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.
	PLIB_EBI_ChipSelectEnableSet	Sets the Chip Select pins for use with the EBI or GPIO.
	PLIB_EBI_ControlEnableGet	Returns the status of the EBI enable bit.
	PLIB_EBI_ControlEnableSet	Sets the EBI bus ON/OFF enable bit.
	PLIB_EBI_DataEnableSet	Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.
	PLIB_EBI_DataTurnAroundTimeGet	Returns the data turn-around time set for the EBI bus.
	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module.
	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module.
	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module.
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module.
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module.
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module.
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module.
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module.
	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module.

	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module.
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module.
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module.
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module.
	PLIB_EBI_FlashPowerDownModeGet	Returns the set power-down state.
	PLIB_EBI_FlashPowerDownModeSet	Sets the pin state for Flash devices on Power-down and Reset.
	PLIB_EBI_FlashResetPinGet	Sets the control use of Flash Reset pin.
	PLIB_EBI_FlashResetPinSet	Sets the control use of the Flash Reset pin.
	PLIB_EBI_FlashTimingGet	Returns the set Flash timing for external Flash.
	PLIB_EBI_FlashTimingSet	Sets the timing for hold in reset for external Flash.
	PLIB_EBI_MemoryCharacteristicsSet	Sets the characteristics for memory or attached devices attached to the specified pin.
	PLIB_EBI_MemoryPageSizeGet	Returns the Paging mode settings.
	PLIB_EBI_MemoryPagingSet	Sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	Sets the cycle time for page reading.
	PLIB_EBI_PageModeGet	Returns the Paging mode settings.
	PLIB_EBI_PageReadCycleTimeGet	Returns the cycle time for a page read.
	PLIB_EBI_ReadCycleTimeGet	Returns the read time in the number of clock cycles.
	PLIB_EBI_ReadyModeGet	Returns whether or not Ready mode was set.
	PLIB_EBI_ReadyModeSet	Sets the use of Ready mode for each pin.
	PLIB_EBI_ReadyPin1ConfigSet	Sets the control use of ReadyPin1 and inverts the status.
	PLIB_EBI_ReadyPin2ConfigSet	Sets the control use of ReadyPin2 and inverts the status.
	PLIB_EBI_ReadyPin3ConfigSet	Sets the control use of ReadyPin3 and inverts the status.
	PLIB_EBI_ReadyPinSensSet	Sets the sensitivity of the Ready pin.
	PLIB_EBI_StaticMemoryWidthRegisterGet	Returns the set width of the data bus.
	PLIB_EBI_StaticMemoryWidthRegisterSet	Sets the data width of static memory.
	PLIB_EBI_WriteOutputControlSet	Sets the Write Enable and Output Enable control pins.
	PLIB_EBI_WritePulseWidthGet	Returns the set hold time in clock cycles.

Description

External Bus Interface (EBI) Peripheral Library Interface

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the EBI Peripheral Library.

File Name

plib_ebi.h

Company

Microchip Technology Inc.

plib_ebi_help.h

Enumerations

	Name	Description
	EBI_ADDRESS_PORT	Selects the EBI address port.
	EBI_CS_TIMING	Selects the timing register set for Chip Select.

	EBI_MEMORY_SIZE	Selects the memory size for Chip Select.
	EBI_MEMORY_TYPE	Selects the memory type for Chip Select.
	EBI_MODULE_ID	Identifies the supported EBI modules.
	EBI_PAGE_SIZE	Selects the page size for the page mode device.
	EBI_STATIC_MEMORY_WIDTH	Selects the static memory width for the register EBISMTx.

Section

Data Types & Constants

Ethernet Peripheral Library

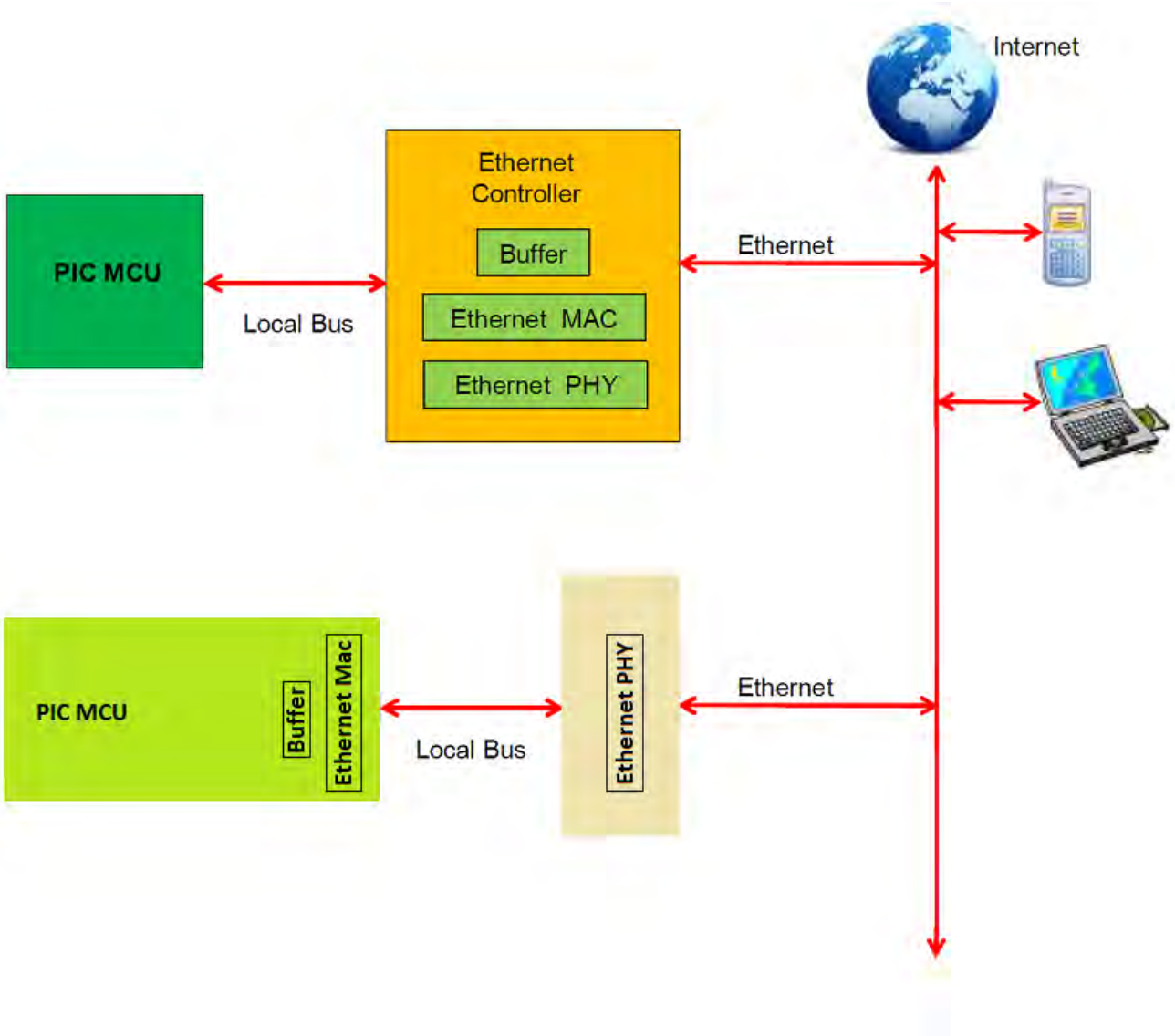
This section describes the Ethernet Peripheral Library.

Introduction

The Ethernet Peripheral Library provides a low-level abstraction of the Ethernet module in the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without interacting directly with the module's registers, thereby hiding the differences in microcontroller variations.

Description

Ethernet is the most widely deployed network in offices and homes. Ethernet's infrastructure, interoperability, and scalability serve to ease development and facilitate communications to the device. Once equipment is connected to an Ethernet network, it can be monitored or controlled through the internet with low latency – "real time" remote delivery.



The Ethernet Controller is a bus master module that interfaces with an off-chip PHY to implement a complete Ethernet node in a system.

Following are some of the key features of this module:

- Supports 10/100 Mbps data transfer rates

- Supports Full-Duplex and Half-Duplex operation
- Supports Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY interface
- Supports MII Management (MIIM) PHY Management interface
- Supports both manual and automatic flow control
- Supports Auto-MDIX and enabled PHYs
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering:
  - CRC Check
  - 64-byte Pattern Match
  - Broadcast, Multicast and Unicast packets
  - Magic Packet™
  - 64-bit Hash Table
  - Runt Packet
- Supports Packet Payload Checksum calculation
- Supports various hardware statistics counters

Using the Library

This topic describes the basic architecture of the Ethernet Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_eth.h`

The interface to the Ethernet Peripheral Library is defined in the `plib_eth.h` header file., which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Ethernet Peripheral Library should include `peripheral.h`.

Library File:

The Ethernet Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

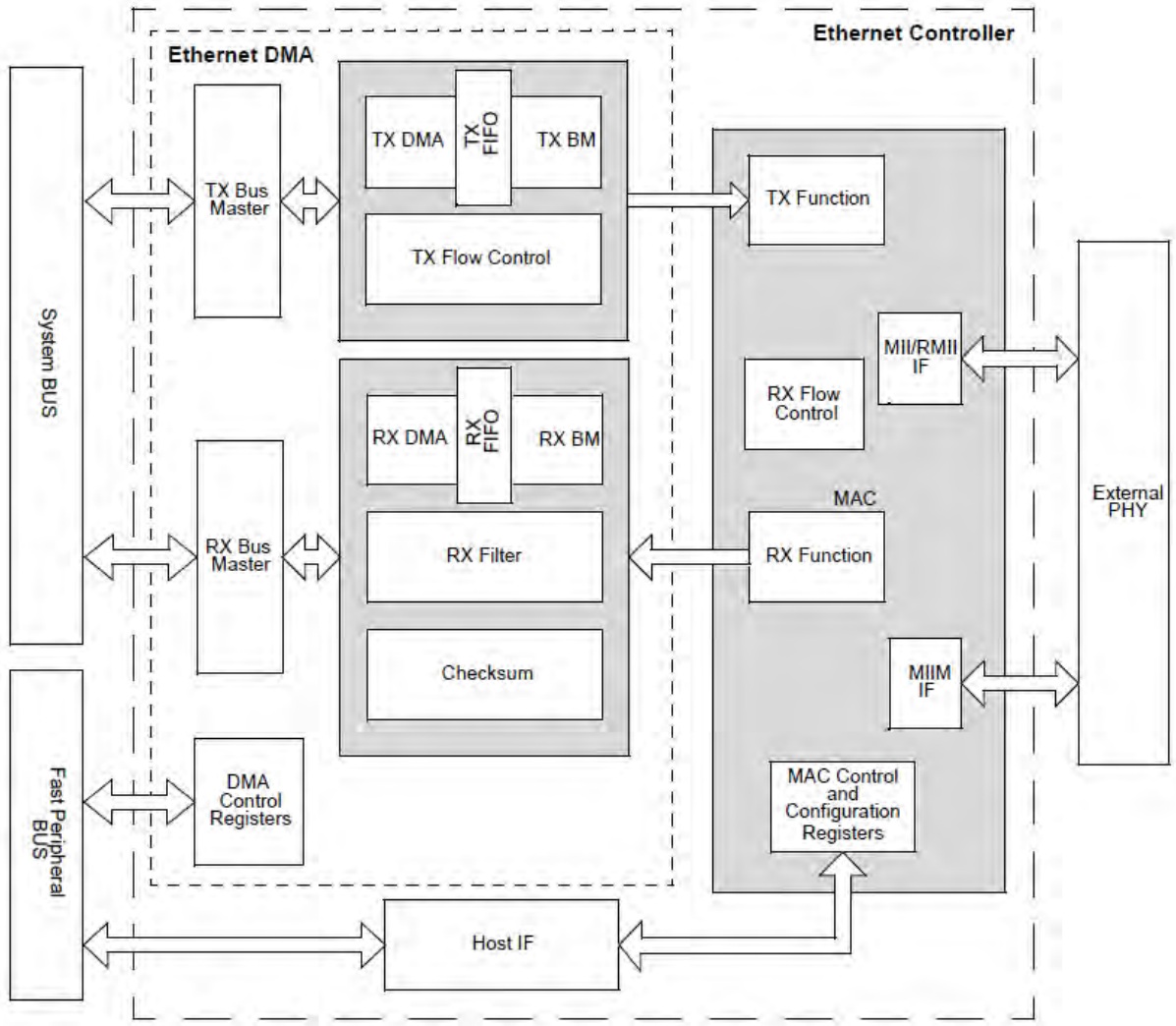
This section describes how the low-level abstraction is modeled in the software and introduces the library interface. The interface provided is a superset of all the functionality of the available Ethernet on the device. Refer to the specific data sheet or family reference manual to determine the set of functions that are supported for each Ethernet module on your device.

Description

Hardware Model

This library provides the low-level abstraction of the Ethernet module on Microchip family of microcontrollers with a convenient C language interface.

Ethernet Controller Block Diagram



Note: For information on off-chip options to provide Ethernet functionality, refer to the specific device data sheet.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet module.

Library Interface Section	Description
Initialization Functions	Provides setup and configuration routines.
Control Functions	Provides control routines.
Status Functions	Provides status routines.
Filtering Functions	Provides filtering routines.
Flow Control Functions	Provides flow control routines.
Interrupt Functions	Provides interrupt routines.
Statistics Functions	Provides statistics routines.
Feature Existence Functions	These functions can be used to determine whether or not a particular feature exists on the device.

How the Library Works

This section provides information on the operation of the Ethernet Peripheral Library.

Description

The Peripheral Library is an abstraction of hardware which can be used to control hardware and obtain status. It is intended to allow the programmer access to hardware registers without knowing the exact bit or register names which may vary on individual devices. For example, instead of turning on the Ethernet module using:

ETHCON1bits.EON = 1; the programmer can turn on the Ethernet module using: PLIB_ETH_Enable(moduleId);. This allows the programmer to use the same command for any device containing an Ethernet module without having to learn the individual device register names.

Basic Overview

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external PHY chip. To off-load the CPU from moving packet data to and from the module, internal descriptor-based DMA engines are included in the controller.

The Ethernet Controller consists of the following modules:

- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block, which provides:
  - Manual flow control
  - Automatic flow control
  - Transmission of PAUSE frames
  - Reception of PAUSE frame is handled within the MAC
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected and includes:
  - CRC Check
  - 64-byte Pattern Match
  - Broadcast, Multicast and Unicast packets
  - Magic Packet™
  - 64-bit Hash Table
  - Runt Packet
- TX DMA/TX BM Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX BM Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Basic Ethernet Controller Operation

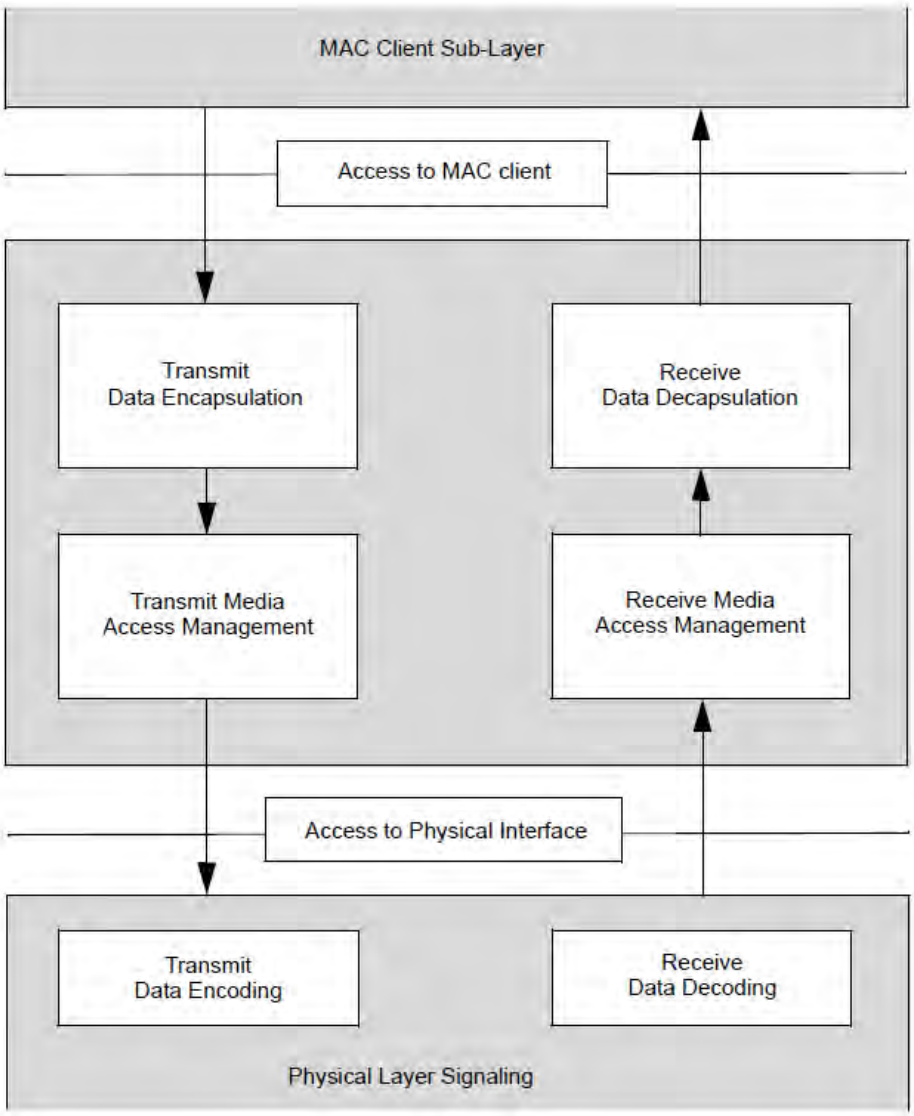
The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORTx latch bit and TRISx bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The Special Function Registers (SFRs) are still accessible and their values are preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

Ethernet Controller Section Block Diagram



MAC Overview

The MAC sub-layer is part of the functionality described in the OSI model for the Data Link Layer. It defines a medium-independent facility, built on the medium-dependent physical facility provided by the Physical Layer, and under the access-layer-independent LLC sub-layer or other MAC client. It is applicable to a general class of local area broadcast media suitable for use with the Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

The CSMA/CD MAC sub-layer provides services to the MAC client required for the transmission and reception of frames. The CSMA/CD MAC sub-layer makes a best effort to acquire the medium and transfer a serial stream of bits to the Physical Layer. Although certain errors are reported to the client, error recovery is not provided by the MAC.

The following is a summary of the functional capabilities of the CSMA/CD MAC sub-layer as shown in the previous diagram.

- For Frame Transmission:
  - Accepts data from the MAC client and constructs a frame
  - Presents a bit-serial data stream to the Physical Layer for transmission on the medium
  - In half-duplex mode, defers transmission of a bit-serial stream whenever the physical medium is busy
  - It can append proper Frame Check Sequence (FCS) value to outgoing frames and verifies full octet boundary alignment
  - Delays transmission of frame bit stream for specified inter-frame gap period
  - In half-duplex mode, halts transmission when collision is detected
  - In half-duplex mode, schedules retransmission after a collision until a specified retry limit is reached
  - In half-duplex mode, enforces collision to ensure propagation throughout network by sending jam message
  - Prepends preamble and Start Frame Delimiter and appends FCS to all frame Appends PAD field for frames whose data length is less than a minimum value
- For Frame Reception:

- Receives a bit-serial data stream from the Physical Layer
- Presents the received frames to the MAC client (broadcast, multicast, unicast frames, etc.)
- Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment
- Removes preamble, Start Frame Delimiter and PAD field (if necessary) from received frames
- Implements the MII Management block that provides control/status connection to the external MII PHY

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the set of modes supported for your device. In addition, refer to **Section 35. "Ethernet Controller"** (DS60001155) of the *"PIC32 Family Reference Manual"* or the Ethernet MAC Driver Library for more information.

Ethernet Frame Overview

Ethernet Frame Overview

IEEE 802.3-compliant Ethernet frames (packets) are between 64 and 1518 bytes long (Preamble and Start -of-Frame Delimiter not included). Frames containing less than 64 bytes are known as "runt" frames, while frames containing more than 1518 bytes are known as "huge" frames.

An Ethernet frame is made up of the following fields:

- Start-of-Stream/Preamble
- Start-of-Frame Delimiter (SFD)
- Destination MAC address (DA)
- Source MAC address (SA)
- Type/Length field
- Data Payload
- Optional Padding field
- Frame Check Sequence (FCS)
- Traffic on the actual physical cable

Please refer to the IEEE 802.3 Specification for detailed information about the Ethernet protocol.

Ethernet Controller Operation

Basic Ethernet Controller Operation

The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORT latch bit and TRIS bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The SFRs are still accessible and their values preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

Media Independent Interface (MII)

This section describes the Media Independent Interface (MII).

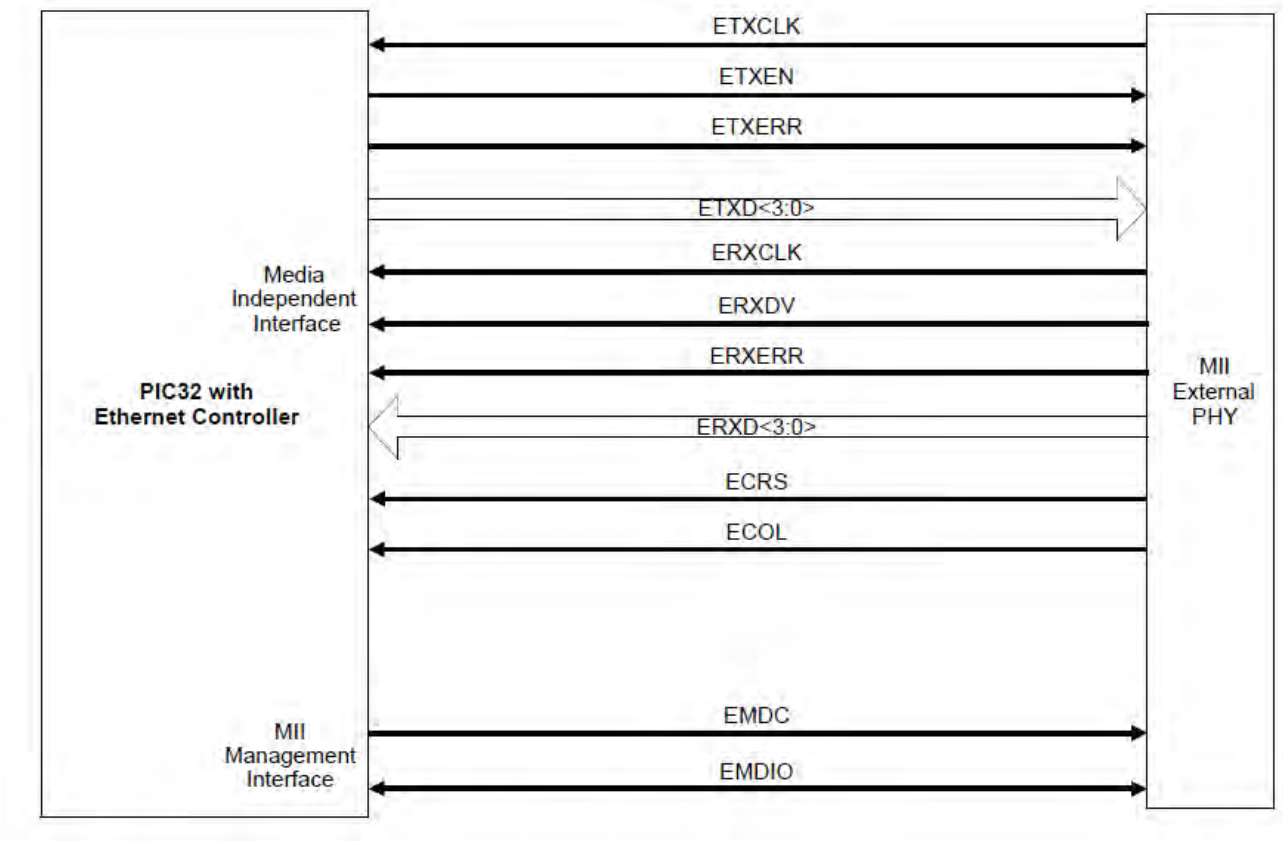
Description

Media Independent Interface (MII)

The Media Independent Interface (MII) is a standard interconnection between the MAC and the PHY for communicating TX and RX frame data. This MII has the following important characteristics:

- Capable of supporting 10/100 Mbps rates for data transfer, and offers support for management functions
- Provides independent four bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation

In 10 Mbps mode, the MII runs at 2.5 MHz; in 100 Mbps mode it runs at 25 MHz. PHYs that provide MII are not required to support both data rates, and may support either one or both.



Reduced Media Independent Interface (RMII)

This section describes the Reduced Media Independent Interface (RMII).

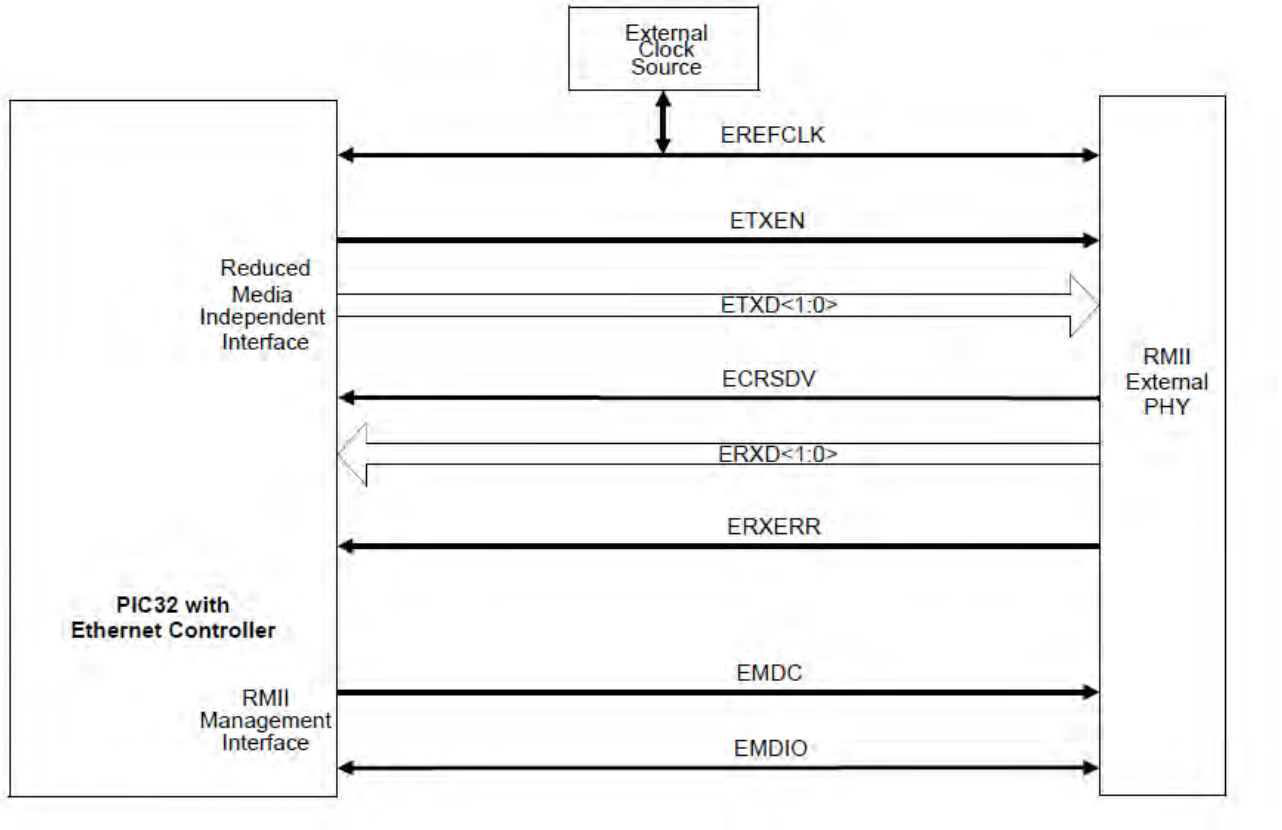
Description

Reduced Media Independent Interface (RMII)

The management interface (MDIO/MDC) is assumed to be identical to that defined in MII.

The RMII has the following characteristics:

- Capable of supporting 10 Mbps and 100 Mbps data rates
- Single clock reference for both MAC and the PHY (can be sourced from the MAC or from an external source)
- Provides independent 2 bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation
- The interface runs at 50 MHz



Flow Control Overview

Flow Control Overview

Ethernet flow control consists of the ability to send and receive PAUSE frames, which cause the receiving node to stop transmitting for a specific amount of time.

On the transmit side, the Flow Control (FC) block handles the hardware handshaking between the MAC and the CPU when the transmit flow control is enabled. Flow control for the received packets is part of the MAC functionality.

The PIC32 MAC supports both Symmetric PAUSE and Asymmetric PAUSE as described in Clause 28, Table 28B-2, and Clause 31 and Annex 31B of the IEEE 802.3 Specification.

The FC block supports two modes of operation: manual and automatic. In addition, the mode of transmission (Full-Duplex or Half-Duplex) programmed into the MAC registers, is used by the FC block.

Receive Filtering Overview

Receive Filtering Overview

The Receive Filter (RXF) block examines all incoming receive packets and accepts or rejects the packet, based on user-selectable filters. The following RX filters are supported:

- CRC Error Acceptance Filter
- Runt Error Acceptance Filter
- CRC Check Rejection Filter
- Runt Rejection Filter
- Unicast Acceptance Filter
- Not Me Unicast Acceptance Filter
- Multicast Acceptance Filter
- Broadcast Acceptance Filter
- Hash Table Acceptance Filter
- Magic Packet Acceptance Filter
- Pattern Match Acceptance Filter with logical inversion

Ethernet DMA and Buffer Management Engine

Ethernet DMA and Buffer Management Engines

To reduce the overhead on the CPU to move the packet data between data memory and the Ethernet controller, internal receive (RX) and transmit (TX) DMA engines are integrated into the module. The DMA engines are responsible for transferring data from system memory to the MAC for transmit packets and for transferring data from the MAC to system memory for receive packets. The DMA engines each have access to the system memory by acting as two different bus masters, one bus master for transmit and one for receive.

The DMA engines use separate Ethernet Descriptor Tables (EDTs) for TX and RX operations to determine where the TX/RX packet buffer resides in the system memory.

Both transmit and receive descriptors, called Ethernet Descriptors (EDs), used by the DMA engines have a similar format, with only the status word formats being different.

Refer to the **"Ethernet"** section in the family reference manual for more information.

Sample Code

This section provides information for initialization, transmit, and receive using code samples.

Description

Sample Code

The basic functionality of the Ethernet Peripheral Library is grouped into these sections:

- Initialization
- Transmit
- Receive

For a properly functioning Ethernet port, development of a data queue and use of interrupts will be necessary.

For this additional functionality, see the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

Initialization Sequence

The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset.

After being Enabled, disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. All registers remain accessible and their values are preserved. It is important to reset the PHY and MII Management registers to ensure the PHY is in a known initialized state.

Before enabling the module for transmitting or receiving packets, there are several registers that must be initialized. To initialize the Ethernet Controller to receive and transmit, Microchip recommends the following sequence:

1. Begin with the module.
- Disable the CPU Ethernet interrupt
- Turn the Ethernet Controller off
- Disable receive
- Disable transmit
- Wait until the Ethernet module is no longer busy
- Disable all Ethernet Interrupts
- Clear the Ethernet Interrupt Pending Flag(s)
- Clear the transmit and receive start addresses

2. MAC initialize.
- Reset the MAC
- A configuration fuse (FETHIO) setting shows which pins will be used for connection to the PHY
- A configuration fuse (FMIIEN) settings shows whether MII or RMII will be used

3. PHY Initialize is PHY dependent. Common procedures (which may vary by PHY vender) include the following:
- Reset PHY (using Control Register 0)
- Set the MII/RMII operation mode
- Set the normal, swapped, or auto (preferred)MDIX
- Check the PY capabilities (using Status Register 1)
- Preferably the auto-negotiation should be selected if the PHY supports it.
  - Expose the supported capabilities (extended Register 4)
  - Start the negotiation (using Control Register 0)
  - wait for the negotiation to complete and get the link partner capabilities (extended Register 5)

- negotiation result (vendor-specific register)

- If auto-negotiation is not supported (or not selected)
  - update the PHY Duplex and speed settings (use Control Register 0 or vender-specific register)

4. MAC Configuration

- Enable MAC receive and pause both transmit and receive direction control frames.
- Select the desired auto-padding, duplex, huge frame, and CRC capabilities.
- Program back-to-back inter-packet gap and non-back-to-back interpacket gap.
- Program collision window and maximum retransmissions.
- Set Maximum frame length.
- Optionally set the station MAC address(es) (or let them default to the factory value).

5. Continue Ethernet Controller initialization:

- If planning to turn on flow control, update the Pause Timer Value
- If using auto-flow control, set up the full/empty watermarks and enable it
- Set the receive filters
- Set receive buffer size - using packets that are too small impacts performance
- Prepare the list/ring of transmit descriptors - refer to the device data sheet for more information
- Prepare the list of receive descriptors populated with valid buffers for messages to be received - refer to the device data sheet for more information
- Update the transmit buffer start address and the receive buffer start address
- Enable the Ethernet Controller and wait until the module is ready
- Enable Ethernet Receive.

Receive Loop

1. Wait until the receive buffer count has been incremented.
2. Inspect the receive descriptor to see whether it is still owned by the Ethernet Module.
3. If the receive descriptor is still owned by the Ethernet module, nothing has been received. Loop to 1.
4. Otherwise, a message was received and software has control of the memory where it is stored.
5. Extract a message and do something with it.
6. Reinitialize buffer descriptor and return receive buffer to list and decrement the receive packet buffer count.
7. Loop.

Transmit Loop

1. Software writes the packet data to packet data memory. A pointer to the packet data is inserted into the transmit buffer descriptor.
2. Update the necessary number of transmit descriptors, starting from the head of the list - setting the data buffer address (large packet fragmentation will impact performance).
3. Update the byte count for each descriptor with the number of bytes contained in each buffer.
4. Set the flag in each transmit descriptor that belongs to the packet to give the descriptor to the Ethernet module for transmission.
5. Enable the transmission.
6. Wait until the Ethernet module returns the transmit buffer descriptor ownership to software.
7. Insect the results of the transmission.
8. Loop.

The packet transmit process is as follows:

1. Software writes the packet data to a packet buffer in data memory and programs a buffer descriptor entry to point to the packet data buffer. The buffer descriptor is used to define the Transmission Packet data buffer location and length, and its address is written to the Ethernet Controller.
2. Software then calls the TransmitRequestToSendEnable function, which starts the TX DMA and TXBM logic.
3. After the TX DMA engine reads the DATA_BUFFER_ADDRESS value, the engine will begin reading the packet data from the location read from the descriptor.
4. The TXBM indicates the start-of-frame to the MAC and transmits the entire frame data from the transmit buffer, until the end address is reached. The TXBM simply transmits from the start address until the specified number of bytes has been transmitted to the MAC transmit interface.
5. The MAC can retry a transmission due to an early collision.
6. The MAC can abort a transmission due to a late collision, excessive collisions or excessive defers. This condition is signaled by TXABORT bit (ETHIRQ<2>).
7. Once transmission has completed, the TX DMA engine stores the relevant bits of the TSV into the first descriptor entry of the packet.
8. After the packet has been transmitted, all the descriptors used for the transmission are released to the software via the EOWN bits being cleared.
9. If more valid TX descriptors are available (EOWN = 1), the DMA engine will go back to step 3 to begin the next packet's transmission. Otherwise, if the next descriptor is still owned by hardware (EOWN = 0), transmission will halt and wait for the software to set the TXRTS bit again.

Note that any collision that occurs within the CWINDOW bit (EMAC1CLRT<8:14>) boundary is an early collision and results in a Retry operation. A

collision that happens beyond the CWINDOW boundary will be treated as a late collision and will cause an abort. The CWINDOW bit in the EMAC1CLRT register is typically set to 64 bytes. An abort condition can also result from reaching the maximum collision count RETX bit (EMAC1CLRT<0:3>) for a packet trying to be sent.

See the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

Initialization

```
#include "plib_eth.h"

/*****************************************************************************
*/
void my_ETH_Initialize(ETH_MODULE_ID id)
{

    //Disable System (CPU) Interrupt associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example: PLIB_INT_SourceDisable(PLIB_INT_SOURCE_ETH_1);

    //Disable Ethernet Peripheral in case it was previously enabled
    PLIB_ETH_Disable(ETH_MODULE_ID1);

    //Disable any Ethernet Controller interrupt generation:
    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    PLIB_ETH_RxDisable(ETH_MODULE_ID1);
    PLIB_ETH_TxRTSDisable(ETH_MODULE_ID1);

    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1)) {/*wait*/ }

    //Clear the System Interrupt Flag associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example:     PLIB_INT_SourceFlagClear(PLIB_INT_SOURCE_ETH_1);

    ////////////////////////////////////////////////
    //MAC Initialize
    PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);

    //A configuration fuse (FETHIO) setting shows which digital pins need to be configured
    if (CONFIGURATION_FUSE_PRIMARY_ETH_PINS==1)
    {
        ConfigurePrimaryEthernetPinsAsDigital();
    }
    else //(CONFIGURATION_FUSE_PRIMARY_ETH_PINS==0)
    {
        ConfigureAlternateEthernetPinsAsDigital();
    }

    //A configuration fuse (FMIIEN) settings shows which digital pins need to be configured
    if (CONFIGURATION_FUSE_MII_STYLE==RMII)
    {
        PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID1);
        PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID1);
        PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID1, ETH_RMII_100Mps);
    }
    PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);
    PLIB_ETH_MIIMClockSet(ETH_MODULE_ID1, ETH_MII_SYSCLK_DIV_BY_10);


    ////////////////////////////////////////////////
    //PHY Initialize is PHY dependent.
    // Common procedures (which may vary by PHY vender) include the following:
    //
    //Reset PHY (using Control Register 0)
    //Set the MII/RMII operation mode
    //Set the normal, swapped, or auto (preferred)MDIX
    //Check the PY capabilities (using Status Register 1)
```

```
    //Preferably the auto-negotiation should be selected if the PHY supports it.
    //Expose the supported capabilities (extended Register 4)
    //Start the negotiation (using Control Register 0)
    //wait for the negotiation to complete and get the link partner capabilities (extended Register 5)
    //negotiation result (vendor-specific register)
    //If auto-negotiation is not supported (or not selected)
    //update the PHY Duplex and speed settings (use Control Register 0 or vender-specific register)

    ////////////////////////////////////////////////
    //MAC Configuration
    PLIB_ETH_RxEnable(ETH_MODULE_ID1);
    PLIB_ETH_TxPauseEnable(ETH_MODULE_ID1);
    PLIB_ETH_RxPauseEnable(ETH_MODULE_ID1);

    //Select the desired auto-padding, duplex, huge frame, and CRC capabilities
    PLIB_ETH_AutoDetecPadSet(ETH_MODULE_ID1, ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC );
    PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID1); //or disable for half-duplex
    PLIB_ETH_CRCEnable(ETH_MODULE_ID1); //or disable if EMAC does not add CRC when no padding is necessary
    PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID1); //or enable to receive huge frames

    //Program back-to-back inter-packet gap and non-back-to-back interpacket gap
    PLIB_ETH_BackToBackIPGpSet(ETH_MODULE_ID1, 0x15 /*backToBackGapValue*/);
    PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID1, 0x0C /*nonBackToBackGap1Value*/);
    PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID1, 0x12 /*nonBackToBackGap2Value*/);

    //Set Maximum frame length
    PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID1, 1527 /*maxFrameLenValue*/);

    //Optionally set the station MAC address(es) (or let them default to the factory value)
    PLIB_ETH_StationAddressSet(ETH_MODULE_ID1, 1, 0x13 /*stationAddr1Value*/);


    ////////////////////////////////////////////////
    //Continue Ethernet Controller Initialization

    //Set receive buffer size - using packets that are too small impacts performance
    PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID1, 200 /*receiveBufSizeValue*/);

    //Prepare the list/ring of transmit descriptors - refer to the device data sheet for more information
    //Prepare the list of receive descriptors populated with valid buffers for messages to be received -
refer
    //to the device data sheet for more information
    //Update the transmit buffer start address and the receive buffer start address
    PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t *)0xadd1add0/*txPktDescStartAddrValue*/);
    PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t *)0xadd3add2/*rxPktDescStartAddrValue*/);

    //If using auto flow control, set up the full/empty watermarks and enable it
    PLIB_ETH_PauseTimerSet(ETH_MODULE_ID1, 0x64 /*pauseTimerValue*/);
    PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID1, 0x3F/*emptyRxWatermarkValue*/);
    PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID1, 0x10 /*fullRxWatermarkValue*/);
    PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID1);

    PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID1,
                                 ETH_MULTICAST_FILT |   // enable to accept multicast address packets
                                 ETH_BROADCAST_FILTER );// enable to accept all broadcast address packets

    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    //Enable the Ethernet Controller and wait until the module is ready
    PLIB_ETH_Enable(ETH_MODULE_ID1);
    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1))
    {
        //Wait
    }

    PLIB_ETH_RxEnable(ETH_MODULE_ID1);

    return (EXIT_SUCCESS);
}
```

Descriptor Table Example

```
typedef volatile struct
{
    union
    { //Common to both Receive and Transmit
        struct
        {
            unsigned: 7;
            unsigned EOWN: 1; //0=Software owns, 1=Ethernet Controller owns
            unsigned NPV: 1;  //0=Next Desc follows this, 1=Next Desc pointed to by NEXT_ED
            unsigned: 7;
            unsigned bCount: 11;
            unsigned: 3;
            unsigned EOP: 1;
            unsigned SOP: 1;
        };
        uint32_t w;
    };                      // descriptor header

    //Common to both Receive and Transmit
    uint8_t* pEDBuff;     // pointer to data buffer

    union
    { //This portion differs from Receive to Transmit
        uint64_t RX_STATUS;    //Receive Status
        struct
        {
            unsigned RX_PKT_CHECKSUM: 16;
            unsigned: 8;
            unsigned RXF_RSV: 8;
            unsigned RSV: 32;
        };

        uint64_t TX_STATUS;    // Transmit Status
        struct //Transmit Struct
        {
            unsigned TSV_TRANSMIT_BYTE_COUNT: 16;
            unsigned TSV_TX_COLLISION_COUNT: 4;
            unsigned TSV_TX_CRC_ERROR: 1;
            unsigned TSV_TX_LENGTH_CHECK_ERROR: 1;
            unsigned TSV_TX_DONE: 1;
            unsigned TSV_TX_MULTICAST: 1;
            unsigned TSV_TX_BROADCAST: 1;
            unsigned TSV_TX_PACKET_DEFER: 1;
            unsigned TSV_TX_EXCESSIVE_DEFER: 1;
            unsigned TSV_TX_MAXIMUM_COLLISION: 1;
            unsigned TSV_TX_LATE_COLLISION: 1;
            unsigned TSV_TX_GIANT: 1;
            unsigned TSV_TX_UNDER_RUN: 1;
            unsigned TSV_TX_STATUS_VECTOR:1;

            unsigned TSV_TOTAL_BYTES_TX_ON_WIRE: 16;
            unsigned TSV_TX_CONTROL_FRAME: 1;
            unsigned TSV_TX_PAUSE_CONTROL_FRAME: 1;
            unsigned TSV_TX_BACKPRESSURE_APPLIED: 1;
            unsigned TSV_TX_VLAN_TAGGED_FRAME: 1;
            unsigned TSV_IGNORED: 4;
            unsigned TSV_USER_DEFINED: 8;
        };
    };                  // descriptor header

    //Common to both Receive and Transmit
    uint8_t* next_ed;       // next descriptor (NPV==1);

}__attribute__((__packed__)) eth_buffer_descriptor_t; // hardware RX descriptor (linked).
```

For more information on this structure, refer to the **"Ethernet"** section in the family reference manual.

Transmit

C

Description

```
#include "plib_eth.h"

/*****************************************************************
//
*/

void my_ETH_Transmit(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
                     uint8_t nArrayItems, uint16_t *pArrSize)
{
    //Input Parameters:
    //   *pArrDcpt : pointer to an array that holds free descriptors
    //   *pArrBuff: pointer to an array that holds buffers to be transmitted
    //   *pArrSize: pointer to an array that holds the sizes of the buffers to be transmitted
    //   nArrayItems: number of buffers to be transmitted
    //
    // Before attempting to transmit, the data must be placed into a packet
    // buffer (list) and the point(s) must be placed into transmit descriptor(s)
    // Refer to the DRV_ETH or the device family reference manual for more information

    extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
                                        // address of the input virtual address parameter

    uint16_t ix;                        // loop index
    eth_buffer_descriptor_t *pEDcpt;    // current Ethernet descriptor
    eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
    uint8_t *pBuff;                     // current data buffer to be transmitted
    uint16_t *pSize;                    // current data buffer size

    pEDcpt=pArrDcpt;
    pBuff=pArrBuff;
    pSize=pArrSize;
    tailDcpt=0;

    for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++, pSize++)
    {
        // pass the descriptor to hw, use linked descriptors, set proper size
        pEDcpt->pEDBuff = (uint8_t *)VA_TO_PA(pBuff); // set buffer
        pEDcpt->w=0;                                  // clear all the fields
        pEDcpt->NPV=1;                                // set next pointer valid
        pEDcpt->EOWN=1;                               // set hw ownership
        pEDcpt->bCount = *pSize;                      // set proper size
        if(tailDcpt)
        {
            tailDcpt->next_ed = VA_TO_PA(&pEDcpt);
        }
        tailDcpt=pEDcpt;
    }
    // at this moment pEDcpt is an extra descriptor we use to end the descriptors list
    pEDcpt->w=0;                                                  // software ownership
    tailDcpt->next_ed= VA_TO_PA(&pEDcpt);
    pArrDcpt[0].SOP=1;                                            // start of packet
    pArrDcpt[nArrayItems-1].EOP=1;                                // end of packet

    PLIB_ETH_TransmitPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt)); // set the transmit address
    PLIB_ETH_TransmitRequestToSendIsEnabled(id);                     // set the TXRTS

    // the ETHC will transmit the buffers we just programmed
    /* do something else in between */
    while (PLIB_ETH_TxRTSIsEnabled(id))
    {
       //Wait until transmit is not busy
```

```
    }

    // check the ETHSTAT register to see the transfer result
 }
```

Receive

```
#include "plib_eth.h"

/*****************************************************************
//
*/
void my_ETH_Receive(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
                    uint8_t nArrayItems, uint16_t rxBuffSize)
{
   extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
                                       // address of the input virtual address parameter
    int ix;// index
    eth_buffer_descriptor_t *pEDcpt;   // current Ethernet descriptor
    eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
    char* pBuff;// current data buffer to be transmitted

    pEDcpt=pArrDcpt;
    pBuff=pArrBuff;
    tailDcpt=0;

    //Prepare to receive
    PLIB_ETH_ReceiveBufferSizeSet(id, (rxBuffSize/16));

    for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++)
    {// pass the descriptor to hw, use linked descriptors, set proper size
        pEDcpt->pEDBuff=(unsigned char*)VA_TO_PA(pBuff);// set buffer
        pEDcpt->w=0;// clear all the fields
        pEDcpt->NPV= 1;// set next pointer valid
        pEDcpt->EOWN= 1;// set hw ownership

        if(tailDcpt)
        {
            tailDcpt->next_ed=VA_TO_PA(&pEDcpt);
        }
        tailDcpt=pEDcpt;
    }
    // at this moment pEDcpt is an extra descriptor we use to end the descriptors list
    pEDcpt->w=0;// software ownership
    tailDcpt->next_ed= VA_TO_PA(&pEDcpt);

    PLIB_ETH_RxPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt));
    // set the address of the first RX descriptor
    PLIB_ETH_ReceiveEnable(id);
    // the Ethernet Controller will receive frames and place them in the receive buffers
    // we just programmed
    /* do something else in between */

    while(PLIB_ETH_RxPacketCountGet(id)==0)
    {
        //Wait until something is received.
        // Packets are received as set by the receive filter
    }

    //Receive the packet
    //doSomethingWithReceivedPacket()
    // After the received packet has been processed, restore EOWN to 1 and
    PLIB_ETH_RxBufferCountDecrement(id);
}
```

Support for Legacy "Ethernet Controller Library"

These routines (PLIB_ETH_Legacy*) provide MPLAB Harmony support for applications that use the "Ethernet Controller Library for Microchip

PIC32MX Microcontrollers" that is installed with XC32 compilers. Refer to the related compiled Help, `hlpETH.chm`, which is located in the `../Microchip/xc32/<version>/docs/pic32-lib-help` folder of your XC32 compiler installation for legacy information.

An additional argument, an index value from the ETH_MODULE_ID enumeration, has been added to allow these functions to work from within the context of this MPLAB Harmony Peripheral Library.

Legacy Controller Library	MPLAB Harmony
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Peripheral Library	Legacy Controller Library
PLIB_ETH_EventsClr	EthEventsClr
PLIB_ETH_EventsEnableClr	EthEventsEnableClr
PLIB_ETH_EventsEnableGet	EthEventsEnableGet
PLIB_ETH_EventsEnableSet	EthEventsEnableSet
PLIB_ETH_EventsEnableWrite	EthEventsEnableWrite
PLIB_ETH_EventsGet	EthEventsGet
PLIB_ETH_MACGetAddress	EthMACGetAddress
PLIB_ETH_MACSetAddress	EthMACSetAddress
PLIB_ETH_MACSetMaxFrame	EthMACSetMaxFrame
PLIB_ETH_RxFiltersClr	EthRxFiltersClr
PLIB_ETH_RxFiltersHTSet	EthRxFiltersHTSet
PLIB_ETH_RxFiltersPMClr	EthRxFiltersPMClr
PLIB_ETH_RxFiltersPMSet	EthRxFiltersPMSet
PLIB_ETH_RxFiltersSet	EthRxFiltersSet
PLIB_ETH_RxFiltersWrite	EthRxFiltersWrite
PLIB_ETH_RxSetBufferSize	EthRxSetBufferSize
PLIB_ETH_StatRxAlgnErrCnt	EthStatRxAlgnErrCnt
PLIB_ETH_StatRxFcsErrCnt	EthStatRxFcsErrCnt
PLIB_ETH_StatRxOkCnt	EthStatRxOkCnt
PLIB_ETH_StatRxOvflCnt	EthStatRxOvflCnt
PLIB_ETH_StatTxMColCnt	EthStatTxMColCnt
PLIB_ETH_StatTxOkCnt	EthStatTxOkCnt
PLIB_ETH_StatTxSColCnt	EthStatTxSColCnt

Configuring the Library

The library is configured for the supported Ethernet module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.
	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full duplex operation.

	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.
	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.
	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.
	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.
	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMIISpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMIISpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.

b) Control Functions

	Name	Description
	PLIB_ETH_MCSRxResetDisable	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetEnable	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetIsEnabled	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	PLIB_ETH_MCSTxResetDisable	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetEnable	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.

	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.
	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send.
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.
	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.

c) Status Functions

	Name	Description
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.
	PLIB_ETH_ReceiveIsBusy	Gets the Ethernet receive logic busy status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status

d) Filtering Functions

	Name	Description
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.

	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.
	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.
	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.

e) Flow Control Functions

	Name	Description
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlIsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC backpressure/no back-off.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC backpressure/no back-off enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlIsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no back-off.
	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.
	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.
	PLIB_ETH_TxPauseEnable	Enables the transmission Pause frames.
	PLIB_ETH_TxPauseIsEnabled	Gets the Ethernet module enable status.

f) Interrupt Functions

	Name	Description
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.

	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.
	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.

g) Statistics Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.

h) Feature Existence Functions

	Name	Description
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.
	PLIB_ETH_ExistsFCSErrorCount	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramesTransmittedOK	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramexReceivedOK	Identifies whether the FramexReceivedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsHashTable	Identifies whether the HashTable feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterPacketGaps	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterrupt	Identifies whether the Interrupt feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Configuration	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Resets	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Testing	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	PLIB_ETH_ExistsManualFlowControl	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsMaxFrameLength	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Config	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Indicators	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMAddresses	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMReadWrite	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMScanMode	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIWriteReadData	Identifies whether the MIIWriteReadData feature exists on the Ethernet module.
	PLIB_ETH_ExistsPatternMatch	Identifies whether the PatternMatch feature exists on the Ethernet module.
	PLIB_ETH_ExistsPauseTimer	Identifies whether the PauseTimer feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveBufferSize	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveFilters	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveOverflowCount	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.

	PLIB_ETH_ExistsReceiveWmarks	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.
	PLIB_ETH_ExistsRetransmissionMaximum	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	PLIB_ETH_ExistsRMII_Support	Identifies whether the RMII_Support feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxBufferCountDecrement	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.

i) Data Types and Constants

	Name	Description
	ETH_AUTOPAD_OPTION	Lists the possible automatic padding configurations.
	ETH_INTERRUPT_SOURCES	Lists the possible values of ETH_INTERRUPT_SOURCES.
	ETH_MIIM_CLK	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	ETH_PATTERN_MATCH_MODE	Lists the possible pattern match values.
	_ETH_PATTERN_MATCH_MODE_	Lists the possible pattern match values.
	ETH_RECEIVE_FILTER	Lists the possible values of the receive filter.
	ETH_RMII_SPEED	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Description

This section describes the Application Programming Interface (API) functions of the Ethernet Peripheral Library.

Refer to each section for a detailed description.

a) Initialization Functions

PLIB_ETH_AutoDetectPadClear Function

Clears the EMAC Auto-Pad option.

File

plib_eth.h

C

```
void PLIB_ETH_AutoDetectPadClear(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);
```

Returns

None.

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than ETH_AUTOPAD_DISABLED_CHECK_CRC is selected, the CRC must be enabled by PLIB_ETH_CRCEnable.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoDetectPadClear(MY_ETH_INSTANCE, PAD_OPTION);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoDetectPadClear(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);

PLIB_ETH_AutoDetectPadGet Function

Gets the current EMAC Auto-Pad setting.

File

plib_eth.h

C

```
ETH_AUTOPAD_OPTION PLIB_ETH_AutoDetectPadGet(ETH_MODULE_ID index);
```

Returns

ETH_AUTOPAD_OPTION.

Description

This function gets the current EMAC Auto-Pad setting.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
autoPadOption = PLIB_ETH_AutoDetectPadGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_AUTOPAD_OPTION PLIB_ETH_AutoDetectPadGet(ETH_MODULE_ID index);

PLIB_ETH_AutoDetectPadSet Function

Sets the EMAC Auto-Pad option.

File

plib_eth.h

C

```
void PLIB_ETH_AutoDetectPadSet(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);
```

Returns

None.

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than ETH_AUTOPAD_DISABLED_CHECK_CRC is selected, the CRC must be enabled by PLIB_ETH_CRCEnable.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoDetectPadSet(MY_ETH_INSTANCE, PAD_OPTION);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoDetectPadSet(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);

PLIB_ETH_BackToBackIPGGet Function

Gets the EMAC back-to-back interpacket gap.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_BackToBackIPGGet(ETH_MODULE_ID index);
```

Returns

- backToBackIPGValue - Back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC back-to-back interpacket gap.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_BackToBackIPGGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_BackToBackIPGGet(ETH_MODULE_ID index)

PLIB_ETH_BackToBackIPGSet Function

Sets the EMAC back-to-back interpacket gap.

File

plib_eth.h

C

```
void PLIB_ETH_BackToBackIPGSet(ETH_MODULE_ID index, uint8_t backToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC back-to-back interpacket gap.

Remarks

In Full-Duplex mode, the register value should be the desired period in nibble times minus 3. In Full-Duplex, the recommended setting is 0x15.

In Half-Duplex mode, the register value should be the desired period in nibble times minus 6. n Half-Duplex, the recommended setting is 0x12.
This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackToBackIPGSet(MY_ETH_INSTANCE, backToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
backToBackIPGValue	Back-to-back interpacket gap

Function

void PLIB_ETH_BackToBackIPGSet(ETH_MODULE_ID index, uint8_t backToBackIPGValue)

PLIB_ETH_CollisionWindowGet Function

Gets the EMAC collision window.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_CollisionWindowGet(ETH_MODULE_ID index);
```

Returns

- collisionWindowValue - A uint8_t (6-bit) value.

Description

This function gets the EMAC collision window.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_CollisionWindowGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_CollisionWindowGet(ETH_MODULE_ID index)

PLIB_ETH_CollisionWindowSet Function

Sets the EMAC collision window.

File

plib_eth.h

C

```
void PLIB_ETH_CollisionWindowSet(ETH_MODULE_ID index, uint8_t collisionWindowValue);
```

Returns

None.

Description

This function sets the EMAC collision window.

Remarks

This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Since the collision window starts at the beginning of transmission, the preamble and SFD is included. Its default of 0x37 (55d) corresponds to the count of frame bytes at the end of the window.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CollisionWindowSet(MY_ETH_INSTANCE, collisionWindowValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
collisionWindowValue	Collision window

Function

void PLIB_ETH_CollisionWindowSet(ETH_MODULE_ID index, uint8_t collisionWindowValue)

PLIB_ETH_DelayedCRCDisable Function

Disables EMAC delayed CRC.

File

plib_eth.h

C

```
void PLIB_ETH_DelayedCRCDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC delayed CRC. No proprietary header exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_DelayedCRCDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_DelayedCRCDisable(ETH_MODULE_ID index)

PLIB_ETH_DelayedCRCEnable Function

Enables EMAC delayed CRC.

File

plib_eth.h

C

```
void PLIB_ETH_DelayedCRCEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC delayed CRC. A proprietary packet header exists. Four bytes of the packet header are ignored by the CRC function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_DelayedCRCEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_DelayedCRCEnable(ETH_MODULE_ID index)

PLIB_ETH_DelayedCRCIsEnabled Function

Gets the EMAC delayed CRC enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_DelayedCRCIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - A proprietary header exists. Four bytes of packet header are ignored by the CRC function.
- false - No proprietary header

Description

This function returns the EMAC delayed CRC enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_DelayedCRCIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_DelayedCRCIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_Disable Function

Disables the Ethernet module.

File

plib_eth.h

C

```
void PLIB_ETH_Disable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet module.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_Disable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_Disable(ETH_MODULE_ID index)

PLIB_ETH_Enable Function

Enables the Ethernet module.

File

plib_eth.h

C

```
void PLIB_ETH_Enable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet module.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_Enable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_Enable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferDisable Function

Disables EMAC excess defer.

File

plib_eth.h

C

```
void PLIB_ETH_ExcessDeferDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC excess defer. The EMAC will abort when the excessive deferral limit is reached.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ExcessDeferDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ExcessDeferDisable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferEnable Function

Enables EMAC excess defer.

File

plib_eth.h

C

```
void PLIB_ETH_ExcessDeferEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC excess defer. The EMAC will defer to the carrier indefinitely.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ExcessDeferEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ExcessDeferEnable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferIsEnabled Function

Gets the EMAC excess defer enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ExcessDeferIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will defer to carrier indefinitely as per the Standard
- false - The EMAC will abort when the excessive deferral limit is reached

Description

This function gets the EMAC excess defer enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ExcessDeferIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ExcessDeferIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckDisable Function

Disables the EMAC frame length check.

File

plib_eth.h

C

```
void PLIB_ETH_FrameLengthCheckDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC frame length check.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FrameLengthCheckDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FrameLengthCheckDisable(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckEnable Function

Enables the EMAC frame length check.

File

plib_eth.h

C

```
void PLIB_ETH_FrameLengthCheckEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC frame length check. Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length, the check is performed. Mismatches are reported on the transmit/receive statistics vector.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FrameLengthCheckEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FrameLengthCheckEnable(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckIsEnabled Function

Gets the EMAC frame length check status.

File

plib_eth.h

C

```
bool PLIB_ETH_FrameLengthCheckIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length then the check is performed. Mismatches are reported on the transmit/receive statistics vector.
- false - Length/Type field check is not performed

Description

This function returns the EMAC frame length check status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_FrameLengthCheckIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_FrameLengthCheckIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexDisable Function

Disables the EMAC full duplex operation.

File

plib_eth.h

C

```
void PLIB_ETH_FullDuplexDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC full-duplex operation, which results in the EMAC operating in half-duplex.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FullDuplexDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FullDuplexDisable(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexEnable Function

Enables the EMAC full duplex operation.

File

plib_eth.h

C

```
void PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC full duplex operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FullDuplexEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexIsEnabled Function

Gets the EMAC full duplex enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_FullDuplexIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC operates in Full-Duplex mode
- false - The EMAC operates in Half-Duplex mode

Description

This function returns the EMAC full duplex enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_FullDuplexIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_FullDuplexIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameDisable Function

Disables the EMAC from receiving huge frames.

File

plib_eth.h

C

```
void PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC from receiving huge frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_HugeFrameDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameEnable Function

Enables the EMAC to receive huge frames.

File

plib_eth.h

C

```
void PLIB_ETH_HugeFrameEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC receiving the frames of any length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_HugeFrameEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_HugeFrameEnable(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameIsEnabled Function

Gets the EMAC huge frame enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_HugeFrameIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Frames of any length are transmitted and received
- false - Huge frames are not allowed for receive or transmit

Description

This function returns the EMAC huge frame enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_HugeFrameIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_HugeFrameIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_IsEnabled Function

Gets the Ethernet module enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_IsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module is enabled
- false - Ethernet module is disabled

Description

This function returns the Ethernet module enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_IsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_IsEnabled(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleDisable Function

Disables EMAC long preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_LongPreambleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC long preamble enforcement. The EMAC allows any length preamble.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LongPreambleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LongPreambleDisable(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleEnable Function

Enables EMAC long preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_LongPreambleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC long preamble enforcement. The EMAC only allows receive packets which contain preamble fields less than 12 bytes in length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LongPreambleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LongPreambleEnable(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleIsEnabled Function

Gets the EMAC long preamble enforcement enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_LongPreambleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC only allows receive packets, which contain preamble fields less than 12 bytes in length
- false - The EMAC allows any length preamble

Description

This function gets the EMAC long preamble enforcement enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LongPreambleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LongPreambleIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_LoopbackDisable Function

Disables the EMAC loopback logic.

File

plib_eth.h

C

```
void PLIB_ETH_LoopbackDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC loopback. EMAC resumes normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LoopbackDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LoopbackDisable(ETH_MODULE_ID index)

PLIB_ETH_LoopbackEnable Function

Enables the EMAC loopback logic.

File

plib_eth.h

C

```
void PLIB_ETH_LoopbackEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC loopback logic. The EMAC transmit interface is looped back to the EMAC receive interface.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LoopbackEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LoopbackEnable(ETH_MODULE_ID index)

PLIB_ETH_LoopbackIsEnabled Function

Gets the EMAC Loopback interface enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_LoopbackIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC transmit interface is looped back to the EMAC receive interface

- false - EMAC normal operation

Description

This function gets the EMAC Loopback interface enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LoopbackIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LoopbackIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MaxFrameLengthGet Function

Gets the EMAC maximum frame length.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MaxFrameLengthGet(ETH_MODULE_ID index);
```

Returns

- MaxFrameLength - Maximum frame length

Description

This function gets the EMAC maximum frame length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_MaxFrameLengthGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MaxFrameLengthGet(ETH_MODULE_ID index)

PLIB_ETH_MaxFrameLengthSet Function

Sets the EMAC maximum frame length.

File

plib_eth.h

C

```
void PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID index, uint16_t MaxFrameLength);
```

Returns

None.

Description

This function sets the EMAC maximum frame length.

Remarks

This field resets to 0x05EE, which represents a maximum receive frame of 1518 bytes. An untagged maximum size Ethernet frame is 1518 bytes. A tagged frame adds four octets for a total of 1522 bytes. If a shorter/longer maximum length restriction is desired, program this 16-bit field.

If a proprietary header is allowed, this field should be adjusted accordingly. For example, if 4-byte headers are prepended to frames, MACMAXF could be set to 1527 bytes. This would allow the maximum VLAN tagged frame plus the 4-byte header.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MaxFrameLengthSet(MY_ETH_INSTANCE, MaxFrameLength);
```

Parameters

Parameters	Description
index	Identifier for the device instance
MaxFrameLength	Maximum frame length

Function

void PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID index, uint16_t MaxFrameLength)

PLIB_ETH_MIIMClockGet Function

Gets the current EMAC MIIM clock selection.

File

plib_eth.h

C

```
ETH_MIIM_CLK PLIB_ETH_MIIMClockGet(ETH_MODULE_ID index);
```

Returns

- MIIMClock - Of type PLIB_ETH_MIIM_CLK

Description

This function returns the current EMAC MIIM clock selection.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
clkstat = PLIB_ETH_MIIMClockGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_MII_CLK PLIB_ETH_MIIMClockGet(ETH_MODULE_ID index)

PLIB_ETH_MIIMClockSet Function

Sets the EMAC MIM clock selection.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMClockSet(ETH_MODULE_ID index, ETH_MIIM_CLK MIIMClock);
```

Returns

None.

Description

This function sets the EMAC MIIM clock selection.

Remarks

This field is used by the clock divide logic in creating the MII Management Clock (MDC), which the IEEE 802.3 Specification defines to be no faster than 2.5 MHz. Some PHYs support clock rates up to 12.5 MHz.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMClockSet(MY_ETH_INSTANCE, MIIMClock);
```

Parameters

Parameters	Description
index	Identifier for the device instance
MIIMClock	of type ETH_MIIM_CLK - the system clock divisor for MII

Function

void PLIB_ETH_MIIMClockSet(ETH_MODULE_ID index, ETH_MIIM_CLK MIIMClock)

PLIB_ETH_MIIMNoPreDisable Function

Disables EMAC No Preamble (allows preamble).

File

plib_eth.h

C

```
void PLIB_ETH_MIIMNoPreDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC No preamble (allows preamble). Normal read/write cycles are performed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMNoPreDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMNoPreDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMNoPreEnable Function

Enables EMAC MIIM No Preamble (suppresses preamble).

File

plib_eth.h

C

```
void PLIB_ETH_MIIMNoPreEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC MIIM No Preamble (suppresses preamble). The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMNoPreEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMNoPreEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMNoPreIsEnabled Function

Gets the EMAC MIIM No Preamble enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMNoPreIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.
- false - Normal read/write cycles are performed

Description

This function gets the EMAC MIIM No Preamble enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMNoPreIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMNoPreIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG1Get Function

Gets the EMAC non-back-to-back interpacket gap register 1.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_NonBackToBackIPG1Get(ETH_MODULE_ID index);
```

Returns

- nonBackToBackIPGValue - Non-back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC non-back-to-back interpacket gap register 1.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_NonBackToBackIPG1Get(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_NonBackToBackIPG1Get(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG1Set Function

Sets the EMAC non-back-to-back interpacket gap register 1.

File

plib_eth.h

C

```
void PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC non-back-to-back interpacket gap register 1. A value of 0x0C is recommended.

Remarks

This is a programmable field representing the optional carrierSense window referenced in the IEEE 802.3 Specification. If a carrier is detected during the timing of IPGR1, the MAC defers to the carrier. If, however, the carrier comes after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thereby ensuring fair access to the medium. Its range of values is 0x0 to IPGR2. Its recommended value is 0xC (12d).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NonBackToBackIPG1Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back interpacket gap

Function

void PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue)

PLIB_ETH_NonBackToBackIPG2Get Function

Gets the EMAC non-back-to-back interpacket gap register 2.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_NonBackToBackIPG2Get(ETH_MODULE_ID index);
```

Returns

- nonBackToBackIPGValue - Non-back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC non-back-to-back interpacket gap register 2.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_NonBackToBackIPG2Get(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_NonBackToBackIPG2Get(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG2Set Function

Sets the EMAC non-back-to-back interpacket gap register 2.

File

plib_eth.h

C

```
void PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC non-back-to-back interpacket gap register 2. A value of 0x12 is recommended.

Remarks

This is a programmable field representing the non-back-to-back Inter-Packet-Gap. Its recommended value is 0x12 (18d), which represents the minimum IPG of 0.96 us (in 100 Mbps) or 9.6 us (in 10 Mbps).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NonBackToBackIPG2Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back interpacket gap

Function

void PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue)

PLIB_ETH_PauseTimerGet Function

Gets the Pause Timer value used for flow control.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PauseTimerGet(ETH_MODULE_ID index);
```

Returns

- PauseTimerValue - Pause Timer Value

Description

This function gets the Pause Timer value used for flow control.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PauseTimerValue = PLIB_ETH_PauseTimerGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PauseTimerGet(ETH_MODULE_ID index)

PLIB_ETH_PauseTimerSet Function

Sets the Pause Timer value used for flow control.

File

plib_eth.h

C

```
void PLIB_ETH_PauseTimerSet(ETH_MODULE_ID index, uint16_t PauseTimerValue);
```

Returns

None.

Description

This function sets the Pause Timer value used for flow control.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Write to the Pause Timer register before enabling the receiver. Call this function before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PauseTimerSet(MY_ETH_INSTANCE, PauseTimerValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
PauseTimerValue	used for Flow Control

Function

void PLIB_ETH_PauseTimerSet(ETH_MODULE_ID index, uint16_t PauseTimerValue)

PLIB_ETH_ReceiveBufferSizeGet Function

Gets the Ethernet module receive buffer size.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_ReceiveBufferSizeGet(ETH_MODULE_ID index);
```

Returns

ReceiveBufferSize - receive buffer size divided by 16

Description

This function gets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
size = PLIB_ETH_ReceiveBufferSizeGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_ReceiveBufferSizeGet(ETH_MODULE_ID index)

PLIB_ETH_ReceiveBufferSizeSet Function

Sets the Ethernet module receive buffer size.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID index, uint8_t ReceiveBufferSize);
```

Returns

None.

Description

This function sets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveBufferSizeSet(MY_ETH_INSTANCE, ReceiveBufferSize/16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance
ReceiveBufferSize	In units of 16 bytes, must be a value of 0x01 to 0x7F, 0x00 is invalid

Function

void PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID index, uint8_t ReceiveBufferSize)

PLIB_ETH_ReceiveDisable Function

Disables the EMAC from receiving frames.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC from receiving frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ReceiveDisable(ETH_MODULE_ID index)

PLIB_ETH_ReceiveEnable Function

Enables the EMAC to receive the frames.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC to receive the frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ReceiveEnable(ETH_MODULE_ID index)

PLIB_ETH_ReceiveIsEnabled Function

Gets the Ethernet EMAC receive enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Enable the EMAC to receive frames
- false - Disable the EMAC to receive frames

Description

This function gets the Ethernet EMAC receive enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ReceiveIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ReTxMaxGet Function

Gets the EMAC maximum retransmissions.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_ReTxMaxGet(ETH_MODULE_ID index);
```

Returns

- retransmitMax - Maximum number of retransmissions

Description

This function gets the EMAC maximum retransmissions.

Remarks

The maximum number of attempts is limited to 0x0F.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_ReTxMaxGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_ReTxMaxGet(ETH_MODULE_ID index)

PLIB_ETH_ReTxMaxSet Function

Sets the EMAC maximum retransmissions.

File

plib_eth.h

C

```
void PLIB_ETH_ReTxMaxSet(ETH_MODULE_ID index, uint16_t retransmitMax);
```

Returns

None.

Description

This function sets the EMAC maximum retransmissions.

Remarks

This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The IEEE 802.3 Specification standard specifies the maximum number of attempts (attemptLimit) to be 0xF (15d). Its default is 000.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReTxMaxSet(retransmitMax);
```

Parameters

Parameters	Description
index	Identifier for the device instance
retransmitMax	Maximum number of retransmissions

Function

void PLIB_ETH_ReTxMaxSet(ETH_MODULE_ID index, uint8_t retransmitMax)

PLIB_ETH_RMIISpeedGet Function

Gets the current EMAC RMII speed.

File

plib_eth.h

C

```
ETH_RMII_SPEED PLIB_ETH_RMIISpeedGet(ETH_MODULE_ID index);
```

Returns

- RMII_100Mbps - RMII running at 100 Mbps
- RMII_10Mbps - RMII running at 10 Mbps

Description

This function gets the current EMAC RMII speed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIISpeedGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_RMII_SPEED PLIB_ETH_RMIISpeedGet(ETH_MODULE_ID index)

PLIB_ETH_RMIISpeedSet Function

Sets EMAC RMII Speed.

File

plib_eth.h

C

```
void PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID index, ETH_RMII_SPEED RMIISpeed);
```

Returns

None.

Description

This function sets EMAC RMII speed. RMII speed can be either RMII_100Mbps or RMII_10Mbps.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIISpeedSet(MY_ETH_INSTANCE, ETH_RMII_SPEED RMIISpeed);
```

Parameters

Parameters	Description
index	Identifier for the device instance
RMIISpeed	RMII_100Mbps or RMII_10Mbps of type ETH_RMII_SPEED

Function

void PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID index, ETH_RMII_SPEED RMIISpeed)

PLIB_ETH_StopInIdleDisable Function

Ethernet module operation will continue when the device enters Idle mode.

File

plib_eth.h

C

```
void PLIB_ETH_StopInIdleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function directs the Ethernet module to continue operation when the device enters Idle mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StopInIdleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_StopInIdleDisable(ETH_MODULE_ID index)

PLIB_ETH_StopInIdleEnable Function

Ethernet module operation will stop when the device enters Idle mode.

File

plib_eth.h

C

```
void PLIB_ETH_StopInIdleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function directs the Ethernet module to stop operation when the device enters Idle mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StopInIdleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_StopInIdleEnable(ETH_MODULE_ID index)

PLIB_ETH_StopInIdleIsEnabled Function

Gets the Ethernet module Stop in Idle mode enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_StopInIdleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module transfers stop during Idle mode
- false - Ethernet module transfers continue to run during Idle mode

Description

This function returns the Ethernet module Stop in Idle mode enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_StopInIdleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_StopInIdleIsEnabled(ETH_MODULE_ID index)

b) Control Functions

PLIB_ETH_MCSRxResetDisable Function

Disables the reset EMAC Control Sub-layer/Receive domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSRxResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSRxResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSRxResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MCSRxResetEnable Function

Enables the reset EMAC Control Sub-layer/Receive domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSRxResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSRxResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSRxResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MCSRxResetIsEnabled Function

Gets the EMAC Control Sub-layer/Receive domain logic reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_MCSRxResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Control Sub-layer/Receive domain logic is in reset
- false - EMAC Control Sub-layer/Receive domain logic is not in reset

Description

This function gets the EMAC Control Sub-layer/Receive domain logic reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MCSRxResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MCSRxResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetDisable Function

Disables the reset EMAC Control Sub-layer/Transmit domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSTxResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the reset EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSTxResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSTxResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetEnable Function

Enables the reset of EMAC Control Sub-layer/Transmit domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSTxResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function resets the EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSTxResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSTxResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetIsEnabled Function

Gets the EMAC Control Sub-layer/Transmit domain logic reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_MCSTxResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Control Sub-layer/Transmit domain logic is held in reset
- false - EMAC Control Sub-layer/Transmit domain logic is not in reset

Description

This function gets the EMAC Control Sub-layer/Transmit domain logic reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MCSTxResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MCSTxResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMReadDataGet Function

Gets EMAC MIIM management read data after a MII read cycle has completed.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MIIMReadDataGet(ETH_MODULE_ID index);
```

Returns

- readData - MII read data

Description

This function gets EMAC MIIM management read data after a MII read cycle has completed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
readData = PLIB_ETH_MIIMReadDataGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MIIMReadDataGet(ETH_MODULE_ID index)

PLIB_ETH_MIIMReadStart Function

Initiates an MII management read command.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMReadStart(ETH_MODULE_ID index);
```

Returns

None.

Description

This function initiates an MII read command. The MII Management module will perform a single read cycle. To get data, use PLIB_ETH_MIIReadDataGet.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
while (PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE))
{
    //Wait until MII Link is not busy
}
PLIB_ETH_MIIReadStart(MY_ETH_INSTANCE);
while (PLIB_ETH_MIIReadIsDataNotValid(MY_ETH_INSTANCE))
{
    //Wait until read is complete
}
data = PLIB_ETH_MIIReadDataGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMReadStart(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetDisable Function

Disables EMAC Reset MII Management.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Reset MII Management. EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetEnable Function

Enables EMAC Reset Media Independent Interface (MII) Management.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Reset MII Management and holds the MII Management module in reset while enabled.

Remarks

MII Management held in Reset after This function is called. Disable ResetMIIanagement to return to normal operation.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetIsEnabled Function

Gets the EMAC Reset MII Management enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Reset the MII Management module
- false - Normal operation

Description

This function gets the EMAC Reset MII Management enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrEnable Function

Enables EMAC MIIM Scan Increment.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanIncrEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC MIIM Scan Increment. The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.

Remarks

The read cycles will start at PHY address 1 and continue through the value set for as the PHY address register.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanIncrEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanIncrEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrDisable Function

Disables the EMAC MIIM Scan Increment.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanIncrDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC MIIM Scan Increment. Allows continuous reads of the same PHY.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanIncrDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanIncrDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrIsEnabled Function

Gets the EMAC MIIM scan increment enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMScanIncrIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.
- false - Continuous reads of the same PHY

Description

This function gets the EMAC MIIM scan increment enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
ScanIncrement = PLIB_ETH_MIIMScanIncrIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMScanIncrIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeDisable Function

Disables MIIM scan mode.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanModeDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables MIIM scan mode. Scan is disabled for Normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanModeDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanModeDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeEnable Function

Enables MIIM scan mode.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanModeEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables MIIM scan mode. The MII Management module will perform read cycles continuously. (Useful for monitoring the Link Fail.)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanModeEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanModeEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeIsEnabled Function

Gets the MII management scan enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMScanModeIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module will perform read cycles continuously (for example, useful for monitoring the Link Fail)
- false - Normal operation

Description

This function returns the MII management scan enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMScanModeIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMScanModeIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMWriteDataSet Function

Sets the EMAC MIIM write data before initiating an MII write cycle.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMWriteDataSet(ETH_MODULE_ID index, uint16_t writeData);
```

Returns

None.

Description

This function sets the EMAC MIIM write data before initiating write cycle.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Prior to a call to this routine, the PHY and Register addresses should be set using PLIB_ETH_MIIPHYAddressSet and PLIB_ETH_MIIRegisterAddressSet.

Example

```
PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, WriteData);
```

Parameters

Parameters	Description
index	Identifier for the device instance
writeData	MII write data

Function

void PLIB_ETH_MIIMWriteDataSet(ETH_MODULE_ID index, uint16_t writeData)

PLIB_ETH_MIIMWriteStart Function

Initiates an MII management write command.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMWriteStart(ETH_MODULE_ID index);
```

Returns

None.

Description

This function initiates an MII management read command. The MII Management module will perform a write cycle.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

The PHY address and MII register address must be configured before a write using PLIB_ETH_MIIPHYAddressSet(MY_ETH_INSTANCE, phyAddr) and PLIB_ETH_MIIRegisterAddressSet(MY_ETH_INSTANCE, regAddr)

Data to be written must be first loaded into the MII write register using PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, writeData)

Example

```
PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, dataToWrite);
PLIB_ETH_MIIMWriteStart(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMWriteStart(ETH_MODULE_ID index)

PLIB_ETH_MIIResetDisable Function

Disables the EMAC Soft reset.

File

plib_eth.h

C

```
void PLIB_ETH_MIIResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC MIIM Soft reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIResetEnable Function

Enables the EMAC MIIM Soft reset.

File

plib_eth.h

C

```
void PLIB_ETH_MIIResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC MIIM soft reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIResetIsEnabled Function

Gets EMAC MIIM Soft Reset enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - MAC MII is in reset

- false - MAC MII is not in reset

Description

This function gets EMAC MIIM Soft reset enable status. By default this bit is set to '1'.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_PHYAddressGet Function

Gets the EMAC MIIM management PHY address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_PHYAddressGet(ETH_MODULE_ID index);
```

Returns

- phyAddr - A 5-bit address of the PHY

Description

This function gets the EMAC MIIM management PHY address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
phyAddr = PLIB_ETH_PHYAddressGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_PHYAddressGet(ETH_MODULE_ID index)

PLIB_ETH_PHYAddressSet Function

Sets the EMAC MIIM PHY address.

File

plib_eth.h

C

```
void PLIB_ETH_PHYAddressSet(ETH_MODULE_ID index, uint8_t phyAddr);
```

Returns

None.

Description

This function sets the EMAC MIIM PHY address. This field represents the 5-bit PHY Address field of Management cycles. Up to 31 PHYs can be addressed (0 is reserved).

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PHYAddressSet(MY_ETH_INSTANCE, PhyAddr);
```

Parameters

Parameters	Description
index	Identifier for the device instance
phyAddr	A 5-bit address of the PHY

Function

void PLIB_ETH_PHYAddressSet(ETH_MODULE_ID index, uint8_t phyAddr)

PLIB_ETH_RegisterAddressGet Function

Gets the EMAC MIIM management register address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RegisterAddressGet(ETH_MODULE_ID index);
```

Returns

- regAddr - The (5-bit) address of the MII Registers

Description

This function gets the EMAC MIIM management register address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
regAddr = PLIB_ETH_RegisterAddressGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RegisterAddressGet(ETH_MODULE_ID index)

PLIB_ETH_RegisterAddressSet Function

Sets EMAC MIIM register address.

File

plib_eth.h

C

```
void PLIB_ETH_RegisterAddressSet(ETH_MODULE_ID index, uint8_t regAddr);
```

Returns

None.

Description

This function sets the EMAC MIIM register address. Up to 32 registers may be accessed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RegisterAddressSet(MY_ETH_INSTANCE, regAddr);
```

Parameters

Parameters	Description
index	Identifier for the device instance
regAddr	The (5-bit) address of the MII Registers.

Function

void PLIB_ETH_RegisterAddressSet(ETH_MODULE_ID index, uint8_t regAddr)

PLIB_ETH_RMIIResetDisable Function

Disables EMAC Reset RMII.

File

plib_eth.h

C

```
void PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Reset RMII for normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIIResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID index)

PLIB_ETH_RMIIResetEnable Function

Enables EMAC Reset RMII.

File

plib_eth.h

C

```
void PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Reset RMII.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIIResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID index)

PLIB_ETH_RMIIResetIsEnabled Function

Gets the EMAC Reset RMII enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RMIIResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Reset the EMAC RMII module
- false - Normal operation

Description

This function gets the EMAC Reset RMII enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_RMIIResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RMIIResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxBufferCountDecrement Function

Causes the Receive Descriptor Buffer Counter to decrement by 1.

File

plib_eth.h

C

```
void PLIB_ETH_RxBufferCountDecrement(ETH_MODULE_ID index);
```

Returns

None.

Description

This function causes the Receive Descriptor Buffer Counter to decrement by 1.

Remarks

Hardware increments the receive buffer counter and software decrements it. If the receive buffer counter is incremented by the receive logic at the same time, the receive buffer counter will appear unchanged.

Always reads as '0', so there is no get value routine.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxBufferCountDecrement(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxBufferCountDecrement(ETH_MODULE_ID index)

PLIB_ETH_RxDisable Function

Disables the Ethernet module receive logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet receive logic.

Remarks

Disabling Ethernet receive is not recommended for making changes to any receive-related registers. After the receiver has been enabled, the

Ethernet module must be reinitialized to implement changes.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxDisable(ETH_MODULE_ID index)

PLIB_ETH_RxEnable Function

Enables the Ethernet receive logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet receive logic. Packets are received and stored in the receive buffer as controlled by the filter configuration.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

All receive registers must be configured before calling this function.

Example

```
PLIB_ETH_RxEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxEnable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetDisable Function

Disables the EMAC reset receive function logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxFuncResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC reset receive function logic. The reset receive function logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFuncResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxFuncResetDisable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetEnable Function

Enables the EMAC reset receive function logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxFuncResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC reset receive function logic. The receive function logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFuncResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxFuncResetEnable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetIsEnabled Function

Gets the EMAC reset receive function status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxFuncResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Receive function logic is held in reset
- false - EMAC Receive function logic is not in reset

Description

This function gets the EMAC reset receive function status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_RxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxFuncResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxIsEnabled Function

Gets the Ethernet module receive enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module receive is enabled
- false - Ethernet module receive is disabled

Description

This function returns the Ethernet module receive enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxPacketDescAddrGet Function

Gets the address of the next receive descriptor.

File

plib_eth.h

C

```
uint8_t * PLIB_ETH_RxPacketDescAddrGet(ETH_MODULE_ID index);
```

Returns

- ReceivePacketDescriptorAddress - receive packet descriptor address

Description

This function gets the address of the next receive descriptor to be used.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
RxPacketDescAddr = PLIB_ETH_RxPacketDescAddrGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t *PLIB_ETH_RxPacketDescAddrGet(ETH_MODULE_ID index)

PLIB_ETH_RxPacketDescAddrSet Function

Sets the Ethernet module receive packet descriptor start address.

File

plib_eth.h

C

```
void PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t * rxPacketDescStartAddr);
```

Returns

None.

Description

This function sets the Ethernet receive packet descriptor start address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Example

```
PLIB_ETH_RxPacketDescAddrSet(MY_ETH_INSTANCE, rxPacketDescStartAddr)
```

Parameters

Parameters	Description
index	Identifier for the device instance
rxPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Function

void PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t *rxPacketDescStartAddr)

PLIB_ETH_TestPauseDisable Function

Disables EMAC test pause.

File

plib_eth.h

C

```
void PLIB_ETH_TestPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Test Pause. The EMAC will resume normal operation.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_TestPauseEnable Function

Enables EMAC test pause.

File

plib_eth.h

C

```
void PLIB_ETH_TestPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC test pause. The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_TestPauseIsEnabled Function

Gets the EMAC test pause enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TestPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received
- false - Normal operation

Description

This function returns the EMAC test pause enable status.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TestPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TestPauseIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetDisable Function

Disables the EMAC Transmit function reset.

File

plib_eth.h

C

```
void PLIB_ETH_TxFuncResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC Transmit function reset. The EMAC transmit function is released from reset for normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxFuncResetDisable(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetEnable Function

Enables the EMAC transmit function reset.

File

plib_eth.h

C

```
void PLIB_ETH_TxFuncResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC transmit function reset. The transmit function is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxFuncResetEnable(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetIsEnabled Function

Gets the EMAC Transmit function reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxFuncResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC transmit function logic is held in reset
- false - EMAC transmit function logic is not in reset

Description

This function gets the EMAC Transmit function reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxFuncResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxPacketDescAddrGet Function

Gets the address of the next descriptor to be transmitted.

File

plib_eth.h

C

```
uint8_t * PLIB_ETH_TxPacketDescAddrGet(ETH_MODULE_ID index);
```

Returns

Transmit Packet Descriptor Start Address.

Description

This function gets the address of the next transmit descriptor.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
txPacketDescAddr = PLIB_ETH_TxPacketDescAddrGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t *PLIB_ETH_TxPacketDescAddrGet(ETH_MODULE_ID index)

PLIB_ETH_TxPacketDescAddrSet Function

Sets the Ethernet module transmit packet descriptor start address.

File

plib_eth.h

C

```
void PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t * txPacketDescStartAddr);
```

Returns

None.

Description

This function sets the Ethernet transmit packet descriptor start address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Example

```
PLIB_ETH_TxPacketDescAddrSet(MY_ETH_INSTANCE, txPacketDescStartAddr)
```

Parameters

Parameters	Description
index	Identifier for the device instance
txPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Function

void PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t *txPacketDescStartAddr)

PLIB_ETH_TxRTSDisable Function

Aborts an Ethernet module transmission.

File

plib_eth.h

C

```
void PLIB_ETH_TxRTSDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function aborts an Ethernet module transmission and disables the transmitter after the current packet has completed.

Remarks

When disabled by software, transmission stops after the current packet has been completed.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxRTSDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxRTSDisable(ETH_MODULE_ID index)

PLIB_ETH_TxRTSEnable Function

Enables the Ethernet transmit request to send.

File

plib_eth.h

C

```
void PLIB_ETH_TxRTSEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet request to send. Transmit logic is activated and any packets defined in the Ethernet descriptor table are transmitted.

Remarks

This status is cleared by hardware when the transmission is complete.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

The TX descriptor list and TX DMA must be initialized using PLIB_ETH_TransmitPacketDescStartAddrSet.

Example

```
PLIB_ETH_TxRTSEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxRTSEnable(ETH_MODULE_ID index)

PLIB_ETH_TxRTSIsEnabled Function

Gets the Ethernet module transmit request to send status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxRTSIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module transmit is active
- false - Ethernet module transmission has stopped or has completed

Description

This function returns the Ethernet module transmit request to send status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TransmitRTSIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxRTSIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_CRCDisable Function

Disables EMAC CRC.

File

plib_eth.h

C

```
void PLIB_ETH_CRCDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC CRC.

Remarks

The frames presented to the EMAC have a valid CRC

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CRCDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_CRCDisable(ETH_MODULE_ID index)

PLIB_ETH_CRCEnable Function

Enables EMAC CRC.

File

plib_eth.h

C

```
void PLIB_ETH_CRCEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC CRC. The EMAC will append CRC whether padding is required or not. This must be enabled if auto-padding is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CRCEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_CRCEnable(ETH_MODULE_ID index)

PLIB_ETH_CRCIsEnabled Function

Gets the EMAC CRC enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_CRCIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will append a CRC to every frame whether or not padding was required. Must be set if auto-padding is enabled.
- false - The frames presented to the EMAC have a valid CRC

Description

This function returns the EMAC CRC enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_CRCIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_CRCIsEnabled(ETH_MODULE_ID index)

c) Status Functions

PLIB_ETH_DataNotValid Function

Gets the MII management read data not valid status.

File

plib_eth.h

C

```
bool PLIB_ETH_DataNotValid(ETH_MODULE_ID index);
```

Returns

- true - The MII Management read cycle has not completed and the read data is not yet valid
- false - The MII Management read cycle is complete and the read data is valid

Description

This function gets the MII management read data not valid status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_DataNotValid(MY_ETH_INSTANCE);
```

Also see the example for function: PLIB_ETH_MIIReadStart().

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_DataNotValid(ETH_MODULE_ID index)

PLIB_ETH_EthernetIsBusy Function

Gets the status value of the Ethernet logic busy.

File

plib_eth.h

C

```
bool PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Ethernet logic has been turned ON or is completing a transaction
- false - Ethernet logic is idle

Description

This function sets the value of the Ethernet logic busy. A request indicates that the module has just been turned ON or is completing a transaction after being turned OFF.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_EthernetIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID index)

PLIB_ETH_LinkHasFailed Function

Gets the MII management link fail status.

File

plib_eth.h

C

```
bool PLIB_ETH_LinkHasFailed(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module link fail has occurred
- false - The MII Management module link fail has not occurred

Description

This function returns the MII management link fail status. This value reflects the last read from the PHY status register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LinkHasFailed(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LinkHasFailed(ETH_MODULE_ID index)

PLIB_ETH_MIIMIsBusy Function

Gets the MII management busy status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMIsBusy(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module is currently performing an MII Management read or write cycle
- false - The MII Management is free

Description

This function returns the MII management busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMIsBusy(ETH_MODULE_ID index)

PLIB_ETH_MIIMIsScanning Function

Gets the MII Management scanning status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMIsScanning(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module scan operation (continuous MII Management Read cycles) is in progress
- false - The MII Management scan operation is not in progress

Description

This function gets the MII Management scanning status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMIsScanning(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMIsScanning(ETH_MODULE_ID index)

PLIB_ETH_ReceiveIsBusy Function

Gets the Ethernet receive logic busy status.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Receive logic is receiving data
- false - Receive logic is idle

Description

This function gets the Ethernet receive logic busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ReceiveIsBusy(ETH_MODULE_ID index)

PLIB_ETH_TransmitIsBusy Function

Gets the status value of the Ethernet transmit logic busy status

File

plib_eth.h

C

```
bool PLIB_ETH_TransmitIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Transmit logic is sending data
- false - Transmit logic is idle

Description

This function gets the value of the Ethernet transmit logic busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TransmitIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TransmitIsBusy(ETH_MODULE_ID index)

d) Filtering Functions

PLIB_ETH_HashTableGet Function

Gets the value of the Hash table.

File

plib_eth.h

C

```
uint32_t PLIB_ETH_HashTableGet(ETH_MODULE_ID index);
```

Returns

- hashTable - Hash table value (64-bits)

Description

This function gets the value of the Hash table.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_HashTableGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint64_t PLIB_ETH_HashTableGet(ETH_MODULE_ID index)

PLIB_ETH_HashTableSet Function

Sets the Ethernet module Hash table with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_HashTableSet(ETH_MODULE_ID index, uint64_t hashTableValue);
```

Returns

None.

Description

This function sets the Hash table with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_HashTableFilterDisable to disable the Hash table filter before changing the hash table, or set the Hash Table prior to calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_HashTableSet(MY_ETH_INSTANCE, hashTableValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
hashTableValue	hash table value (64-bits)

Function

void PLIB_ETH_HashTableSet(ETH_MODULE_ID index, uint64_t hashTableValue)

PLIB_ETH_PassAllDisable Function

Disables the EMAC PassAll.

File

plib_eth.h

C

```
void PLIB_ETH_PassAllDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC PassAll. Control frames are ignored.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PassAllDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PassAllDisable(ETH_MODULE_ID index)

PLIB_ETH_PassAllEnable Function

Enables the EMAC PassAll.

File

plib_eth.h

C

```
void PLIB_ETH_PassAllEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC PassAll, which enables both the normal and the control frames to be passed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PassAllEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PassAllEnable(ETH_MODULE_ID index)

PLIB_ETH_PassAllIsEnabled Function

Gets the EMAC PassAll enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_PassAllIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will accept all frames regardless of type (normal vs. Control)
- false - The received Control frames are ignored

Description

This function gets the EMAC PassAll enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_PassAllIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_PassAllIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchChecksumGet Function

Gets the value of the pattern match checksum register.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PatternMatchChecksumGet(ETH_MODULE_ID index);
```

Returns

The pattern match checksum.

Description

This function gets the value of the patter match checksum register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchChecksumGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PatternMatchChecksumGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchChecksumSet Function

Sets the Ethernet module pattern match checksum register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchChecksumSet(ETH_MODULE_ID index, uint16_t PatternMatchChecksumValue);
```

Returns

None.

Description

This function sets the pattern match checksum register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchChecksumSet(MY_ETH_INSTANCE, PatternMatchChecksumValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PatternMatchChecksumValue	Pattern match checksum value

Function

void PLIB_ETH_PatternMatchChecksumSet(ETH_MODULE_ID index, uint16_t PatternMatchChecksumValue)

PLIB_ETH_PatternMatchGet Function

Gets the value of the selected pattern match mask register.

File

plib_eth.h

C

```
uint64_t PLIB_ETH_PatternMatchGet(ETH_MODULE_ID index);
```

Returns

- patternMatchMaskValue - Pattern Match Mask Values

Description

This function gets the selected value of the patter match mask register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint64_t PLIB_ETH_PatternMatchGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchModeGet Function

Gets the value of the selected pattern match mask register.

File

plib_eth.h

C

```
ETH_PATTERN_MATCH_MODE PLIB_ETH_PatternMatchModeGet(ETH_MODULE_ID index);
```

Returns

- modeSel - The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Description

This function gets the selected value of the patter match mask register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchModeGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_PATTERN_MATCH_MODE PLIB_ETH_PatternMatchModeGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchModeSet Function

Sets the Ethernet module pattern match mode.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchModeSet(ETH_MODULE_ID index, ETH_PATTERN_MATCH_MODE modeSel);
```

Returns

None.

Description

This function sets the pattern match mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Set the value before calling PLIB_ETH_ReceiveEnable().

Example

```
PLIB_ETH_PatternMatchModeSet(MY_ETH_INSTANCE, ETH_PATTERN_MATCH_DISABLED);
```

Parameters

Parameters	Description
index	Identifier for the device instance
modeSel	The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Function

void PLIB_ETH_PatternMatchModeSet(ETH_MODULE_ID index, ETH_PATTERN_MATCH_MODE modeSel)

PLIB_ETH_PatternMatchOffsetGet Function

Gets the value of the patter match offset register.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PatternMatchOffsetGet(ETH_MODULE_ID index);
```

Returns

- PatternMatchOffsetValue - Pattern match offset value

Description

This function gets the value of the patter match offset register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchOffsetGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PatternMatchOffsetGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchOffsetSet Function

Sets the Ethernet module patter match offset register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchOffsetSet(ETH_MODULE_ID index, uint16_t PatternMatchOffsetValue);
```

Returns

None.

Description

This function sets the pattern match offset register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchOffsetSet(MY_ETH_INSTANCE, PatternMatchOffsetValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance

PatternMatchOffsetValue	Pattern match offset value

Function

void PLIB_ETH_PatternMatchOffsetSet(ETH_MODULE_ID index, uint16_t PatternMatchOffsetValue)

PLIB_ETH_PatternMatchSet Function

Sets the Ethernet module pattern match mask register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchSet(ETH_MODULE_ID index, uint64_t patternMatchMaskValue);
```

Returns

None.

Description

This function sets the pattern match mask register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(index,ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchSet(MY_ETH_INSTANCE, patternMatchMaskValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
patternMatchMaskValue	Pattern Match Mask Values (64-bits)

Function

void PLIB_ETH_PatternMatchSet(ETH_MODULE_ID index, uint64_t patternMatchMaskValue)

PLIB_ETH_PurePreambleDisable Function

Disables EMAC pure preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_PurePreambleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC pure preamble enforcement. The EMAC does not perform any preamble checking.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PurePreambleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PurePreambleDisable(ETH_MODULE_ID index)

PLIB_ETH_PurePreambleEnable Function

Enables EMAC pure preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_PurePreambleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC pure preamble enforcement. The EMAC will verify the contents of the preamble and discard packets with errors in the preamble.

Remarks

The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PurePreambleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PurePreambleEnable(ETH_MODULE_ID index)

PLIB_ETH_PurePreambleIsEnabled Function

Gets EMAC pure preamble enforcement enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_PurePreambleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.
- false - The EMAC does not perform any preamble checking

Description

This function gets the EMAC pure preamble enforcement enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_PurePreambleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_PurePreambleIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ReceiveFilterDisable Function

Disables the specified receive filter.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveFilterDisable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

None.

Description

This function disables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveFilterDisable(index,
                              ETH_CRC_OK_FILTER      |
                              ETH_RUNT_ENABLE_FILTER |
                              ETH_UNICAST_FILTER     );
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Function

void PLIB_ETH_ReceiveFilterDisable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_ReceiveFilterEnable Function

Enables the specified receive filter.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

None.

Description

This function enables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveFilterEnable(index,
                             ETH_CRC_OK_FILTER      |
                             ETH_RUNT_ENABLE_FILTER |
                             ETH_UNICAST_FILTER     );
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be enabled from the enumerated selection ETH_RECEIVE_FILTER

Function

void PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_ReceiveFilterIsEnable Function

Disables the specified receive filter.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveFilterIsEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

- true - If at least one of the specified filters is enabled
- false - If no specified filter is enabled

Description

This function disables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if (PLIB_ETH_ReceiveFilterIsEnable(index, ETH_UNICAST_FILTER))
{
    PLIB_ETH_ReceiveFilterDisable(MY_MODULE_ID, ETH_UNICAST_FILTER);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Function

bool PLIB_ETH_ReceiveFilterIsEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_StationAddressGet Function

Gets the selected EMAC station address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_StationAddressGet(ETH_MODULE_ID index, uint8_t which);
```

Returns

- stationAddress1 - Station address

Description

This function gets the selected EMAC station address.

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stationAddr1 = PLIB_ETH_StationAddress1Get(MY_ETH_INSTANCE, which);
```

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.

Function

uint8_t PLIB_ETH_StationAddressGet(ETH_MODULE_ID index, uint8_t which)

PLIB_ETH_StationAddressSet Function

Sets the selected EMAC Station Address.

File

plib_eth.h

C

```
void PLIB_ETH_StationAddressSet(ETH_MODULE_ID index, uint8_t which, uint8_t stationAddress);
```

Returns

None.

Description

This function sets the selected EMAC Station Address.

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StationAddressSet(MY_ETH_INSTANCE, which, stationAddress);
```

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.
stationAddress	Station Address.

Function

void PLIB_ETH_StationAddressSet(ETH_MODULE_ID index, uint8_t which, uint8_t stationAddress)

e) Flow Control Functions

PLIB_ETH_AutoFlowControlDisable Function

Disables the Ethernet module Automatic Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_AutoFlowControlDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet Automatic Flow Control logic.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoFlowControlDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoFlowControlDisable(ETH_MODULE_ID index)

PLIB_ETH_AutoFlowControlEnable Function

Enables the Ethernet Automatic Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables Ethernet Automatic Flow Control logic.

Remarks

The full and empty watermarks are used to automatically enable and disable flow control, respectively. When the number of received buffers rises to the full watermark, flow control is automatically enabled. When the receive buffer count falls to the empty watermark, flow control is automatically disabled.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoFlowControlEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID index)

PLIB_ETH_AutoFlowControlIsEnabled Function

Gets the Ethernet module Automatic Flow Control status.

File

plib_eth.h

C

```
bool PLIB_ETH_AutoFlowControlIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module Automatic Flow Control is enabled
- false - Ethernet module Automatic Flow Control is disabled

Description

This function gets the Ethernet Automatic Flow Control enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_AutoFlowControlIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_AutoFlowControlIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffDisable Function

Disables EMAC backpressure/no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_BackPresNoBackoffDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC backpressure/no back-off. The EMAC will back-off when there is backpressure and collisions occur.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackPresNoBackoffDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_BackPresNoBackoffDisable(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffEnable Function

Enables EMAC back pressure/no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_BackPresNoBackoffEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC backpressure/no back-off. The EMAC will not back-off when there is backpressure and collisions occur.

Remarks

The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackPresNoBackoffEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_BackPresNoBackoffEnable(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffIsEnabled Function

Gets the EMAC backpressure/no back-off enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_BackPresNoBackoffIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent
- false - The EMAC will not remove the back-off

Description

This function gets the EMAC backpressure/no back-off enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_BackPresNoBackoffIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_BackPresNoBackoffIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlDisable Function

Disable Ethernet module Manual Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_ManualFlowControlDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet Manual Flow Control logic and automatically sends a Pause frame with a 0x0000 Pause Timer value. This function affects both transmit and receive operations.

Remarks

Disabling Manual Flow Control will automatically send a Pause frame with a 0x0000 Pause Timer value to disable flow control.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ManualFlowControlDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ManualFlowControlDisable(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlEnable Function

Enables the Ethernet Manual Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_ManualFlowControlEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet Manual Flow Control logic. Enabling Manual Flow Control will send a Pause frame using the Pause Timer value. While enabled, a Pause frame is repeated every 128 x (Pause timer value)/2 transmit clock cycles.

Affects both transmit and receive operations.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ManualFlowControlEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ManualFlowControlEnable(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlIsEnabled Function

Gets the Ethernet module Manual Flow Control enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ManualFlowControlIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module Manual Flow Control is enabled
- false - Ethernet module Manual Flow Control is disabled

Description

This function returns the Ethernet module Manual Flow Control enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ManualFlowControlIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ManualFlowControlIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffDisable Function

Disables EMAC no back-offs.

File

plib_eth.h

C

```
void PLIB_ETH_NoBackoffDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC no back-off. The EMAC will back-off when a collision occurs.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NoBackoffDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_NoBackoffDisable(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffEnable Function

Enables EMAC no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_NoBackoffEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC no back-off. The EMAC will not back-off when a collision occurs.

Remarks

Following a collision, the EMAC will immediately retransmit rather than using the Binary Exponential Back-off algorithm as specified in the Standard.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NoBackoffEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_NoBackoffEnable(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffIsEnabled Function

Gets the EMAC no back-off enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_NoBackoffIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Following a collision, the EMAC will immediately retransmit
- false - Following a collision, the EMAC will use the Binary Exponential Back-off algorithm

Description

This function gets the EMAC no back-off enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_NoBackoffIsEnabled(MY_ETsH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_NoBackoffIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxEmptyWmarkGet Function

Gets the Ethernet module receive empty watermark.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxEmptyWmarkGet(ETH_MODULE_ID index);
```

Returns

- receiveEmptyWmark - Empty watermark value

Description

This function gets the Ethernet receive empty watermark.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
watermarkValue = PLIB_ETH_RxEmptyWmarkGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxEmptyWmarkGet(ETH_MODULE_ID index)

PLIB_ETH_RxEmptyWmarkSet Function

Sets the Ethernet module receive empty water mark.

File

plib_eth.h

C

```
void PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue);
```

Returns

None.

Description

This function sets the Ethernet receive empty water mark with a new value. The software controlled Receive Buffer Empty Wmark is compared against the receive buffer count to determine the empty watermark condition for the empty watermark interrupt and for disabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxEmptyWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Empty watermark value

Function

void PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue)

PLIB_ETH_RxFullWmarkGet Function

Gets the Ethernet module to receive a full watermark.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxFullWmarkGet(ETH_MODULE_ID index);
```

Returns

- receiveFullWmark - Full watermark value

Description

This function gets the Ethernet to receive a full watermark.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
watermarkValue = PLIB_ETH_RxFullWmarkGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxFullWmarkGet(ETH_MODULE_ID index)

PLIB_ETH_RxFullWmarkSet Function

Sets the Ethernet module to receive a full watermark.

File

plib_eth.h

C

```
void PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue);
```

Returns

None.

Description

This function sets the Ethernet to receive a full watermark with a new value.

The software controlled RX Buffer Full Watermark (Wmark) Pointer is compared against the receive buffer count to determine the full watermark condition for the full watermark interrupt and for enabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFullWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Full watermark value

Function

void PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue)

PLIB_ETH_RxPauseDisable Function

Disables the EMAC receive flow control.

File

plib_eth.h

C

```
void PLIB_ETH_RxPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC receive flow control. The received Pause Flow control frames are ignored.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_RxPauseEnable Function

Enables the EMAC receive flow control.

File

plib_eth.h

C

```
void PLIB_ETH_RxPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC receive flow control. The EMAC will act upon the received Pause frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_RxPauseIsEnabled Function

Gets the EMAC receive flow pause enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC acts upon received Pause Flow Control frames
- false - Received Pause Flow Control frames are ignored

Description

This function gets the EMAC receive flow pause enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
status = PLIB_ETH_RxPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxPauseIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaDisable Function

Disables EMAC shortcut pause quanta.

File

plib_eth.h

C

```
void PLIB_ETH_ShortcutQuantaDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC shortcut pause quanta. The EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ShortcutQuantaDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ShortcutQuantaDisable(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaEnable Function

Enables EMAC shortcut pause quanta.

File

plib_eth.h

C

```
void PLIB_ETH_ShortcutQuantaEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC shortcut pause quanta. The EMAC reduces the effective pause quanta from 64 byte-times to 1 byte-time.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ShortcutQuantaEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ShortcutQuantaEnable(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaIsEnabled Function

Gets EMAC shortcut pause quanta enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ShortcutQuantaIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC reduces the effective Pause Quanta from 64 byte-times to 1 byte-time
- false - Normal operation

Description

This function returns EMAC shortcut pause quanta enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ShortcutQuantaIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ShortcutQuantaIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_SimResetDisable Function

Disables the EMAC simulation reset.

File

plib_eth.h

C

```
void PLIB_ETH_SimResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC simulation reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SimResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_SimResetDisable(ETH_MODULE_ID index)

PLIB_ETH_SimResetEnable Function

Enables the EMAC simulation reset.

File

plib_eth.h

C

```
void PLIB_ETH_SimResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC simulation reset and resets the random number generator within the transmit (TX) function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SimResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_SimResetEnable(ETH_MODULE_ID index)

PLIB_ETH_SimResetIsEnabled Function

Gets the EMAC simulation reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_SimResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Simulation Reset is enabled
- false - Simulation Reset is disabled

Description

This function returns the EMAC simulation reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_SimResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_SimResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressDisable Function

Disables EMAC Test backpressure.

File

plib_eth.h

C

```
void PLIB_ETH_TestBackPressDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Test backpressure. The EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestBackPressDisable(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressEnable Function

Enables EMAC Test backpressure.

File

plib_eth.h

C

```
void PLIB_ETH_TestBackPressEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Test backpressure. The EMAC will assert backpressure on the link. Backpressure causes preamble to be transmitted, raising carrier sense. A transmit packet from the system will be sent during backpressure.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestBackPressEnable(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressIsEnabled Function

Gets the EMAC test backpressure enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TestBackPressIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will assert backpressure on the link. Backpressure causes the preamble to be transmitted, raising the carrier sense. A transmit packet from the system will be sent during backpressure.
- false - Normal operation

Description

This function gets the EMAC test backpressure enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TestBackPressIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxPauseDisable Function

Disables the transmission of Pause frames.

File

plib_eth.h

C

```
void PLIB_ETH_TxPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the transmit Pause frames. The Pause frames are blocked from being transmitted.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_TxPauseEnable Function

Enables the transmission Pause frames.

File

plib_eth.h

C

```
void PLIB_ETH_TxPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the transmission Pause frames. The Pause frames are allowed for transmission.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_TxPauseIsEnabled Function

Gets the Ethernet module enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Pause Flow Control frames are allowed to be transmitted
- false - Pause Flow Control frames are blocked

Description

This function gets the Ethernet module enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
status = PLIB_ETH_TxPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxPauseIsEnabled(ETH_MODULE_ID index)

f) Interrupt Functions

PLIB_ETH_InterruptClear Function

Clears the Ethernet module interrupt source status as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptClear(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function clears the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not ORed with the others will be disabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To clear several interrupts with one command, execute the following:
PLIB_ETH_InterruptClear(MY_ETH_INSTANCE,
                                    ETH_EMPTY_WATERMARK_INTERRUPT      |
                                    ETH_FULL_WATERMARK_INTERRUPT       |
                                    ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptClear(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSet Function

Sets the Ethernet module interrupt source status as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function sets the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not OR'd with the others will be disabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To set several interrupt flags with one command, execute the following:
PLIB_ETH_InterruptSet(MY_ETH_INSTANCE,
                                    ETH_EMPTY_WATERMARK_INTERRUPT      |
                                    ETH_FULL_WATERMARK_INTERRUPT       |
                                    ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptsGet Function

Gets the Ethernet module Interrupt Flag register as a group.

File

plib_eth.h

C

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptsGet(ETH_MODULE_ID index);
```

Returns

Interrupt bit map, as defined in ETH_INTERRUPT_SOURCES, showing which interrupts have fired.

Description

Gets the Ethernet module Interrupt Flag register as a group.

Remarks

none.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptsGet(ETH_MODULE_ID index)

PLIB_ETH_InterruptSourceDisable Function

Clears the Ethernet module Interrupt Enable register as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function disables the Ethernet module interrupts by using a mask to select one or multiple interrupts to disable. Logically 'OR' elements of ETH_INTERRUPT_SOURCES together.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To disable some interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE,
                                ETH_TRANSMIT_DONE_INTERRUPT                |
                                ETH_TRANSMIT_ABORT_INTERRUPT               |
                                ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT |
                                ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT );

// Or to disable one interrupt with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);

// Or to disable all interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourceEnable Function

Sets the Ethernet module Interrupt Enable register in a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSourceEnable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function enables the Ethernet module interrupts by using a mask to enable one or multiple interrupt enables. Logically 'OR' elements of ETH_INTERRUPT_SOURCES together.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To enable some interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE,
                                    ETH_TRANSMIT_DONE_INTERRUPT                |
                                    ETH_TRANSMIT_ABORT_INTERRUPT               |
                                    ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT |
                                    ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT);

// Or to enable one interrupt with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);

// Or to enable all interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSourceEnable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourceIsEnabled Function

Gets the Ethernet module Interrupt Enable register singularly or as a group.

File

plib_eth.h

C

```
bool PLIB_ETH_InterruptSourceIsEnabled(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

- true - If any of the specified sources are enabled
- false - If none of the specified sources are enabled

Description

This function returns a true if any of the specified interrupt sources are enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_ETH_InterruptSourceIsEnabled(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    //If the interrupt is enabled, disable it...
    PLIB_ETH_InterruptSourceDisable(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
mask	The interrupt mask(s) to be checked

Function

bool PLIB_ETH_InterruptSourceIsEnabled(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourcesGet Function

Returns the entire interrupt enable register.

File

plib_eth.h

C

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptSourcesGet(ETH_MODULE_ID index);
```

Returns

Bit map of interrupt sources, all ORed together, as defined by ETH_INTERRUPT_SOURCES.

Description

This function returns the entire interrupt enable register.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptSourcesGet(ETH_MODULE_ID index)

PLIB_ETH_InterruptStatusGet Function

Gets the Ethernet module Interrupt Flag register as a group, using a mask.

File

plib_eth.h

C

```
bool PLIB_ETH_InterruptStatusGet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

- true - If any of the specified source statuses are enabled

- false - If none of the specified source statuses are enabled

Description

This function gets the Ethernet module Interrupt Flag register using a mask.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if (PLIB_ETH_InterruptStatusGet(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    PLIB_ETH_InterruptClear(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

bool PLIB_ETH_InterruptStatusGet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);

g) Statistics Functions

PLIB_ETH_AlignErrorCountClear Function

Sets the count of Ethernet alignment errors to zero.

File

plib_eth.h

C

```
void PLIB_ETH_AlignErrorCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the count of Ethernet alignment errors to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AlignErrorCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	count of alignment errors

Function

void PLIB_ETH_AlignErrorCountClear(ETH_MODULE_ID index)

PLIB_ETH_AlignErrorCountGet Function

Gets the count of Ethernet alignment errors.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_AlignErrorCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of alignment errors

Description

This function gets the count of Ethernet alignment errors.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_AlignErrorCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_AlignErrorCountGet(ETH_MODULE_ID index)

PLIB_ETH_FCSErrorCountClear Function

Sets the value of the Ethernet frame check sequence error to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FCSErrorCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet frame check sequence error to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FCSErrorCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of FCS Errors

Function

void PLIB_ETH_FCSErrorCountClear(ETH_MODULE_ID index)

PLIB_ETH_FCSErrorCountGet Function

Gets the count of the frame check sequence error.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FCSErrorCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of FCS Errors

Description

This function gets the count of the frame check sequence error.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FCSErrorCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FCSErrorCountGet(ETH_MODULE_ID index)

PLIB_ETH_FramesRxdOkCountClear Function

Sets the value of the Ethernet received frames 'OK' count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FramesRxdOkCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet frames 'OK' count to zero.

Remarks

Changing the value of this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FramesRxdOkCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of frames received correctly

Function

void PLIB_ETH_FramesRxdOkCountClear(ETH_MODULE_ID index)

PLIB_ETH_FramesRxdOkCountGet Function

Gets the count of the frames frames received successfully.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FramesRxdOkCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of frames received correctly

Description

This function gets the count of the frames received successfully. Increment count for frames received successfully by the RX Filter. This count will not be incremented if there is a Frame Check Sequence (FCS) or Alignment error.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FramesRxdOkCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FramesRxdOkCountGet(ETH_MODULE_ID index)

PLIB_ETH_FramesTxdOkCountClear Function

Sets the count of Ethernet Control frames transmitted to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FramesTxdOkCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the count of Ethernet Control frames transmitted to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FramesTxdOkCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of control frames transmitted correctly

Function

void PLIB_ETH_FramesTxdOkCountClear(ETH_MODULE_ID index)

PLIB_ETH_FramesTxdOkCountGet Function

Gets the count of Ethernet Control Frames transmitted successfully.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FramesTxdOkCountGet(ETH_MODULE_ID index);
```

Returns

- count - count of control frames transmitted correctly

Description

This function gets the count of Ethernet Control Frames transmitted successfully.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FramesTxdOkCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FramesTxdOkCountGet(ETH_MODULE_ID index)

PLIB_ETH_MultipleCollisionCountClear Function

Sets the value of the Ethernet multiple collision frame count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_MultipleCollisionCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet multiple collision frame count to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MultipleCollisionCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of multiple collision frames

Function

void PLIB_ETH_MultipleCollisionCountClear(ETH_MODULE_ID index)

PLIB_ETH_MultipleCollisionCountGet Function

Gets the count of the frames transmitted successfully after there was more than one collision.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MultipleCollisionCountGet(ETH_MODULE_ID index);
```

Returns

- count - Count of multiple collision frames

Description

This function gets the count of the frames transmitted successfully after there was more than one collision.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_MultipleCollisionCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MultipleCollisionCountGet(ETH_MODULE_ID index)

PLIB_ETH_RxOverflowCountClear Function

Sets the value of the Ethernet receive overflow count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_RxOverflowCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet receive overflow count to zero. Increment counter for frames accepted by the RX filter and subsequently dropped due to internal receive error. This event also sets the receive overflow interrupt flag.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxOverflowCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxOverflowCountClear(ETH_MODULE_ID index)

PLIB_ETH_RxOverflowCountGet Function

Gets the count of the dropped Ethernet receive frames.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_RxOverflowCountGet(ETH_MODULE_ID index);
```

Returns

- count - uint16_t receiver overflow counts

Description

This function gets the count of the dropped Ethernet receive frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_RxOverflowCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_RxOverflowCountGet(ETH_MODULE_ID index)

PLIB_ETH_RxPacketCountGet Function

Gets the value of the receive packet buffer count.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxPacketCountGet(ETH_MODULE_ID index);
```

Returns

- RxPacketCount - Number of received packets in memory

Description

This function gets the value of the receive packet buffer count. When a packet has been successfully received, this value is incremented by hardware. Software decrements the counter after a packet has been read. Call PLIB_ETH_ReceiveBufferCountDecrement(ETH_MODULE_ID index) to decrement.

Remarks

Receive Packet Buffer Counter cannot be decremented below 0x00 Cleared when the Ethernet module is reset.

The Receive Packet Buffer Count is not set to '0' when the Ethernet module is turned OFF. This allows software to continue to utilize and decrement the count.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_RxPacketCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxPacketCountGet(ETH_MODULE_ID index)

PLIB_ETH_SingleCollisionCountClear Function

Sets the value of the Ethernet single collision frame count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_SingleCollisionCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet single collision frame count to zero.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SingleCollisionCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of single collision frames

Function

void PLIB_ETH_SingleCollisionCountClear(ETH_MODULE_ID index)

PLIB_ETH_SingleCollisionCountGet Function

Gets the count of the frames transmitted successfully on a second attempt.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_SingleCollisionCountGet(ETH_MODULE_ID index);
```

Returns

- count - Count of frames transmitted successfully on second attempt

Description

This function gets the count of the frames transmitted successfully on a second attempt.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_SingleCollisionCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_SingleCollisionCountGet(ETH_MODULE_ID index)

h) Feature Existence Functions

PLIB_ETH_ExistsAlignmentErrorCount Function

Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsAlignmentErrorCount(ETH_MODULE_ID index);
```

Returns

- true - The AlignmentErrorCount feature is supported on the device
- false - The AlignmentErrorCount feature is not supported on the device

Description

This function identifies whether the AlignmentErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_AlignErrorCountClear
- PLIB_ETH_AlignErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsAlignmentErrorCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsAutoFlowControl Function

Identifies whether the AutoFlowControl feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsAutoFlowControl(ETH_MODULE_ID index);
```

Returns

- true - The AutoFlowControl feature is supported on the device
- false - The AutoFlowControl feature is not supported on the device

Description

This function identifies whether the AutoFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_AutoFlowControlEnable
- PLIB_ETH_AutoFlowControlDisable
- PLIB_ETH_AutoFlowControlIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsAutoFlowControl(ETH_MODULE_ID index)

PLIB_ETH_ExistsCollisionCounts Function

Identifies whether the CollisionCounts feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsCollisionCounts(ETH_MODULE_ID index);
```

Returns

- true - The CollisionCounts feature is supported on the device
- false - The CollisionCounts feature is not supported on the device

Description

This function identifies whether the CollisionCounts feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_SingleCollisionCountClear
- PLIB_ETH_SingleCollisionCountGet
- PLIB_ETH_MultipleCollisionCountClear
- PLIB_ETH_MultipleCollisionCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsCollisionCounts(ETH_MODULE_ID index)

PLIB_ETH_ExistsCollisionWindow Function

Identifies whether the CollisionWindow feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsCollisionWindow(ETH_MODULE_ID index);
```

Returns

- true - The CollisionWindow feature is supported on the device
- false - The CollisionWindow feature is not supported on the device

Description

This function identifies whether the CollisionWindow feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_CollisionWindowGet
- PLIB_ETH_CollisionWindowSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsCollisionWindow(ETH_MODULE_ID index)

PLIB_ETH_ExistsEnable Function

Identifies whether the Enable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsEnable(ETH_MODULE_ID index);
```

Returns

- true - The Enable feature is supported on the device
- false - The Enable feature is not supported on the device

Description

This function identifies whether the Enable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_Enable
- PLIB_ETH_Disable
- PLIB_ETH_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsEnable(ETH_MODULE_ID index)

PLIB_ETH_ExistsEthernetControllerStatus Function

Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsEthernetControllerStatus(ETH_MODULE_ID index);
```

Returns

- true - The EthernetControllerStatus feature is supported on the device
- false - The EthernetControllerStatus feature is not supported on the device

Description

This function identifies whether the EthernetControllerStatus feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxPacketCountGet
- PLIB_ETH_EthernetIsBus

- PLIB_ETH_TransmitIsBusy
- PLIB_ETH_ReceiveIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsEthernetControllerStatus(ETH_MODULE_ID index)

PLIB_ETH_ExistsFCSErrorCount Function

Identifies whether the FCSErrorCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFCSErrorCount(ETH_MODULE_ID index);
```

Returns

- true - The FCSErrorCount feature is supported on the device
- false - The FCSErrorCount feature is not supported on the device

Description

This function identifies whether the FCSErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FCSErrorCountClear
- PLIB_ETH_FCSErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFCSErrorCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsFramesTransmittedOK Function

Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFramesTransmittedOK(ETH_MODULE_ID index);
```

Returns

- true - The FramesTransmittedOK feature is supported on the device

- false - The FramesTransmittedOK feature is not supported on the device

Description

This function identifies whether the FramesTransmittedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FramesTxdOkCountClear
- PLIB_ETH_FramesTxdOkCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFramesTransmittedOK(ETH_MODULE_ID index)

PLIB_ETH_ExistsFramexReceivedOK Function

Identifies whether the FramexReceivedOK feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFramexReceivedOK(ETH_MODULE_ID index);
```

Returns

- true - The FramexReceivedOK feature is supported on the device
- false - The FramexReceivedOK feature is not supported on the device

Description

This function identifies whether the FramexReceivedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FramesRxdOkCountClear
- PLIB_ETH_FramesRxdOkCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFramexReceivedOK(ETH_MODULE_ID index)

PLIB_ETH_ExistsHashTable Function

Identifies whether the HashTable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsHashTable(ETH_MODULE_ID index);
```

Returns

- true - The HashTable feature is supported on the device
- false - The HashTable feature is not supported on the device

Description

This function identifies whether the HashTable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_HashTableSet
- PLIB_ETH_HashTableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsHashTable(ETH_MODULE_ID index)

PLIB_ETH_ExistsInterPacketGaps Function

Identifies whether the InterPacketGaps feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsInterPacketGaps(ETH_MODULE_ID index);
```

Returns

- true - The InterPacketGaps feature is supported on the device
- false - The InterPacketGaps feature is not supported on the device

Description

This function identifies whether the InterPacketGaps feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_BackToBackIPGGet
- PLIB_ETH_BackToBackIPGSet
- PLIB_ETH_NonBackToBackIPG1Get
- PLIB_ETH_NonBackToBackIPG1Set
- PLIB_ETH_NonBackToBackIPG2Get
- PLIB_ETH_NonBackToBackIPG2Set

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsInterPacketGaps(ETH_MODULE_ID index)

PLIB_ETH_ExistsInterrupt Function

Identifies whether the Interrupt feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsInterrupt(ETH_MODULE_ID index);
```

Returns

- true - The Interrupt feature is supported on the device
- false - The Interrupt feature is not supported on the device

Description

This function identifies whether the Interrupt feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_InterruptSourceEnable
- PLIB_ETH_InterruptSourceDisable
- PLIB_ETH_InterruptSourceIsEnabled
- PLIB_ETH_InterruptSet
- PLIB_ETH_InterruptClear
- PLIB_ETH_InterruptStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsInterrupt(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Configuration Function

Identifies whether the MAC_Configuration feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Configuration(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Configuration feature is supported on the device
- false - The MAC_Configuration feature is not supported on the device

Description

This function identifies whether the MAC_Configuration feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_LoopbackEnable
- PLIB_ETH_LoopbackDisable
- PLIB_ETH_LoopbackIsEnabled

- PLIB_ETH_TxPauseEnable
- PLIB_ETH_TxPauseDisable
- PLIB_ETH_TxPauseIsEnabled
- PLIB_ETH_RxPauseEnable
- PLIB_ETH_RxPauseDisable
- PLIB_ETH_RxPauseIsEnabled
- PLIB_ETH_PassAllEnable
- PLIB_ETH_PassAllDisable
- PLIB_ETH_PassAllIsEnabled
- PLIB_ETH_ReceiveEnable
- PLIB_ETH_ReceiveDisable
- PLIB_ETH_ReceiveIsEnabled
- PLIB_ETH_ExcessDeferEnable
- PLIB_ETH_ExcessDeferDisable
- PLIB_ETH_ExcessDeferIsEnabled
- PLIB_ETH_BackPresNoBackoffEnable
- PLIB_ETH_BackPresNoBackoffDisable
- PLIB_ETH_BackPresNoBackoffIsEnabled
- PLIB_ETH_NoBackoffEnable
- PLIB_ETH_NoBackoffDisable
- PLIB_ETH_NoBackoffIsEnabled
- PLIB_ETH_LongPreambleEnable
- PLIB_ETH_LongPreambleDisable
- PLIB_ETH_LongPreambleIsEnabled
- PLIB_ETH_PurePreambleEnable
- PLIB_ETH_PurePreambleDisable
- PLIB_ETH_PurePreambleIsEnabled
- PLIB_ETH_AutoDetectPadGet
- PLIB_ETH_AutoDetectPadSet
- PLIB_ETH_AutoDetectPadClear
- PLIB_ETH_CRCEnable
- PLIB_ETH_CRCDisable
- PLIB_ETH_CRCIsEnabled
- PLIB_ETH_DelayedCRCEnable
- PLIB_ETH_DelayedCRCDisable
- PLIB_ETH_DelayedCRCIsEnabled
- PLIB_ETH_HugeFrameEnable
- PLIB_ETH_HugeFrameDisable
- PLIB_ETH_HugeFrameIsEnabled
- PLIB_ETH_FrameLengthCheckEnable
- PLIB_ETH_FrameLengthCheckDisable
- PLIB_ETH_FrameLengthCheckIsEnabled
- PLIB_ETH_FullDuplexEnable
- PLIB_ETH_FullDuplexDisable
- PLIB_ETH_FullDuplexIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Configuration(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Resets Function

Identifies whether the MAC_Resets feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Resets(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Resets feature is supported on the device
- false - The MAC_Resets feature is not supported on the device

Description

This function identifies whether the MAC_Resets feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIResetEnable
- PLIB_ETH_MIIResetDisable
- PLIB_ETH_MIIResetIsEnabled
- PLIB_ETH_SimResetEnable
- PLIB_ETH_SimResetDisable
- PLIB_ETH_SimResetIsEnabled
- PLIB_ETH_MCSRxResetEnable
- PLIB_ETH_MCSRxResetDisable
- PLIB_ETH_MCSRxResetIsEnabled
- PLIB_ETH_RxFuncResetEnable
- PLIB_ETH_RxFuncResetDisable
- PLIB_ETH_RxFuncResetIsEnabled
- PLIB_ETH_MCSTxResetEnable
- PLIB_ETH_MCSTxResetDisable
- PLIB_ETH_MCSTxResetIsEnabled
- PLIB_ETH_TxFuncResetEnable
- PLIB_ETH_TxFuncResetDisable
- PLIB_ETH_TxFuncResetIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Resets(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Testing Function

Identifies whether the MAC_Testing feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Testing(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Testing feature is supported on the device
- false - The MAC_Testing feature is not supported on the device

Description

This function identifies whether the MAC_Testing feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TestBackPressEnable
- PLIB_ETH_TestBackPressDisable
- PLIB_ETH_TestBackPressIsEnabled
- PLIB_ETH_TestPauseEnable
- PLIB_ETH_TestPauseDisable
- PLIB_ETH_TestPauseIsEnabled
- PLIB_ETH_ShortcutQuantaEnable
- PLIB_ETH_ShortcutQuantaDisable
- PLIB_ETH_ShortcutQuantaIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Testing(ETH_MODULE_ID index)

PLIB_ETH_ExistsManualFlowControl Function

Identifies whether the ManualFlowControl feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsManualFlowControl(ETH_MODULE_ID index);
```

Returns

- true - The ManualFlowControl feature is supported on the device
- false - The ManualFlowControl feature is not supported on the device

Description

This function identifies whether the ManualFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ManualFlowControlEnable
- PLIB_ETH_ManualFlowControlDisable
- PLIB_ETH_ManualFlowControlIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsManualFlowControl(ETH_MODULE_ID index)

PLIB_ETH_ExistsMaxFrameLength Function

Identifies whether the MaxFrameLength feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMaxFrameLength(ETH_MODULE_ID index);
```

Returns

- true - The MaxFrameLength feature is supported on the device
- false - The MaxFrameLength feature is not supported on the device

Description

This function identifies whether the MaxFrameLength feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MaxFrameLengthGet
- PLIB_ETH_MaxFrameLengthSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMaxFrameLength(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIM_Config Function

Identifies whether the MIIM_Config feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIM_Config(ETH_MODULE_ID index);
```

Returns

- true - The MIIM_Config feature is supported on the device
- false - The MIIM_Config feature is not supported on the device

Description

This function identifies whether the MIIM_Config feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMResetEnable
- PLIB_ETH_MIIMResetDisable
- PLIB_ETH_MIIMResetIsEnabled
- PLIB_ETH_MIIMClockGet
- PLIB_ETH_MIIMClockSet
- PLIB_ETH_MIIMNoPreEnable
- PLIB_ETH_MIIMNoPreDisable

- PLIB_ETH_MIIMNoPreIsEnabled
- PLIB_ETH_MIIMScanIncrEnable
- PLIB_ETH_MIIMScanIncrDisable
- PLIB_ETH_MIIMScanIncrIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIM_Config(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIM_Indicators Function

Identifies whether the MIIM_Indicators feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIM_Indicators(ETH_MODULE_ID index);
```

Returns

- true - The MIIM_Indicators feature is supported on the device
- false - The MIIM_Indicators feature is not supported on the device

Description

This function identifies whether the MIIM_Indicators feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_LinkHasFailed
- PLIB_ETH_DataNotValid
- PLIB_ETH_MIIMIsScanning
- PLIB_ETH_MIIMIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIM_Indicators(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMAddresses Function

Identifies whether the MIIMAddresses feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMAddresses(ETH_MODULE_ID index);
```

Returns

- true - The MIIMAddresses feature is supported on the device
- false - The MIIMAddresses feature is not supported on the device

Description

This function identifies whether the MIIMAddresses feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PHYAddressGet
- PLIB_ETH_PHYAddressSet
- PLIB_ETH_RegisterAddressGet
- PLIB_ETH_RegisterAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMAddresses(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMReadWrite Function

Identifies whether the MIIMReadWrite feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMReadWrite(ETH_MODULE_ID index);
```

Returns

- true - The MIIMReadWrite feature is supported on the device
- false - The MIIMReadWrite feature is not supported on the device

Description

This function identifies whether the MIIMReadWrite feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMReadStart
- PLIB_ETH_MIIMWriteStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMReadWrite(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMScanMode Function

Identifies whether the MIIMScanMode feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMScanMode(ETH_MODULE_ID index);
```

Returns

- true - The MIIMScanMode feature is supported on the device
- false - The MIIMScanMode feature is not supported on the device

Description

This function identifies whether the MIIMScanMode feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMScanModeEnable
- PLIB_ETH_MIIMScanModeDisable
- PLIB_ETH_MIIMScanModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMScanMode(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIWriteReadData Function

Identifies whether the MIIWriteReadData feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIWriteReadData(ETH_MODULE_ID index);
```

Returns

- true - The MIIWriteReadData feature is supported on the device
- false - The MIIWriteReadData feature is not supported on the device

Description

This function identifies whether the MIIWriteReadData feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMWriteDataSet
- PLIB_ETH_MIIMReadDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIWriteReadData(ETH_MODULE_ID index)

PLIB_ETH_ExistsPatternMatch Function

Identifies whether the PatternMatch feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsPatternMatch(ETH_MODULE_ID index);
```

Returns

- true - The PatternMatch feature is supported on the device
- false - The PatternMatch feature is not supported on the device

Description

This function identifies whether the PatternMatch feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PatternMatchSet
- PLIB_ETH_PatternMatchGet
- PLIB_ETH_PatternMatchChecksumSet
- PLIB_ETH_PatternMatchChecksumGet
- PLIB_ETH_PatternMatchOffsetSet
- PLIB_ETH_PatternMatchOffsetGet
- PLIB_ETH_PatternMatchModeSet
- PLIB_ETH_PatternMatchModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsPatternMatch(ETH_MODULE_ID index)

PLIB_ETH_ExistsPauseTimer Function

Identifies whether the PauseTimer feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsPauseTimer(ETH_MODULE_ID index);
```

Returns

- true - The PauseTimer feature is supported on the device
- false - The PauseTimer feature is not supported on the device

Description

This function identifies whether the PauseTimer feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PauseTimerSet
- PLIB_ETH_PauseTimerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsPauseTimer(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveBufferSize Function

Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveBufferSize(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveBufferSize feature is supported on the device
- false - The ReceiveBufferSize feature is not supported on the device

Description

This function identifies whether the ReceiveBufferSize feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReceiveBufferSizeGet
- PLIB_ETH_ReceiveBufferSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveBufferSize(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveFilters Function

Identifies whether the ReceiveFilters feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveFilters(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveFilters feature is supported on the device
- false - The ReceiveFilters feature is not supported on the device

Description

This function identifies whether the ReceiveFilters feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReceiveFilterEnable
- PLIB_ETH_ReceiveFilterDisable
- PLIB_ETH_ReceiveFilterIsEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveFilters(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveOverflowCount Function

Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveOverflowCount(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveOverflowCount feature is supported on the device
- false - The ReceiveOverflowCount feature is not supported on the device

Description

This function identifies whether the ReceiveOverflowCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxOverflowCountClear
- PLIB_ETH_RxOverflowCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveOverflowCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveWmarks Function

Identifies whether the ReceiveWmarks feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveWmarks(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveWmarks feature is supported on the device
- false - The ReceiveWmarks feature is not supported on the device

Description

This function identifies whether the ReceiveWmarks feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxFullWmarkSet
- PLIB_ETH_RxFullWmarkGet
- PLIB_ETH_RxEmptyWmarkSet
- PLIB_ETH_RxEmptyWmarkGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveWmarks(ETH_MODULE_ID index)

PLIB_ETH_ExistsRetransmissionMaximum Function

Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRetransmissionMaximum(ETH_MODULE_ID index);
```

Returns

- true - The RetransmissionMaximum feature is supported on the device
- false - The RetransmissionMaximum feature is not supported on the device

Description

This function identifies whether the RetransmissionMaximum feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReTxMaxGet
- PLIB_ETH_ReTxMaxSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRetransmissionMaximum(ETH_MODULE_ID index)

PLIB_ETH_ExistsRMII_Support Function

Identifies whether the RMII_Support feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRMII_Support(ETH_MODULE_ID index);
```

Returns

- true - The RMII_Support feature is supported on the device
- false - The RMII_Support feature is not supported on the device

Description

This function identifies whether the RMII_Support feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RMIIResetEnable
- PLIB_ETH_RMIIResetDisable
- PLIB_ETH_RMIIResetIsEnabled
- PLIB_ETH_RMIISpeedGet
- PLIB_ETH_RMIISpeedSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRMII_Support(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxBufferCountDecrement Function

Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxBufferCountDecrement(ETH_MODULE_ID index);
```

Returns

- true - The RxBufferCountDecrement feature is supported on the device
- false - The RxBufferCountDecrement feature is not supported on the device

Description

This function identifies whether the RxBufferCountDecrement feature is available on the Ethernet module. When this function returns true, this function is supported on the device:

- PLIB_ETH_RxBufferCountDecrement

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxBufferCountDecrement(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxEnable Function

Identifies whether the ReceiveEnable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxEnable(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveEnable feature is supported on the device
- false - The ReceiveEnable feature is not supported on the device

Description

This function identifies whether the ReceiveEnable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxEnable
- PLIB_ETH_RxDisable
- PLIB_ETH_RxIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxEnable(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxPacketDescriptorAddress Function

Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxPacketDescriptorAddress(ETH_MODULE_ID index);
```

Returns

- true - The RxPacketDescriptorAddress feature is supported on the device

- false - The RxPacketDescriptorAddress feature is not supported on the device

Description

This function identifies whether the RxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxPacketDescAddrSet
- PLIB_ETH_RxPacketDescAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxPacketDescriptorAddress(ETH_MODULE_ID index)

PLIB_ETH_ExistsStationAddress Function

Identifies whether the StationAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsStationAddress(ETH_MODULE_ID index);
```

Returns

- true - The StationAddress feature is supported on the device
- false - The StationAddress feature is not supported on the device

Description

This function identifies whether the StationAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_StationAddressGet
- PLIB_ETH_StationAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsStationAddress(ETH_MODULE_ID index)

PLIB_ETH_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsStopInIdle(ETH_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_StopInIdleEnable
- PLIB_ETH_StopInIdleDisable
- PLIB_ETH_StopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsStopInIdle(ETH_MODULE_ID index)

PLIB_ETH_ExistsTransmitRTS Function

Identifies whether the TransmitRTS feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsTransmitRTS(ETH_MODULE_ID index);
```

Returns

- true - The TransmitRTS feature is supported on the device
- false - The TransmitRTS feature is not supported on the device

Description

This function identifies whether the TransmitRTS feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TxRTSEnable
- PLIB_ETH_TxRTSDisable
- PLIB_ETH_TxRTSIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsTransmitRTS(ETH_MODULE_ID index)

PLIB_ETH_ExistsTxPacketDescriptorAddress Function

Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsTxPacketDescriptorAddress(ETH_MODULE_ID index);
```

Returns

- true - The TxPacketDescriptorAddress feature is supported on the device
- false - The TxPacketDescriptorAddress feature is not supported on the device

Description

This function identifies whether the TxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TxPacketDescAddrSet
- PLIB_ETH_TxPacketDescAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsTxPacketDescriptorAddress(ETH_MODULE_ID index)

i) Data Types and Constants

ETH_AUTOPAD_OPTION Enumeration

Lists the possible automatic padding configurations.

File

plib_eth.h

C

```
typedef enum {
  ETH_AUTOPAD_DISABLED_CHECK_CRC,
  ETH_AUTOPAD_TO_60BYTES_ADD_CRC,
  ETH_AUTOPAD_TO_64BYTES_ADD_CRC,
  ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC
} ETH_AUTOPAD_OPTION;
```

Description

AutoDetectPad Selection

This enumeration lists the possible automatic padding configurations.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_INTERRUPT_SOURCES Enumeration

Lists the possible values of ETH_INTERRUPT_SOURCES.

File

plib_eth.h

C

```
typedef enum {
  ETH_TRANSMIT_BVCI_BUS_ERROR_INTERRUPT,
  ETH_RECEIVE_BVCI_BUS_ERROR_INTERRUPT,
  ETH_EMPTY_WATERMARK_INTERRUPT,
  ETH_FULL_WATERMARK_INTERRUPT,
  ETH_RECEIVE_DONE_INTERRUPT,
  ETH_PACKET_PENDING_INTERRUPT,
  ETH_RECEIVE_ACTIVITY_INTERRUPT,
  ETH_TRANSMIT_DONE_INTERRUPT,
  ETH_TRANSMIT_ABORT_INTERRUPT,
  ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT,
  ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT,
  ETH_ALL_INTERRUPT_SOURCES
} ETH_INTERRUPT_SOURCES;
```

Description

ETH Module Interrupt Mask

This enumeration lists the possible values of ETH_INTERRUPT_SOURCES.

Remarks

See also PLIB_ETH_EVENTS in plib_eth_lib.h.

ETH_MIIM_CLK Enumeration

Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

File

plib_eth.h

C

```
typedef enum {
  ETH_MIIM_SYSCLK_DIV_BY_4,
  ETH_MIIM_SYSCLK_DIV_RSVD,
  ETH_MIIM_SYSCLK_DIV_BY_6,
  ETH_MIIM_SYSCLK_DIV_BY_8,
  ETH_MIIM_SYSCLK_DIV_BY_10,
  ETH_MIIM_SYSCLK_DIV_BY_14,
  ETH_MIIM_SYSCLK_DIV_BY_20,
  ETH_MIIM_SYSCLK_DIV_BY_28,
  ETH_MIIM_SYSCLK_DIV_BY_40
} ETH_MIIM_CLK;
```

Description

MII Clock Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_PATTERN_MATCH_MODE Enumeration

Lists the possible pattern match values.

File

plib_eth.h

C

```
typedef enum _ETH_PATTERN_MATCH_MODE_ {
  ETH_PATTERN_MATCH_DISABLED,
  ETH_PATTERN_MATCH_MODE_CHECKSUM_MATCH,
  ETH_PATTERN_MATCH_MODE_STATION_ADDR,
```

```
  ETH_PATTERN_MATCH_MODE_UNICAST_ADDR,
  ETH_PATTERN_MATCH_MODE_BROADCAST_ADDR,
  ETH_PATTERN_MATCH_MODE_HASH_MATCH,
  ETH_PATTERN_MATCH_MODE_MAGIC_PACKET
} ETH_PATTERN_MATCH_MODE;
```

Members

Members	Description
ETH_PATTERN_MATCH_DISABLED	Pattern Match is disabled; pattern match is always unsuccessful
ETH_PATTERN_MATCH_MODE_CHECKSUM_MATCH	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches)
ETH_PATTERN_MATCH_MODE_STATION_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Station Address)
ETH_PATTERN_MATCH_MODE_UNICAST_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Unicast Address)
ETH_PATTERN_MATCH_MODE_BROADCAST_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Broadcast Address)
ETH_PATTERN_MATCH_MODE_HASH_MATCH	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Hash Table Filter match)
ETH_PATTERN_MATCH_MODE_MAGIC_PACKET	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Packet = Magic Packet)

Description

Pattern Match Modes

This enumeration lists the pattern match mode values.

Remarks

In all pattern match enabled cases, it is the pattern match inversion XOR pattern match checksum AND .

Some states were omitted from the enumeration since they were duplicates.

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_RECEIVE_FILTER Enumeration

Lists the possible values of the receive filter.

File

plib_eth.h

C

```
typedef enum {
  ETH_HASH_FILTER,
  ETH_MAGIC_PACKET_FILTER,
  ETH_PATTERN_MATCH_INVERSION,
  ETH_CRC_ERROR_FILTER,
  ETH_CRC_OK_FILTER,
  ETH_RUNT_ERROR_FILTER,
  ETH_RUNT_ENABLE_FILTER,
  ETH_UNICAST_FILTER,
  ETH_NOT_ME_UNICAST_FILTER,
  ETH_MULTICAST_FILTER,
  ETH_BROADCAST_FILTER,
  ETH_ENABLE_ALL_FILTER
} ETH_RECEIVE_FILTER;
```

Members

Members	Description
ETH_HASH_FILTER	Enable: Hash Table Filtering.
ETH_MAGIC_PACKET_FILTER	Enable: Magic Packet Filtering.
ETH_PATTERN_MATCH_INVERSION	Enable: Pattern Match Checksum must not match to be accepted.
ETH_CRC_ERROR_FILTER	Enable: Received packet CRC must be invalid to be accepted.
ETH_CRC_OK_FILTER	Enable: Received packet CRC must be valid to be accepted.
ETH_RUNT_ERROR_FILTER	Enable: Received packet must be runt packet to be accepted.
ETH_RUNT_ENABLE_FILTER	Enable: Received packet must not be a runt packet to be accepted.

ETH_UNICAST_FILTER	Enable: Unicast Packets whose destination address matches the station address are accepted.
ETH_NOT_ME_UNICAST_FILTER	Enable: Unicast Packets whose destination address do NOT match the station address are accepted.
ETH_MULTICAST_FILTER	Enable: Multicast Address Packets are accepted.
ETH_BROADCAST_FILTER	Enable: Broadcast Address Packets are accepted.
ETH_ENABLE_ALL_FILTER	Enable: All of the above packets are accepted.

Description

ETH Module Receive Filter Mask

This enumeration lists the possible values of the receive filter.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_RMII_SPEED Enumeration

Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

File

plib_eth.h

C

```
typedef enum {
  ETH_RMII_10Mbps,
  ETH_RMII_100Mps
} ETH_RMII_SPEED;
```

Description

RMII Speed Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

Files

Files

Name	Description
plib_eth.h	Defines the Ethernet Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_eth.h

Defines the Ethernet Peripheral Library Interface.

Enumerations

	Name	Description
	_ETH_PATTERN_MATCH_MODE_	Lists the possible pattern match values.
	ETH_AUTOPAD_OPTION	Lists the possible automatic padding configurations.
	ETH_INTERRUPT_SOURCES	Lists the possible values of ETH_INTERRUPT_SOURCES.
	ETH_MIIM_CLK	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	ETH_PATTERN_MATCH_MODE	Lists the possible pattern match values.
	ETH_RECEIVE_FILTER	Lists the possible values of the receive filter.
	ETH_RMII_SPEED	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlIsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC backpressure/no back-off.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC backpressure/no back-off enable status.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.
	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.
	PLIB_ETH_ExistsFCSErrorCount	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramesTransmittedOK	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramexReceivedOK	Identifies whether the FramexReceivedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsHashTable	Identifies whether the HashTable feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterPacketGaps	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterrupt	Identifies whether the Interrupt feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Configuration	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Resets	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Testing	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	PLIB_ETH_ExistsManualFlowControl	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsMaxFrameLength	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Config	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Indicators	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMAddresses	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMReadWrite	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMScanMode	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIWriteReadData	Identifies whether the MIIWriteReadData feature exists on the Ethernet module.

	PLIB_ETH_ExistsPatternMatch	Identifies whether the PatternMatch feature exists on the Ethernet module.
	PLIB_ETH_ExistsPauseTimer	Identifies whether the PauseTimer feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveBufferSize	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveFilters	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveOverflowCount	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveWmarks	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.
	PLIB_ETH_ExistsRetransmissionMaximum	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	PLIB_ETH_ExistsRMII_Support	Identifies whether the RMII_Support feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxBufferCountDecrement	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.
	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.
	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.
	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlIsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.

	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MCSRxResetDisable	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetEnable	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetIsEnabled	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	PLIB_ETH_MCSTxResetDisable	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetEnable	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.
	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.
	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no back-off.
	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.
	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.
	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.

	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_ReceiveIsBusy	Gets the Ethernet receive logic busy status.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RMIISpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMIISpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.
	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.

	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.
	PLIB_ETH_TxPauseEnable	Enables the transmission Pause frames.
	PLIB_ETH_TxPauseIsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send.
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.

Description

Ethernet Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ethernet Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Ethernet module.

File Name

plib_eth.h

Company

Microchip Technology Inc.

GLCD Controller Peripheral Library

This section describes the GLCD Controller (GLCD Controller) Peripheral Library.

Introduction

This library provides a low-level abstraction of the GLCD Controller Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Graphics LCD (GLCD) controller is to directly interface with display glasses and to control pixels on a screen. The GLCD controller transfers display data from a memory device and formats it for a display device. It also provides accelerated on-the-fly rendering of vertical and horizontal lines, rectangles, and copying of a rectangular area between different locations on the screen.

After initialization, the GLCD controller will perform rendering through DMA, thereby off-loading the CPU.

The GLCD controller can support three rendering layers. Each layer supports the configurable stride and alpha blending feature. Each layer supports different input pixel formats such as RGBA8888, ARGB8888, RGB888, RGB565, RGBA5551, YUYV, RGB232, LUT8 and grayscale. These layers can be combined together into a single layer with a configurable output format. The output format can be RGB888, RGB666, RGB323, YUYV, Serial 3-beat (RGB), Serial 4-beat (RGBA), Two-phase 12-bit mode, and BT656. For the LUT8 input format, the Look-up Table (LUT) resides within the device and is configurable.

The device supports a programmable cursor and the cursor data can be set in the device memory. The polarity of signals HSYNC, VSYNC, DE, and PCLK is configurable.

Using the Library

This topic describes the basic architecture of the GLCD Controller Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_glcd.h`

The interface to the GLCD Controller Peripheral Library is defined in the `plib_glcd.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the GLCD Controller Peripheral Library must include `peripheral.h`.

Library File:

The GLCD Controller Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

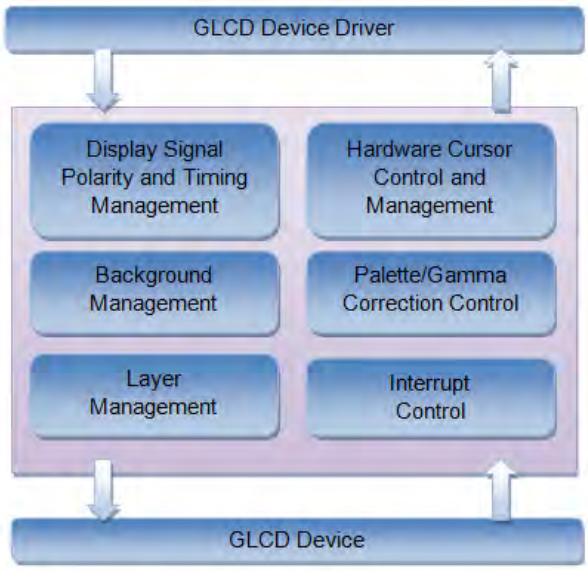
Hardware Abstraction Model

This library provides the low-level abstraction of the GLCD Controller module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The GLCD Peripheral Library software is depicted by the following block diagram:

GLCD Controller Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Deadman Timer module.

Library Interface Section	Description
General Configuration Functions	Functions that enable/disable the GLCD and set the RGB Sequential mode.
Display Signal Polarity and Timing Management Functions	Functions that handles polarity and timing of display signals such as VSYNC, HSYNC, DE, and PCLK.
Background Management Functions	Functions that handle setting the background color.
Layer Management	Functions that manage layer configuration and overlay.
Hardware Cursor Control and Management Functions	Functions that handle cursor data settings and cursor drawing position control.
Palette and Gamma Correction Control Functions	Functions that control hardware color palette/gamma table data settings.
Interrupt Control Functions	Functions that control device interrupt settings.
Miscellaneous Control Functions	Functions that control miscellaneous GLCD features.
Feature Existence Functions	Functions that determine existence of certain features.

How the Library Works

This section provides information on how the GLCD Controller Peripheral Library works.

General Configuration

Provides general configuration information.

Description

General configuration of the GLCD Controller includes enabling and disabling the GLCD module globally.

Enable/Disable GLCD

Enabling and disabling the GLCD are the basic routines that may be called during initialization and exit routines of the application.

Example:

```
// Enable the GLCD controller
PLIB_GLCD_Enable ( GLCD_ID_0 );
// Disable the GLCD
PLIB_GLCD_Disable ( GLCD_ID_0 );
// Check the enable status of the GLCD controller.
```

```
if(PLIB_GLCD_IsEnabled( GLCD_ID_0 ))
{
// application code
}
```

GLCD RGB Sequential Mode Set

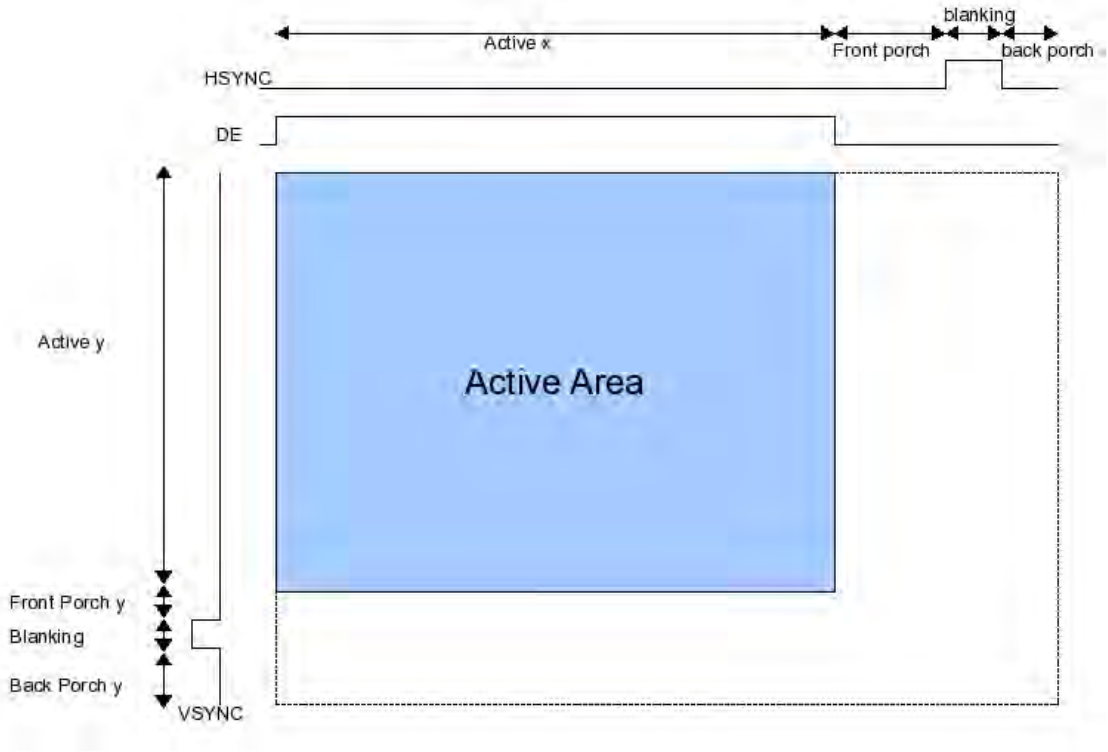
The GLCD RGB Sequential mode controls the GLCD output pins. The function used to set the RGB sequential mode is PLIB_GLCD_RGBSequentialModeSet. This basic function may be called during initialization and exit routines of the application.

Display Signal Polarity and Timing Management

Provides information on display signal polarity.

Description

A GLCD controller peripheral is connected the display using the following pins: HSYNC, VSYNC, GCLK, GEN and GD<23:0>. The HSYNC, VSYNC, GCLK and GEN polarity can be set using the PLIB_GLCD_SignalPolaritySet function. The signal polarity can be either positive or negative based on the display requirements. The GLCD source clock and pixel clock frequency can be controlled using the PLIB_GLCD_FormattingClockDivideEnable and PLIB_GLCD_ClockDividerSet functions. The timing of the signals, HSYNC, VSYNC, and GEN, is based on the display requirements and can be programmed to suit different display resolutions. The functions used to modify the HSYNC, VSYNC and GEN timing parameters are PLIB_GLCD_ResolutionXYSet, PLIB_GLCD_BlankingXYSet PLIB_GLCD_FrontPorchXYSet, and PLIB_GLCD_BackPorchXYSet.

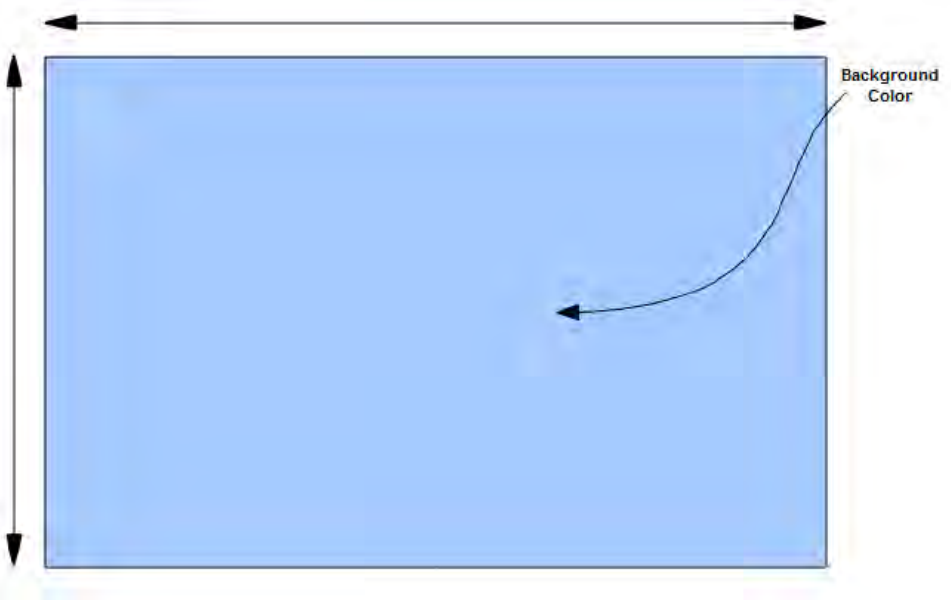


Display Background Management

Provides information on display background management.

Description

The GLCD Controller supports three layers and a background layer. A background layer is a basic layer that can be filled with one color of format RGBA8888 using the PLIB_GLCD_BackgroundColorSet function. The other layers will be applied on top of the background layer with a requested blending mode.

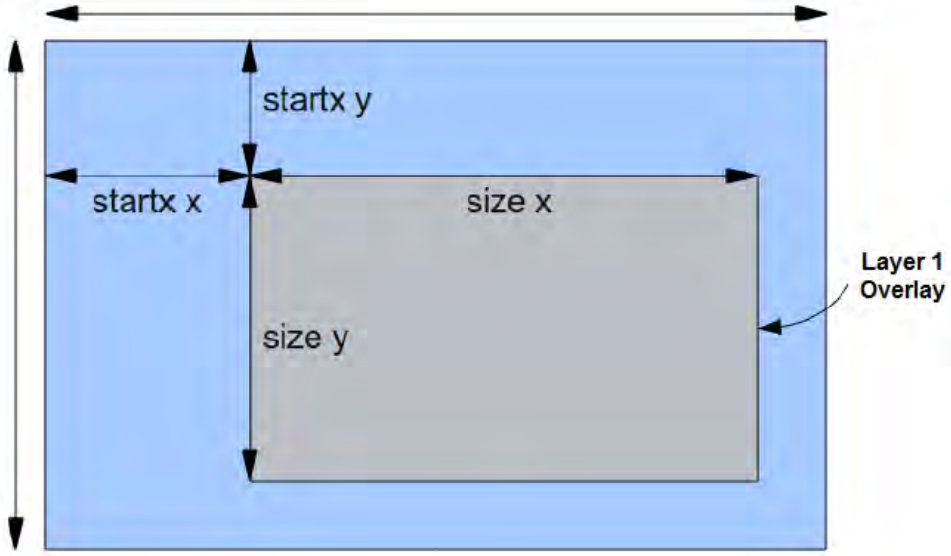


Layer Management

Provides information on layer management.

Description

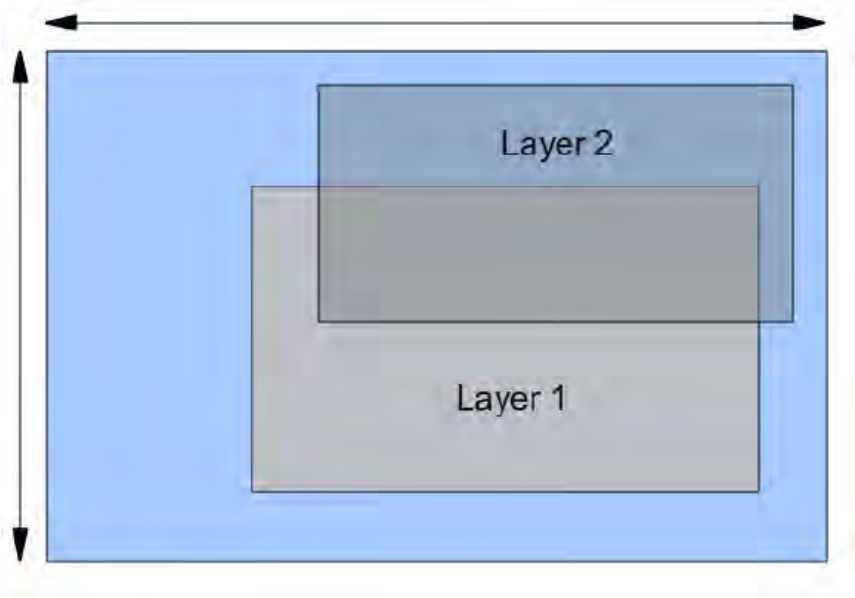
The GLCD Controller supports three drawing layers. Each layer can be enabled or disabled using the PLIB_GLCD_LayerEnable or PLIB_GLCD_LayerDisable functions, respectively. Each layer's start location, start (x, y), can be set using the PLIB_GLCD_LayerStartXYSet function and the size, size(x, y), can be set using the PLIB_GLCD_LayerSizeXYSet function.



Layer resolution and stride can be set using the PLIB_GLCD_LayerResXYSet and PLIB_GLCD_LayerStrideSet functions, respectively. A bi-linear filter can be applied to each layer using the PLIB_GLCD_LayerBilinearFilterEnable function.

Each layer supports different color formats and the color format can be set using the PLIB_GLCD_LayerColorModeSet function. Each Layer can have a different global alpha value, which can be set using the PLIB_GLCD_LayerGlobalAlphaSet function. Each Layer alpha can be premultiplied by global alpha using the PLIB_GLCD_LayerPreMultiplyImageAlphaEnable function; otherwise, the PLIB_GLCD_LayerPreMultiplyImageAlphaDisable function is used. To force global alpha value as the layer alpha the PLIB_GLCD_LayerForceWithGlobalAlphaEnable function is called; otherwise, the PLIB_GLCD_LayerForceWithGlobalAlphaDisble function is called.

The next layer is overlayed on a previous layer. Two layers can be blended based on source and destination layer blending functions. The source and destination blending functions can be set using the PLIB_GLCD_LayerSrcBlendFuncSet and PLIB_GLCD_LayerDestBlendFuncSet functions, respectively. The layer overlay and blending is not restricted by the layer color format. Two layers with a different color format can be blended together.



With the exception of the background layer, the other three layers have no memory space allocated on its own into GLCD device. The application must allocate GLCD accessible memory into the controller RAM and pass the address of the allocated memory to the GLCD. The address of the allocated memory for the layer can be set into the GLCD using the PLIB_GLCD_LayerBaseAddressSet function.

Hardware Cursor Control and Management

Provides information on hardware cursor control and management.

Description

The GLCD Controller supports a fully programmable 32 x 32 16-color hardware cursor. The cursor can be enabled or disabled using the PLIB_GLCD_CursorEnable or PLIB_GLCD_CursorDisbale functions, respectively.

The position of the cursor is set using the PLIB_GLCD_CursorXYSet function. Each pixel value of the cursor bitmap is represented by 4 bits. The first 32-bit word contains 8 pixels of the cursor bitmap data starting from pixel position (0, 7) to pixel (0, 0). The remainder of the cursor bitmap data continues similarly to the first 32-bit word. The complete cursor bitmap data is represented by 128 x 32 bit words.

This cursor data is set using the PLIB_GLCD_CursorDataSet function. The 4-bit value representing the pixel in the cursor bitmap data is used as the index of the cursor Color Look-up Table (CLUT). The cursor CLUT is set using the PLIB_GLCD_CursorLUTSet function. The color format of the cursor CLUT is XRGB, where 'X' = unimplemented. The size of cursor CLUT is 16 colors.

Palette and Gamma Correction Control

Provides information on palette/gamma correction control.

Description

For a layer with a source format set to 8-bit color palette look-up table (LUT8), the global CLUT can be set in the GLCD using the PLIB_GLCD_GlobalColorLUTSet function. The format of each color in the CLUT is XRGB8888, where 'X' = unimplemented. The size of the CLUT is 256 colors. The same global CLUT is used to map RGB values to new RGB values for the purpose of gamma correction. The PLIB_GLCD_PaletteGammaRampEnable or PLIB_GLCD_PaletteGammaRampDisable function can be used to enable or disable the Palette/Gamma correction feature.

Interrupt Control

Provides information interrupt control

Description

The GLCD Controller can trigger an interrupt on HSYNC and VSYNC. To enable or disable interrupt on HSYNC the PLIB_GLCD_HSyncInterruptEnable or PLIB_GLCD_HSyncInterruptDisable functions is used, respectively. To enable or disable interrupt on VSYNC the PLIB_GLCD_VSyncInterruptEnable or PLIB_GLCD_VSyncInterruptDisable function is used, respectively. The interrupt trigger can be set to either level or edge triggered using the PLIB_GLCD_IRQTriggerControlSet function.

Miscellaneous Control

Provides information on miscellaneous control features.

Description

The other GLCD feature controls that were not covered in the previous sections are: dithering, VSYNC for single cycle per line, force output blank, and YUV output.

The dithering feature can be enabled or disabled using the PLIB_GLCD_DitheringEnable or PLIB_GLCD_DitheringDisable functions, respectively.

VSYNC for single cycle per line is enabled or disabled using the PLIB_GLCD_SingleCyclePerLineVsyncEnable or PLIB_GLCD_SingleCyclePerLineVsyncDisable functions, respectively.

The forcing of GLCD output to blank is enabled or disabled using the PLIB_GLCD_ForceOutputBlankEnable or PLIB_GLCD_ForceOutputBlankDisable functions, respectively.

If the GLCD RGB Sequential mode is set to YUYU or BT656 format, the function, PLIB_GLCD_YUVOutputEnable, must be called; otherwise, the PLIB_GLCD_YUVOutputDisable function must be called.

The GLCD Peripheral Library also provides access to various GLCD status flags. To verify if the last row is currently displayed, the function, PLIB_GLCD_IsLastRowm is called. To determine the signal level of DE, HSYNC and VSYNC, the functions, PLIB_GLCD_DESignalLevelGet, PLIB_GLCD_HSyncSignalLevelGet, and PLIB_GLCD_VSyncSignalLevelGet are called, respectively. To know whether GLCD is in active vertical blanking state the PLIB_GLCD_IsVerticalBlankingActive function is called.

Configuring the Library

The library is configured for the supported GLCD Controller module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_GLCD_Disable	Disables the Graphics LCD Controller.
	PLIB_GLCD_Enable	Enables the Graphics LCD Controller.
	PLIB_GLCD_RGBSequentialModeSet	Sets the RGB output sequential mode.

b) Display Signal Polarity and Timing Management Functions

	Name	Description
	PLIB_GLCD_SignalPolarityGet	Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SignalPolaritySet	Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_BackPorchXGet	Gets X Axis Back Porch.
	PLIB_GLCD_BackPorchXYSet	Sets the back porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_BackPorchYGet	Gets Y Axis Back Porch.
	PLIB_GLCD_BlankingXYSet	Sets the blanking period on the x and y axis of the Graphics LCD Controller.
	PLIB_GLCD_FormattingClockDivideDisable	Disbale the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideEnable	Enable the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideIsEnabled	Verify whether Clock Formatting feature is enabled.
	PLIB_GLCD_FrontPorchXGet	Gets X Axis Front Porch.
	PLIB_GLCD_FrontPorchXYSet	Sets the front porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_FrontPorchYGet	Gets Y Axis Front Porch.
	PLIB_GLCD_LinesPrefetchGet	Gets value of lines to be prefetched.
	PLIB_GLCD_ResolutionXGet	Gets X Axis Resolution.
	PLIB_GLCD_ResolutionYGet	Gets Y Axis Resolution.
	PLIB_GLCD_BlankingXGet	Gets the X Axis Blanking Period.
	PLIB_GLCD_BlankingYGet	Gets the Y Axis Blanking Period.
	PLIB_GLCD_LinesPrefetchSet	Sets the clock controls of the Graphics LCD Controller.
	PLIB_GLCD_ClockDividerSet	Sets clock controls of the Graphics LCD Controller.
	PLIB_GLCD_ClockDividerGet	Gets Clock Divider value.
	PLIB_GLCD_ResolutionXYSet	Sets the display resolution for the Graphics LCD Controller.

c) Background Management Functions

	Name	Description
	PLIB_GLCD_BackgroundColorGet	Gets the value of the Background Color.

	PLIB_GLCD_BackgroundColorSet	Sets the background color.

d) Layer Management Functions

	Name	Description
	PLIB_GLCD_LayerBaseAddressGet	Gets the Layer Base Address.
	PLIB_GLCD_LayerBaseAddressSet	Sets the base address of layer surface of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterDisable	Disables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterEnable	Enables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterIsEnabled	Verify whether Layer Bilinear Filter is enabled.
	PLIB_GLCD_LayerColorModeGet	Gets the Layer Color Mode.
	PLIB_GLCD_LayerColorModeSet	Sets the layer color mode of the Graphics LCD Controller.
	PLIB_GLCD_LayerDestBlendFuncGet	Gets the Layer Destination Blend Function.
	PLIB_GLCD_LayerDestBlendFuncSet	Sets the layer destination blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerDisable	Disables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerEnable	Enables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerForceWithGlobalAlphaDisable	Disables the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaEnable	Enable the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled	Verify whether Layer Force with Global Alpha Feature is enabled.
	PLIB_GLCD_LayerGlobalAlphaGet	Gets the Layer Global Alpha value.
	PLIB_GLCD_LayerGlobalAlphaSet	Sets the layer global alpha of the Graphics LCD Controller.
	PLIB_GLCD_LayerIsEnabled	Verify whether Layer is enabled.
	PLIB_GLCD_LayerPreMultiplyImageAlphaDisable	Disable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaEnable	Enable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled	Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.
	PLIB_GLCD_LayerResXGet	Gets the Layer X Axis Resolution.
	PLIB_GLCD_LayerResXYSet	Sets the layer resolution in pixels.
	PLIB_GLCD_LayerResYGet	Gets the Layer Y Axis Resolution.
	PLIB_GLCD_LayerSizeXGet	Gets the Layer X Axis Size.
	PLIB_GLCD_LayerSizeXYSet	Sets the layer size x and size y of the Graphics LCD Controller.
	PLIB_GLCD_LayerSizeYGet	Gets the Layer Y Axis Size.
	PLIB_GLCD_LayerSrcBlendFuncGet	Gets the Layer Source Blend Function.
	PLIB_GLCD_LayerSrcBlendFuncSet	Sets the layer source blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartXGet	Gets the Layer X Axis Start Position.
	PLIB_GLCD_LayerStartXYSet	Sets the layer start x and start y of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartYGet	Gets the Layer Y Axis Start Position.
	PLIB_GLCD_LayerStrideGet	Gets the Layer Stride value.
	PLIB_GLCD_LayerStrideSet	Sets the layer surface stride of the Graphics LCD Controller.

e) Hardware Cursor Control and Management Functions

	Name	Description
	PLIB_GLCD_CursorDataGet	Gets the Cursor Data at given Index.
	PLIB_GLCD_CursorDataSet	Sets the cursor image data.
	PLIB_GLCD_CursorDisable	Disables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorEnable	Enables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorIsEnabled	Verifies whether the cursor is enabled.
	PLIB_GLCD_CursorLUTGet	Gets the color from Cursor LUT at given Index.
	PLIB_GLCD_CursorLUTSet	Sets the cursor color look-up table (LUT) in XRGB8888 format.
	PLIB_GLCD_CursorXGet	Gets the cursor X Axis Position.
	PLIB_GLCD_CursorXYSet	Sets the x and y coordinates of the Graphics LCD Controller cursor.
	PLIB_GLCD_CursorYGet	Gets the Cursor Y Axis Position.

f) Palette and Gamma Correction Control Functions

	Name	Description
	PLIB_GLCD_GlobalColorLUTGet	Gets the Color from Global Color LUT at given Index.
	PLIB_GLCD_GlobalColorLUTSet	Set Global Color LUT.

	PLIB_GLCD_PaletteGammaRampDisable	Disables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampEnable	Enables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampIsEnabled	Verify whether Palette / Gamma Ramp feature is enabled.

g) Interrupt Control Functions

	Name	Description
	PLIB_GLCD_HSyncInterruptDisable	Disables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptEnable	Enables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptIsEnabled	Verify whether HSYNC Interrupt is enabled.
	PLIB_GLCD_HSyncSignalLevelGet	Gets the Hsync signal level.
	PLIB_GLCD_IRQTriggerControlGet	Gets the IRQ Trigger Control value.
	PLIB_GLCD_IRQTriggerControlSet	Sets the IRQ trigger control.
	PLIB_GLCD_VSyncInterruptDisable	Disables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptEnable	Enables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptIsEnabled	Verify whether VSYNC Interrupt is enabled.
	PLIB_GLCD_VSyncSignalLevelGet	Gets the Vsync signal level.

h) Miscellaneous Control Functions

	Name	Description
	PLIB_GLCD_DitheringDisable	Disables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringEnable	Enables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_ForceOutputBlankDisable	Disable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankEnable	Enable Force Output Blank feature.
	PLIB_GLCD_SingleCyclePerLineVsyncDisable	Clears VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncEnable	Sets VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled	Verify whether VSYNC on Single Cycle Per Line feature is enabled.
	PLIB_GLCD_YUVOutputDisable	Disables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputEnable	Enables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_DESignalLevelGet	Gets the display enable signal level.
	PLIB_GLCD_DitheringIsEnabled	Verifies whether Dithering is enabled.
	PLIB_GLCD_ForceOutputBlankIsEnabled	Verify whether the Force Output Blank feature is enabled.
	PLIB_GLCD_IsEnabled	Verifies whether the Graphics LCD Controller is Enabled.
	PLIB_GLCD_IsLastRow	Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.
	PLIB_GLCD_IsVerticalBlankingActive	Get the active status.
	PLIB_GLCD_RGBSequentialModeGet	Get the RGB Sequential Mode already set.
	PLIB_GLCD_YUVOutputIsEnabled	Verify whether the YUV Output feature is enabled.

i) Feature Existence Functions

	Name	Description
	PLIB_GLCD_ExistsBackgroundColor	Verify whether the Background Color Feature is supported.
	PLIB_GLCD_ExistsBackPorchXY	Verify whether Back Porch Feature is supported.
	PLIB_GLCD_ExistsBlankingXY	Verify whether Blanking Period Feature is supported.
	PLIB_GLCD_ExistsClockDivider	Verify whether the Clock Divider feature is supported.
	PLIB_GLCD_ExistsCursor	Verifies whether the cursor feature exists.
	PLIB_GLCD_ExistsCursorData	
	PLIB_GLCD_ExistsCursorLUT	Verify whether Cursor LUT feature is supported.
	PLIB_GLCD_ExistsCursorXY	Verify whether Cursor XY Position feature is supported.
	PLIB_GLCD_ExistsDESignalLevel	Verify whether DE Signal Level feature is supported.
	PLIB_GLCD_ExistsDithering	Verifies whether the Dithering feature exists.
	PLIB_GLCD_ExistsEnable	Identifies whether the GLCD Enable feature exists on the GLCD module.
	PLIB_GLCD_ExistsForceOutputBlank	Verify whether Force Output Blank feature is supported.
	PLIB_GLCD_ExistsFormattingClockDivide	Verify whether Clock Formatting feature is available.
	PLIB_GLCD_ExistsFrontPorchXY	Verify whether Front Porch Feature is supported.
	PLIB_GLCD_ExistsGlobalColorLUT	Verify whether Global Color LUT feature is supported.

	PLIB_GLCD_ExistsHSyncInterruptEnable	Verify whether the HSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsHSyncSignalLevel	Verify whether HSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsIRQTriggerControl	Verify whether the IRQ Trigger Control feature is supported.
	PLIB_GLCD_ExistsIsLastRow	Verify whether Is Last Row Feature is supported.
	PLIB_GLCD_ExistsIsVerticalBlankingActive	Verify whether Is Vertical Blanking Active feature is supported.
	PLIB_GLCD_ExistsLayerBaseAddress	Verify whether the Layer Base Address feature is supported.
	PLIB_GLCD_ExistsLayerBilinearFilterEnable	Enable Layer Bilinear Filter Feature.
	PLIB_GLCD_ExistsLayerColorMode	Verify whether Layer Color Mode is supported.
	PLIB_GLCD_ExistsLayerDestBlendFunc	Verify whether Layer Destination Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerEnable	Verify whether Layer Enable Feature is supported.
	PLIB_GLCD_ExistsLayerForceWithGlobalAlpha	Verify whether Layer Force with Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerGlobalAlpha	Verify whether Layer Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha	Verify whether Layer Pre-Multiply Image Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerResXY	
	PLIB_GLCD_ExistsLayerSizeXY	Verify whether Layer X Axis and Y Axis Size feature is supported.
	PLIB_GLCD_ExistsLayerSrcBlendFunc	Verify whether Layer Source Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerStartXY	Verify whether Layer Start XY Position feature is supported.
	PLIB_GLCD_ExistsLayerStride	Verify whether the Layer Stride Feature is supported.
	PLIB_GLCD_ExistsLinesPrefetch	Verify whether Lines Prefetch Set Feature supported.
	PLIB_GLCD_ExistsPaletteGammaRamp	Verifies whether the palette / gamma ramp feature is supported.
	PLIB_GLCD_ExistsResolutionXY	Verify whether YUV Output feature is supported.
	PLIB_GLCD_ExistsRGBSequentialMode	Verify whether RGB Sequential Mode feature is supported.
	PLIB_GLCD_ExistsSignalPolarity	Verifies whether the Signal Polarity Selection feature exists.
	PLIB_GLCD_ExistsSingleCyclePerLineVsync	Verifies whether VSYNC on Single cycle Per Line Feature exists.
	PLIB_GLCD_ExistsVSyncInterruptEnable	Verify whether VSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsVSyncSignalLevel	Verify whether VSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsYUVOutput	Verify whether YUV output feature is supported.

j) Data Types and Constants

	Name	Description
	GLCD_IRQ_TRIGGER_CONTROL	Possible values of GLCD Interrupt Trigger Mode.
	GLCD_LAYER_COLOR_MODE	Possible values of GLCD color Mode.
	GLCD_LAYER_DEST_BLEND_FUNC	Possible values of GLCD Layer Destination Blend Functions.
	GLCD_LAYER_ID	Possible values of GLCD Layer Ids
	GLCD_LAYER_SRC_BLEND_FUNC	Possible values of GLCD Layer Source Blend Functions.
	GLCD_MODULE_ID	Possible instances of the GLCD module
	GLCD_RGB_MODE	GLCD RGB Sequential Mode
	GLCD_SIGNAL_POLARITY	Polarity values of different output signals.

Description

This section describes the Application Programming Interface (API) functions of the GLCD Controller Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_GLCD_Disable Function

Disables the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_Disable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_Disable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_Disable(GLCD_MODULE_ID index)

PLIB_GLCD_Enable Function

Enables the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_Enable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_Enable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_Enable(GLCD_MODULE_ID index)

PLIB_GLCD_RGBSequentialModeSet Function

Sets the RGB output sequential mode.

File

plib_glcd.h

C

```
void PLIB_GLCD_RGBSequentialModeSet(GLCD_MODULE_ID index, GLCD_RGB_MODE mode);
```

Returns

None.

Description

This function sets the RGB output sequential mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_RGBSequentialModeSet( GLCD_MODULE_ID_0,
                                GLCD_RGB_MODE_PARALLEL );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
mode	RGB sequential mode defined by GLCD_RGB_MODE.

Function

void PLIB_GLCD_RGBSequentialModeSet(GLCD_MODULE_ID index, GLCD_RGB_MODE mode)

b) Display Signal Polarity and Timing Management Functions

PLIB_GLCD_SignalPolarityGet Function

Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

File

plib_glcd.h

C

```
GLCD_SIGNAL_POLARITY PLIB_GLCD_SignalPolarityGet(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t signalPolarity = GLCD_POLARITY_POSITIVE;

signalPolarity = PLIB_GLCD_VSyncPolarityGet( GLCD_MODULE_ID_0 );

if( signalPolarity & GLCD_DE_POLARITY_NEGATIVE )
{
    // DE Polarity Negative
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_SIGNAL_POLARITY PLIB_GLCD_SignalPolarityGet(GLCD_MODULE_ID index)

PLIB_GLCD_SignalPolaritySet Function

Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_SignalPolaritySet(GLCD_MODULE_ID index, GLCD_SIGNAL_POLARITY polarity);
```

Returns

None.

Description

This function sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SignalPolaritySet( GLCD_MODULE_ID_0,
                             GLCD_POLARITY_POSITIVE |
                             GLCD_DE_POLARITY_NEGATIVE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
polarity	Enum variable specifying polarity.

Function

void PLIB_GLCD_SignalPolaritySet(GLCD_MODULE_ID index,
GLCD_SIGNAL_POLARITY polarity)

PLIB_GLCD_BackPorchXGet Function

Gets X Axis Back Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackPorchXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Back Porch.

Description

This function gets X Axis Back Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backPorchX;

backPorchX = PLIB_GLCD_BackPorchXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackPorchXGet(GLCD_MODULE_ID index)

PLIB_GLCD_BackPorchXYSet Function

Sets the back porch on the x and y axis for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_BackPorchXYSet(GLCD_MODULE_ID index, uint32_t backPorchX, uint32_t backPorchY);
```

Returns

None.

Description

This function sets the back porch on the x and y axis for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set back porch as: on x axis: 40 and on y axis: 20
PLIB_GLCD_BackPorchXYSet( GLCD_MODULE_ID_0, 40, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
backPorchX	Back porch pulse width on x axis.
backPorchY	Back porch pulse width on y axis.

Function

void PLIB_GLCD_BackPorchXYSet(GLCD_MODULE_ID index,
uint32_t backPorchX
uint32_t backPorchY)

PLIB_GLCD_BackPorchYGet Function

Gets Y Axis Back Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackPorchYGet(GLCD_MODULE_ID index);
```

Returns

- value of Y Axis Back Porch.

Description

This function gets Y Axis Back Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backPorchY;

backPorchY = PLIB_GLCD_BackPorchYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackPorchYGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingXYSet Function

Sets the blanking period on the x and y axis of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_BlankingXYSet(GLCD_MODULE_ID index, uint32_t blankingX, uint32_t blankingY);
```

Returns

None.

Description

This function sets the blanking period on the x and y axis of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set blanking period: on x axis: 10 and on y axis: 10
PLIB_GLCD_BlankingXY_Set( GLCD_MODULE_ID_0, 10, 10 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
blankingX	Blanking period on x axis.
blankingY	Blanking period on y axis.

Function

void PLIB_GLCD_BlankingXYSet(GLCD_MODULE_ID index,
uint32_t blankingX
uint32_t blankingY)

PLIB_GLCD_FormattingClockDivideDisable Function

Disbale the Clock Formatting feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_FormattingClockDivideDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disbales the Clock Formatting feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_FormattingClockDivideDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_FormattingClockDivideDisable(GLCD_MODULE_ID index)

PLIB_GLCD_FormattingClockDivideEnable Function

Enable the Clock Formatting feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_FormattingClockDivideEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function Enables the Clock Formatting feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_FormattingClockDivideEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_FormattingClockDivideEnable(GLCD_MODULE_ID index)

PLIB_GLCD_FormattingClockDivideIsEnabled Function

Verify whether Clock Formatting feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_FormattingClockDivideIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Clock Formatting feature is enabled.
- false - The Clock Formatting feature is disabled.

Description

This function verifies if the Clock Formatting feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_FormattingClockDivideIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_FormattingClockDivideDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_FormattingClockDivideIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_FrontPorchXGet Function

Gets X Axis Front Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_FrontPorchXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Front Porch.

Description

This Function gets X Axis Front Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t porchX;

porchX = PLIB_GLCD_FrontPorchXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_FrontPorchXGet(GLCD_MODULE_ID index)

PLIB_GLCD_FrontPorchXYSet Function

Sets the front porch on the x and y axis for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_FrontPorchXYSet(GLCD_MODULE_ID index, uint32_t frontPorchX, uint32_t frontPorchY);
```

Returns

None.

Description

This function sets the front porch on the x and y axis for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set front porch as: on x axis: 40 and on y axis: 20
PLIB_GLCD_FrontPorchXYSet( GLCD_MODULE_ID_0, 40, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
frontPorchX	Front porch pulse width on x axis.
frontPorchY	Front porch pulse width on y axis.

Function

void PLIB_GLCD_FrontPorchXYSet(GLCD_MODULE_ID index,
uint32_t frontPorchX,
uint32_t frontPorchY)

PLIB_GLCD_FrontPorchYGet Function

Gets Y Axis Front Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_FrontPorchYGet(GLCD_MODULE_ID index);
```

Returns

- value of y axis front porch.

Description

This function gets Y Axis Front Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t porchY;

porchY = PLIB_GLCD_FrontPorchYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_FrontPorchYGet(GLCD_MODULE_ID index)

PLIB_GLCD_LinesPrefetchGet Function

Gets value of lines to be prefetched.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LinesPrefetchGet(GLCD_MODULE_ID index);
```

Returns

- value of lines to be prefetched.

Description

This function gets the value of lines to be prefetched.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t linesPrefetch;

linesPrefetch = PLIB_GLCD_LinesPrefetchGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_LinesPrefetchGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionXGet Function

Gets X Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ResolutionXGet(GLCD_MODULE_ID index);
```

Returns

- value of x axis resolution.

Description

This function gets the X Axis resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionX;

resolutionX = PLIB_GLCD_ResolutionXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ResolutionXGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionYGet Function

Gets Y Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ResolutionYGet(GLCD_MODULE_ID index);
```

Returns

- Value of Y Axis Resolution.

Description

This function gets the Y Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionY;

resolutionY = PLIB_GLCD_ResolutionYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ResolutionYGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingXGet Function

Gets the X Axis Blanking Period.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BlankingXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Blanking Period.

Description

This function gets the X Axis Blanking Period.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t blankingX;

blankingX = PLIB_GLCD_BlankingXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BlankingXGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingYGet Function

Gets the Y Axis Blanking Period.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BlankingYGet(GLCD_MODULE_ID index);
```

Returns

- value of Y Axis Blanking Period.

Description

This function gets the Y Axis Blanking Period.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t blankingY;

blankingY = PLIB_GLCD_BlankingYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BlankingYGet(GLCD_MODULE_ID index)

PLIB_GLCD_LinesPrefetchSet Function

Sets the clock controls of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LinesPrefetchSet(GLCD_MODULE_ID index, uint32_t linesPrefetch);
```

Returns

None.

Description

This function sets the clock controls of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LinesPrefetchSet( GLCD_MODULE_ID_0,
                            MY_LINES_PREFETCH_VALUE );
```

Parameters

Parameters	Description
index	Identifier of the device instance
linesPrefetch	clock lines prefetch

Function

void PLIB_GLCD_LinesPrefetchSet(GLCD_MODULE_ID index, uint32_t linesPrefetch)

PLIB_GLCD_ClockDividerSet Function

Sets clock controls of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_ClockDividerSet(GLCD_MODULE_ID index, uint32_t clockDivider);
```

Returns

None.

Description

This function sets clock controls of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ClockDividerSet( GLCD_MODULE_ID_0, MY_CLOCK_DIVIDER_VALUE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
clockDivider	Factor dividing the GLCD clock.

Function

void PLIB_GLCD_ClockDividerSet(GLCD_MODULE_ID index, uint32_t clockDivider)

PLIB_GLCD_ClockDividerGet Function

Gets Clock Divider value.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ClockDividerGet(GLCD_MODULE_ID index);
```

Returns

- value of clock divider.

Description

This function gets the Clock Divider value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t dividerValue;

dividerValue = PLIB_GLCD_ClockDividerGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ClockDividerGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionXYSet Function

Sets the display resolution for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_ResolutionXYSet(GLCD_MODULE_ID index, uint32_t resolutionX, uint32_t resolutionY);
```

Returns

None.

Description

This function sets the display resolution for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set display resolution as: width: 640 pixels and height: 480 pixels
PLIB_GLCD_ResolutionXYSet( GLCD_MODULE_ID_0, 640, 480 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
resolutionX	Display resolution on x axis in terms of number of pixels.
resolutionY	Display resolution on y axis in terms of number of pixels.

Function

void PLIB_GLCD_ResolutionXYSet(GLCD_MODULE_ID index,
uint32_t resolutionX,
uint32_t resolutionY)

c) Background Management Functions

PLIB_GLCD_BackgroundColorGet Function

Gets the value of the Background Color.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackgroundColorGet(GLCD_MODULE_ID index);
```

Returns

- value of the Background Color.

Description

This function gets the value of the Background Color.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backgroundColor;

backgroundColor = PLIB_GLCD_BackgroundColorGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackgroundColorGet(GLCD_MODULE_ID index);

PLIB_GLCD_BackgroundColorSet Function

Sets the background color.

File

plib_glcd.h

C

```
void PLIB_GLCD_BackgroundColorSet(GLCD_MODULE_ID index, uint32_t bgColor);
```

Returns

None.

Description

This function sets the background color of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_BGColorSet( GLCD_MODULE_ID_0, 0x000000FF );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
bgColor	Background color.

Function

void ExistsBackgroundColor(GLCD_MODULE_ID index)

d) Layer Management Functions

PLIB_GLCD_LayerBaseAddressGet Function

Gets the Layer Base Address.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerBaseAddressGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer base address value.

Description

This function sets the Layer Base Address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t layerBaseAddress;

layerBaseAddress = PLIB_GLCD_LayerBaseAddressGet( GLCD_MODULE_ID_0
                                                  GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerBaseAddressGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBaseAddressSet Function

Sets the base address of layer surface of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBaseAddressSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t baseAddress);
```

Returns

None.

Description

This function sets the layer base address of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set layer 0 surface base address as 0x20000000
PLIB_GLCD_LayerBaseAddressSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 0x20000000 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
baseAddress	baseAddress of layer surface.

Function

void PLIB_GLCD_LayerBaseAddressSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint32_t baseAddress)

PLIB_GLCD_LayerBilinearFilterDisable Function

Disables the layer Bilinear filter of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBilinearFilterDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the layer Bilinear filter of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerBilinearFilterDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerBilinearFilterDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBilinearFilterEnable Function

Enables the layer Bilinear filter of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBilinearFilterEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the layer Bilinear filter of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerBilinearFilterEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerBilinearFilterEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBilinearFilterIsEnabled Function

Verify whether Layer Bilinear Filter is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerBilinearFilterIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Bilinear feature is enabled.
- false - The Layer Bilinear feature is disabled.

Description

This function verifies whether Layer Bilinear Filter is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerBilinearFilterIsEnabled( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerBilinearFilterDisable( GLCD_MODULE_ID_0,
                                          GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerBilinearFilterIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerColorModeGet Function

Gets the Layer Color Mode.

File

plib_glcd.h

C

```
GLCD_LAYER_COLOR_MODE PLIB_GLCD_LayerColorModeGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_COLOR_MODE_LUT8 - Layer Color Mode set to LUT8
- GLCD_LAYER_COLOR_MODE_RGBA5551 - Layer Color Mode set to RGBA5551
- GLCD_LAYER_COLOR_MODE_RGBA8888 - Layer Color Mode set to RGBA8888
- GLCD_LAYER_COLOR_MODE_RGB332 - Layer Color Mode set to RGB 332
- GLCD_LAYER_COLOR_MODE_RGB565 - Layer Color Mode set to RGB565
- GLCD_LAYER_COLOR_MODE_ARGB8888 - Layer Color Mode set to ARGB8888
- GLCD_LAYER_COLOR_MODE_L8 - Layer Color Mode set to L8
- GLCD_LAYER_COLOR_MODE_L1 - Layer Color Mode set to L1
- GLCD_LAYER_COLOR_MODE_L4 - Layer Color Mode set to L4
- GLCD_LAYER_COLOR_MODE_YUYV - Layer Color Mode set to YUYV
- GLCD_LAYER_COLOR_MODE_RGB888 - Layer Color Mode set to RGB888

Description

This function gets the Layer Color Mode

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t colorMode;

colorMode = PLIB_GLCD_LayerColorModeGet( GLCD_MODULE_ID_0
                                         GLCD_LAYER_0   );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
GLCD_LAYER_COLOR_MODE PLIB_GLCD_LayerColorModeGet( GLCD_MODULE_ID index,
                                                   GLCD_LAYER_ID  layerId );
```

PLIB_GLCD_LayerColorModeSet Function

Sets the layer color mode of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerColorModeSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, GLCD_LAYER_COLOR_MODE
colorMode);
```

Returns

None.

Description

This function sets the layer color mode of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set layer color mode to RGB232
PLIB_GLCD_LayerColorModeSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                             GLCD_LAYER_COLOR_MODE_RGB232 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
colorMode	Color mode defined by GLCD_LAYER_COLOR_MODE.

Function

```
void PLIB_GLCD_LayerColorModeSet( GLCD_MODULE_ID       index,
                                  GLCD_LAYER_ID        layerId,
                                  GLCD_LAYER_COLOR_MODE colorMode )
```

PLIB_GLCD_LayerDestBlendFuncGet Function

Gets the Layer Destination Blend Function.

File

plib_glcd.h

C

```
GLCD_LAYER_DEST_BLEND_FUNC PLIB_GLCD_LayerDestBlendFuncGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_DEST_BLEND_BLACK - Destination Blend function set to Black = 0.
- GLCD_LAYER_DEST_BLEND_WHITE - Destination Blend function set to White = 255.
- GLCD_LAYER_DEST_BLEND_ALPHA_SRC - Destination Blend function set to Source Alpha.
- GLCD_LAYER_DEST_BLEND_ALPHA_GBL - Destination Blend function set to Global Alpha.
- GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL - Destination Blend function set to (Source * Global Alpha)
- GLCD_LAYER_DEST_BLEND_INV_SRC - Destination Blend function set to (1 - Source Alpha)
- GLCD_LAYER_DEST_BLEND_INV_GBL - Destination Blend function set to (1 - Global Alpha)
- GLCD_LAYER_DEST_BLEND_INV_SRCGBL - Destination Blend function set to (1 - (Source Alpha * Global Alpha)).
- GLCD_LAYER_DEST_BLEND_ALPHA_DST - Destination Blend function set to Destination Alpha.
- GLCD_LAYER_DEST_BLEND_INV_DST - Destination Blend function set to (1 - Destination Alpha)

Description

This function gets the Layer Destination Blend Function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t destBlendFunc;

destBlendFunc = PLIB_GLCD_LayerDestBlendFuncGet( GLCD_MODULE_ID_0
                                                 GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
GLCD_LAYER_DEST_BLEND_FUNC PLIB_GLCD_LayerDestBlendFuncGet( GLCD_MODULE_ID index,
```

GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerDestBlendFuncSet Function

Sets the layer destination blend function of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerDestBlendFuncSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId,
GLCD_LAYER_DEST_BLEND_FUNC blendFunc);
```

Returns

None.

Description

This function sets the layer destination blend function of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerDestBlendFuncSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                                 GLCD_BLEND_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
blendFunc	Blend function defined by GLCD_LAYER_BLEND_FUNCTION.

Function

void PLIB_GLCD_LayerDestBlendFuncSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
GLCD_LAYER_DEST_BLEND_FUNC blendFunc)

PLIB_GLCD_LayerDisable Function

Disables the layer of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the layer of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerEnable Function

Enables the layer of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the layer of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaDisable Function

Disables the Layer Force Global Alpha feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerForceWithGlobalAlphaDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the Layer Force Global Alpha feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerForceWithGlobalAlphaDisable ( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerForceWithGlobalAlphaDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaEnable Function

Enable the Layer Force Global Alpha feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerForceWithGlobalAlphaEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the Layer Force Global Alpha feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerForceWithGlobalAlphaEnable ( GLCD_MODULE_ID_0,
                                            GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerForceWithGlobalAlphaEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled Function

Verify whether Layer Force with Global Alpha Feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Force Global Alpha feature is enabled.
- false - The Layer Force Global Alpha feature is disabled.

Description

This function verifies whether Layer Force with Global Alpha Feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled( GLCD_MODULE_ID_0,
                                                   GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerForceWithGlobalAlphaDisable( GLCD_MODULE_ID_0,
                                                GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerGlobalAlphaGet Function

Gets the Layer Global Alpha value.

File

plib_glcd.h

C

```
uint8_t PLIB_GLCD_LayerGlobalAlphaGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Alpha value

Description

This function gets the Layer Global Alpha value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint8_t layerAlpha;

layerAlpha = PLIB_GLCD_LayerGlobalAlphaGet( GLCD_MODULE_ID_0
                                            GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint8_t PLIB_GLCD_LayerGlobalAlphaGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerGlobalAlphaSet Function

Sets the layer global alpha of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerGlobalAlphaSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint8_t value);
```

Returns

None.

Description

This function sets the layer global alpha of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set global alpha value of layer 0 to 255
PLIB_GLCD_Layer_GlobalAlpha_Set( GLCD_MODULE_ID_0, GLCD_LAYER_0, 255 )
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
value	Global alpha value.

Function

void PLIB_GLCD_LayerGlobalAlphaSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint8_t value)

PLIB_GLCD_LayerIsEnabled Function

Verify whether Layer is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The given Layer is enabled.
- false - The given Layer is disabled.

Description

This function verifies whether a given Layer is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerIsEnabled( GLCD_MODULE_ID_0, GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_LayerIsEnabled (GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaDisable Function

Disable Layer Pre-Multiply Image Alpha Feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerPreMultiplyImageAlphaDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables Layer Pre-Multiply Image Alpha Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerPreMultiplyImageAlphaDisable( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerPreMultiplyImageAlphaDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaEnable Function

Enable Layer Pre-Multiply Image Alpha Feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerPreMultiplyImageAlphaEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the Layer Pre-Multiply Image Alpha Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerPreMultiplyImageAlphaEnable( GLCD_MODULE_ID_0,
                                            GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerPreMultiplyImageAlphaEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled Function

Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Pre-Multiply Image Alpha feature is enabled
- false - The Layer Pre-Multiply Image Alpha feature is disabled.

Description

This function verifies whether the Layer Pre-Multiply Image Alpha Feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled( GLCD_MODULE_ID_0,
                                                    GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerPreMultiplyImageAlphaDisable( GLCD_MODULE_ID_0,
                                                 GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerResXGet Function

Gets the Layer X Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerResXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of layer x axis resolution.

Description

This function gets the Layer X Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionX;

resolutionX = PLIB_GLCD_LayerResXGet( GLCD_MODULE_ID_0
                                      GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerResXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerResXYSet Function

Sets the layer resolution in pixels.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerResXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t resolutionX, uint32_t
resolutionY);
```

Returns

None.

Description

This function sets the layer resolution in pixels. The resolution is defined as the number of pixels on the x axis and the number of pixels on the y

axis.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Layer resolution set to x = 320 pixels and y = 240 pixels
PLIB_GLCD_Layer_ResXY_Set( GLCD_MODULE_ID_0, GLCD_LAYER_0, 320, 240);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
resolutionX	resolution on x axis in terms of pixels.
resolutionY	resolution on y axis in terms of pixels.

Function

void PLIB_GLCD_LayerResXYSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint32_t resolutionX,
uint32_t resolutionY)

PLIB_GLCD_LayerResYGet Function

Gets the Layer Y Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerResYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of layer y axis resolution.

Description

This function gets the Layer Y Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionY;

resolutionY = PLIB_GLCD_LayerResYGet( GLCD_MODULE_ID_0
                                      GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerResYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSizeXGet Function

Gets the Layer X Axis Size.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerSizeXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Size X value.

Description

This function gets the Layer X Axis Size.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t sizeX;

startX  = PLIB_GLCD_LayerSizeXGet( GLCD_MODULE_ID_0
                                   GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerSizeXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSizeXYSet Function

Sets the layer size x and size y of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerSizeXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t sizeX, uint32_t sizeY);
```

Returns

None.

Description

This function sets the layer start x and start y of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set layer size to (640, 480)
PLIB_GLCD_LayerSizeXYSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 640, 480);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
sizeX	Layer size on x axis in terms of pixels.
sizeY	Layer size on y axis in terms of pixels.

Function

void PLIB_GLCD_LayerSizeXYSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint32_t sizeX,
uint32_t sizeY)

PLIB_GLCD_LayerSizeYGet Function

Gets the Layer Y Axis Size.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerSizeYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Size Y value.

Description

This function gets the Layer Y Axis Size.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t sizeY;

sizeY   = PLIB_GLCD_LayerSizeYGet( GLCD_MODULE_ID_0
                                   GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerSizeYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSrcBlendFuncGet Function

Gets the Layer Source Blend Function.

File

plib_glcd.h

C

```
GLCD_LAYER_SRC_BLEND_FUNC PLIB_GLCD_LayerSrcBlendFuncGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_SRC_BLEND_BLACK - Source Blend Function set to Black = 0.
- GLCD_LAYER_SRC_BLEND_WHITE - Source Blend Function set to White = 255.
- GLCD_LAYER_SRC_BLEND_ALPHA_SRC - Source Blend Function set to Source Alpha.
- GLCD_LAYER_SRC_BLEND_ALPHA_GBL - Source Blend Function set to Global Alpha.
- GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL - Source Blend Function set to (Source Alpha * Global Alpha).
- GLCD_LAYER_SRC_BLEND_INV_SRC - Source Blend Function set to (1 - Source Alpha).
- GLCD_LAYER_SRC_BLEND_INV_GBL - Source Blend Function set to (1 - Global Alpha).
- GLCD_LAYER_SRC_BLEND_INV_SRCGBL - Source Blend Function set to (1 - (Source Alpha * Global Alpha))
- GLCD_LAYER_SRC_BLEND_ALPHA_DST - Source Blend Function set to Destination Alpha
- GLCD_LAYER_SRC_BLEND_INV_DST - Source Blend Function set to (1 - Destination Alpha)

Description

This function gets the Layer Source Blend Function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t srcBlendFunc;

srcBlendFunc = PLIB_GLCD_LayerSrcBlendFuncGet( GLCD_MODULE_ID_0
                                               GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

GLCD_LAYER_SRC_BLEND_FUNC PLIB_GLCD_LayerSrcBlendFuncGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSrcBlendFuncSet Function

Sets the layer source blend function of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerSrcBlendFuncSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, GLCD_LAYER_SRC_BLEND_FUNC
blendFunc);
```

Returns

None.

Description

This function sets the layer source blend function of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerSrcBlendFuncSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
```

```
                                    GLCD_BLEND_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
blendFunc	Blend function defined by GLCD_LAYER_BLEND_FUNCTION.

Function

```
void PLIB_GLCD_LayerSrcBlendFuncSet(  GLCD_MODULE_ID          index,
                                      GLCD_LAYER_ID           layerId,
                                      GLCD_LAYER_SRC_BLEND_FUNC  blendFunc )
```

PLIB_GLCD_LayerStartXGet Function

Gets the Layer X Axis Start Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStartXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Start X value.

Description

This function gets the Layer X Axis Start Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t startX;

startX  = PLIB_GLCD_LayerStartXGet( GLCD_MODULE_ID_0
                                    GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
uint32_t PLIB_GLCD_LayerStartXGet(  GLCD_MODULE_ID index,
                                    GLCD_LAYER_ID  layerId );
```

PLIB_GLCD_LayerStartXYSet Function

Sets the layer start x and start y of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerStartXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t startX, uint32_t
startY);
```

Returns

None.

Description

This function sets the layer start x and start y of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None

Example

```
// set layer start to (20, 20)
PLIB_GLCD_LayerStartXYSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 20, 20);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
startX	Layer start on x axis in terms of pixels.
startY	Layer start on y axis in terms of pixels.

Function

```
void PLIB_GLCD_LayerStartXYSet(    GLCD_MODULE_ID index,
                                   GLCD_LAYER_ID  layerId,
uint32_t      startX,
uint32_t      startY )
```

PLIB_GLCD_LayerStartYGet Function

Gets the Layer Y Axis Start Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStartYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Start Y value.

Description

This function gets the Layer Y Axis Start Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t startY;

startY  = PLIB_GLCD_LayerStartYGet( GLCD_MODULE_ID_0
                                    GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerStartYGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId);

PLIB_GLCD_LayerStrideGet Function

Gets the Layer Stride value.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStrideGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of the stride.

Description

This function gets the Layer Stride value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t layerStride;

layerStride = PLIB_GLCD_LayerStrideGet( GLCD_MODULE_ID_0
                                        GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerStrideGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerStrideSet Function

Sets the layer surface stride of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerStrideSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t stride);
```

Returns

None.

Description

This function sets the layer surface stride of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t imageStride = IMAGE_WIDTH * IMAGE_BYTES_PER_PIXEL;

// To make stride 4 byte aligned
imageStride += ( imageStride % 4 );

PLIB_GLCD_LayerStrideSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                          imageStride );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
stride	line width in bytes including padding.

Function

void PLIB_GLCD_LayerStrideSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint32_t stride)

e) Hardware Cursor Control and Management Functions

PLIB_GLCD_CursorDataGet Function

Gets the Cursor Data at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorDataGet(GLCD_MODULE_ID index, uint32_t dataIndex);
```

Returns

- value of the 8 pixels of the cursor at given index.

Description

This function gets the Cursor Data at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorData;

cursorData = PLIB_GLCD_CursorDataGet( GLCD_MODULE_ID_0,
                                      MY_CURSOR_DATA_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
dataIndex	Index of the Cursor data.

Function

uint32_t PLIB_GLCD_CursorDataGet(GLCD_MODULE_ID index, uint32_t dataIndex)

PLIB_GLCD_CursorDataSet Function

Sets the cursor image data.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorDataSet(GLCD_MODULE_ID index, uint32_t * cursorData);
```

Returns

None.

Description

This function sets the cursor image data. The image data format is 4-bit black, white and 50 percent gray color.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t *cursorData = cursorDataAddress;

// Set cursor data in the format: 4-bit CLUT Index
PLIB_GLCD_CursorDataSet( GLCD_MODULE_ID_0, cursorData );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorData	pointer to cursor data in the format 4-bit CLUT Index.

Function

void PLIB_GLCD_CursorDataSet(GLCD_MODULE_ID index, uint32_t * cursorData)

PLIB_GLCD_CursorDisable Function

Disables the cursor of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the cursor of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_CursorDisable(GLCD_MODULE_ID index)

PLIB_GLCD_CursorEnable Function

Enables the cursor of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the cursor of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_CursorEnable(GLCD_MODULE_ID index)

PLIB_GLCD_CursorIsEnabled Function

Verifies whether the cursor is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_CursorIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - Cursor is enabled.
- false - Cursor is disabled.

Description

This function verifies whether the cursor is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_CursorIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_CursorIsEnabled (GLCD_MODULE_ID index)

PLIB_GLCD_CursorLUTGet Function

Gets the color from Cursor LUT at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex);
```

Returns

- value of color from color LUT at given index.

Description

This function gets the color from Cursor LUT at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorLUTColor;

cursorLUTColor = PLIB_GLCD_CursorLUTGet( GLCD_MODULE_ID_0,
                                         MY_CURSOR_CLUT_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
lutIndex	Cursor Color LUT index.

Function

uint32_t PLIB_GLCD_CursorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex)

PLIB_GLCD_CursorLUTSet Function

Sets the cursor color look-up table (LUT) in XRGB8888 format.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorLUTSet(GLCD_MODULE_ID index, uint32_t * cursorLUT);
```

Returns

None.

Description

This function sets the cursor color LUT of the Graphics LCD Controller in XRGB8888 format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t * cursorLUTColors = cursorLUTAddress;

PLIB_GLCD_CursorLUTSet( GLCD_MODULE_ID_0, cursorLUTColors );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorLUT	Pointer to the cursor color LUT in XRGB8888 format.

Function

void PLIB_GLCD_CursorLUTSet(GLCD_MODULE_ID index, uint32_t * cursorLUT)

PLIB_GLCD_CursorXGet Function

Gets the cursor X Axis Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorXGet(GLCD_MODULE_ID index);
```

Returns

- Cursor X Axis Position.

Description

This function gets the cursor X Axis Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorX;

cursorX = PLIB_GLCD_CursorXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_CursorXGet(GLCD_MODULE_ID index)

PLIB_GLCD_CursorXYSet Function

Sets the x and y coordinates of the Graphics LCD Controller cursor.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorXYSet(GLCD_MODULE_ID index, uint32_t cursorX, uint32_t cursorY);
```

Returns

None.

Description

This function sets the x and y coordinates of the Graphics Display Controller cursor.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// update cursor position at x:20 pixels and y:20 pixels
PLIB_GLCD_CursorXYSet( GLCD_MODULE_ID_0, 20, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorX	cursor coordinate on x axis in terms of pixels.
cursorY	cursor coordinate on y axis in terms of pixels.

Function

void PLIB_GLCD_CursorXYSet(GLCD_MODULE_ID index,
uint32_t cursorX,
uint32_t cursorY)

PLIB_GLCD_CursorYGet Function

Gets the Cursor Y Axis Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorYGet(GLCD_MODULE_ID index);
```

Returns

- Cursor Y Axis Position.

Description

This function gets the Cursor Y Axis Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorY;

cursorY = PLIB_GLCD_CursorYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_CursorYGet(GLCD_MODULE_ID index)

f) Palette and Gamma Correction Control Functions

PLIB_GLCD_GlobalColorLUTGet Function

Gets the Color from Global Color LUT at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_GlobalColorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex);
```

Returns

- value of Color from the Color Lookup Table.

Description

This function gets the Color from Global Color LUT at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t lutColor;

lutColor = PLIB_GLCD_GlobalColorLUTGet( GLCD_MODULE_ID_0,
                                        MY_LUT_COLOR_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
lutIndex	Color Lookup table index.

Function

uint32_t PLIB_GLCD_GlobalColorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex)

PLIB_GLCD_GlobalColorLUTSet Function

Set Global Color LUT.

File

plib_glcd.h

C

```
void PLIB_GLCD_GlobalColorLUTSet(GLCD_MODULE_ID index, uint32_t * globalLUT);
```

Returns

None.

Description

This function sets the Global Color LUT.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t globalColorLUT[256] = { // Initial values
                               };
PLIB_GLCD_GlobalColorLUTSet( GLCD_MODULE_ID_0, globalColorLUT );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
globalLUT	pointer of the Global Color Lookup table data array.

Function

void PLIB_GLCD_GlobalColorLUTSet(GLCD_MODULE_ID index, uint32_t *globalLUT)

PLIB_GLCD_PaletteGammaRampDisable Function

Disables the palette / gamma ramp of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_PaletteGammaRampDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the palette / gamma ramp of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_PaletteGammaRampDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_PaletteGammaRampDisable(GLCD_MODULE_ID index)

PLIB_GLCD_PaletteGammaRampEnable Function

Enables the palette / gamma ramp of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_PaletteGammaRampEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the palette / gamma ramp of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_PaletteGammaRampEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_PaletteGammaRampEnable(GLCD_MODULE_ID index)

PLIB_GLCD_PaletteGammaRampIsEnabled Function

Verify whether Palette / Gamma Ramp feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_PaletteGammaRampIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The palette / gamma ramp feature is enabled.
- false - The palette / gamma ramp feature is disabled.

Description

This function verifies whether Palette / Gamma Ramp feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_PaletteGammaRampIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_PaletteGammaRampDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_PaletteGammaRampIsEnabled(GLCD_MODULE_ID index)

g) Interrupt Control Functions

PLIB_GLCD_HSyncInterruptDisable Function

Disables interrupts at Hsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_HSyncInterruptDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables interrupts at Hsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_HsyncInterruptDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_HSyncInterruptDisable(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncInterruptEnable Function

Enables interrupts at Hsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_HSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables interrupts at Hsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_HSyncInterruptEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_HSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncInterruptIsEnabled Function

Verify whether HSYNC Interrupt is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_HSyncInterruptIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Interrupt feature is enabled.
- false - The HSYNC Interrupt feature is disabled.

Description

This function verifies whether HSYNC Interrupt is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_HSyncInterruptIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_HSyncInterruptDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_HSyncInterruptIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncSignalLevelGet Function

Gets the Hsync signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_HSyncSignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The Hsync signal level is high.

- false - The Hsync signal level is low.

Description

This function returns the Hsync signal level.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_HSyncSignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //Hsync signal level is low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_HSyncSignalLevelGet(GLCD_MODULE_ID index)

PLIB_GLCD_IRQTriggerControlGet Function

Gets the IRQ Trigger Control value.

File

plib_glcd.h

C

```
GLCD_IRQ_TRIGGER_CONTROL PLIB_GLCD_IRQTriggerControlGet(GLCD_MODULE_ID index);
```

Returns

- GLCD_IRQ_TRIGGER_LEVEL - IRQ is Level Triggered.
- GLCD_IRQ_TRIGGER_EDGE - IRQ is edge Triggered.

Description

This function gets the IRQ Trigger Control value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t irqTrigger;

irqTrigger = PLIB_GLCD_IRQTriggerControlGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_IRQ_TRIGGER_CONTROL PLIB_GLCD_IRQTriggerControlGet(GLCD_MODULE_ID index)

PLIB_GLCD_IRQTriggerControlSet Function

Sets the IRQ trigger control.

File

plib_glcd.h

C

```
void PLIB_GLCD_IRQTriggerControlSet(GLCD_MODULE_ID index, GLCD_IRQ_TRIGGER_CONTROL irqControl);
```

Returns

None.

Description

This function sets the IRQ trigger control. IRQ trigger detection is defined at the edge or level of the trigger pulse.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set trigger detection at edge
PLIB_GLCD_IRQTriggerControlSet( GLCD_MODULE_ID_0, GLCD_IRQ_TRIGGER_EDGE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
control	Trigger control defined by GLCD_IRQ_TRIGGER_CONTROL.

Function

void PLIB_GLCD_IRQTriggerControlSet(GLCD_MODULE_ID index,
GLCD_IRQ_TRIGGER_CONTROL irqControl)

PLIB_GLCD_VSyncInterruptDisable Function

Disables interrupts at Vsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_VSyncInterruptDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables interrupts at Vsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_VsyncInterruptDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_VSyncInterruptDisable(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncInterruptEnable Function

Enables interrupts at Vsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_VSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables interrupts at Vsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None

Example

```
PLIB_GLCD_VSyncInterruptEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_VSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncInterruptIsEnabled Function

Verify whether VSYNC Interrupt is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_VSyncInterruptIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Interrupt feature is enabled.
- false - The VSYNC Interrupt feature is disabled.

Description

This function verifies whether the VSYNC Interrupt is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_VSyncInterruptIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_VSyncInterruptDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_VSyncInterruptIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncSignalLevelGet Function

Gets the Vsync signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_VSyncSignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The Vsync signal level is high.
- false - The Vsync signal level is low.

Description

This function returns the Vsync signal level.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_VSyncSignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //Vsync signal level is low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_VSyncSignalLevelGet(GLCD_MODULE_ID index)

h) Miscellaneous Control Functions

PLIB_GLCD_DitheringDisable Function

Disables the Dithering feature of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_DitheringDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Dithering feature of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_DitheringDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_DitheringDisable(GLCD_MODULE_ID index)

PLIB_GLCD_DitheringEnable Function

Enables the Dithering feature of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_DitheringEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Dithering feature of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_DitheringEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_DitheringEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankDisable Function

Disable Force Output Blank feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_ForceOutputBlankDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Force Output Blank feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ForceOutputBlankDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_ForceOutputBlankDisable(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankEnable Function

Enable Force Output Blank feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_ForceOutputBlankEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Force Output Blank feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ForceOutputBlankEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_ForceOutputBlankEnable(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncDisable Function

Clears VSYNC on single cycle per line.

File

plib_glcd.h

C

```
void PLIB_GLCD_SingleCyclePerLineVsyncDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function clears VSYNC on single cycle per line.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SingleCyclePerLineVsyncClear( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_SingleCyclePerLineVsyncClear(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncEnable Function

Sets VSYNC on single cycle per line.

File

plib_glcd.h

C

```
void PLIB_GLCD_SingleCyclePerLineVsyncEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function sets VSYNC on single cycle per line.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SingleCyclePerLineVsyncEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_SingleCyclePerLineVsyncSet(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled Function

Verify whether VSYNC on Single Cycle Per Line feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - VSYNC on Single Cycle Per Line is enabled.
- false - VSYNC on Single Cycle Per Line is disabled.

Description

This function verifies whether VSYNC on Single Cycle Per Line feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID_0))
{
    PLIB_GLCD_SingleCyclePerLineVsyncDisable(GLCD_MODULE_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputDisable Function

Disables the output of the Graphics LCD Controller in YUV format.

File

plib_glcd.h

C

```
void PLIB_GLCD_YUVOutputDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function Disables the output of the Graphics LCD Controller in YUV format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_YUVOutputDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_YUVOutputDisable(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputEnable Function

Enables the output of the Graphics LCD Controller in YUV format.

File

plib_glcd.h

C

```
void PLIB_GLCD_YUVOutputEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the output of the Graphics LCD Controller in YUV format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_YUVOutputEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_YUVOutputEnable(GLCD_MODULE_ID index)

PLIB_GLCD_DESignalLevelGet Function

Gets the display enable signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_DESignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The DE signal level high.
- false - The DE signal level low.

Description

This function returns the display enable (DE) signal of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_DESignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //DE signal low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_DESignalLevelGet(GLCD_MODULE_ID index)

PLIB_GLCD_DitheringIsEnabled Function

Verifies whether Dithering is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Dithering feature is enabled.
- false - The Dithering feature is disabled.

Description

This function verifies whether Dithering is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID_0 ))
{
    PLIB_GLCD_DitheringDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankIsEnabled Function

Verify whether the Force Output Blank feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ForceOutputBlankIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Force Output Blank feature is enabled.
- false - The Force Output Blank feature is disabled.

Description

This function verifies whether the Force Output Blank feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ForceOutputBlankIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ForceOutputBlankDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ForceOutputBlankIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_IsEnabled Function

Verifies whether the Graphics LCD Controller is Enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - Graphics LCD Controller is Enabled
- false - Graphics LCD Controller is Disabled

Description

This function verifies whether the Graphics LCD Controller is Enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_IsEnabled( GLCD_MODULE_ID_0 ) )
```

```
{
    PLIB_GLCD_Disable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsEnabled (GLCD_MODULE_ID index)

PLIB_GLCD_IsLastRow Function

Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsLastRow(GLCD_MODULE_ID index);
```

Returns

- true - The last row is currently displayed.
- false - The last row is not currently displayed.

Description

This function returns the status indicating whether a last row is currently displayed by the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_IsLastRow( GLCD_MODULE_ID_0 ));
{
    // The last row is not currently displayed
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsLastRow(GLCD_MODULE_ID index)

PLIB_GLCD_IsVerticalBlankingActive Function

Get the active status.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsVerticalBlankingActive(GLCD_MODULE_ID index);
```

Returns

- true - The Vertical Blanking is active.
- false - The Vertical Blanking is inactive.

Description

This function returns the active status of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_IsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
{
    //GLCD Vertical Blanking active
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsVerticalBlankingActive(GLCD_MODULE_ID index)

PLIB_GLCD_RGBSequentialModeGet Function

Get the RGB Sequential Mode already set.

File

plib_glcd.h

C

```
GLCD_RGB_MODE PLIB_GLCD_RGBSequentialModeGet(GLCD_MODULE_ID index);
```

Returns

- GLCD_RGB_MODE_PARALLEL_RGB565 - Parallel RGB 565 Mode
- GLCD_RGB_MODE_PARALLEL_RGB888 - Parallel RGB 888 Mode
- GLCD_RGB_MODE_SERIAL_RGB_3 - Byte Serial RGB 3 Mode
- GLCD_RGB_MODE_SERIAL_RGBA_4 - Byte Serial RGB 4 Mode
- GLCD_RGB_MODE_SERIAL_12BIT - Byte Two-Phase 12-bit Mode
- GLCD_RGB_MODE_YUYV_16BIT - YUYV 16 bit Mode
- GLCD_RGB_MODE_BT_656 - BT 656 Mode

Description

This function gets the RGB Sequential Mode already set.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t rgbMode;

rgbMode = PLIB_GLCD_RGBSequentialModeGet( GLCD_MODULE_ID_0 );

if( rgbMode == GLCD_RGB_MODE_PARALLEL_RGB565 )
{
    // RGB Mode set to RGB 565 format
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_RGB_MODE PLIB_GLCD_RGBSequentialModeGet(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputIsEnabled Function

Verify whether the YUV Output feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_YUVOutputIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The YUV output feature is enabled.
- false - The YUV output feature is disabled.

Description

This function verifies whether the YUV Output feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_YUVOutputIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_YUVOutputDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_YUVOutputIsEnabled(GLCD_MODULE_ID index)

i) Feature Existence Functions

PLIB_GLCD_ExistsBackgroundColor Function

Verify whether the Background Color Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBackgroundColor(GLCD_MODULE_ID index);
```

Returns

- true - The Background Color feature is supported.
- false - The Background Color feature is not supported.

Description

This function verifies whether the Background Color Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBackgroundColor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BackgroundColorSet( GLCD_MODULE_ID_0,
                                  MY_BACKGROUND_COLOR_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBackgroundColor(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsBackPorchXY Function

Verify whether Back Porch Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBackPorchXY(GLCD_MODULE_ID index);
```

Returns

- true - The Back Porch feature is supported.
- false - The Back Porch feature is not supported.

Description

This function verifies whether Back Porch Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBackPorchXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BackPorchXYSet( GLCD_MODULE_ID_0,
                              MY_BACK_PORCH_X_VALUE,
                              MY_BACK_PORCH_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBackPorchXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsBlankingXY Function

Verify whether Blanking Period Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBlankingXY(GLCD_MODULE_ID index);
```

Returns

- true - The Blanking feature is supported.
- false - The Blanking feature is not supported.

Description

This function verifies whether Blanking Period Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBlankingXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BlankingXYSet( GLCD_MODULE_ID_0,
                             MY_BLANKING_PERIOD_X_VALUE,
                             MY_BLANKING_PERIOD_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBlankingXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsClockDivider Function

Verify whether the Clock Divider feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsClockDivider(GLCD_MODULE_ID index);
```

Returns

- true - The Clock divide feature is supported.
- false - The Clock divide feature is not supported.

Description

This function verifies whether the Clock Divider feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsClockDivider( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ClockDividerSet( GLCD_MODULE_ID_0, MY_CLOCK_DIVIDER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsClockDivider(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursor Function

Verifies whether the cursor feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursor(GLCD_MODULE_ID index);
```

Returns

- true - Cursor feature exists.
- false - Cursor feature does not exits.

Description

This function verifies whether the cursor feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursor(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorData Function

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorData(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor Data feature is supported.
- false - The Cursor Data feature is not supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorData( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDataSet( GLCD_MODULE_ID_0,
                             MY_CURSOR_DATA_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorData(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorLUT Function

Verify whether Cursor LUT feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorLUT(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor LUT feature is supported.
- false - The Cursor LUT feature is not supported.

Description

This function verifies whether Cursor LUT feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorLUT( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorLUTSet( GLCD_MODULE_ID_0,
                            MY_CURSOR_LUT_DATA_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorLUT(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorXY Function

Verify whether Cursor XY Position feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorXY(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor Position feature is supported.
- false - The Cursor Position feature is not supported.

Description

This function verifies whether Cursor XY Position feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorXYSet( GLCD_MODULE_ID_0,
                           MY_CURSOR_POSITION_X_VALUE,
                           MY_CURSOR_POSITION_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsDESignalLevel Function

Verify whether DE Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsDESignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The DE Signal Level feature is supported.
- false - The DE Signal Level feature is not supported.

Description

This function verifies whether DE Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsDESignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_DESignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
```

```
            // DE Signal Level High
        }
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsDESignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsDithering Function

Verifies whether the Dithering feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsDithering(GLCD_MODULE_ID index);
```

Returns

- true - The Dithering feature is supported on the device
- false - The Dithering feature is not supported on the device

Description

This function verifies whether the Dithering feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsDithering( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_DitheringEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsDithering(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsEnable Function

Identifies whether the GLCD Enable feature exists on the GLCD module.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsEnable(GLCD_MODULE_ID index);
```

Returns

- true - The GLCD Enable feature is supported on the device
- false - The GLCD Enable feature is not supported on the device

Description

This function identifies whether the GLCD Enable feature is available on the GLCD module. When this function returns true, these functions are supported on the device:

- PLIB_GLCD_Enable
- PLIB_GLCD_Disable

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_Enable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_GLCD_ExistsEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsForceOutputBlank Function

Verify whether Force Output Blank feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsForceOutputBlank(GLCD_MODULE_ID index);
```

Returns

- true - The force output blank feature supported
- false - The force output blank feature not supported

Description

This function verifies whether Force Output Blank feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsForceOutputBlank( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ForceOutputBlankEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsForceOutputBlank(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsFormattingClockDivide Function

Verify whether Clock Formatting feature is available.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsFormattingClockDivide(GLCD_MODULE_ID index);
```

Returns

- true - The Clock formatting feature is supported.
- false - The Clock formatting feature is not supported.

Description

This function verifies whether Clock formatting feature is available.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsFormattingClockDivide( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_FormattingClockDivideEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsFormattingClockDivide(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsFrontPorchXY Function

Verify whether Front Porch Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsFrontPorchXY(GLCD_MODULE_ID index);
```

Returns

- true - The Front Porch feature is supported.
- false - The Front Porch feature is not supported.

Description

This funtion verifies whether Front Porch Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsFrontPorchXY( GLCD_MODULE_ID_0 ) )
```

```
{
    PLIB_GLCD_FrontPorchXYSet( GLCD_MODULE_ID_0,
                               MY_FRONT_PORCH_X_VALUE,
                               MY_FRONT_PORCH_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsFrontPorchXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsGlobalColorLUT Function

Verify whether Global Color LUT feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsGlobalColorLUT(GLCD_MODULE_ID index);
```

Returns

- true - The Global Color LUT feature is supported.
- false - The Global Color LUT feature is not supported.

Description

This function verifies whether Global Color LUT feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsGlobalColorLUT( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_GlobalColorLUTSet( GLCD_MODULE_ID_0,
                                 MY_GLOBAL_LUT_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsGlobalColorLUT(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsHSyncInterruptEnable Function

Verify whether the HSYNC Interrupt Enable feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsHSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Interrupt Enable feature is supported.

- false - The HSYNC Interrupt Enable feature is not supported.

Description

This function verifies whether the HSYNC Interrupt Enable feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsHSyncInterruptEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_HSyncInterruptEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsHSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsHSyncSignalLevel Function

Verify whether HSYNC Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsHSyncSignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Signal Level feature is supported.
- false - The HSYNC Signal Level feature is not supported.

Description

This function verifies whether the HSYNC Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsHSyncSignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_HSyncSignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
       // HSYNC Signal Level High
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsHSyncSignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIRQTriggerControl Function

Verify whether the IRQ Trigger Control feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIRQTriggerControl(GLCD_MODULE_ID index);
```

Returns

- true - The IRQ Trigger Control feature is supported.
- false - The IRQ Trigger Control feature is not supported.

Description

This function verifies whether the IRQ Trigger Control feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIRQTriggerControl( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_IRQTriggerControlSet( GLCD_MODULE_ID_0,
                                    MY_IRQ_TRIGGER_CONTROL );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIRQTriggerControl(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIsLastRow Function

Verify whether Is Last Row Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIsLastRow(GLCD_MODULE_ID index);
```

Returns

- true - The Is Last Row feature is supported.
- false - The Is Last Row feature is not supported.

Description

This function verifies whether Is Last Row Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIsLastRow( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_IsLastRow( GLCD_MODULE_ID_0 ) )
    {
       // Last row currently displayed
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIsLastRow(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIsVerticalBlankingActive Function

Verify whether Is Vertical Blanking Active feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIsVerticalBlankingActive(GLCD_MODULE_ID index);
```

Returns

- true - The Is Vertical Blanking feature is supported.
- false - The Is Vertical Blanking feature is not supported.

Description

This function verifies whether Is Vertical Blanking Active feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_IsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
    {
       // Vertical Blanking Active
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIsVerticalBlankingActive(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerBaseAddress Function

Verify whether the Layer Base Address feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerBaseAddress(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Base Address feature is supported.
- false - The Layer Base Address feature is not supported.

Description

This function verifies whether the Layer Base Address feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerBaseAddress( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerBaseAddressSet( GLCD_MODULE_ID_0,
                                   GLCD_LAYER_0,
                                   MY_LAYER_BASE_ADDRESS_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerBaseAddress (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerBilinearFilterEnable Function

Enable Layer Bilinear Filter Feature.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerBilinearFilterEnable(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Bilinear Filter feature is supported.
- false - The Layer Bilinear Filter feature is not supported.

Description

This function enables the Layer Bilinear Filter Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerBilinearFilterEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerBilinearFilterEnable( GLCD_MODULE_ID_0,
                                         GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerBilinearFilterEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerColorMode Function

Verify whether Layer Color Mode is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerColorMode(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Color Mode feature is supported.
- false - The Layer Color Mode feature is not supported.

Description

This function verifies whether Layer Color Mode is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerColorMode( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerColorModeSet( GLCD_MODULE_ID_0,
                                 GLCD_LAYER_0,
                                 MY_LAYER_COLOR_MODE_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerColorMode (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerDestBlendFunc Function

Verify whether Layer Destination Blend Function Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerDestBlendFunc(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Destination Blend Function feature is supported.
- false - The Layer Destination Blend Function feature is not supported.

Description

This function verifies whether Layer Destination Blend Function Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerDestBlendFunc( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerDestBlendFuncSet( GLCD_MODULE_ID_0,
                                     GLCD_LAYER_0,
                                     MY_LAYER_DEST_BLEND_FUNC_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerDestBlendFunc (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerEnable Function

Verify whether Layer Enable Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerEnable(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Enable feature is supported.
- false - The Layer Enable feature is not supported.

Description

This function verifies whether Layer enable Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerForceWithGlobalAlpha Function

Verify whether Layer Force with Global Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerForceWithGlobalAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Force Global Alpha feature is supported.
- false - The Layer Force Global Alpha feature is not supported.

Description

This function verifies whether Layer Force with Global Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerForceWithGlobalAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerForceWithGlobalAlphaEnable( GLCD_MODULE_ID_0,
                                               GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerForceWithGlobalAlpha (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerGlobalAlpha Function

Verify whether Layer Global Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerGlobalAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Global Alpha feature is supported.
- false - The Layer Global Alpha feature is not supported.

Description

This function verifies whether Layer Global Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerGlobalAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerGlobalAlphaSet( GLCD_MODULE_ID_0,
                                   GLCD_LAYER_0,
                                   MY_LAYER_GLOBAL_ALPHA_VALUE );
```

```
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerGlobalAlpha(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha Function

Verify whether Layer Pre-Multiply Image Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Pre-Multiply Image Alpha feature is supported.
- false - The Layer Pre-Multiply Image Alpha feature is not supported.

Description

This function verifies whether Layer Pre-Multiply Image Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerPreMultiplyImageAlphaEnable( GLCD_MODULE_ID_0,
                                                GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerResXY Function

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerResXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Resolution feature is supported.
- false - The Layer Resolution feature is not supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerResXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerResXYSet( GLCD_MODULE_ID_0,
                             GLCD_LAYER_0,
                             MY_LAYER_RES_X_VALUE,
                             MY_LAYER_RES_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerResXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerSizeXY Function

Verify whether Layer X Axis and Y Axis Size feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerSizeXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Size feature is supported.
- false - The Layer Size feature is not supported.

Description

This function verifies whether Layer X Axis and Y Axis Size feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerSizeXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerSizeXYSet( GLCD_MODULE_ID_0,
                              GLCD_LAYER_0,
                              MY_LAYER_SIZE_X_VALUE,
                              MY_LAYER_SIZE_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerSizeXY (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerSrcBlendFunc Function

Verify whether Layer Source Blend Function Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerSrcBlendFunc(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Source Blend Function feature is supported.
- false - The Layer Source Blend Function feature is not supported.

Description

This function verifies whether Layer Source Blend Function Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerSrcBlendFunc( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerSrcBlendFuncSet( GLCD_MODULE_ID_0,
                                    GLCD_LAYER_0,
                                    MY_LAYER_SRC_BLEND_FUNC_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerSrcBlendFunc (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerStartXY Function

Verify whether Layer Start XY Position feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerStartXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Start feature is supported.
- false - The Layer Start feature is not supported.

Description

This function verifies whether Layer Start XY Position feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerStartXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerStartXYSet( GLCD_MODULE_ID_0,
                               GLCD_LAYER_0,
```

```
                                MY_LAYER_START_X_VALUE,
                                MY_LAYER_START_Y_VALUE );
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerStartXY (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerStride Function

Verify whether the Layer Stride Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerStride(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Stride feature is supported.
- false - The Layer Stride feature is not supported.

Description

This function verifies whether the Layer Stride Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
    if( PLIB_GLCD_ExistsLayerStride( GLCD_MODULE_ID_0 ) )
    {
        PLIB_GLCD_LayerStrideSet( GLCD_MODULE_ID_0,
                                  GLCD_LAYER_0,
                                  MY_LAYER_STRIDE_VALUE );
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerStride(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLinesPrefetch Function

Verify whether Lines Prefetch Set Feature supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLinesPrefetch(GLCD_MODULE_ID index);
```

Returns

- true - The Lines Prefetch feature is supported.
- false - The Lines Prefetch feature is not supported.

Description

This function verifies whether Lines Prefetch Set Feature supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLinesPrefetch( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LinesPrefetchSet( GLCD_MODULE_ID_0,
                                MY_LINES_PREFETCH_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLinesPrefetch(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsPaletteGammaRamp Function

Verifies whether the palette / gamma ramp feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsPaletteGammaRamp(GLCD_MODULE_ID index);
```

Returns

- true - The palette / gamma ramp enable feature supported.
- false - The palette / gamma ramp enable feature not supported.

Description

This function verifies whether the palette / gamma ramp feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsPaletteGammaRamp( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_PaletteGammaRampEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsPaletteGammaRamp(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsResolutionXY Function

Verify whether YUV Output feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsResolutionXY(GLCD_MODULE_ID index);
```

Returns

- true - The Resolution feature is supported.
- false - The Resolution feature is not supported.

Description

This function verifies whether YUV Output feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsResolutionXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ResolutionXYSet( GLCD_MODULE_ID_0,
                               MY_RESOLUTION_X_VALUE,
                               MY_RESOLUTION_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsRGBSequentialMode Function

Verify whether RGB Sequential Mode feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsRGBSequentialMode(GLCD_MODULE_ID index);
```

Returns

- true - The RGB Sequential Mode feature is supported.
- false - The RGB Sequential Mode feature is not supported.

Description

This function verifies whether RGB Sequential Mode feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsRGBSequentialMode( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_RGBSequentialModeSet( GLCD_MODULE_ID_0,
                                    GLCD_RGB_MODE_PARALLEL );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsRGBSequentialMode(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsSignalPolarity Function

Verifies whether the Signal Polarity Selection feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsSignalPolarity(GLCD_MODULE_ID index);
```

Returns

- true - Feature exists
- false - Feature does not exist

Description

This function verifies whether the Signal Polarity Selection feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsSignalPolarity( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_SignalPolaritySet( GLCD_MODULE_ID_0,
                                 GLCD_POLARITY_POSITIVE |
                                 GLCD_DE_POLARITY_NEGATIVE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsSignalPolarity(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsSingleCyclePerLineVsync Function

Verifies whether VSYNC on Single cycle Per Line Feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsSingleCyclePerLineVsync(GLCD_MODULE_ID index);
```

Returns

- true - The feature: VSYNC on Single Cycle Per Line is supported on the device.
- false - The feature: VSYNC on Single Cycle Per Line is not supported on the device.

Description

This function verifies whether VSYNC on Single cycle Per Line Feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_ExistsSingleCyclePerLineVsync( GLCD_MODULE_ID_0))
{
    PLIB_GLCD_SingleCyclePerLineVsyncEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsSingleCycleVsync(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsVSyncInterruptEnable Function

Verify whether VSYNC Interrupt Enable feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsVSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Interrupt Enable feature is supported.
- false - The VSYNC Interrupt Enable feature is not supported.

Description

This function verifies whether VSYNC Interrupt Enable feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsVSyncInterruptEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_VSyncInterruptEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsVSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsVSyncSignalLevel Function

Verify whether VSYNC Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsVSyncSignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Signal Level feature is supported.
- false - The VSYNC Signal Level feature is not supported.

Description

This function verifies whether VSYNC Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsVSyncSignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_VSyncSignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
       // VSYNC Signal Level High
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsVSyncSignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsYUVOutput Function

Verify whether YUV output feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index);
```

Returns

- true - The YUV output feature is supported.
- false - The YUV output feature is not supported.

Description

This function verifies whether YUV output feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsYUVOutput( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_YUVOutputEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index)

j) Data Types and Constants

GLCD_IRQ_TRIGGER_CONTROL Enumeration

Possible values of GLCD Interrupt Trigger Mode.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_IRQ_TRIGGER_LEVEL = 0x00000000,
  GLCD_IRQ_TRIGGER_EDGE = 0x80000000
} GLCD_IRQ_TRIGGER_CONTROL;
```

Members

Members	Description
GLCD_IRQ_TRIGGER_LEVEL = 0x00000000	Interrupt Level Triggered
GLCD_IRQ_TRIGGER_EDGE = 0x80000000	Interrupt Edge Triggered

Description

GLCD Interrupt Trigger Mode

This data type defines the possible values of GLCD Interrupt Trigger modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_COLOR_MODE Enumeration

Possible values of GLCD color Mode.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_COLOR_MODE_LUT8 = 0x0,
  GLCD_LAYER_COLOR_MODE_RGBA5551 = 0x1,
  GLCD_LAYER_COLOR_MODE_RGBA8888 = 0x2,
  GLCD_LAYER_COLOR_MODE_RGB332 = 0x4,
  GLCD_LAYER_COLOR_MODE_RGB565 = 0x5,
  GLCD_LAYER_COLOR_MODE_ARGB8888 = 0x6,
  GLCD_LAYER_COLOR_MODE_L8 = 0x7,
  GLCD_LAYER_COLOR_MODE_L1 = 0x8,
  GLCD_LAYER_COLOR_MODE_L4 = 0x9,
  GLCD_LAYER_COLOR_MODE_YUYV = 0xA,
  GLCD_LAYER_COLOR_MODE_RGB888 = 0xB
} GLCD_LAYER_COLOR_MODE;
```

Members

Members	Description
GLCD_LAYER_COLOR_MODE_LUT8 = 0x0	GLCD Layer color mode LUT 8
GLCD_LAYER_COLOR_MODE_RGBA5551 = 0x1	GLCD Layer color mode RGBA 5551
GLCD_LAYER_COLOR_MODE_RGBA8888 = 0x2	GLCD Layer color mode RGBA 8888
GLCD_LAYER_COLOR_MODE_RGB332 = 0x4	GLCD Layer color mode RGB 332
GLCD_LAYER_COLOR_MODE_RGB565 = 0x5	GLCD Layer color mode RGB 565
GLCD_LAYER_COLOR_MODE_ARGB8888 = 0x6	GLCD Layer color mode ARGB 8888
GLCD_LAYER_COLOR_MODE_L8 = 0x7	GLCD Layer color mode L8
GLCD_LAYER_COLOR_MODE_L1 = 0x8	GLCD Layer color mode L1
GLCD_LAYER_COLOR_MODE_L4 = 0x9	GLCD Layer color mode L4
GLCD_LAYER_COLOR_MODE_YUYV = 0xA	GLCD Layer color mode YUYV
GLCD_LAYER_COLOR_MODE_RGB888 = 0xB	GLCD Layer color mode RGB888

Description

GLCD Layer color Mode

This data type defines the possible values of GLCD Layer color modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_DEST_BLEND_FUNC Enumeration

Possible values of GLCD Layer Destination Blend Functions.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_DEST_BLEND_BLACK = 0x0000,
  GLCD_LAYER_DEST_BLEND_WHITE = 0x1000,
  GLCD_LAYER_DEST_BLEND_ALPHA_SRC = 0x2000,
  GLCD_LAYER_DEST_BLEND_ALPHA_GBL = 0x3000,
  GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL = 0x4000,
  GLCD_LAYER_DEST_BLEND_INV_SRC = 0x5000,
  GLCD_LAYER_DEST_BLEND_INV_GBL = 0x6000,
  GLCD_LAYER_DEST_BLEND_INV_SRCGBL = 0x7000,
  GLCD_LAYER_DEST_BLEND_ALPHA_DST = 0xA000,
  GLCD_LAYER_DEST_BLEND_INV_DST = 0xD000
} GLCD_LAYER_DEST_BLEND_FUNC;
```

Members

Members	Description
GLCD_LAYER_DEST_BLEND_BLACK = 0x0000	Layer Destination Blend Function: Black, Destination Alpha = 0
GLCD_LAYER_DEST_BLEND_WHITE = 0x1000	Layer Destination Blend Function: White, Destination Alpha = 255
GLCD_LAYER_DEST_BLEND_ALPHA_SRC = 0x2000	Layer Destination Blend Function: Source Alpha
GLCD_LAYER_DEST_BLEND_ALPHA_GBL = 0x3000	Layer Destination Blend Function: Global Alpha
GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL = 0x4000	Layer Destination Blend Function: (Source Alpha * Global Alpha)
GLCD_LAYER_DEST_BLEND_INV_SRC = 0x5000	Layer Destination Blend Function: (1 - Source Alpha)
GLCD_LAYER_DEST_BLEND_INV_GBL = 0x6000	Layer Destination Blend Function: (1 - Global Alpha)
GLCD_LAYER_DEST_BLEND_INV_SRCGBL = 0x7000	Layer Destination Blend Function: (1 - (Source Alpha * Global Alpha))

GLCD_LAYER_DEST_BLEND_ALPHA_DST = 0xA000	Layer Destination Blend Function: Destination Alpha
GLCD_LAYER_DEST_BLEND_INV_DST = 0xD000	Layer Destination Blend Function: (1 - Destination Alpha)

Description

GLCD Layer Destination Blend Function

This data type defines the possible values of GLCD Layer Destination Blend Functions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_ID Enumeration

Possible values of GLCD Layer Ids

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_ID_0 = 0x0,
  GLCD_LAYER_ID_1 = 0x1,
  GLCD_LAYER_ID_2 = 0x2,
  GLCD_LAYER_ID_MAX = 0x3
} GLCD_LAYER_ID;
```

Members

Members	Description
GLCD_LAYER_ID_0 = 0x0	GLCD Layer 0 Id
GLCD_LAYER_ID_1 = 0x1	GLCD Layer 1 Id
GLCD_LAYER_ID_2 = 0x2	GLCD Layer 2 Id
GLCD_LAYER_ID_MAX = 0x3	Maximum number of Layers supported by GLCD

Description

GLCD Layer Ids

This data type defines the possible values of GLCD Layer Ids.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_SRC_BLEND_FUNC Enumeration

Possible values of GLCD Layer Source Blend Functions.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_SRC_BLEND_BLACK = 0x0000,
  GLCD_LAYER_SRC_BLEND_WHITE = 0x0100,
  GLCD_LAYER_SRC_BLEND_ALPHA_SRC = 0x0200,
  GLCD_LAYER_SRC_BLEND_ALPHA_GBL = 0x0300,
  GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL = 0x0400,
  GLCD_LAYER_SRC_BLEND_INV_SRC = 0x0500,
  GLCD_LAYER_SRC_BLEND_INV_GBL = 0x0600,
  GLCD_LAYER_SRC_BLEND_INV_SRCGBL = 0x0700,
  GLCD_LAYER_SRC_BLEND_ALPHA_DST = 0x0A00,
  GLCD_LAYER_SRC_BLEND_INV_DST = 0x0D00
} GLCD_LAYER_SRC_BLEND_FUNC;
```

Members

Members	Description
GLCD_LAYER_SRC_BLEND_BLACK = 0x0000	Layer Source Blend Function: Black, Source Alpha = 0
GLCD_LAYER_SRC_BLEND_WHITE = 0x0100	Layer Source Blend Function: White, Source Alpha = 255
GLCD_LAYER_SRC_BLEND_ALPHA_SRC = 0x0200	Layer Source Blend Function: Source Alpha
GLCD_LAYER_SRC_BLEND_ALPHA_GBL = 0x0300	Layer Source Blend Function: Global Alpha
GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL = 0x0400	Layer Source Blend Function: (Source Alpha * Global Alpha)
GLCD_LAYER_SRC_BLEND_INV_SRC = 0x0500	Layer Source Blend Function: (1 - Source Alpha)
GLCD_LAYER_SRC_BLEND_INV_GBL = 0x0600	Layer Source Blend Function: (1 - Global Alpha)
GLCD_LAYER_SRC_BLEND_INV_SRCGBL = 0x0700	Layer Source Blend Function: (1 - (Source Alpha * Global Alpha))
GLCD_LAYER_SRC_BLEND_ALPHA_DST = 0x0A00	Layer Source Blend Function: Destination Alpha
GLCD_LAYER_SRC_BLEND_INV_DST = 0x0D00	Layer Source Blend Function: (1 - Destination Alpha)

Description

GLCD Layer Source Blend Function

This data type defines the possible values of GLCD Layer Source Blend Functions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_MODULE_ID Enumeration

Possible instances of the GLCD module

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_ID_0 = 0,
  GLCD_NUMBER_OF_MODULES
} GLCD_MODULE_ID;
```

Members

Members	Description
GLCD_ID_0 = 0	first instance of the GLCD
GLCD_NUMBER_OF_MODULES	number of GLCD modules

Description

GLCD module ID

This data type defines the possible instances of the GLCD module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_RGB_MODE Enumeration

GLCD RGB Sequential Mode

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_RGB_MODE_PARALLEL_RGB565 = 0x00000000,
  GLCD_RGB_MODE_PARALLEL_RGB888 = 0x00000001,
  GLCD_RGB_MODE_SERIAL_RGB_3 = 0x00000002,
  GLCD_RGB_MODE_SERIAL_RGBA_4 = 0x00000003,
  GLCD_RGB_MODE_SERIAL_12BIT = 0x00000004,
  GLCD_RGB_MODE_YUYV_16BIT = 0x00000005,
  GLCD_RGB_MODE_BT_656 = 0x00000006
} GLCD_RGB_MODE;
```

Members

Members	Description
GLCD_RGB_MODE_PARALLEL_RGB565 = 0x00000000	Parallel RGB 565 Mode
GLCD_RGB_MODE_PARALLEL_RGB888 = 0x00000001	Parallel RGB 888 Mode
GLCD_RGB_MODE_SERIAL_RGB_3 = 0x00000002	Byte Serial RGB 3 Mode
GLCD_RGB_MODE_SERIAL_RGBA_4 = 0x00000003	Byte Serial RGB 4 Mode
GLCD_RGB_MODE_SERIAL_12BIT = 0x00000004	Byte Two-Phase 12-bit Mode
GLCD_RGB_MODE_YUYV_16BIT = 0x00000005	YUYV 16 bit Mode
GLCD_RGB_MODE_BT_656 = 0x00000006	BT 656 Mode

Description

GLCD RGB Sequential Mode

This data type defines the possible RGB Sequential Mode of GLCD.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_SIGNAL_POLARITY Enumeration

Polarity values of different output signals.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_POLARITY_POSITIVE = 0x00000000,
  GLCD_PIXEL_CLOCK_POLARITY_NEGATIVE = 0x00400000,
  GLCD_DE_POLARITY_NEGATIVE = 0x04000000,
  GLCD_HSYNC_POLARITY_NEGATIVE = 0x08000000,
  GLCD_VSYNC_POLARITY_NEGATIVE = 0x10000000
} GLCD_SIGNAL_POLARITY;
```

Members

Members	Description
GLCD_POLARITY_POSITIVE = 0x00000000	Positive polarity of PCLK, DE, HSYNC and VSYNC pins
GLCD_PIXEL_CLOCK_POLARITY_NEGATIVE = 0x00400000	Negative Pixel Clock Pin Polarity
GLCD_DE_POLARITY_NEGATIVE = 0x04000000	Negative DE Pin Polarity
GLCD_HSYNC_POLARITY_NEGATIVE = 0x08000000	Negative HSYNC Pin Polarity
GLCD_VSYNC_POLARITY_NEGATIVE = 0x10000000	Negative VSYNC Pin Polarity

Description

GLCD Output Signal Polarity

This data type defines the polarity values of different output signals.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_glcd.h	Defines the Graphics LCD Controller (GLCD) Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_glcd.h

Defines the Graphics LCD Controller (GLCD) Peripheral Library Interface.

Functions

	Name	Description
	PLIB_GLCD_BackgroundColorGet	Gets the value of the Background Color.
	PLIB_GLCD_BackgroundColorSet	Sets the background color.
	PLIB_GLCD_BackPorchXGet	Gets X Axis Back Porch.
	PLIB_GLCD_BackPorchXYSet	Sets the back porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_BackPorchYGet	Gets Y Axis Back Porch.
	PLIB_GLCD_BlankingXGet	Gets the X Axis Blanking Period.
	PLIB_GLCD_BlankingXYSet	Sets the blanking period on the x and y axis of the Graphics LCD Controller.
	PLIB_GLCD_BlankingYGet	Gets the Y Axis Blanking Period.
	PLIB_GLCD_ClockDividerGet	Gets Clock Divider value.
	PLIB_GLCD_ClockDividerSet	Sets clock controls of the Graphics LCD Controller.
	PLIB_GLCD_CursorDataGet	Gets the Cursor Data at given Index.
	PLIB_GLCD_CursorDataSet	Sets the cursor image data.
	PLIB_GLCD_CursorDisable	Disables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorEnable	Enables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorIsEnabled	Verifies whether the cursor is enabled.
	PLIB_GLCD_CursorLUTGet	Gets the color from Cursor LUT at given Index.
	PLIB_GLCD_CursorLUTSet	Sets the cursor color look-up table (LUT) in XRGB8888 format.
	PLIB_GLCD_CursorXGet	Gets the cursor X Axis Position.
	PLIB_GLCD_CursorXYSet	Sets the x and y coordinates of the Graphics LCD Controller cursor.
	PLIB_GLCD_CursorYGet	Gets the Cursor Y Axis Position.
	PLIB_GLCD_DESignalLevelGet	Gets the display enable signal level.
	PLIB_GLCD_Disable	Disables the Graphics LCD Controller.
	PLIB_GLCD_DitheringDisable	Disables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringEnable	Enables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringIsEnabled	Verifies whether Dithering is enabled.
	PLIB_GLCD_Enable	Enables the Graphics LCD Controller.
	PLIB_GLCD_ExistsBackgroundColor	Verify whether the Background Color Feature is supported.
	PLIB_GLCD_ExistsBackPorchXY	Verify whether Back Porch Feature is supported.
	PLIB_GLCD_ExistsBlankingXY	Verify whether Blanking Period Feature is supported.
	PLIB_GLCD_ExistsClockDivider	Verify whether the Clock Divider feature is supported.
	PLIB_GLCD_ExistsCursor	Verifies whether the cursor feature exists.
	PLIB_GLCD_ExistsCursorData	
	PLIB_GLCD_ExistsCursorLUT	Verify whether Cursor LUT feature is supported.
	PLIB_GLCD_ExistsCursorXY	Verify whether Cursor XY Position feature is supported.
	PLIB_GLCD_ExistsDESignalLevel	Verify whether DE Signal Level feature is supported.
	PLIB_GLCD_ExistsDithering	Verifies whether the Dithering feature exists.

	PLIB_GLCD_ExistsEnable	Identifies whether the GLCD Enable feature exists on the GLCD module.
	PLIB_GLCD_ExistsForceOutputBlank	Verify whether Force Output Blank feature is supported.
	PLIB_GLCD_ExistsFormattingClockDivide	Verify whether Clock Formatting feature is available.
	PLIB_GLCD_ExistsFrontPorchXY	Verify whether Front Porch Feature is supported.
	PLIB_GLCD_ExistsGlobalColorLUT	Verify whether Global Color LUT feature is supported.
	PLIB_GLCD_ExistsHSyncInterruptEnable	Verify whether the HSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsHSyncSignalLevel	Verify whether HSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsIRQTriggerControl	Verify whether the IRQ Trigger Control feature is supported.
	PLIB_GLCD_ExistsIsLastRow	Verify whether Is Last Row Feature is supported.
	PLIB_GLCD_ExistsIsVerticalBlankingActive	Verify whether Is Vertical Blanking Active feature is supported.
	PLIB_GLCD_ExistsLayerBaseAddress	Verify whether the Layer Base Address feature is supported.
	PLIB_GLCD_ExistsLayerBilinearFilterEnable	Enable Layer Bilinear Filter Feature.
	PLIB_GLCD_ExistsLayerColorMode	Verify whether Layer Color Mode is supported.
	PLIB_GLCD_ExistsLayerDestBlendFunc	Verify whether Layer Destination Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerEnable	Verify whether Layer Enable Feature is supported.
	PLIB_GLCD_ExistsLayerForceWithGlobalAlpha	Verify whether Layer Force with Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerGlobalAlpha	Verify whether Layer Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha	Verify whether Layer Pre-Multiply Image Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerResXY	
	PLIB_GLCD_ExistsLayerSizeXY	Verify whether Layer X Axis and Y Axis Size feature is supported.
	PLIB_GLCD_ExistsLayerSrcBlendFunc	Verify whether Layer Source Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerStartXY	Verify whether Layer Start XY Position feature is supported.
	PLIB_GLCD_ExistsLayerStride	Verify whether the Layer Stride Feature is supported.
	PLIB_GLCD_ExistsLinesPrefetch	Verify whether Lines Prefetch Set Feature supported.
	PLIB_GLCD_ExistsPaletteGammaRamp	Verifies whether the palette / gamma ramp feature is supported.
	PLIB_GLCD_ExistsResolutionXY	Verify whether YUV Output feature is supported.
	PLIB_GLCD_ExistsRGBSequentialMode	Verify whether RGB Sequential Mode feature is supported.
	PLIB_GLCD_ExistsSignalPolarity	Verifies whether the Signal Polarity Selection feature exists.
	PLIB_GLCD_ExistsSingleCyclePerLineVsync	Verifies whether VSYNC on Single cycle Per Line Feature exists.
	PLIB_GLCD_ExistsVSyncInterruptEnable	Verify whether VSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsVSyncSignalLevel	Verify whether VSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsYUVOutput	Verify whether YUV output feature is supported.
	PLIB_GLCD_ForceOutputBlankDisable	Disable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankEnable	Enable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankIsEnabled	Verify whether the Force Output Blank feature is enabled.
	PLIB_GLCD_FormattingClockDivideDisable	Disbale the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideEnable	Enable the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideIsEnabled	Verify whether Clock Formatting feature is enabled.
	PLIB_GLCD_FrontPorchXGet	Gets X Axis Front Porch.
	PLIB_GLCD_FrontPorchXYSet	Sets the front porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_FrontPorchYGet	Gets Y Axis Front Porch.
	PLIB_GLCD_GlobalColorLUTGet	Gets the Color from Global Color LUT at given Index.
	PLIB_GLCD_GlobalColorLUTSet	Set Global Color LUT.
	PLIB_GLCD_HSyncInterruptDisable	Disables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptEnable	Enables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptIsEnabled	Verify whether HSYNC Interrupt is enabled.
	PLIB_GLCD_HSyncSignalLevelGet	Gets the Hsync signal level.
	PLIB_GLCD_IRQTriggerControlGet	Gets the IRQ Trigger Control value.
	PLIB_GLCD_IRQTriggerControlSet	Sets the IRQ trigger control.
	PLIB_GLCD_IsEnabled	Verifies whether the Graphics LCD Controller is Enabled.
	PLIB_GLCD_IsLastRow	Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.
	PLIB_GLCD_IsVerticalBlankingActive	Get the active status.
	PLIB_GLCD_LayerBaseAddressGet	Gets the Layer Base Address.
	PLIB_GLCD_LayerBaseAddressSet	Sets the base address of layer surface of the Graphics LCD Controller.

	PLIB_GLCD_LayerBilinearFilterDisable	Disables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterEnable	Enables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterIsEnabled	Verify whether Layer Bilinear Filter is enabled.
	PLIB_GLCD_LayerColorModeGet	Gets the Layer Color Mode.
	PLIB_GLCD_LayerColorModeSet	Sets the layer color mode of the Graphics LCD Controller.
	PLIB_GLCD_LayerDestBlendFuncGet	Gets the Layer Destination Blend Function.
	PLIB_GLCD_LayerDestBlendFuncSet	Sets the layer destination blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerDisable	Disables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerEnable	Enables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerForceWithGlobalAlphaDisable	Disables the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaEnable	Enable the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled	Verify whether Layer Force with Global Alpha Feature is enabled.
	PLIB_GLCD_LayerGlobalAlphaGet	Gets the Layer Global Alpha value.
	PLIB_GLCD_LayerGlobalAlphaSet	Sets the layer global alpha of the Graphics LCD Controller.
	PLIB_GLCD_LayerIsEnabled	Verify whether Layer is enabled.
	PLIB_GLCD_LayerPreMultiplyImageAlphaDisable	Disable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaEnable	Enable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled	Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.
	PLIB_GLCD_LayerResXGet	Gets the Layer X Axis Resolution.
	PLIB_GLCD_LayerResXYSet	Sets the layer resolution in pixels.
	PLIB_GLCD_LayerResYGet	Gets the Layer Y Axis Resolution.
	PLIB_GLCD_LayerSizeXGet	Gets the Layer X Axis Size.
	PLIB_GLCD_LayerSizeXYSet	Sets the layer size x and size y of the Graphics LCD Controller.
	PLIB_GLCD_LayerSizeYGet	Gets the Layer Y Axis Size.
	PLIB_GLCD_LayerSrcBlendFuncGet	Gets the Layer Source Blend Function.
	PLIB_GLCD_LayerSrcBlendFuncSet	Sets the layer source blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartXGet	Gets the Layer X Axis Start Position.
	PLIB_GLCD_LayerStartXYSet	Sets the layer start x and start y of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartYGet	Gets the Layer Y Axis Start Position.
	PLIB_GLCD_LayerStrideGet	Gets the Layer Stride value.
	PLIB_GLCD_LayerStrideSet	Sets the layer surface stride of the Graphics LCD Controller.
	PLIB_GLCD_LinesPrefetchGet	Gets value of lines to be prefetched.
	PLIB_GLCD_LinesPrefetchSet	Sets the clock controls of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampDisable	Disables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampEnable	Enables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampIsEnabled	Verify whether Palette / Gamma Ramp feature is enabled.
	PLIB_GLCD_ResolutionXGet	Gets X Axis Resolution.
	PLIB_GLCD_ResolutionXYSet	Sets the display resolution for the Graphics LCD Controller.
	PLIB_GLCD_ResolutionYGet	Gets Y Axis Resolution.
	PLIB_GLCD_RGBSequentialModeGet	Get the RGB Sequential Mode already set.
	PLIB_GLCD_RGBSequentialModeSet	Sets the RGB output sequential mode.
	PLIB_GLCD_SignalPolarityGet	Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SignalPolaritySet	Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SingleCyclePerLineVsyncDisable	Clears VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncEnable	Sets VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled	Verify whether VSYNC on Single Cycle Per Line feature is enabled.
	PLIB_GLCD_VSyncInterruptDisable	Disables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptEnable	Enables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptIsEnabled	Verify whether VSYNC Interrupt is enabled.
	PLIB_GLCD_VSyncSignalLevelGet	Gets the Vsync signal level.
	PLIB_GLCD_YUVOutputDisable	Disables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputEnable	Enables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputIsEnabled	Verify whether the YUV Output feature is enabled.

Description

Graphics LCD Controller (GLCD) Peripheral Library Interface Header.

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Graphics LCD Controller Peripheral Library for Microchip micro-controllers. The definitions in this file are only for the directly addressable memory mapped Graphics LCD Controller module.

File Name

plib_glcd.h

Company

Microchip Technology Inc.

I2C Peripheral Library

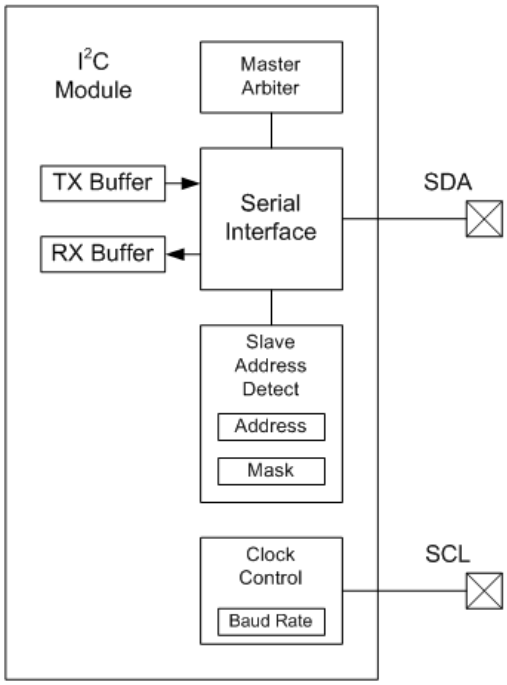
This section describes the Inter-Integrated Circuit (I2C) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The I2C module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, display drivers, analog-to-digital converters, etc.



Please refer to the I2C-Bus Specification (v2.1), available from Philips Semiconductor (see the following **Note**) for complete details on the operation and requirements for the I2C bus.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the I2C Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_i2c.h`

The interface to the I2C library is defined in the `plib_i2c.h` header file, which is included by the peripheral library header file,`peripheral.h`. Any C language source (`.c`) file that uses the I2C library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the I2C Peripheral Library, the module instance IDs are defined by the I2C_MODULE_ID enumeration. Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific device data sheet for more information.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

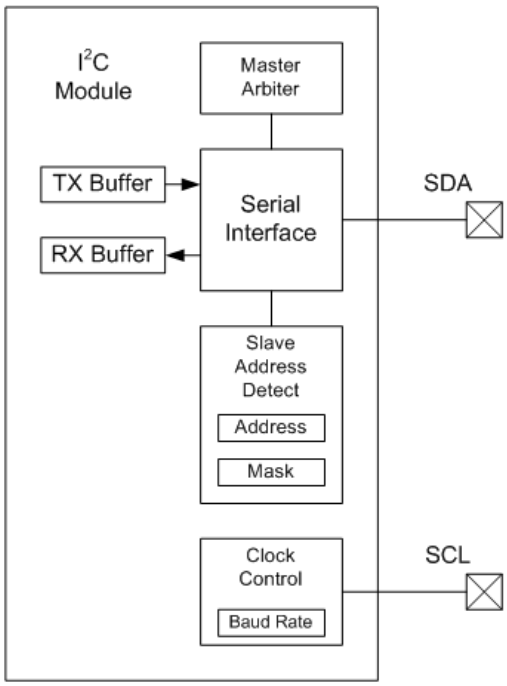
This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Description

The I2C Peripheral Library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model Diagram



The **TX Buffer** and **RX Buffer** are 1-byte (8-bit) buffers for data transmitted (TX) or received (RX) by the **Serial Interface** to the I2C bus over the **SDA** line synchronized with the **SCL** line by the **Clock Control** logic.

The desired baud rate frequency can be programmed in the **Clock Control** logic (within I2C bus specifications of 0-100 kHz for Low-to-Fast I2C mode and up to 3.4 MHz for High-Speed I2C mode) by setting the **Baud Rate** value. However, the nature of the I2C bus allows slower targets to stretch the clock to slower rates as required, in real time.

The I2C bus is a transfer-based multi-master bus, allowing multiple masters to initiate transfers at any time the bus is not currently busy. The **Master Arbiter** logic monitors transfers and determines if arbitration has been lost. There is no loss of data for the winning master. However, the losing master must retry the entire transfer again later, when the bus is idle. Slave devices are identified by either 7- or-10 bit addresses. The I2C module can act as either a bus master that can initiate transfers or as a slave, responding to transfers initiated by other masters on the bus.

The **Slave Address Detect** logic monitors transfers initiated by other masters to recognize when other masters have addressed this module when it is acting as a slave. The Slave **Address** is programmed into the I2C module by the software along with an (optional) address **Mask**, specifying bits in the address that can be ignored. This ability to "mask off" certain bits in the address allows the I2C module to identify multiple addresses and respond as if it were more than one slave device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2C module.

Library Interface Section	Description
General Initialization Functions	Provides configuration routines for: • Slave Mode Clock Stretching • General Call and Other Reserved Address Support • System Management Bus (SMBus) support • High Frequency Support • Stop-in-Idle Control • Module Enable/Disable Control

General Status Functions	Provides status routines for I2C bus conditions.
Baud Rate Generator Control Functions	Provides configuration and access routines for baud rate control.
Slave Address Control Functions	Provides configuration, access, and status routines for control of the slave address to which the I2C module will respond when operating in Slave mode.
Slave Mask Control Functions	Provides configuration and access routines for control of the "ignore" bit mask for slave addressing. The "ignore" mask determines which bits in the incoming transfer's address are ignored, allowing the module to respond to multiple I2C addresses when operating in slave mode. **Note:** This feature is not available on all devices. Refer to the specific device data sheet determine whether slave address masking is supported for your device.
Slave Control Functions	Provides status and control routines for operating in slave mode (either as a receiver or as a transmitter).
Master Control Functions	Provides routines for control of the module when operating in master mode (either as a receiver or as a transmitter). **Note:** The necessary status routines are either part of the general, transmitter, or receiver groups depending on their purpose.
Transmitter Control Functions	Provides control, status, and data transfer routines for transmitting data (either as a master or a slave).
Receiver Control Functions	Provides control, status, and data transfer routines for receiving data (either as a master or a slave).
Feature Existence Functions	Determine whether particular features exist on a device.

How the Library Works

Provides information on how the library works.

Description

Before enabling the I2C module the caller must initialize basic configuration, clock frequency, and Slave address recognition features (see Initializing the I2C).

After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see Operating as a Slave Receiver). A slave device only transfers data on the bus when it has been addressed by a master (see Operating as a Slave Transmitter). If the module is used to start a transfer, it is operating as a master. A master addresses a slave and controls the transfer of data by providing the clock (see Operating as a Master Transmitter and Operating as a Master Receiver).

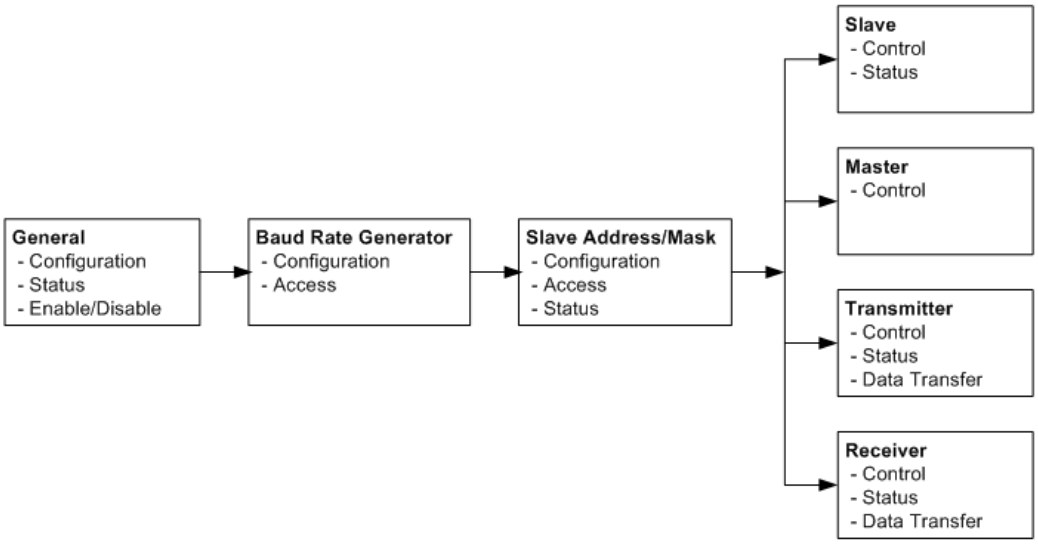
Some operations in the I2C library initiate actions on the I2C bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. Interrupt State Machine describes the different states that can cause an interrupt and show what causes the transition from one state to another. Handling Errors describes the various errors that can occur and how to deal with them.

Data transfers on the I2C bus must follow specific formats defined by the I2C bus protocol. (Refer to Forming Transfers for details)

Note: The I2C peripheral library does not directly respond to interrupts. The client software (usually a driver, middleware layer, or application) should implement the Interrupt Service Routine (ISR) and call the I2C library's interface routines from that ISR to manage the state of the I2C module.

Usage Model

The following diagram illustrates the library usage model for the I2C Peripheral Library.



Initializing the I2C

To initialize the I2C module, perform the following sequence:

1. Select the desired general configuration options using General Initialization Functions (see the Library Interface section) to select the desired operation of the following features:
   - Slave Clock Stretching
   - General Call Address Recognition
   - System Management Bus (SMBus) Support
   - High Frequency Operation
   - Reserved Address Protection
   - Stop-in-Idle Operation
2. Program the Baud Rate Generator to set the desired baud rate using the PLIB_I2C_BaudRateSet function.
3. If operating in slave mode, set the desired slave address using the PLIB_I2C_SlaveAddress7BitSet function (for 7-bit mode operation) or PLIB_I2C_SlaveAddress10BitSet function (for 10-bit mode operation).
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and error) I2C interrupts.

Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for more information.

5. Enable the module for operation using the PLIB_I2C_Enable function.

Example

```
#define MY_I2C_ID  I2C_ID_1
uint32_t  actualClock;

// Configure General I2C Options
PLIB_I2C_SlaveClockStretchingEnable(MY_I2C_ID);
PLIB_I2C_GeneralCallEnable(MY_I2C_ID);
PLIB_I2C_SMBDisable(MY_I2C_ID);
PLIB_I2C_HighFrequencyDisable(MY_I2C_ID);
PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
PLIB_I2C_StopInIdleEnable(MY_I2C_ID);

// Set Desired baud rate
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_PERIPHEAL_CLOCK_FREQ, MY_I2C_BAUD_RATE);

// Set the appropriate slave address (slave only)
PLIB_I2C_SlaveAddress7BitSet(MY_I2C_ID, MY_SLAVE_ADDRESS);

// Optional:  Clear and enable interrupts before enabling the I2C module.

// Enable the module
PLIB_I2C_Enable(MY_I2C_ID);
```

Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section referenced by that chapter for more information.

The previous example uses the following constants:

Symbol	Description
I2C_ID_1	Defined as the selected I2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
MY_I2C_PERIPHERAL_CLOCK	Defined as the peripheral clock frequency (in Hz) of the I2C module. Example: `#define MY_I2C_PERIPHERAL_CLOCK 80000000`
MY_I2C_BAUD_RATE	Defined as desired I2C baud rate. Example: #define MY_I2C_BAUD_RATE 100000
MY_SLAVE_ADDRESS	Defined as the desired 7-bit I2C slave address. Example: `#define 0x1A // Slave address 0011010` in binary **Note:** If '0' is passed as the mask value, the I2C module will only respond to single slave address passed in the second parameter.

Operating as a Slave Receiver

Before enabling the module, the module's "local" slave address and mask should be programmed using the PLIB_I2C_SlaveAddress7BitSet function or the PLIB_I2C_SlaveAddress10BitSet function, depending on the desired address width. If slave address masking is supported on the chosen processor the "don't care bits" mask can be set using the PLIB_I2C_SlaveMask7BitSet function or the PLIB_I2C_SlaveMask10BitSet function, also depending on the desired address width. Once the I2C module has been enabled, it will begin waiting for an I2C master to address it. When the programmed local slave address is recognized (ignoring any bits in the programmed mask value, if supported), the module will ACK the address and notify software that it has received an address byte. If the R/W bit of the address is 0, the module enters receive mode automatically. As additional bytes are received, the module will automatically ACK them if the previous byte has been read from the RX buffer before the new byte has been completely received and notify software of the newly available byte. Both address and data bytes received must be read from the RX buffer by software.

Preconditions

1. The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).
2. An external master must have started a transfer.
3. The address of the transfer must have matched with the address (ignoring any bits in the mask) programmed into the module by software.

Process

Software must follow this sequence:

1. Ensure that a byte has been received either by polling the PLIB_I2C_ReceivedByteIsAvailable function or by waiting for a slave interrupt (at which time, this function should still be checked).
2. Identify if the byte is a data byte or an address byte using the PLIB_I2C_SlaveAddressIsDetected function.
3. If the byte is an address byte, software must identify if it is starting a read or write operation using the PLIB_I2C_SlaveReadIsRequested function.
4. Software must read the byte from the RX buffer using the PLIB_I2C_ReceivedByteGet function, even if it was an address byte.
5. If clock stretching was enabled during initialization, software must call the PLIB_I2C_SlaveClockRelease function to release the SCL line.

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    // Check to see if the byte is data or an address
    if (PLIB_I2C_SlaveAddressWasDetected(MY_I2C_ID))
    {
        if ( PLIB_I2C_SlaveReadWasRequested(MY_I2C_ID) )
        {
            // Beginning a slave receive sequence
        }
        else
        {
            // Beginning a slave transmit sequence
        }
    }
    else
    {
        // Data Byte Received
    }
    // Read the data or address from the RX buffer
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);
```

```
    // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

The previous example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C_MODULE_ID enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the variable used to hold the byte received.

Operating as a Slave Transmitter

Once an external master has addressed the I2C module (see Operating as a Slave Receiver) with the read-write bit set (as determined by calling the PLIB_I2C_SlaveReadIsRequested function), software can begin transmitting data to the master.

Preconditions

1. The module must have had its basic options and clock frequency initialized and have been enabled(see Initializing the I2C)
2. An external master must have started a transfer.
3. The address of the transfer must have matched with the address slave programmed into the I2C module (see Managing Slave Addresses).
4. A slave read must have been requested (as previously described).

Note: When the address has been received and matched the programmed local slave address, a slave interrupt will occur if it has been enabled.

Transmitting a Byte

When operating as a slave transmitter on the I2C bus, the software must use the following sequence to transmit data bytes.

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
2. Write the byte to be transmitted to the TX buffer using the PLIB_I2C_TransmitterByteSend function.
3. If clock stretching was enabled (using the PLIB_I2C_SlaveClockStretchingEnable function) and the previous byte was ACKed, the PLIB_I2C_SlaveClockRelease function must be called to release the SCL line (it is safe to call this even if the previous byte was NAKed.)

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t         data;

// Assign the desired byte value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    // Write the byte to the TX buffer
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);

     // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

Checking Acknowledgement

After a byte has been transmitted, the module latches the ACK/NAK response from the master so that software may use the following sequence to verify it and causes another slave interrupt (if it has been enabled).

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by responding to the slave interrupt).
2. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

Example

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
    {
        // Transmission successful
    }
    else
    {
        // Transmission was not successful
    }
}
```

The previous examples use the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C_MODULE_ID enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the buffer containing byte of data to transmit.

Operating as a Master Transmitter

Whenever the I2C module initiates a transfer on the bus, it is acting as master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration at some point during the transfer and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I2C bus transfer always begins with a Start condition, followed by one or two properly formatted bytes containing the 7-bit or 10-bit target slave address (address of the slave device that is the target of the transfer). The direction of the remainder of transfers (after the address has been sent) is indicated by bit 0 of the address (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). After that, any amount of data may be transferred and a repeated Start condition (followed by another properly formatted address) can occur to change the direction of the transfer or even the target slave device. See Forming Transfers for details of the possible transfer formats. When the master is finished, an I2C bus transfer always ends with a Stop condition. These steps are summarized as follows:

Summary of Steps

1. Send a Start condition.
2. Send a 7-bit or 10-bit address (one or two bytes with a write transfer indication in bit 0).
3. Send data bytes.
4. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I2C master interrupt. The software must always keep track of which state it was in, check the I2C status, and respond to the interrupt by performing the next appropriate step.

Notes:

1. Arbitration loss can occur after each step has completed. Software should check for arbitration loss by calling the PLIB_I2C_ArbitrationLossHasOccurred function. If arbitration loss occurs, it must be cleared by calling the PLIB_I2C_ArbitrationLossClear function.
2. After any step completes, but before sending the Stop condition (see step 4), the master may send a repeated Start condition and start over at step 2 to change slave targets and/or transfer direction.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).

Process

1. Verify that the bus is idle using the PLIB_I2C_BusIsIdle function.
2. Send the start signal using the PLIB_I2C_MasterStart function.
3. Check for arbitration loss after the start condition has completed using the PLIB_I2C_ArbitrationLossHasOccurred function.

Example: Sending a Start Condition

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Sending a 7-bit Address

To send a 7-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Write the address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function. The address byte that is supplied as the parameter to PLIB_I2C_TransmitterByteSend should also include the R/W bit.

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device. (If the slave device does not answer, an implicit NAK will occur.)

2. Once the address byte has been ACKed or NAKed by the slave target, software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to send a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID I2C_ID_1
#define SLAVE_TARGET_ADDRESS 0x22
uint8_t slaveAddress = 0;

// Format the 7-bit address byte with the write indication in bit 0
slaveAddress = SLAVE_TARGET_ADDRESS;

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddress);
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To send a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer).
2. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
3. Write the first address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

4. Ensure that the first address byte was ACKed by the slave target (to do this, follow the Acknowledgement process described in a following section.
5. Write the second address byte to the TX buffer using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function.

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

6. Ensure that the second address byte was ACKed by the slave target (to do this, follow the process in the Acknowledgement and Arbitration, which is described later in this section).

Example: Formatting the 10-bit Address in Two Bytes

```
/* first byte of 10-bit address which is inclusive of R/W bit */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE1 0xF6

/* second byte of 10-bit address */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE2 0x22

uint8_t slaveAddressByte1 = 0;
uint8_t slaveAddressByte2 = 0;

/* include the R/W in bit 0 of the first byte */
slaveAddressByte1 = SLAVE_TARGET_10BIT_ADDRESS_BYTE1;
slaveAddressByte2 = SLAVE_TARGET_10BIT_ADDRESS_BYTE2;
```

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte1);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte2);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Sending a Data Byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address has been fully transmitted (with the write indication in bit 0) and ACKed by the targeted slave device

Process

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady Function.
2. Write the byte to be transmitted to the TX buffer using the PLIB_I2C_TransmitterByteSend Function.

Note: After the byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the PLIB_I2C_TransmitterByteHasCompleted function).

3. Check for arbitration loss or an ACK or NAK from the slave as described in the next section, Acknowledgement and Arbitration.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t         data;

// Assign the data value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as previously described), the master transmitter must check to see if arbitration was lost and if the byte was ACKed or NAKed. If the byte was ACKed, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot accept any more bytes at this time.

Preconditions

- The start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the previous processes

Process

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by waiting for the master interrupt)
2. Determine if arbitration loss has occurred using the PLIB_I2C_ArbitrationLossHasOccurred function.
3. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

Example

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
        // Handle arbitration loss

        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
```

```
    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}
```

To handle arbitration loss, the master transmitter must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it can send a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Process

Send the repeated Start condition using the PLIB_I2C_MasterStartRepeat function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated.

The repeated Start must be followed by another target slave address (either 7-bit or 10-bit, as appropriate).

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a stop condition using the PLIB_I2C_MasterStop function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the Stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous examples use the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I^2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`

`data`	This is the buffer containing byte of data to transmit.
`slaveTargetAddress7Bit`	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
`SLAVE_TARGET_7BIT_ADDRESS`	This constant is used as a placeholder for the actual 7-bit target slave address that must be defined by the client code.
`slaveAddress10Bit`	This variable is used to demonstrate how to properly format and access a 10-bit slave address.
`SLAVE_TARGET_10BIT_ADDRESS`	This constant is used as a placeholder for the actual 10-bit slave address that must be defined by the client code.

Operating as a Master Receiver

Any time the I2C module initiates a transfer on the bus, it is acting as a master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I2C bus transfer always begins with a Start condition, followed by a 7-bit or 10-bit (1-byte or 2-byte) target slave address. Therefore, every master transfer starts off transmitting at least one byte. The direction of transfer of subsequent bytes (after the first byte has been sent) is indicated by bit 0 of the first address byte (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). If 10-bit addressing is used or additional data (such as an internal address) is to be transmitted after the first address byte, the first address byte must indicate a write transfer. To change the direction after transmitting more than just a single address byte, a repeated Start condition (followed by another single address byte) must be sent (see Forming Transfers for details of the possible transfer formats). When the master is finished, an I2C bus transfer always ends with a Stop condition. This results in two basic formats for receiving data as a master, summarized as follows:

Summary of Steps: Read Only

1. Send a Start condition.
2. Send a 7-bit address with a read transfer indication in bit 0.
3. Receive data bytes.
4. Send a Stop condition.

Summary of Steps: Combined Read-Write

1. Send a Start condition.
2. Send a 7-bit address (or the first byte of a 10-bit address) with a write transfer indication in bit 0.
3. Send the second byte of a 10-bit address (and/or any additional data to be transmitted).
4. Send a repeated Start condition.
5. Resend the first byte of the 10-bit address (or a 7-bit address) with a read indication in bit 0.
6. Receive data bytes.
7. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I2C master interrupt. The software must always keep track of which state it was in and respond to the interrupt by performing the next appropriate step.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).

Process

1. Verify that the bus is idle using the PLIB_I2C_BusIsIdle function.
2. Send the start signal using the PLIB_I2C_MasterStart function.

Example

```
#define MY_I2C_ID  I2C_ID_1
if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the Start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1
if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
```

```
}
```

Sending a 7-bit Address

To send a 7-bit address (or the first byte of a 10-bit address), the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
2. Format the address byte correctly with the 7-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer).
3. Write the address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device (or if arbitration was lost). If the slave device does not answer, an implicit NAK will occur.

4. Once the address byte has been ACKed or NAKed by the slave target (or if arbitration was lost), software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to send a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID  I2C_ID_1
#define SLAVE_TARGET_ADDRESS 0x22
uint8_t slaveAddress = 0;

// Format the 7-bit address byte with the read indication in bit 0
slaveAddress = (SLAVE_TARGET_ADDRESS | 0x01);

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddress);
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To send a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit cleared, indicating a write transfer.
2. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
3. Write the first address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the PLIB_I2C_TransmitterByteHasCompleted function).

4. Ensure that the first address byte was ACKed by the slave target and check for arbitration loss. To do this, use the Acknowledgement and Arbitration process, which described in the next section.
5. Write the second address byte to the TX buffer using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function.

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

6. Ensure that the second address byte was ACKed by the slave target and that arbitration was not lost. To do this, use the Acknowledgement and Arbitration process, which described in the next section.

Example: Formatting the 10-bit Address in Two Bytes

```
#define MY_I2C_ID I2C_ID_1

/* first byte of 10-bit address which is inclusive of R/W bit */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE1 0xF6

/* second byte of 10-bit address */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE2 0x22

uint8_t slaveAddressByte1 = 0;
```

```
uint8_t slaveAddressByte2 = 0;

/* including the R/W bit in byte-1 of 10-bit address */
slaveAddressByte1 = (SLAVE_TARGET_10BIT_ADDRESS_BYTE1 | 0x01);

/* the second byte is included as is */
slaveAddressByte2 = SLAVE_TARGET_10BIT_ADDRESS_BYTE2;
```

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte1);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte2);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as previously described), the master transmitter must check to see if arbitration was lost and if the byte was ACKed or NAKed. If the byte was ACKed, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot accept any more bytes at this time.

Preconditions

- The Start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the previous processes

Process

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by waiting for the master interrupt).
2. Determine if arbitration loss has occurred using the PLIB_I2C_ArbitrationLossHasOccurred function.
3. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
        // Handle arbitration loss
        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}
```

To handle arbitration loss, the master must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it must send a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.
- Zero or more data bytes may also have been sent or received and ACKed as appropriate.

Process

Send the repeated Start condition using the PLIB_I2C_MasterStartRepeat function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The repeated Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. It must then be followed by another target slave address (either 7-bit or 10-bit, as appropriate). Strictly speaking, a bus collision can still occur while the repeated start condition is being transmitted. To verify that arbitration has not been lost during transmission of the repeated Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the repeated Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Receiving a Data Byte

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address must have been fully transmitted (with the read indication in bit 0) and ACKed by the targeted slave device

Process

1. Start the I2C module cycling the clock line to receive one byte by calling the PLIB_I2C_MasterReceiverClock1Byte function. The I2C module will generate a master interrupt once the eighth clock cycle has completed, but before the ninth cycle has started.
2. Ensure that a byte was received and is available in the RX buffer using the PLIB_I2C_ReceivedByteIsAvailable function.
3. If a byte is available, get it from the RX buffer using the PLIB_I2C_ReceivedByteGet function.
4. ACK or NAK the byte (as determined by software) using the PLIB_I2C_ReceivedByteAcknowledge function. The master receiver indicates to the slave transmitter that it wants to terminate the transfer by NAKing the last byte. All other bytes should be ACKed after reception.
5. Ensure that the ACK/NAK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function or by waiting for the master interrupt.

Note: Arbitration loss cannot occur during a master reception (only during transmission).

Example: Enabling the Receiver

```
#define MY_I2C_ID I2C_ID_1
PLIB_I2C_MasterReceiverClock1Byte(MY_I2C_ID);
```

Wait for reception to complete before attempting to get the byte from the receiver.

Example: Reading a Byte from the RX Buffer

```
#define MY_I2C_ID I2C_ID_1
#define MY_MAX_COUNT  10
uint8_t data;
int     count;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);
    count++;

    // Determine if the data should be ACKed or NAKed
    if (count < MY_MAX_COUNT)
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, true);
    }
    else
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, false);
    }
}
```

Wait for the ACK/NAK to complete by waiting for the next master interrupt.

Example: Verifying ACK/NAK Signal is Complete

```
#define MY_I2C_ID I2C_ID_1
```

```
if (PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(MY_I2C_ID))
{
    // Byte reception sequence complete
}
```

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a Stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.
- Zero or more data bytes may also have been sent or received and ACKed as appropriate

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a Stop condition using the PLIB_I2C_MasterStop function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the buffer to hold byte of data received.
`slaveTargetAddress7Bit`	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
`SLAVE_TARGET_7BIT_ADDRESS`	This constant is used as a place holder for the actual 7-bit target slave address that must be defined by the client code.
`slaveAddress10Bit`	This variable is used to demonstrate how to properly format and access a 10-bit target slave address
`SLAVE_TARGET_10BIT_ADDRESS`	This constant is used as a place holder for the actual 10-bit target slave address that must be defined by the client code.

Interrupt State Machine

The I2C state machine can be used in either a polled or an interrupt-driven manner. However, even when polled, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

The I2C module can cause three different interrupts:

- Slave Interrupt - A byte was sent or received in Slave mode
- Master Interrupt - A bus action has completed in Master mode
- Error Interrupt - An error has occurred in any mode (see Handling Errors)

Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for more information.

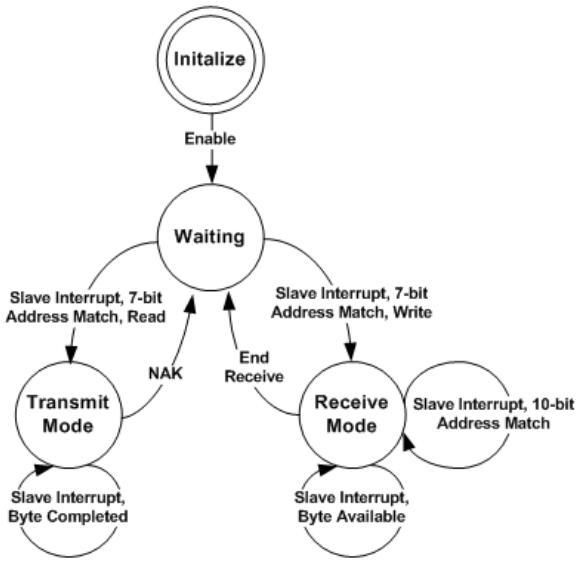
The I2C module will not start generating interrupts (setting the I2C interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described in the following section. Software has to respond appropriately once the interrupt has

occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described following the state machine diagram to which they refer.

Notes:

1. These state machine diagrams show the normal operations. Error and exception cases are explained in the Handling Errors section and are not shown in the diagrams.
2. The transition labels sometimes represent hardware interrupts (or the setting of an interrupt flag) and sometimes represent software actions. Refer to the transition tables to understand the transitions.
3. The Wait state is shared between all state machines (i.e., while in the Wait state, the I2C module can start a new master-mode transfer or be addressed by an external master).
4. The Start state is shared between both read and write master transfer state machines (i.e., when starting a new transfer, software can choose start either a read or a write transfer).

Slave Mode State Machine



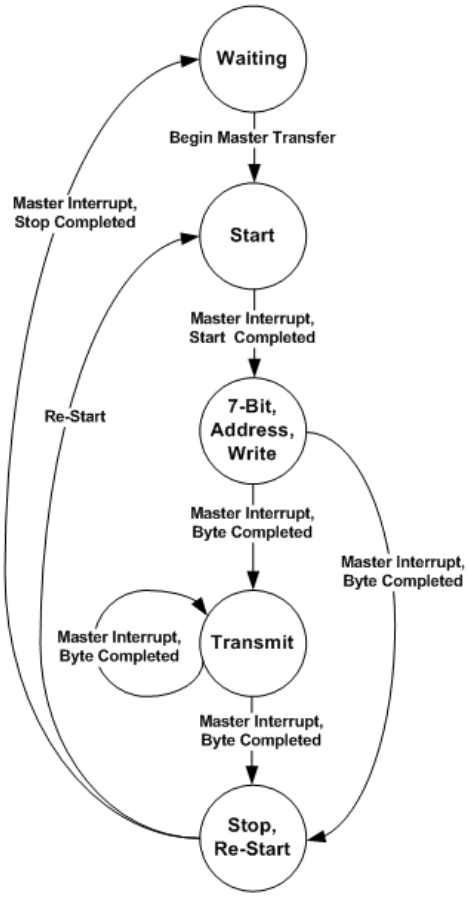
Slave Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the module continues to operate correctly.

Transition	From State	To State	Software Actions
Enable	Initialize	Wait	Enable the module, using the PLIB_I2C_Enable function
Slave Interrupt, 7-bit Address Match, Write	Wait	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that a 7-bit address (or the first byte of a 10-bit address) has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. Software must also release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte. **Note:** This transition occurs on a write transfer with either a 7-bit or 10-bit address (for the first of two address bytes).
Slave Interrupt, Byte Received	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and, if one has, get the byte from the receiver buffer by calling PLIB_I2C_ReceivedByteGet function. Then, software must release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte.
Slave Interrupt, 10-bit Address Match	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and (if 10-bit addressing is being used) call the PLIB_I2C_SlaveAddress10BitWasDetected function to verify that the second byte of the 10-bit address matched the low-order 8-bits of the local-slave 10-bit address. Next, software must get the address byte received from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. **Note:** This transition occurs on write transfer when using 10-bit (2-byte) addressing when the second address byte is received.
End Receive	Receive mode	Wait	None. Software is not aware of this transition. A receive transfer can end with a stop condition, a repeated start, or if the second-half of a 10-bit slave address does not match the programmed local slave address (and possibly the optional mask).

Slave Interrupt, 7-bit Address Match, Read	Wait	Transmit mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that an address has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. **Notes:** 1. This transition occurs on a read transfer with a 7-bit (1-byte) address or after a repeated-start with a 10-bit slave address. 2. The second byte of a 10-bit address is not sent after a repeated-start condition, only the first byte. The 10-bit slave device must remember if it was addressed at the beginning of this transfer. 3. After handling the address byte received, software should call the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the requested byte.
Slave Interrupt, Byte Completed	Transmit mode	Transmit mode	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. If the byte was ACKed, software must send the next byte by calling the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the next byte.
NAK	Transmit mode	Wait	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. The master will NAK the last byte it wants to receive. At that point, the transfer is effectively over and software must not attempt to send any more bytes. The module automatically transitions back to the Wait state when the Stop condition occurs, but software does not receive an interrupt at that point.

Master Mode Write Transfer State Machine

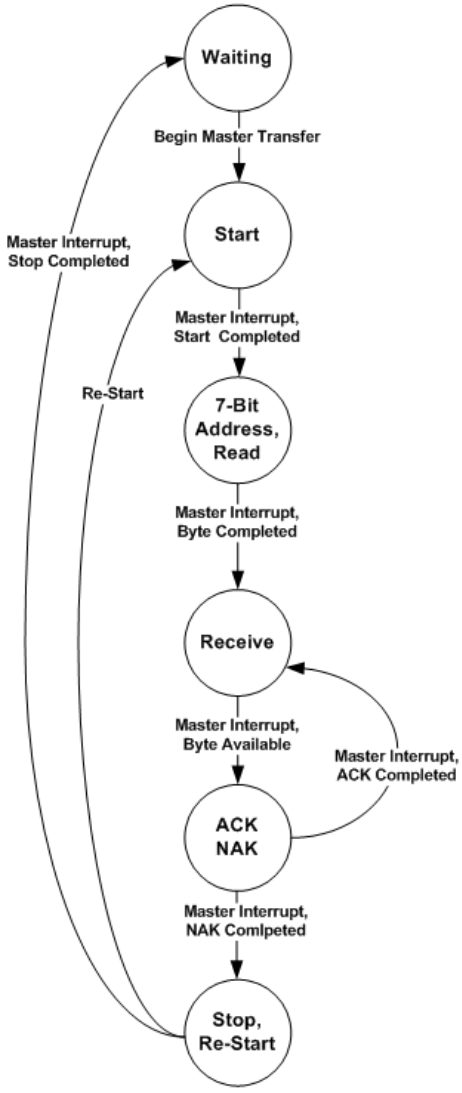


Master-Mode Write-Transfer State Transitions

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIsIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated start) condition.

Master Interrupt, Start Completed	Start	7-Bit Address, Write	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I^2C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReadyfunction to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte.
Master Interrupt, Byte Completed	7-bit Address, Write	Transmit	When the master interrupt occurs (or the PLIB_I2C_TransmitterByteHasCompleted function returns true) to indicate that the last byte sent has completed, software must check to ensure that the slave device ACKed the byte by calling the PLIB_I2C_TransmitterByteWasAcknowledged function. Software should also call the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may then send another byte to the slave using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function. If 10-bit addressing is being used for this transfer, the start condition and first byte of the 10-bit address must be followed by the second byte of the 10-bit address. Multiple slaves may respond to the first byte of the 10-bit address. It requires the second byte to uniquely identify the 10-bit slave. However, this is sent using the same process as if it were a data byte.
Master Interrupt, Byte Completed	Transmit	Transmit	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to ensure that the slave device ACKed the byte and the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may call the PLIB_I2C_TransmitterIsReady function and PLIB_I2C_TransmitterByteSendfunction to send another byte to the slave and remain in the Transmit state. If not, software should abort the transfer by sending a Stop condition and transition to the Stop, repeated Start state.
Master Interrupt, Byte Completed	7-bit Address, Write	Stop, repeated Start	If the slave NAKed any byte sent or if software wants to end or restart the transfer (perhaps to change direction) it can do so by sending a stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Master Interrupt, Byte Completed	Transmit	Stop, repeated Start	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to check to see if the slave ACKed or NAKed the byte. If the slave NAKed the byte or if software wants to end or restart the transfer (perhaps to change direction) it can do so by sending a Stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software transitions back to the Start state where it must decide what sort of transfer it will start next. **Note:** Read transfers will transition to the Start state in the Master Mode Read Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the Wait state (which is actually part of the Slave Mode state machine).

Master-Mode Read-Transfer State Machine



Master Mode Read Transfer State Transitions

In Master mode, transitions are initiated under software control and interrupts occur (the master interrupt flag is set, even when interrupts are disabled) when the action initiated has completed.

If at any point, an address or data byte is NAKed by the slave or an error occurs, usually the safest thing for the master to do is to end the transfer with a Stop condition and release the bus.

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIsIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated Start) condition.
Master Interrupt, Start Completed	Start	7-Bit Address, Read	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I2C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReady function to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte. **Note:** If using 10-bit addressing, this is a repeated transmission of the first byte of the 10-bit address since all 10-bit transfers must start off as write transfers (so both address bytes can be sent) then be restarted to become read transfers.
Master Interrupt, Byte Completed	7-Bit Address, Read	Receive	Software must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I2C module will cycle the SCL line eight times causing the slave to send the requested byte of data.

Master Interrupt, Byte Available	Receive	ACK NAK	When a byte has been received (which software can verify using the PLIB_I2C_ReceivedByteIsAvailable function), software must get the received byte from the RX buffer (using the PLIB_I2C_ReceivedByteGet function) and initiate transmission of the ACK/NAK bit (using the PLIB_I2C_ReceivedByteAcknowledge function). If it intends to receive another byte, software must ACK the byte just received. If it does not intend to receive another byte, software must NAK the byte just received.
Master Interrupt, ACK Completed	ACK, NAK	Receive	Software can verify that the ACK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software ACKed the previously received byte, it must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I2C module will cycle the SCL line eight times so that the slave can send the requested byte. Otherwise, software must have NAKed the previous byte to end the transfer.
Master Interrupt, NAK Completed	ACK, NAK	Stop, Restart	Software can verify that the NAK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software intends to continue the transfer without allowing the bus to go idle, it must initiate a repeated-start (using the PLIB_I2C_MasterStartRepeat function) and transition immediately to the Start state. If software intends to end the transfer and allow the bus to go idle, it must initiate a Stop condition using the PLIB_I2C_MasterStop function.
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software automatically transitions back to the "Start" state where it must decide if it will restart with a read or write transfer. **Note:** Write transfers will transition to the Start state in the Master Mode Write Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the waiting state (which is actually part of the Slave Mode state machine).

Managing Slave Addresses

An I2C address identifies which slave device is being targeted by a master device on a transfer-by-transfer basis. I2C addresses can be either 7-bits or 10-bits, as shown in the following sections, 7-Bit Slave Addresses and 10-Bit Slave Addresses.

Addresses that are 7-bits long are sent in a single byte where bit 0 identifies the direction of the remaining data in the transfer. Addresses that are 10-bits long are sent in two bytes using a reserved range in the 7-bit address space (addresses 0x78 through 0x7C) to encode the two high-order bits (bits A9 and A8) in the first byte. The second byte encodes the eight low-order bites (A7 through A0). These formats are in the following two sections.

7-Bit Slave Addresses

When using 7-bit slave addressing, the address byte is the first byte sent by any I2C master device immediately following a "Start" condition. The 7-bit address is shifted left by 1 bit to fill bits 7-to-1 in the address byte. Bit 0 in the 7-bit address byte indicates the transfer direction, as follows.

7-Bit Address Format

A6	A5	A4	A3	A2	A1	A0	R/$\overline{W}$

Bits	Usage
A6:A0	Bits 7 through 1 of the address byte contain the 7-bit slave address.
R/W	Bit 0 of the address byte is the read/write bit (0 = Read, 1 = Write).

See Operating as a Master Transmitter and Operating as a Master Receiver for examples.

10-Bit Slave Addresses

When using 10-bit slave addresses, the first address byte must be sent immediately following a start or repeated Start condition. The second byte must immediately follow the first byte when performing a write transfer. Read transfers must begin as write transfers so that both bytes may be sent. Immediately after sending the second address byte, the master will issue a repeated Start condition and resend the first address byte only (with the read/write bit cleared, indicating a read transfer. The second byte is not retransmitted. Since the transfer has been restarted, all listening 10-bit slave devices know that they will not be address again until the transfer has been ended with a Stop condition. However, devices that respond to 7-bit addresses might respond, so the first byte of the 10-bit address must be repeated to prevent that. The format of the 10-bit address is shown as follows.

10-Bit Address Format

1st Byte								2nd Byte							
1	1	1	1	0	A9	A8	R/$\overline{W}$	A7	A6	A5	A4	A3	A2	A1	A0

Bits	Usage
11110	Bits 7 through 3 of the first byte are constants. This guarantees that the first byte of the 10-bit address will fall within the reserved range of the 7-bit addresses.
A9:A8	Bits 2 through 1 of the first byte are the two high-order bits of the 10-bit address.

R/W	Bit 0 of the first byte is the Read/Write bit (0 = Read, 1 = Write).
A7:A0	The second byte contains the 8 low-order bits of the 10-bit address.

See Operating as a Master Transmitter and Operating as a Master Receiver for examples.

Operating in Slave Mode

In Slave mode, the "slave address" refers to the address (and optional mask) programmed into the I2C module to determine to which address or addresses the module will respond when addressed by an external master. The "raw" (unformatted) address is passed to or returned from the Managing Slave Addresses routines (without being formatted as previously shown). Likewise, the mask passed to or returned from the Slave Mask Control Functions (see the Library Interface section) is unformatted.

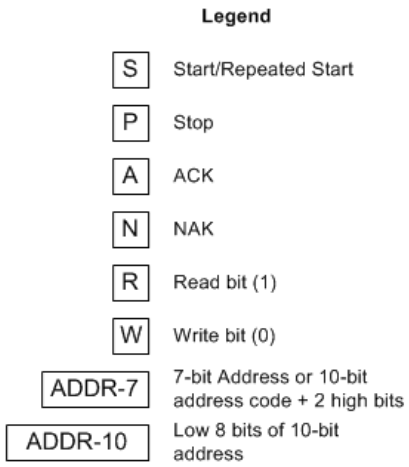
Operating in Master Mode

In Master mode, the "slave address" refers to the address transmitted on the I2C bus to identify the slave target of the transfer. In these cases, the slave address must be correctly formatted (as previously shown) and sent on the bus one byte at a time.

Forming Transfers

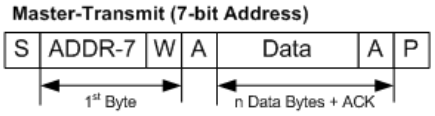
Forming Transfers

Transfers are formed by chaining the basic "building blocks" shown in the following transfer format legend.

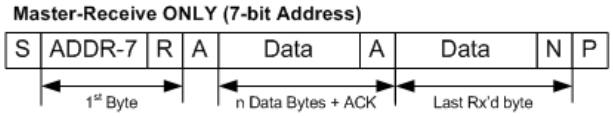


The following are all valid I2C transfer formats:

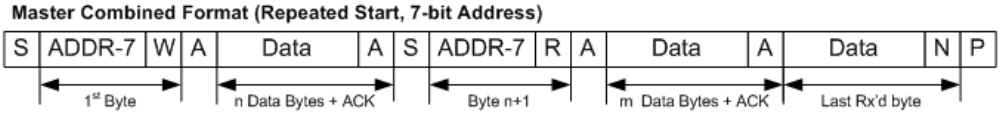
- This format allows a master to transmit *n* bytes of data to an I2C slave supporting a 7-bit address.



- This format allows a master to receive *n*+1 bytes of data from a "streaming" slave. Such a slave has no internal addressing capability. That is, it has no internal registers or addresses. It can only "stream" data to a master that reads from it.



- This format allows a master to write *n* bytes of data to a slave, followed immediately by reading *n*+1 bytes of data from a slave by sending a repeated Start condition after the first transfer. This format is useful for specifying an internal address to a device, and then reading data from that address. A repeated Start condition can also be used to chain transfers together without letting the bus go idle and potentially losing arbitration.

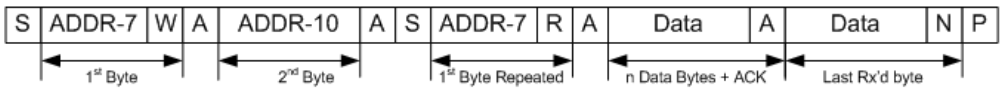


- The following format allows a master to send *n* bytes of data to a target supporting a 10-bit address.

Master Transmit (10-bit Address)

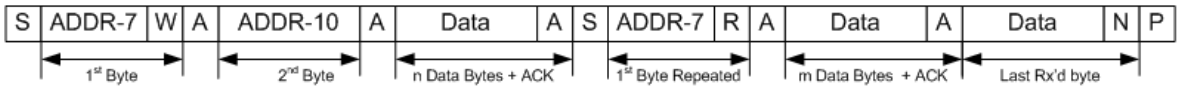
- This format is used to receive *n*+1 bytes of data from a slave supporting a 10-bit address. The repeated Start condition must be used because the second byte of the 10-bit address must be sent after the first byte of the 10-bit address. Then, the direction of the transfer must be switched to a "read". To do this, after the repeated Start, the master repeats the first byte of the 10-bit address with a "read" indication in the low-order bit and 10-bit slave devices must "remember" if they were being addressed once the transfer has started.

Master Receive (10-bit Address)



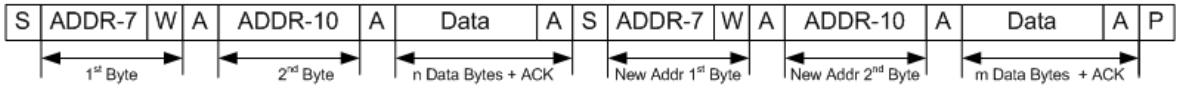
- This format allows a master to send *n* bytes of data to a slave device supporting a 10-bit address and immediately read *n*+1 bytes of data from the same device using the repeated Start and resending the first byte of the 10-bit address technique as shown in the previous format. This format is useful when the master needs to read data from a specific address within a device and it needs to send that address before performing the read.

Master Combined Format (10-bit write, 10-bit read)



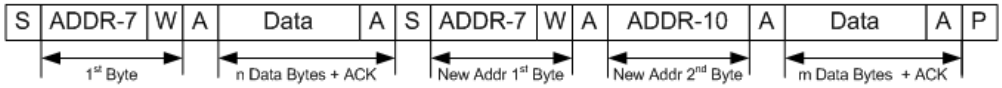
- This format allows a master to chain writes to multiple slaves supporting 10-bit addresses.

Master Combined Format (10-bit write, 10-bit write)



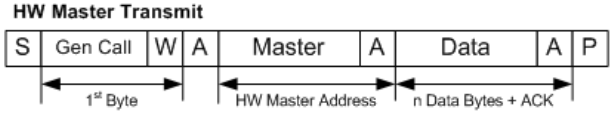
- This format allows a master to chain multiple writes to different slaves. Note that only the first write can be to a slave supporting a 7-bit address. The second (and any subsequent) writes must be to slaves with 10-bit addresses once any 10-bit device has been addressed.

Master Combined Format (7-bit write, 10-bit write)



- This format can be used by a "hardware" master that does not have a programmable address. This allows the hardware master to broadcast to all slaves on the bus using the "General Call" address, followed by the master's address. (The master address is generally the same as the device's slave address if it can also act as a slave device.) This format is not normally used by a microcontroller acting as a master. However, it may be received by a microcontroller if such a hardware master exists on the bus.

HW Master Transmit



Handling Errors

There are three basic types of errors that can occur during various I2C operations:

- Transmitter Overflow
- Receiver Overflow
- Arbitration Loss

Handling Transmitter Overflow Errors

A transmitter overflow error occurs when the software attempts to write to the TX buffer (by calling the PLIB_I2C_TransmitterByteSend function) while the transmitter is busy. When this occurs, the write is not allowed and the transmitter overflow status bit is set. This can be identified by calling the PLIB_I2C_TransmitterOverflowHasOccurred function. Additional attempts to write to the TX buffer will not be allowed until the transmitter overflow error is cleared by calling the PLIB_I2C_TransmitterOverflowClear function. This sort of error should be either avoided by checking the PLIB_I2C_TransmitterIsReady function before attempting to send a byte or by checking for the error after attempting to send a byte.

Interrupts: The transmitter overflow error does not trigger an interrupt.

Handling Receiver Overflow Errors

A receiver overflow error occurs when an incoming byte has been completely received and is ready to be transferred to the RX buffer, but software has not yet gotten the previous byte from the RX buffer. When this happens the receiver overflow status bit is set and the incoming byte will be lost. This can happen when software does not respond quickly enough to a slave or master interrupt when receiving data. Software should check for this error before calling the PLIB_I2C_ReceivedByteGet function to get a byte from the RX buffer to determine if any data has been lost. This error can be identified by calling the PLIB_I2C_ReceiverOverflowHasOccurred function. This error must be cleared by calling the PLIB_I2C_ReceiverOverflowClear function or no additional data will be received.

Interrupts: The receiver overflow error does not trigger an interrupt (although the incoming data byte does, even though it is lost.)

Handling Arbitration Loss Errors

Strictly speaking, arbitration loss is not an error. Rather it is a normal part of bus operation in a multi-master environment. Arbitration loss occurs when more than one master attempts to transmit on the bus at the same (or nearly the same) time. When this happens, at some point one of the masters will attempt to transmit a binary one ('1') when the other attempts to transmit a binary zero ('0'). At that point, the master transmitting the zero will win the transmission, because the zero transmission will dominate on the bus. The master detecting that the bus signals a zero when it attempted to transmit a one will lose arbitration and must immediately stop transmitting on the bus (which happens automatically).

Arbitration loss can be detected during any of the following actions:

- Transmission of a Start condition
- Transmission of a repeated Start condition
- Transmission of an address or data byte
- Transmission of an acknowledge bit
- Transmission of a Stop condition

Arbitration loss can be identified by calling the PLIB_I2C_ArbitrationLossHasOccurred function. The arbitration loss condition can be cleared by calling the PLIB_I2C_ArbitrationLossClear function. Once arbitration loss occurs, the losing master must wait until the bus is once again idle (at which time a master interrupt will occur if arbitration loss has occurred) and retransmit the transfer from the beginning.

Interrupts: The I2C module will trigger an error interrupt when arbitration loss occurs.

Other Features

Other Features

Note: The following features are not available on all devices. Please refer to the specific device data sheet to determine which features are supported by your device.

General Call Address Support

The I2C General Call address (reserved address 0) allows a master to address all slave devices on the bus. During slave operation the module can be made to respond to or ignore the general call address using the PLIB_I2C_GeneralCallEnable function or the PLIB_I2C_GeneralCallDisable function. When a slave address byte is received, software can identify if it was the general call address using the PLIB_I2C_SlaveAddressIsGeneralCall function (or by checking the value of the address byte received).

Note: The general call address is always 7-bits.

SMBus Support

The I2C module can support communication on a System Management Bus (SMBus) by calling the PLIB_I2C_SMBEnable function to adjust the electrical characteristics of the SMBus.

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the I2C modules are shut down. If the device enters Sleep mode in the middle of an I2C master transmission or reception operation, the operation is aborted. If the device enters Sleep mode when the module is operating in slave mode, the external master's clock will continue run the slave state machine. If a slave address match occurs, an interrupt will be provided to wake up the device.

Note: Indeterminate results can occur if an error occurs during Sleep mode.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

I2C operation during Idle mode can be controlled using the PLIB_I2C_StopInIdleEnable function or the PLIB_I2C_StopInIdleDisable function.

By default, the I2C module continues to operate in Idle mode and provide full functionality.

Operation with DMA

Software activity must occur between the transfer of each byte, making DMA inappropriate for use with the I2C module.

Configuring the Library

This library is appropriate for microcontrollers with a dedicated I2C module (not an SSP or MSSP module). The library is configured for the supported I2C module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Initialization Functions

	Name	Description
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification
	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode

b) General Status Functions

	Name	Description
	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
	PLIB_I2C_BusIsIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected

c) Baud Rate Generator Control Functions

	Name	Description
	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
	PLIB_I2C_BaudRateSet	Sets the desired baud rate.

d) Slave Address Control Functions

	Name	Description
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.

	PLIB_I2C_SlaveAddressModeIs10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.

e) Slave Mask Control Functions

	Name	Description
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.
	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .

f) Slave Control Functions

	Name	Description
	PLIB_I2C_AcksequenceIsInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.

g) Master Control Functions

	Name	Description
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.
	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.
	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.

h) Transmitter Control Functions

	Name	Description
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.
	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

i) Receiver Control Functions

	Name	Description
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedByteIsAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.

j) Feature Existence Functions

	Name	Description
	PLIB_I2C_ExistsAcksequenceProgress	Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.
	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module.
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module.
	PLIB_I2C_ExistsBusIsIdle	Identifies whether the BusIdle feature exists on the I2C module.
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module.
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module.
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module.
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module.
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module.
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module.
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataHoldEnable	Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.

k) Data Types and Constants

	Name	Description
	I2C_MODULE_ID	This is type I2C_MODULE_ID.

Description

This section describes the Application Programming Interface (API) functions of the I2C Peripheral Library.

Refer to each section for a detailed description.

a) General Initialization Functions

PLIB_I2C_Disable Function

Disables the specified I2C module.

File

plib_i2c.h

C

```
void PLIB_I2C_Disable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the specified I2C module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_Disable( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_Disable(I2C_MODULE_ID index)

PLIB_I2C_Enable Function

Enables the specified I2C module.

File

plib_i2c.h

C

```
void PLIB_I2C_Enable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the specified I2C module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_I2C_ID               I2C_ID_1
#define MY_CLOCK_FREQUENCY      80000000
#define MY_BAUD_RATE            10000
#define MY_SLAVE_ADDRESS        0x23

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
PLIB_I2C_SMBDisable ( MY_I2C_ID );
PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_ReservedAddressProtectEnable ( MY_I2C_ID );
PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY, MY_BAUD_RATE );

PLIB_I2C_SlaveAddress7BitSet( MY_I2C_ID, MY_SLAVE_ADDRESS );
PLIB_I2C_Enable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_Enable(I2C_MODULE_ID index)

PLIB_I2C_GeneralCallDisable Function

Disables the I2C module from recognizing the general call address.

File

plib_i2c.h

C

```
void PLIB_I2C_GeneralCallDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from recognizing the general call address when operating as a slave receiver.

Remarks

The General-call feature can be re-enabled by calling the PLIB_I2C_GeneralCallEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCall in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallDisable(MY_I2C_ID);
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_GeneralCallDisable (I2C_MODULE_ID index)

PLIB_I2C_GeneralCallEnable Function

Enables the I2C module to recognize the general call address.

File

plib_i2c.h

C

```
void PLIB_I2C_GeneralCallEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to recognize the general call address when operating as a slave receiver.

Remarks

The General-call feature can be disabled by calling the PLIB_I2C_GeneralCallDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCall in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_GeneralCallEnable (I2C_MODULE_ID index)

PLIB_I2C_HighFrequencyDisable Function

Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.

File

plib_i2c.h

C

```
void PLIB_I2C_HighFrequencyDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from using high-frequency signaling, preventing it from using the 400 kHz and 1 MHz signaling rates.

Remarks

The high-frequency feature can be re-enabled by calling the PLIB_I2C_HighFrequencyEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsHighFrequency in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_HighFrequencyDisable (I2C_MODULE_ID index)

PLIB_I2C_HighFrequencyEnable Function

Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.

File

plib_i2c.h

C

```
void PLIB_I2C_HighFrequencyEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to use high-frequency signaling, allowing it to use the 400 kHz and 1 MHz signaling rates.

Remarks

The high-speed feature can be disabled by calling the PLIB_I2C_HighSpeedEnable function.

This feature must be enabled if frequencies higher than 100 kHz programmed using the PLIB_I2C_BaudRateSet function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsHighFrequency in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_HighFrequencyEnable (I2C_MODULE_ID index)

PLIB_I2C_IPMIDisable Function

Disables the I2C module's support for the IPMI specification

File

plib_i2c.h

C

```
void PLIB_I2C_IPMIDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module's support for the IPMI specification.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsIPMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_IPMIDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_IPMIDisable (I2C_MODULE_ID index)

PLIB_I2C_IPMIEnable Function

Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).

File

plib_i2c.h

C

```
void PLIB_I2C_IPMIEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to support the IPMI specification.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsIPMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
PLIB_I2C_IPMIEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_IPMIEnable (I2C_MODULE_ID index)

PLIB_I2C_ReservedAddressProtectDisable Function

Disables the I2C module from protecting reserved addresses, allowing it to respond to them.

File

plib_i2c.h

C

```
void PLIB_I2C_ReservedAddressProtectDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from protecting reserved addresses, allowing it to respond to them when they match the module's slave address and mask.

Remarks

The reserved-address-protect feature can be re-enabled by calling the PLIB_I2C_ReservedAddressProtectEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReservedAddressProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReservedAddressProtectDisable (I2C_MODULE_ID index)

PLIB_I2C_ReservedAddressProtectEnable Function

Enables the I2C module to protect (not respond to) reserved addresses.

File

plib_i2c.h

C

```
void PLIB_I2C_ReservedAddressProtectEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to protect reserved addresses by not responding to them, even if they match the module's slave address and mask.

Remarks

The reserved-address-protect feature can be disabled by calling the PLIB_I2C_ReservedAddressProtectDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReservedAddressProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReservedAddressProtectEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockStretchingDisable Function

Disables the I2C module from stretching the slave clock.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockStretchingDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from stretching the slave clock to allow time for software to respond to bytes received from the master.

Remarks

The clock stretching feature can be re-enabled by calling the PLIB_I2C_SlaveClockStretchingEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsClockStretching in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockStretchingDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockStretchingEnable Function

Enables the I2C module to stretch the slave clock.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockStretchingEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to stretch the slave clock to allow time for software to respond to bytes received from the master.

Remarks

The clock stretching feature can be disabled by calling the PLIB_I2C_SlaveClockStretchingDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsClockStretching in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockStretchingEnable (I2C_MODULE_ID index)

PLIB_I2C_SMBDisable Function

Disable the I2C module support for SMBus electrical signaling levels.

File

plib_i2c.h

C

```
void PLIB_I2C_SMBDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module support for SMBus electrical signaling levels.

Remarks

The SMB feature can be re-enabled by calling the PLIB_I2C_SMBEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSMBus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SMBDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SMBDisable (I2C_MODULE_ID index)

PLIB_I2C_SMBEnable Function

Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.

File

plib_i2c.h

C

```
void PLIB_I2C_SMBEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to support SMBus electrical signaling levels.

Remarks

The SMB feature can be disabled by calling the PLIB_I2C_SMBDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSMBus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SMBEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SMBEnable (I2C_MODULE_ID index)

PLIB_I2C_StopInIdleDisable Function

Disables the Stop-in-Idle feature.

File

plib_i2c.h

C

```
void PLIB_I2C_StopInIdleDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the Stop-in-Idle feature, preventing the I2C module from stopping when the processor enters Idle mode.

Remarks

The Stop-in-Idle feature can be re-enabled by calling the PLIB_I2C_StopInIdleEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopInIdleDisable (I2C_MODULE_ID index)

PLIB_I2C_StopInIdleEnable Function

Enables the I2C module to stop when the processor enters Idle mode

File

plib_i2c.h

C

```
void PLIB_I2C_StopInIdleEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to stop when the processor enters Idle mode.

Remarks

The Stop-in-Idle feature can be disabled by calling the PLIB_I2C_StopInIdleDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_StopInIdleEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopInIdleEnable (I2C_MODULE_ID index)

b) General Status Functions

PLIB_I2C_ArbitrationLossClear Function

Clears the arbitration loss status flag

File

plib_i2c.h

C

```
void PLIB_I2C_ArbitrationLossClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the arbitration loss status flag.

Remarks

This flag is set by hardware when bus arbitration loss occurs. Its status can be tested using the PLIB_I2C_ArbitrationLossHasOccurred function. This flag should be cleared by software after the arbitration loss has been handled. To handle the error, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsArbitrationLoss in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ArbitrationLossClear (I2C_MODULE_ID index)

PLIB_I2C_ArbitrationLossHasOccurred Function

Identifies if bus arbitration has been lost.

File

plib_i2c.h

C

```
bool PLIB_I2C_ArbitrationLossHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If software if a bus collision occurred, resulting in loss of bus arbitration
- false - If no bus collision occurred

Description

This function identifies if bus arbitration has been lost.

Remarks

The arbitration status should be checked after any Master mode transmission or if an error interrupt occurs. If a bus collision occurs, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

This flag should be cleared by software using the PLIB_I2C_ArbitrationLossClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsArbitrationLoss in your application to determine whether this feature is available.

Preconditions

Bus collisions can occur during any master-mode transmission including:

- Sending a Start condition
- Sending a repeated Start condition
- Sending an address or data byte
- sending an ACK/NAK bit
- Sending a Stop condition

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ArbitrationLossHasOccurred (I2C_MODULE_ID index)

PLIB_I2C_BusIsIdle Function

Determines whether the I2C bus is idle or busy.

File

plib_i2c.h

C

```
bool PLIB_I2C_BusIsIdle(I2C_MODULE_ID index);
```

Returns

- true - The bus is currently idle. It is safe to start a master transfer.
- false - The bus is currently busy. Do not start a master transfer.

Description

This function checks to see if the I2C bus is currently idle or if there is some activity currently taking place.

Remarks

When this function returns true it does not guarantee that a bus arbitration loss cannot occur. Two or more masters can start a transfer within the minimum start signal hold time. (Refer to the I2C specification for a definition of the minimum Start condition hold time.)

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBusIsIdle in your application to determine whether this feature is available.

Preconditions

The module must be configured appropriately and enabled.

Example

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_BusIsIdle ( MY_I2C_ID ))
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_BusIsIdle (I2C_MODULE_ID index)

PLIB_I2C_StartClear Function

Clears the start status flag

File

plib_i2c.h

C

```
void PLIB_I2C_StartClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the start status flag.

Remarks

This flag is cleared automatically by the hardware when a Stop condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStartDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected( MY_I2C_ID ) )
{
    // Handle Start condition

    PLIB_I2C_StartClear(MY_I2C_ID);
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StartClear (I2C_MODULE_ID index)

PLIB_I2C_StartWasDetected Function

Identifies when a Start condition has been detected.

File

plib_i2c.h

C

```
bool PLIB_I2C_StartWasDetected(I2C_MODULE_ID index);
```

Returns

- true - A Start Condition has been detected
- false - No Start condition has been detected since the last time a Stop condition was detected (or the module was initialized)

Description

This function identifies when a Start condition has been detected.

Remarks

This flag is cleared automatically by the hardware when a stop condition is detected, but it can also be cleared by software using PLIB_I2C_StartClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStartDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected ( MY_I2C_ID ) )
{
    // Handle Start condition
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_StartWasDetected (I2C_MODULE_ID index)

PLIB_I2C_StopClear Function

Clears the stop status flag

File

plib_i2c.h

C

```
void PLIB_I2C_StopClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the stop status flag.

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition

    PLIB_I2C_StopClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopClear (I2C_MODULE_ID index)

PLIB_I2C_StopWasDetected Function

Identifies when a Stop condition has been detected

File

plib_i2c.h

C

```
bool PLIB_I2C_StopWasDetected(I2C_MODULE_ID index);
```

Returns

- true - A Stop condition has been detected
- false - No Stop condition has been detected since the last time a Start condition was detected (or the module was initialized)

Description

This function identifies when a Stop condition has been detected.

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software using the PLIB_I2C_StopClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_StopWasDetected (I2C_MODULE_ID index)

c) Baud Rate Generator Control Functions

PLIB_I2C_BaudRateGet Function

Calculates the I2C module's current SCL clock frequency.

File

plib_i2c.h

C

```
I2C_BAUD_RATE PLIB_I2C_BaudRateGet(I2C_MODULE_ID index, uint32_t clockFrequencyHz);
```

Returns

SCL frequency currently used

Description

This function calculates the I2C module's current SCL clock frequency.

Remarks

The actual frequency provided may be slightly different than the frequency requested due to truncation errors. The actual frequency observed on the SCL line may be lower due to clock stretching.

The MY_CLOCK_FREQUENCY macro in the example is used as placeholder for the actual clock frequency.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

The returned value may not be valid if PLIB_I2C_BaudRateSet has not been previously called to set the SCL clock frequency.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_CLOCK_FREQ_INPUT     80000000
uint32_t myBaudRate;

myBaudRate = PLIB_I2C_BaudRateGet ( MY_I2C_ID, MY_CLOCK_FREQ_INPUT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz) provided for the I2C module

Function

I2C_BAUD_RATE PLIB_I2C_BaudRateGet (I2C_MODULE_ID index, uint32_t clockFrequencyHz)

PLIB_I2C_BaudRateSet Function

Sets the desired baud rate.

File

plib_i2c.h

C

```
void PLIB_I2C_BaudRateSet(I2C_MODULE_ID index, uint32_t clockFrequencyHz, I2C_BAUD_RATE baudRate);
```

Returns

None.

Description

This function sets the desired baud rate so that the I2C module will operate at the desired clock frequency (on the SCL line of the bus.)

Remarks

IMPORTANT: The I2C module's high-frequency mode must be enabled for frequencies higher than 100 kHz using the PLIB_I2C_HighFrequencyEnable function. Otherwise, the high-frequency mode must be disabled using the PLIB_I2C_HighFrequencyDisable function.

The actual frequency selected may be slightly different than the frequency requested due to truncation errors. Use the PLIB_I2C_BaudRateGet function to obtain the actual baud rate value that has been programmed.

The actual frequency observed on the SCL line may be lower due to clock stretching.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

The source clock, being sent to the I2C module (internal to the microcontroller) must be operating at the frequency passed.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_CLOCK_FREQUENCY_INPUT 80000000

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY_INPUT, 40000 );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz)
baudRate	The desired baud rate value. This should be an appropriate value for the frequency and microcontroller in use.

Function

void PLIB_I2C_BaudRateSet (I2C_MODULE_ID index, uint32_t clockFrequencyHz,

I2C_BAUD_RATE baudRate)

d) Slave Address Control Functions

PLIB_I2C_SlaveAddress10BitGet Function

Identifies the current 10-bit Slave mode address.

File

plib_i2c.h

C

```
uint16_t PLIB_I2C_SlaveAddress10BitGet(I2C_MODULE_ID index);
```

Returns

The 10-bit slave address to which the module is currently set to respond.

Description

This function identifies the 10-bit slave address to which the module will currently respond.

Remarks

The 16-bit address will be right-aligned in the 16-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

The address provided may not be valid if it has not been previously set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint16_t slaveAddress;

if (PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddress = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint16_t PLIB_I2C_SlaveAddress10BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress10BitSet Function

Selects 10-bit Slave mode addressing and sets the address value.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddress10BitSet(I2C_MODULE_ID index, uint16_t address);
```

Returns

None.

Description

This function selects 10-bit Slave mode addressing sets the slave address to which the module will respond when operating in Slave mode.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the PLIB_I2C_SlaveAddress10BitSetMasked may be used to support the mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_10_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT    0x23

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 10-bit slave address to which the module will respond (The address should be right-aligned in the 16-bit parameter, without any read/write bit in the 0 position)

Function

void PLIB_I2C_SlaveAddress10BitSet (I2C_MODULE_ID index, uint16_t address)

PLIB_I2C_SlaveAddress10BitWasDetected Function

Detects reception of the second byte of a 10-bit slave address.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddress10BitWasDetected(I2C_MODULE_ID index);
```

Returns

- true - If the second byte of the local 10-bit address has been received
- false - If the second byte of the local 10-bit address has not been received

Description

This function detects if the second byte of a 10-bit slave address (containing the low-order 8 bits) matching the local address has been received.

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddressDetect in your application to determine whether this feature is available.

Preconditions

The module must have been configured for 10-bit addressing in Slave mode, enabled, and the first byte of the 10-bit local slave address must have already been received and matched.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t  i2cReadData;

if ( PLIB_I2C_SlaveAddress10BitWasDetected ( MY_I2C_ID ) )
{
    i2cReadData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddress10BitWasDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress7BitGet Function

Identifies the current 7-bit Slave mode address.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_SlaveAddress7BitGet(I2C_MODULE_ID index);
```

Returns

The 7-bit slave address to which the module is currently set to respond.

Description

This function identifies the 7-bit slave address to which the module will currently respond.

Remarks

The 7-bit address will be right-aligned in the 8-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress7Bit in your application to determine whether this feature is available.

Preconditions

The address provided may not be valid if it has not been previously set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slave_address7bit;

if (!PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address7bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_SlaveAddress7BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress7BitSet Function

Selects 7-bit Slave mode addressing and sets the slave address value.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddress7BitSet(I2C_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function selects 7-bit Slave mode addressing sets the address to which the module will respond when operating in Slave mode.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the PLIB_I2C_SlaveAddress7BitSetMasked may be used to support the mask feature. Refer to the specific device

data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_7_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress7Bit in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDRESS_7_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 7-bit slave address to which the module will respond (The address should be right-aligned in the 8-bit parameter, without any read/write bit in the 0 position)

Function

void PLIB_I2C_SlaveAddress7BitSet (I2C_MODULE_ID index, uint8_t address)

PLIB_I2C_SlaveAddressHoldDisable Function

Disables Address holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddressHoldDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables address holding.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveAddressHoldEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveAddressHoldDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressHoldEnable Function

Enables address holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddressHoldEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables address holding. If address byte is received, following the 8th falling edge of clock for a clock line will be held low.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveAddressHoldEnable in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveAddressHoldEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressIsDetected Function

Detects if the most recent byte received is a data or an address byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressIsDetected(I2C_MODULE_ID index);
```

Returns

- true - If the byte received is an address byte
- false - If the byte received is a data byte

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveReadRequest in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t getData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsDetected ( MY_I2C_ID ) )
    {
        if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
        {
            // Begin slave transmit mode
        }
        else
        {
            // Begin slave receive mode
        }
    }

    // Read the data and release the bus
    getData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressIsDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressIsGeneralCall Function

Identifies if the current slave address received is the general call address.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressIsGeneralCall(I2C_MODULE_ID index);
```

Returns

- true - If the slave address received is the general call address
- false - if the slave address received is not the general call address

Description

This function identifies if the current slave address received is the general call address.

Remarks

This bit is set when the general call address has been received.

This bit is automatically cleared by hardware when a Stop condition occurs and cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCallAddressDetect in your application to determine whether this feature is available.

Preconditions

A slave address must have been received.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
uint8_t  slaveAddress7Bit;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsGeneralCall( MY_I2C_ID ) )
    {
        // Handle general call address
    }

    slaveAddress7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressIsGeneralCall (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressModeIs10Bits Function

Identifies if the current slave address mode is 7-bits or 10-bits.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressModeIs10Bits(I2C_MODULE_ID index);
```

Returns

- true - If the current slave addressing mode is 10-bits
- false - if the current slave addressing mode is 7-bits

Description

This function identifies if the current slave addressing mode is 7-bits or 10-bits.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

The mode provided may not be valid if the address mode has not been previously set.

Example

```
uint8_t  slave_address7Bit;
uint16_t slave_address10Bit;

if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address10Bit = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
}
else
{
    slave_address7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressModeIs10Bits (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataIsDetected Function

Detects if the most recent byte received is a data or an address byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveDataIsDetected(I2C_MODULE_ID index);
```

Returns

- true - If the byte received is a data byte
- false - If the byte received is an address byte

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Remarks

This function should only be used by slave receivers.

This function returns the opposite of the PLIB_I2C_SlaveAddressIsDetected function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveDataDetect in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t dataReceived;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveDataIsDetected( MY_I2C_ID ) )
    {
        // Read the data and release the bus
        dataReceived = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
        PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module.

Function

bool PLIB_I2C_SlaveDataIsDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveReadIsRequested Function

Detects if the request from the master was a read or write.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveReadIsRequested(I2C_MODULE_ID index);
```

Returns

- true - If an external master is requesting data (slave read/transmit)

- false - If an external master is sending data (slave write/receive)

Description

This function identifies if a slave read (transmit) or a slave write (receive) was requested by the master that addressed the module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveReadRequest in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveTxData;
uint8_t slaveRxData;

if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
    {
        PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, slaveTxData );
    }
}
else
{
    slaveRxData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveReadIsRequested (I2C_MODULE_ID index)

e) Slave Mask Control Functions

PLIB_I2C_SlaveMask10BitGet Function

Identifies the current 10-bit Slave mode address mask.

File

plib_i2c.h

C

```
uint16_t PLIB_I2C_SlaveMask10BitGet(I2C_MODULE_ID index);
```

Returns

The 10-bit slave address mask that the module is currently using to determine to which addresses the module will respond.

Description

This function identifies the 10-bit slave address mask that is currently being used to determine which slave addresses the module will respond.

Remarks

The 10-bit address mask will be Right-aligned in the 16-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Example

```
#define MY_I2C_ID I2C_ID_1

uint16_t slave_address;
uint16_t slave_addressMask;

if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address     = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
    slave_addressMask = PLIB_I2C_SlaveMask10BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint16_t PLIB_I2C_SlaveMask10BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveMask10BitSet Function

Sets the 10-bit mask for Slave mode addressing.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveMask10BitSet(I2C_MODULE_ID index, uint16_t mask);
```

Returns

None.

Description

This function sets the 10-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and thus respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the PLIB_I2C_SlaveMask10BitSet function will not be supported. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_10_BIT and MY_SLAVE_ADDRESS_MASK_10_BIT macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID                       I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT            0x23
#define MY_SLAVE_ADDR_MASK_10_BIT       0x12

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
PLIB_I2C_SlaveMask10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_10_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this
value must be zero (0).) (This value must also be right	aligned in the 16-bit parameter.)

Function

void PLIB_I2C_SlaveMask10BitSet (I2C_MODULE_ID index, uint16_t mask)

PLIB_I2C_SlaveMask7BitGet Function

Identifies the current 7-bit Slave mode address mask.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_SlaveMask7BitGet(I2C_MODULE_ID index);
```

Returns

The 7-bit Slave mode address mask that the module is currently using to determine to which addresses the module will respond.

Description

This function identifies the 7-bit Slave mode address mask that is currently being used to determine which slave addresses the module will respond.

Remarks

The 7-bit address mask will be right-aligned in the 8-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveAddr;
uint8_t slaveAddressMsk;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddr     = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
    slaveAddressMsk = PLIB_I2C_SlaveMask7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_SlaveMask7BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveMask7BitSet Function

Sets the 7-bit mask for Slave mode addressing .

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveMask7BitSet(I2C_MODULE_ID index, uint8_t mask);
```

Returns

None.

Description

This function sets the 7-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and will respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the PLIB_I2C_SlaveMask7BitSet function will not be supported. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_7_BIT and MY_SLAVE_ADDRESS_MASK_7_BIT macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID                   I2C_ID_1
#define MY_SLAVE_ADDR_7_BIT         0x23
#define MY_SLAVE_ADDR_MASK_7_BIT    0x12

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_7_BIT );
PLIB_I2C_SlaveMask7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_7_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this parameter must be zero (0).)

Function

void PLIB_I2C_SlaveMask7BitSet (I2C_MODULE_ID index, uint8_t mask)

f) Slave Control Functions

PLIB_I2C_AcksequenceIsInProgress Function

Checks whether the acknowledge sequence is in progress or it is completed.

File

plib_i2c.h

C

```
bool PLIB_I2C_AcksequenceIsInProgress(I2C_MODULE_ID index);
```

Returns

- true - Acknowledge sequence is in progress.
- false - Acknowledge sequence is not started or completed.

Description

This function checks whether the acknowledge sequence is in progress or it is completed.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsAcksequenceProgress in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_AcksequenceIsInProgress ( MY_I2C_ID ) )
{
  //Transmission completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_AcksequenceIsInProgress (I2C_MODULE_ID index)

PLIB_I2C_DataLineHoldTimeSet Function

Sets the data line hold time.

File

plib_i2c.h

C

```
void PLIB_I2C_DataLineHoldTimeSet(I2C_MODULE_ID index, I2C_SDA_MIN_HOLD_TIME sdaHoldTimeNs);
```

Returns

None.

Description

This function sets the data line hold time.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsDataLineHoldTime in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_PLIB_I2C_DataLineHoldTimeSet ( MY_I2C_ID, I2C_SDA_MIN_HOLD_TIME_300NS );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module.
sdaHoldTimeNs	SDA hold time in nanoseconds.

Function

void PLIB_I2C_DataLineHoldTimeSet (I2C_MODULE_ID index, I2C_SDA_MIN_HOLD_TIME sdaHoldTimeNs)

PLIB_I2C_SlaveBufferOverwriteDisable Function

Disables buffer overwrite.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBufferOverwriteDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables buffer overwrite. If the buffer overwrite is disabled, on data receive, when the previous data is not read, the buffer will not be overwritten.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveBufferOverwrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBufferOverwriteDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBufferOverwriteEnable Function

Enables buffer overwrite.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBufferOverwriteEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables buffer overwrite. If the buffer overwrite is enabled, on data receive, even if the previous data is not read, the buffer will be

filled with new data and acknowledge will be generated.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveBufferOverwrite in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBufferOverwriteEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBusCollisionDetectDisable Function

Disables bus collision detect interrupts.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBusCollisionDetectDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables bus collision detect interrupts.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveBusCollisionDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBusCollisionDetectDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_SlaveBusCollisionDetectDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBusCollisionDetectEnable Function

Enables slave bus collision interrupts.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBusCollisionDetectEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables bus collision interrupts.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveBusCollisionDetect in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBusCollisionDetectEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockHold Function

Holds the SCL clock line low after receiving a byte.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockHold(I2C_MODULE_ID index);
```

Returns

None.

Description

This function allows an I2C slave to stretch the SCL clock line, holding it low to throttle data transfer from a master transmitter.

Remarks

This function will cause the SCL line to be forced low, after the current byte has been fully received.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveClockHold in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockHold ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockHold (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockRelease Function

Releases a previously held SCL clock line.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockRelease(I2C_MODULE_ID index);
```

Returns

None.

Description

This function allows a slave receiver to release a previously held SCL clock line, allowing it to go high and allowing data transfer to continue.

Remarks

Calling this function when the clock has not been held will not cause any problems.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveClockHold in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master, and the SCL clock line should have been previously held (forced low) by the I2C module.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockRelease (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataHoldDisable Function

Disables data holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveDataHoldDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables data holding.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveDataHoldEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveDataHoldDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataHoldEnable Function

Enables data holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveDataHoldEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables data holding. If a data byte is received, following the 8th falling edge of clock for a clock line will be held low.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveDataHoldEnable in your application to determine whether this feature is available.

This API is supported only for the slave mode of I2C.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveDataHoldEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStartDisable Function

Disables the feature of generating interrupt on start condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStartDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the feature of generating interrupt on start condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStartDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStartDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStartEnable Function

Enables the feature of generating interrupt on start condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStartEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the feature of generating interrupt on start condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStart in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStartEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStartEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStopDisable Function

Disables the feature of generating interrupt on stop condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStopDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the feature of generating interrupt on stop condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStop in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStopDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStopEnable Function

Enables the feature of generating interrupt on stop condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStopEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the feature of generating interrupt on stop condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStop in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStopEnable (I2C_MODULE_ID index)

g) Master Control Functions

PLIB_I2C_MasterReceiverClock1Byte Function

Drives the bus clock to receive 1 byte of data from a slave device.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterReceiverClock1Byte(I2C_MODULE_ID index);
```

Returns

None.

Description

This function drives the bus clock to receive 1 byte of data from a slave device.

Remarks

The module stops driving the bus clock after the reception of a single byte of data and this function must be called again to receive another byte.

After the module has finished receiving a data byte (determined by responding to an I2C master interrupt and/or by checking the

PLIB_I2C_ReceivedByteIsAvailable function), software should check the PLIB_I2C_ReceiverOverflowHasOccurred function to ensure that no data was lost because the receiver buffer was full when the byte was completely received and ready to be placed into the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterReceiverClock1Byte in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the module must be the intended receiver of the next byte of data.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterReceiverClock1Byte ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterReceiverClock1Byte (I2C_MODULE_ID index)

PLIB_I2C_MasterStart Function

Sends an I2C Start condition on the I2C bus in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStart(I2C_MODULE_ID index);
```

Returns

None.

Description

This function sends the start signal (a falling edge on SDA while SCL is high) to start a transfer on the I2C bus when acting in Master mode.

Remarks

Only an I2C master can start a transfer on the bus. The bus is considered to be busy after a Start condition.

After the Start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by calling the PLIB_I2C_ArbitrationLossHasOccurred function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStart in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled and the I2C bus must currently be idle.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_BusIsIdle ( MY_I2C_ID ) )
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStart (I2C_MODULE_ID index)

PLIB_I2C_MasterStartRepeat Function

Sends a repeated Start condition during an ongoing transfer in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStartRepeat(I2C_MODULE_ID index);
```

Returns

None.

Description

This function supports sending a repeated Start condition to change slave targets or transfer direction to support certain I2C transfer formats in Master mode.

Remarks

Only an I2C master that has already started a transfer can send a repeated Start condition.

After the repeated-start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by the PLIB_I2C_ArbitrationLossHasOccurred function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStartRepeat in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStartRepeat (I2C_MODULE_ID index)

PLIB_I2C_MasterStop Function

Sends an I2C Stop condition to terminate a transfer in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStop(I2C_MODULE_ID index);
```

Returns

None.

Description

This function sends the stop signal (a rising edge on SDA while SCL is high) on the I2C bus, to end a transfer on the I2C bus when in Master mode.

Remarks

Only an I2C master that has already started a transfer can send a Stop condition.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStop in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately, enabled, and a previously started transfer must be completed.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStop (I2C_MODULE_ID index)

h) Transmitter Control Functions

PLIB_I2C_TransmitterByteHasCompleted Function

Detects whether the module has finished transmitting the most recent byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterByteHasCompleted(I2C_MODULE_ID index);
```

Returns

- true - If the transmitter has completed sending the data byte
- false - If the transmitter is still busy sending the data byte

Description

This function determines if the transmitter has finished sending the most recently sent byte on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteComplete in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a data or address byte must have been sent.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
  //Transmission completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterByteHasCompleted (I2C_MODULE_ID index)

PLIB_I2C_TransmitterByteSend Function

Sends a byte of data on the I2C bus.

File

plib_i2c.h

C

```
void PLIB_I2C_TransmitterByteSend(I2C_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function allows the caller to send a byte of data on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

The caller must either first call the PLIB_I2C_TransmitterIsReady function before calling this function to ensure that the transmitter is ready to receive a new byte to transmit or call the PLIB_I2C_TransmitterOverflowHasOccurred function immediately after calling this function to ensure that a transmitter write collision did not occur.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteSend in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the I2C module's transmitter must be ready to accept a byte of data to send.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t transmitData;

//set 'transmitData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, transmitData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
data	Data byte to send on the I2C bus

Function

void PLIB_I2C_TransmitterByteSend (I2C_MODULE_ID index, uint8_t data)

PLIB_I2C_TransmitterByteWasAcknowledged Function

Determines whether the most recently sent byte was acknowledged.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterByteWasAcknowledged(I2C_MODULE_ID index);
```

Returns

- true - If the receiver ACKed the byte
- false - If the receiver NAKed the byte

Description

This function allows a transmitter to determine if the byte just sent was positively acknowledged (ACKed) or negatively acknowledged (NAKed) by the receiver.

Remarks

This function can be used by both master or slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been sent on the I2C bus, and the transmission must have completed.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterByteWasAcknowledged ( MY_I2C_ID ) )
    {
        // transmission successful
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterByteWasAcknowledged (I2C_MODULE_ID index)

PLIB_I2C_TransmitterIsBusy Function

Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterIsBusy(I2C_MODULE_ID index);
```

Returns

- true - The transmitter is busy (unable to accept new data)
- false - The transmitter is ready (able to accept new data)

Description

This function identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).

Remarks

This function returns the inverse of the PLIB_I2C_TransmitterIsReady function.

This flag is cleared automatically by the hardware when the transmitter is ready. It cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterIsBusy in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t transData;

//set 'transData' variable

if ( !PLIB_I2C_TransmitterIsBusy( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend( MY_I2C_ID, transData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterIsBusy (I2C_MODULE_ID index)

PLIB_I2C_TransmitterIsReady Function

Detects if the transmitter is ready to accept data to transmit.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterIsReady(I2C_MODULE_ID index);
```

Returns

- true - If the transmitter is ready to accept more data
- false - If the transmitter is not ready to accept more data

Description

This function determines if the transmitter is ready to accept more data to be transmitted on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

This function returns the inverse of the PLIB_I2C_TransmitterIsBusy function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterIsBusy in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately, enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t sendData;

//set 'sendData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
```

```
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, sendData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterIsReady (I2C_MODULE_ID index)

PLIB_I2C_TransmitterOverflowClear Function

Clears the transmitter overflow status flag.

File

plib_i2c.h

C

```
void PLIB_I2C_TransmitterOverflowClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the transmitter overflow status flag.

Remarks

This flag is set by hardware when an overflow error occurs. Its status can be tested using the PLIB_I2C_TransmitterOverflowHasOccurred function. This flag must be cleared be cleared by software after the overflow error has been handled. To handle the error, software must retry the write later after the PLIB_I2C_TransmitterByteSend function returns TRUE.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t my_data;

//set 'my_data' variable
PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, my_data );
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_TransmitterOverflowClear (I2C_MODULE_ID index)

PLIB_I2C_TransmitterOverflowHasOccurred Function

Identifies if a transmitter overflow error has occurred.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterOverflowHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If software attempted to write a byte to the transmitter buffer while the transmitter was busy and unable to accept a new byte (i.e., the write will not occur)
- false - If no transmitter overflow occurred when software attempted to write a byte to the transmitter buffer (i.e., the write occurred successfully)

Description

This function identifies if a transmitter overflow error has occurred.

Remarks

This flag must be cleared be cleared by software using the PLIB_I2C_TransmitterOverflowClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t txData;

//set 'txData' variable

PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, txData );
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterOverflowHasOccurred (I2C_MODULE_ID index)

i) Receiver Control Functions

PLIB_I2C_ReceivedByteAcknowledge Function

Allows a receiver to acknowledge a that a byte of data has been received.

File

plib_i2c.h

C

```
void PLIB_I2C_ReceivedByteAcknowledge(I2C_MODULE_ID index, bool ack);
```

Returns

None.

Description

This function allows a receiver to positively acknowledge (ACK) or negatively acknowledge (NAK) a byte of data that has been received from the I2C bus.

Remarks

This function can only be used by master receivers. Slave receivers automatically ACK or NAK bytes.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge ( MY_I2C_ID ) )
    {
        PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
        data = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
ack	Determines how the byte should be acknowledged: • If true, positively acknowledges (ACK) the byte of data received • If false, negatively acknowledges (NAK) the byte of data received

Function

void PLIB_I2C_ReceivedByteAcknowledge (I2C_MODULE_ID index, bool ack)

PLIB_I2C_ReceivedByteGet Function

Gets a byte of data received from the I2C bus interface.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_ReceivedByteGet(I2C_MODULE_ID index);
```

Returns

A byte of data received from the I2C bus.

Description

This function allows the caller to read a byte of data received from the I2C bus.

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteGet in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have

been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t receivedData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
    receivedData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_ReceivedByteGet (I2C_MODULE_ID index)

PLIB_I2C_ReceivedByteIsAvailable Function

Detects whether the receiver has data available.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceivedByteIsAvailable(I2C_MODULE_ID index);
```

Returns

- true - If the receiver has data available
- false - If the receiver does not have data available

Description

This function determines if the receiver has data available to be read by software.

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAvailable in your application to determine whether this feature is available.

Preconditions

The I2C module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t readData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    readData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceivedByteIsAvailable (I2C_MODULE_ID index)

PLIB_I2C_ReceiverByteAcknowledgeHasCompleted Function

Determines if the previous acknowledge has completed.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_MODULE_ID index);
```

Returns

- true - If the acknowledgment has completed
- false - If the acknowledgment has not completed

Description

This function allows the receiver to determine if the acknowledgment signal has completed.

Remarks

This function can only be used by master receivers. Slave receivers automatically ACK or NAK bytes.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been received on the I2C bus, and the acknowledgment must have been started (by calling the PLIB_I2C_ReceivedByteAcknowledge function).

Example

```
PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );

if ( PLIB_I2C_ReceiverByteAcknowledgeHasCompleted ( MY_I2C_ID ) )
{
    // acknowledgment completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceiverByteAcknowledgeHasCompleted (I2C_MODULE_ID index)

PLIB_I2C_ReceiverOverflowClear Function

Clears the receiver overflow status flag.

File

plib_i2c.h

C

```
void PLIB_I2C_ReceiverOverflowClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the receiver overflow status flag.

Remarks

This flag is set by hardware when an overflow error occurs. Its status can be tested using the PLIB_I2C_ReceiverOverflowHasOccurred function.

This flag should be cleared be cleared by software after the overflow error has been handled by reading the byte in the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceiverOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReceiverOverflowClear (I2C_MODULE_ID index)

PLIB_I2C_ReceiverOverflowHasOccurred Function

Identifies if a receiver overflow error has occurred.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceiverOverflowHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If a byte was received while the receiver buffer still held a previously received byte (The new byte will be lost)
- false - If no receiver overflow occurred when a byte has been received

Description

This function identifies if a receiver overflow error has occurred.

Remarks

This flag should be cleared be cleared by software using the PLIB_I2C_ReceiverOverflowClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceiverOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceiverOverflowHasOccurred (I2C_MODULE_ID index)

PLIB_I2C_MasterReceiverReadyToAcknowledge Function

Checks whether the hardware is ready to acknowledge.

File

plib_i2c.h

C

```
bool PLIB_I2C_MasterReceiverReadyToAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the hardware status is ready to acknowledge
- false - If the hardware is not ready to acknowledge

Description

This function checks for preconditions before acknowledging a slave.

Remarks

This function can only be used by master receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started and a byte of data must have been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge ( MY_I2C_ID ) )
    {
        PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
        data = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_MasterReceiverReadyToAcknowledge (I2C_MODULE_ID index)

j) Feature Existence Functions

PLIB_I2C_ExistsAcksequenceProgress Function

Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsAcksequenceProgress(I2C_MODULE_ID index);
```

Returns

- true - If the AcksequenceIsInProgress feature is supported on the device
- false - If the AcksequenceIsInProgress feature is not supported on the device

Description

This function identifies whether the AcksequenceIsInProgress feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_AcksequenceIsInProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsAcksequenceProgress(I2C_MODULE_ID index)

PLIB_I2C_ExistsArbitrationLoss Function

Identifies whether the ArbitrationLoss feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsArbitrationLoss(I2C_MODULE_ID index);
```

Returns

- true - If the ArbitrationLoss feature is supported on the device
- false - If the ArbitrationLoss feature is not supported on the device

Description

This function identifies whether the ArbitrationLoss feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ArbitrationLossHasOccurred
- PLIB_I2C_ArbitrationLossClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsArbitrationLoss(I2C_MODULE_ID index)

PLIB_I2C_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsBaudRate(I2C_MODULE_ID index);
```

Returns

- true - If the BaudRate feature is supported on the device
- false - If the BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_BaudRateSet
- PLIB_I2C_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsBaudRate(I2C_MODULE_ID index)

PLIB_I2C_ExistsBusIsIdle Function

Identifies whether the BusIdle feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsBusIsIdle(I2C_MODULE_ID index);
```

Returns

- true - If the BusIdle feature is supported on the device
- false - If the BusIdle feature is not supported on the device

Description

This function identifies whether the BusIdle feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_BusIsIdle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsBusIsIdle(I2C_MODULE_ID index)

PLIB_I2C_ExistsClockStretching Function

Identifies whether the ClockStretching feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsClockStretching(I2C_MODULE_ID index);
```

Returns

- true - If the ClockStretching feature is supported on the device
- false - If the ClockStretching feature is not supported on the device

Description

This function identifies whether the ClockStretching feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveClockStretchingEnable
- PLIB_I2C_SlaveClockStretchingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsClockStretching(I2C_MODULE_ID index)

PLIB_I2C_ExistsDataLineHoldTime Function

Identifies whether the DataLineHoldTime feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsDataLineHoldTime(I2C_MODULE_ID index);
```

Returns

- true - If the DataLineHoldTime feature is supported on the device
- false - If the DataLineHoldTime feature is not supported on the device

Description

This function identifies whether the DataLineHoldTime feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_DataLineHoldTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsDataLineHoldTime(I2C_MODULE_ID index)

PLIB_I2C_ExistsGeneralCall Function

Identifies whether the GeneralCall feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsGeneralCall(I2C_MODULE_ID index);
```

Returns

- true - If the GeneralCall feature is supported on the device
- false - If the GeneralCall feature is not supported on the device

Description

This function identifies whether the GeneralCall feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_GeneralCallEnable
- PLIB_I2C_GeneralCallDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsGeneralCall(I2C_MODULE_ID index)

PLIB_I2C_ExistsGeneralCallAddressDetect Function

Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsGeneralCallAddressDetect(I2C_MODULE_ID index);
```

Returns

- true - If the GeneralCallAddressDetect feature is supported on the device
- false - If the GeneralCallAddressDetect feature is not supported on the device

Description

This function identifies whether the GeneralCallAddressDetect feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveAddressIsGeneralCall

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsGeneralCallAddressDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsHighFrequency Function

Identifies whether the HighFrequency feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsHighFrequency(I2C_MODULE_ID index);
```

Returns

- true - If the HighFrequency feature is supported on the device
- false - If the HighFrequency feature is not supported on the device

Description

This function identifies whether the HighFrequency feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_HighFrequencyEnable
- PLIB_I2C_HighFrequencyDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsHighFrequency(I2C_MODULE_ID index)

PLIB_I2C_ExistsIPMI Function

Identifies whether the IPMI feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsIPMI(I2C_MODULE_ID index);
```

Returns

- true - If the IPMI feature is supported on the device
- false - If the IPMI feature is not supported on the device

Description

This function identifies whether the IPMI feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_IPMIEnable
- PLIB_I2C_IPMIDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsIPMI(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterReceiverClock1Byte Function

Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterReceiverClock1Byte(I2C_MODULE_ID index);
```

Returns

- true - If the MasterReceiverClock1Byte feature is supported on the device
- false - If the MasterReceiverClock1Byte feature is not supported on the device

Description

This function identifies whether the MasterReceiverClock1Byte feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterReceiverClock1Byte

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterReceiverClock1Byte(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStart Function

Identifies whether the MasterStart feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStart(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStart feature is supported on the device
- false - If the MasterStart feature is not supported on the device

Description

This function identifies whether the MasterStart feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStart(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStartRepeat Function

Identifies whether the MasterStartRepeat feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStartRepeat(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStartRepeat feature is supported on the device
- false - If the MasterStartRepeat feature is not supported on the device

Description

This function identifies whether the MasterStartRepeat feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStartRepeat

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStartRepeat(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStop Function

Identifies whether the MasterStop feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStop(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStop feature is supported on the device
- false - If the MasterStop feature is not supported on the device

Description

This function identifies whether the MasterStop feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStop(I2C_MODULE_ID index)

PLIB_I2C_ExistsModuleEnable Function

Identifies whether the ModuleEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsModuleEnable(I2C_MODULE_ID index);
```

Returns

- true - If the ModuleEnable feature is supported on the device
- false - If the ModuleEnable feature is not supported on the device

Description

This function identifies whether the ModuleEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_Enable
- PLIB_I2C_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsModuleEnable(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteAcknowledge Function

Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteAcknowledge feature is supported on the device
- false - If the ReceivedByteAcknowledge feature is not supported on the device

Description

This function identifies whether the ReceivedByteAcknowledge feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReceivedByteAcknowledge
- PLIB_I2C_ReceiverByteAcknowledgeHasCompleted
- PLIB_I2C_MasterReceiverReadyToAcknowledge

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteAcknowledge(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteAvailable Function

Identifies whether the ReceivedByteAvailable feature exists on the I2C module

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteAvailable(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteAvailable feature is supported on the device
- false - If the ReceivedByteAvailable feature is not supported on the device

Description

This function identifies whether the ReceivedByteAvailable feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_ReceivedByteIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteAvailable(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteGet Function

Identifies whether the ReceivedByteGet feature exists on the I2C module

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteGet(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteGet feature is supported on the device
- false - If the ReceivedByteGet feature is not supported on the device

Description

This function identifies whether the ReceivedByteGet feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_ReceivedByteGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteGet(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceiverOverflow Function

Identifies whether the ReceiverOverflow feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceiverOverflow(I2C_MODULE_ID index);
```

Returns

- true - If the ReceiverOverflow feature is supported on the device
- false - If the ReceiverOverflow feature is not supported on the device

Description

This function identifies whether the ReceiverOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReceiverOverflowHasOccurred
- PLIB_I2C_ReceiverOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceiverOverflow(I2C_MODULE_ID index)

PLIB_I2C_ExistsReservedAddressProtect Function

Identifies whether the ReservedAddressProtect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReservedAddressProtect(I2C_MODULE_ID index);
```

Returns

- true - If the ReservedAddressProtect feature is supported on the device
- false - If the ReservedAddressProtect feature is not supported on the device

Description

This function identifies whether the ReservedAddressProtect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReservedAddressProtectEnable
- PLIB_I2C_ReservedAddressProtectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReservedAddressProtect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddress10Bit Function

Identifies whether the SlaveAddress10Bit feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddress10Bit(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddress10Bit feature is supported on the device
- false - If the SlaveAddress10Bit feature is not supported on the device

Description

This function identifies whether the SlaveAddress10Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddress10BitSet
- PLIB_I2C_SlaveAddress10BitGet
- PLIB_I2C_SlaveAddressModeIs10Bits

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddress10Bit(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddress7Bit Function

Identifies whether the SlaveAddress7Bit feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddress7Bit(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddress7Bit feature is supported on the device
- false - If the SlaveAddress7Bit feature is not supported on the device

Description

This function identifies whether the SlaveAddress7Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddress7BitSet
- PLIB_I2C_SlaveAddress7BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddress7Bit(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddressDetect Function

Identifies whether the SlaveAddressDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddressDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddressDetect feature is supported on the device
- false - If the SlaveAddressDetect feature is not supported on the device

Description

This function identifies whether the SlaveAddressDetect feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveAddress10BitWasDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddressDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddressHoldEnable Function

Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddressHoldEnable(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddressHoldEnable feature is supported on the device
- false - If the SlaveAddressHoldEnable feature is not supported on the device

Description

This function identifies whether the SlaveAddressHoldEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddressHoldEnable
- PLIB_I2C_SlaveAddressHoldDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddressHoldEnable(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveBufferOverwrite Function

Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveBufferOverwrite(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveBufferOverwrite feature is supported on the device
- false - If the SlaveBufferOverwrite feature is not supported on the device

Description

This function identifies whether the SlaveBufferOverwrite feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveBufferOverwriteEnable
- PLIB_I2C_SlaveBufferOverwriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveBufferOverwrite(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveBusCollisionDetect Function

Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveBusCollisionDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveBusCollisionDetect feature is supported on the device
- false - If the SlaveBusCollisionDetect feature is not supported on the device

Description

This function identifies whether the SlaveBusCollisionDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveBusCollisionDetectEnable
- PLIB_I2C_SlaveBusCollisionDetectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveBusCollisionDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveClockHold Function

Identifies whether the SlaveClockHold feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveClockHold(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveClockHold feature is supported on the device
- false - If the SlaveClockHold feature is not supported on the device

Description

This function identifies whether the SlaveClockHold feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveClockHold
- PLIB_I2C_SlaveClockRelease

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveClockHold(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveDataDetect Function

Identifies whether the SlaveDataDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveDataDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveDataDetect feature is supported on the device
- false - If the SlaveDataDetect feature is not supported on the device

Description

This function identifies whether the SlaveDataDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveDataIsDetected
- PLIB_I2C_SlaveAddressIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveDataDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveInterruptOnStart Function

Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveInterruptOnStart(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveInterruptOnStart feature is supported on the device
- false - If the SlaveInterruptOnStart feature is not supported on the device

Description

This function identifies whether the SlaveInterruptOnStart feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveInterruptOnStartEnable
- PLIB_I2C_SlaveInterruptOnStartDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveInterruptOnStart(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveInterruptOnStop Function

Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveInterruptOnStop(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveInterruptOnStop feature is supported on the device
- false - If the SlaveInterruptOnStop feature is not supported on the device

Description

This function identifies whether the SlaveInterruptOnStop feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveInterruptOnStopEnable
- PLIB_I2C_SlaveInterruptOnStopDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveInterruptOnStop(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveMask Function

Identifies whether the SlaveMask feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveMask(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveMask feature is supported on the device
- false - If the SlaveMask feature is not supported on the device

Description

This function identifies whether the SlaveMask feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveMask7BitSet
- PLIB_I2C_SlaveMask7BitGet
- PLIB_I2C_SlaveMask10BitSet
- PLIB_I2C_SlaveMask10BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveMask(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveReadRequest Function

Identifies whether the SlaveReadRequest feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveReadRequest(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveReadRequest feature is supported on the device
- false - If the SlaveReadRequest feature is not supported on the device

Description

This function identifies whether the SlaveReadRequest feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveReadIsRequested

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveReadRequest(I2C_MODULE_ID index)

PLIB_I2C_ExistsSMBus Function

Identifies whether the SMBus feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSMBus(I2C_MODULE_ID index);
```

Returns

- true - If the SMBus feature is supported on the device
- false - If the SMBus feature is not supported on the device

Description

This function identifies whether the SMBus feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SMBEnable
- PLIB_I2C_SMBDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSMBus(I2C_MODULE_ID index)

PLIB_I2C_ExistsStartDetect Function

Identifies whether the StartDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStartDetect(I2C_MODULE_ID index);
```

Returns

- true - If the StartDetect feature is supported on the device
- false - If the StartDetect feature is not supported on the device

Description

This function identifies whether the StartDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StartWasDetected
- PLIB_I2C_StartClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStartDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsStopDetect Function

Identifies whether the StopDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStopDetect(I2C_MODULE_ID index);
```

Returns

- true - If the StopDetect feature is supported on the device
- false - If the StopDetect feature is not supported on the device

Description

This function identifies whether the StopDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StopWasDetected
- PLIB_I2C_StopClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStopDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStopInIdle(I2C_MODULE_ID index);
```

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StopInIdleEnable
- PLIB_I2C_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStopInIdle(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteAcknowledge Function

Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteAcknowledge feature is supported on the device
- false - If the TransmitterByteAcknowledge feature is not supported on the device

Description

This function identifies whether the TransmitterByteAcknowledge feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteWasAcknowledged

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteAcknowledge(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteComplete Function

Identifies whether the TransmitterByteComplete feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteComplete(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteComplete feature is supported on the device
- false - If the TransmitterByteComplete feature is not supported on the device

Description

This function identifies whether the TransmitterByteComplete feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteComplete(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteSend Function

Identifies whether the TransmitterByteSend feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteSend(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteSend feature is supported on the device
- false - If the TransmitterByteSend feature is not supported on the device

Description

This function identifies whether the TransmitterByteSend feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteSend(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterIsBusy Function

Identifies whether the TransmitterBusy feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterIsBusy(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterBusy feature is supported on the device
- false - If the TransmitterBusy feature is not supported on the device

Description

This function identifies whether the TransmitterBusy feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_TransmitterIsBusy
- PLIB_I2C_TransmitterIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterIsBusy(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterOverflow Function

Identifies whether the TransmitterOverflow feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterOverflow(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterOverflow feature is supported on the device
- false - If the TransmitterOverflow feature is not supported on the device

Description

This function identifies whether the TransmitterOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_TransmitterOverflowHasOccurred
- PLIB_I2C_TransmitterOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterOverflow(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveDataHoldEnable Function

Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveDataHoldEnable(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveDataHoldEnable feature is supported on the device
- false - If the SlaveDataHoldEnable feature is not supported on the device

Description

This function identifies whether the SlaveDataHoldEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveDataHoldEnable
- PLIB_I2C_SlaveDataHoldDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveDataHoldEnable(I2C_MODULE_ID index)

k) Data Types and Constants

I2C_MODULE_ID Enumeration

File

plib_i2c_help.h

C

```
typedef enum {
  I2C_ID_1,
  I2C_ID_2,
  I2C_ID_3,
  I2C_ID_4,
  I2C_ID_5,
  I2C_NUMBER_OF_MODULES
} I2C_MODULE_ID;
```

Members

Members	Description
I2C_ID_1	I2C Module 1 ID
I2C_ID_2	I2C Module 2 ID
I2C_ID_3	I2C Module 3 ID
I2C_ID_4	I2C Module 4 ID
I2C_ID_5	I2C Module 5 ID
I2C_NUMBER_OF_MODULES	Number of available I2C modules.

Description

This is type I2C_MODULE_ID.

Files

Files

Name	Description
plib_i2c.h	This file contains the interface definition for the I2C Peripheral Library.
plib_i2c_help.h	Identifies the supported I2C modules.

Description

This section lists the source and header files used by the library.

plib_i2c.h

This file contains the interface definition for the I2C Peripheral Library.

Functions

	Name	Description
	PLIB_I2C_AcksequenceIsInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
	PLIB_I2C_BaudRateSet	Sets the desired baud rate.
	PLIB_I2C_BusIsIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_ExistsAcksequenceProgress	Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.

	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module.
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module.
	PLIB_I2C_ExistsBusIsIdle	Identifies whether the BusIdle feature exists on the I2C module.
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module.
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module.
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module.
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module.
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataHoldEnable	Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module.
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module.
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module.
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification

	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.
	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.
	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedByteIsAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.
	PLIB_I2C_SlaveAddressModeIs10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.
	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.

	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

Description

I2C Peripheral Library Interface Header

This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_i2c.h

Company

Microchip Technology Inc.

plib_i2c_help.h

Identifies the supported I2C modules.

Enumerations

	Name	Description
	I2C_MODULE_ID	This is type I2C_MODULE_ID.

Description

I2C Module ID

This enumeration identifies the available I2C modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers specified in the data sheet.

Input Capture Peripheral Library

This section describes the Input Capture Peripheral Library.

Introduction

This library provides a low-level abstraction of the Input Capture (IC) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Input Capture (IC) module is used to capture a timer value on the occurrence of an event on an input pin. The IC module plays a major role in applications requiring frequency measurement.

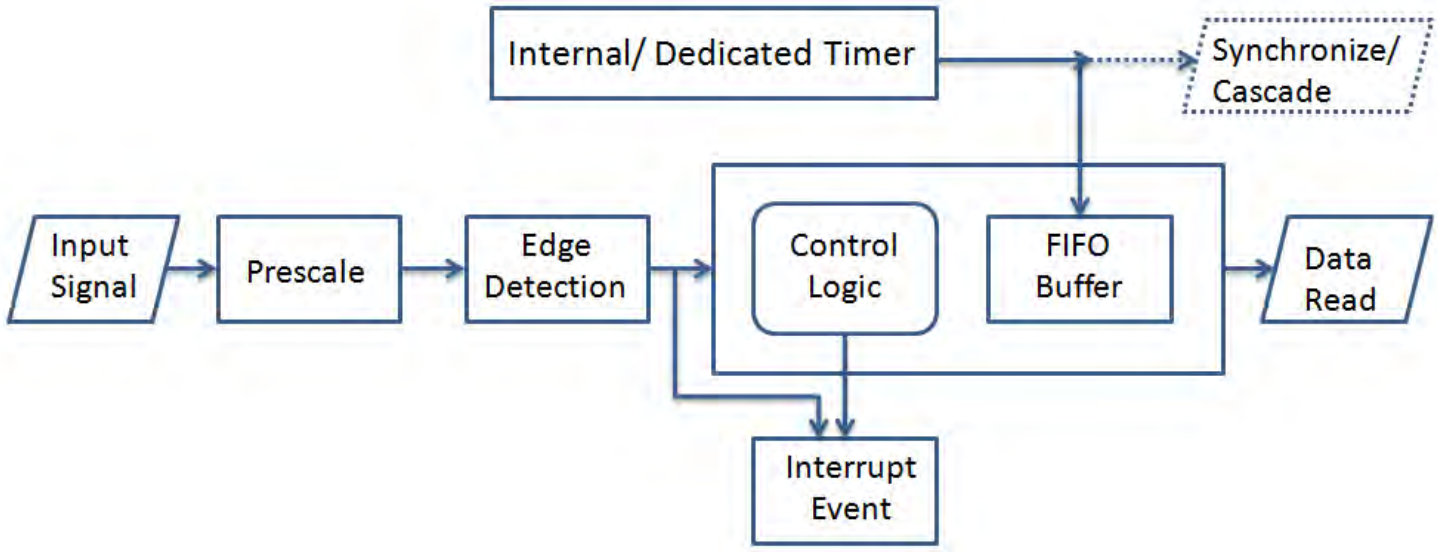


Figure 1: Input Capture Module

Figure 1 shows the functionality of an Input Capture module. An input signal is provided to the input of this module. The input may be prescaled. An edge detection block detects if the input is undergoing a rising or falling transition. Control logic undertakes the capture of the timer value depending upon the operating mode selected.

Some devices have Input Capture modules with dedicated timers that can be used to synchronize multiple IC modules together or cascade 16-bit timers to obtain a 32-bit timer. The timer may be triggered by an external, asynchronous input to start counting as well.

The IC module has multiple modes of operation which can be used for different applications. Operating modes include the following:

- Timer value capture at falling edge of input signal
- Timer value capture at rising edge of input signal
- Timer value capture at both edges of input signal
- Timer value capture on a specified edge of the input signal and every edge thereafter
- Timer value capture on every fourth or sixteenth rising edge of the input signal
- Interrupt-only mode in which Input Capture module acts as a source of external interrupt. This mode is functional only in device Sleep and Idle modes.

The IC module also incorporates a First In First Out (FIFO) buffer, which temporarily holds the captured value until it is read by the user application. The IC module is capable of generating interrupts when a set number of captures have taken place.

Using the Library

This topic describes the basic architecture of the Input Capture Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ic.h`

The interface to the Input Capture Peripheral library is defined in the `plib_ic.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Input Capture Peripheral library must include `peripheral.h`.

Library File: The IC Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

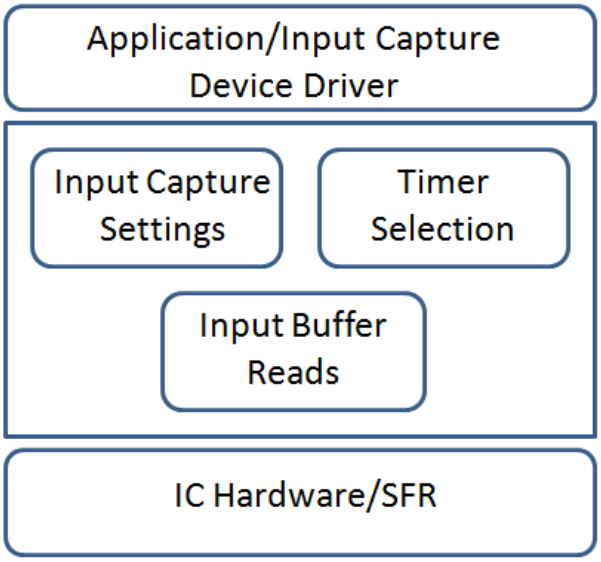
This library provides the low-level abstraction of the Input Capture (IC) module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available IC module on the device. Refer to the **"Input Capture"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each IC module on your device.

Description

The IC Peripheral Library is used to simplify low-level access to the IC module without having the need to directly interact with the module registers. The names of the functions are generic and lead to easier access to the IC module on different device variants.

Input Capture module Software Abstraction Block Diagram



The IC module can be described from a software standpoint as having functions to configure the module itself and select an intended mode of operation. Separate functions exist to select the timer whose value is captured by the IC module. When an external input event occurs, the IC module captures the timer value and stores it in a First In First Out (FIFO) buffer. The captured value can be read from the FIFO buffer by performing an input buffer read by calling the respective function.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Input Capture module.

Library Interface Section	Description
General Setup Functions	Provides setup, configuration, and status interface routines for the overall operation of the Input Capture module.
Timer Functions	Provides timer interface routines.
Input Capture Buffer Functions	Provides buffer interface routines.
Operation Modes Functions	Provides operation mode routines.
Power-Saving Modes Functions	Provides Power-Saving Modes routines.
Feature Existence Functions	Determine whether particular features exist on a device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General setup functions: These functions deal with general setup of the IC module
- Timer functions: These routines are used to select the timer, input for the timer, and configure synchronous or triggered operation for the timer
- Operation Modes routine: The PLIB_IC_ModeSelect function is used to select an operation mode of IC module
- Input capture buffer routines: These routines deal with reading the capture value stored in the IC buffer and performing status checks on the IC buffer
- Power-Saving modes:
  - In Sleep mode, the IC module can function only as an external interrupt source. A rising edge input wakes the device from sleep. An interrupt will be generated if it is enabled for that module.
  - In Idle mode, the IC module can be configured to either stop its operation or continue its operation
- Exists functions: These functions determine whether or not a particular feature is present on a device

Subsequent sections provide examples that depict the use of interface routines to perform general tasks such as IC module setup and reading captured values.

Input Capture Module Setup

This section shows the steps required to initialize and configure the IC module. The IC module is configured such that a capture event occurs at the first falling edge of the input signal, and then at every subsequent edge of the input signal.

Example:

```
/* This example shows how to initialize and configure IC module */
/* Disable IC module */
PLIB_IC_Disable(MY_IC_ID);

/* Disable any peripherals multiplexed with Input Capture pin */

/* Configure I/O pin as input pin. Refer PORTS Peripheral Library */

/* Stop in Idle mode is disabled for IC module. IC mode functions even in Idle mode */
PLIB_IC_StopInIdleDisable(MY_IC_ID);

/* Every Edge Capture mode of operation is selected for IC module */
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);

/* Timer2 is selected as a clock source for the IC module  */
PLIB_IC_TimerSelect(MY_IC_ID, IC_TIMER_TMR2);

/* 32-bit timer is selected*/
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);

/* IC module is enabled */
PLIB_IC_Enable(MY_IC_ID);
```

Reading a Capture Value

This section shows how to read a capture value from the input capture buffer.

Example:

```
/* This example shows how to read a Capture value */
uint32_t bufferVal;

/* Reading a value from an empty Capture Buffer leads to indeterminate data*/
/* Read value from a non-empty buffer */
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
     /* Read the buffer value */
     bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
}

/* Buffer Overflow condition */
while(PLIB_IC_BufferOverflowHasOccurred)
```

```
{
    /*Read buffer values until buffer is empty and overflow condition ceases*/
    bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);

}
```

Configuring the Library

The library is configured for the supported Input Capture module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_IC_Disable	Disables the IC module.
	PLIB_IC_Enable	Enables the IC module.
	PLIB_IC_EventsPerInterruptSelect	Selects the number of capture events before an interrupt is issued.
	PLIB_IC_FirstCaptureEdgeSelect	Captures the timer count value at the first selected edge of input signal.

b) Timer Functions

	Name	Description
	PLIB_IC_TimerSelect	Selects the clock source for the IC module.
	PLIB_IC_AlternateClockDisable	Selects Timer2 and Timer3, instead of the alternate clock source.
	PLIB_IC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_IC_AlternateTimerSelect	Selects an alternate timer as a clock source for the IC module.
	PLIB_IC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the IC module.

c) Input Capture Buffer Functions

	Name	Description
	PLIB_IC_Buffer16BitGet	Obtains the 16-bit input capture buffer value.
	PLIB_IC_Buffer32BitGet	Obtains the 32-bit input capture buffer value.
	PLIB_IC_BufferIsEmpty	Checks whether the input capture buffer is empty.
	PLIB_IC_BufferOverflowHasOccurred	Checks whether an input capture buffer overflow has occurred.
	PLIB_IC_BufferSizeSelect	Sets the input capture buffer size.

d) Operation Modes Functions

	Name	Description
	PLIB_IC_ModeSelect	Selects the input capture mode for IC module.

e) Power-Saving Modes Functions

	Name	Description
	PLIB_IC_StopInIdleDisable	Continues module operation when the device enters Idle mode
	PLIB_IC_StopInIdleEnable	Discontinues IC module operation when the device enters Idle mode.

f) Feature Existence Functions

	Name	Description
	PLIB_IC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the IC module.
	PLIB_IC_ExistsBufferIsEmptyStatus	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
	PLIB_IC_ExistsBufferOverflowStatus	Identifies whether the BufferOverflowStatus feature exists on the IC module.
	PLIB_IC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the IC module.
	PLIB_IC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the IC module.
	PLIB_IC_ExistsCaptureMode	Identifies whether the CaptureMode feature exists on the IC module.
	PLIB_IC_ExistsEdgeCapture	Identifies whether the EdgeCapture feature exists on the IC module.
	PLIB_IC_ExistsEnable	Identifies whether the EnableControl feature exists on the IC module
	PLIB_IC_ExistsEventsPerInterruptSelect	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
	PLIB_IC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the IC module.

	PLIB_IC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the IC module.

g) Data Types and Constants

	Name	Description
	IC_BUFFER_SIZE	Selects the IC Buffer size.
	IC_EDGE_TYPES	Lists the options to select a rising or falling edge for input capture.
	IC_EVENTS_PER_INTERRUPT	Lists the number of input capture events for an interrupt to occur.
	IC_INPUT_CAPTURE_MODES	Lists all of the input capture modes for the IC module.
	IC_MODULE_ID	IC_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller. Refer to the specific device data sheet to determine the correct number of modules defined for the device.
	IC_TIMERS	Lists the different 16-bit timers for the IC module.
	IC_ALT_TIMERS	Lists the different 16-bit alternate timers for the IC module.

Description

This section describes the Application Programming Interface (API) functions of the Input Capture Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_IC_Disable Function

Disables the IC module.

File

plib_ic.h

C

```
void PLIB_IC_Disable(IC_MODULE_ID index);
```

Returns

None.

Description

This function disables the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_Disable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the IC module

Function

void PLIB_IC_Disable(IC_MODULE_ID index)

PLIB_IC_Enable Function

Enables the IC module.

File

plib_ic.h

C

```
void PLIB_IC_Enable(IC_MODULE_ID index);
```

Returns

None.

Description

This function enables the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEnable in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_IC_ID IC_ID_1

//Do all the other configurations before enabling.

PLIB_IC_Enable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_Enable(IC_MODULE_ID index)

PLIB_IC_EventsPerInterruptSelect Function

Selects the number of capture events before an interrupt is issued.

File

plib_ic.h

C

```
void PLIB_IC_EventsPerInterruptSelect(IC_MODULE_ID index, IC_EVENTS_PER_INTERRUPT event);
```

Returns

None.

Description

The IC module can be configured to generate interrupts depending upon the occurrence of a certain number of capture events. The number of capture events before an interrupt is generated is set by this function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEventsPerInterruptSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
// IC module is configured to generate interrupt on every capture event
PLIB_IC_EventsPerInterruptSelect(MY_IC_ID, IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT );
```

Parameters

Parameters	Description
index	Identifies the desired IC module
event	Identifies the interrupt control mode

Function

void PLIB_IC_EventsPerInterruptSelect(IC_MODULE_ID index,IC_EVENTS_PER_INTERRUPT event)

PLIB_IC_FirstCaptureEdgeSelect Function

Captures the timer count value at the first selected edge of input signal.

File

plib_ic.h

C

```
void PLIB_IC_FirstCaptureEdgeSelect(IC_MODULE_ID index, IC_EDGE_TYPES edgeType);
```

Returns

None.

Description

This function captures the timer count value at the first selected edge of the input signal.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEdgeCapture in your application to determine whether this feature is available.

Preconditions

This setting applies only when the Every Edge Capture mode is set. The capture mode is set by the PLIB_IC_ModeSelect function.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_FirstCaptureEdgeSelect(MY_IC_ID,IC_EDGE_FALLING);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
edgeType	Identifies the edge type (i.e., whether rising or falling edge)

Function

void PLIB_IC_FirstCaptureEdgeSelect(IC_MODULE_ID index, IC_EDGE_TYPES edgeType)

b) Timer Functions

PLIB_IC_TimerSelect Function

Selects the clock source for the IC module.

File

plib_ic.h

C

```
void PLIB_IC_TimerSelect(IC_MODULE_ID index, IC_TIMERS tmr);
```

Returns

None.

Description

This function selects the 16-bit timer source for the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set.

Example

```
#define MY_IC_ID IC_ID_1
// 16-bit Timer-2 is selected
PLIB_IC_TimerSelect(MY_IC_ID, IC_TIMER_TMR2);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
tmr	Identifies the 16-bit timer

Function

void PLIB_IC_TimerSelect(IC_MODULE_ID index, IC_TIMERS tmr)

PLIB_IC_AlternateClockDisable Function

Selects Timer2 and Timer3, instead of the alternate clock source.

File

plib_ic.h

C

```
void PLIB_IC_AlternateClockDisable(IC_MODULE_ID index);
```

Returns

None.

Description

Selects Timer2 and Timer3, instead of the alternate clock source.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the IC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_IC_ID IC_ID_1
PLIB_IC_AlternateClockDisable( MY_IC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_AlternateClockDisable(IC_MODULE_ID index)

PLIB_IC_AlternateClockEnable Function

Selects the alternate clock source.

File

plib_ic.h

C

```
void PLIB_IC_AlternateClockEnable(IC_MODULE_ID index);
```

Returns

None.

Description

Selects the alternate clock source instead of Timer2/3.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the IC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_IC_ID IC_ID_1
PLIB_IC_AlternateClockEnable( MY_IC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_AlternateClockEnable(IC_MODULE_ID index)

PLIB_IC_AlternateTimerSelect Function

Selects an alternate timer as a clock source for the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_AlternateTimerSelect(IC_MODULE_ID index, IC_ALT_TIMERS tmr);
```

Returns

Boolean

- true - Alternate timer selected successfully.
- false - Alternate timer selection failure, select appropriate alternate timer for the IC module index.

Description

This function selects an alternate timer as a clock source for the IC module. IC_ID_1,IC_ID_2,IC_ID_3 : Can use Timer4 or Timer5 as alternate timers. IC_ID_4,IC_ID_5,IC_ID_6 : Can use Timer2 or Timer3 as alternate timers. IC_ID_7,IC_ID_8,IC_ID_9 : Can use Timer6 or Timer7 as alternate timers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_IC_ExistsAlternateTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set. PLIB_IC_AlternateClockEnable API should be called for the IC module to enable the alternate clock selection.

Example

```
#define MY_IC_ID   IC_ID_1
bool result;

//Enabling alternate timer selection
// Call system service to unlock oscillator
PLIB_IC_AlternateClockEnable( MY_IC_ID );

// 16-bit Timer4 is selected as the clock source for IC module 1
result = PLIB_IC_AlternateTimerSelect(MY_IC_ID, IC_ALT_TIMER_TMR4);

if(false == result)
{
    //Selected alternate timer does not available for the desired IC module.
    //Select appropriate alternate timer.
}
```

Parameters

Parameters	Description
index	Identifies the desired IC module
tmr	Identifies the alternate timer

Function

bool PLIB_IC_AlternateTimerSelect(IC_MODULE_ID index, IC_ALT_TIMERS tmr)

PLIB_IC_ExistsAlternateTimerSelect Function

Identifies whether the AlternateTimerSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsAlternateTimerSelect(IC_MODULE_ID index);
```

Returns

- true - If the AlternateTimerSelect feature is supported on the device
- false - If the AlternateTimerSelect feature is not supported on the device

Description

This function identifies whether the AlternateTimerSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_AlternateTimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_IC_ExistsAlternateTimerSelect(IC_MODULE_ID index)

c) Input Capture Buffer Functions

PLIB_IC_Buffer16BitGet Function

Obtains the 16-bit input capture buffer value.

File

plib_ic.h

C

```
uint16_t PLIB_IC_Buffer16BitGet(IC_MODULE_ID index);
```

Returns

None.

Description

This function reads the 16-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 16-bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The buffer size should be set to 16 bits (i.e., PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_16BIT).

Example

```
#define MY_IC_ID IC_ID_1
uint16_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer16BitGet(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

uint16_t PLIB_IC_Buffer16BitGet(IC_MODULE_ID index)

PLIB_IC_Buffer32BitGet Function

Obtains the 32-bit input capture buffer value.

File

plib_ic.h

C

```
uint32_t PLIB_IC_Buffer32BitGet(IC_MODULE_ID index);
```

Returns

None.

Description

This function reads the 32-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 32 bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The buffer size should be set to 32 bits (i.e., PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_32BIT).

Example

```
#define MY_IC_ID IC_ID_1
uint32_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

uint32_t PLIB_IC_Buffer32BitGet(IC_MODULE_ID index)

PLIB_IC_BufferIsEmpty Function

Checks whether the input capture buffer is empty.

File

plib_ic.h

C

```
bool PLIB_IC_BufferIsEmpty(IC_MODULE_ID index);
```

Returns

Boolean

- true - The input capture buffer is empty
- false - The input capture buffer is not empty

Description

This function returns 'true' if the input capture buffer is empty and 'false' if the buffer is not empty.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferIsEmptyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

bool PLIB_IC_BufferIsEmpty(IC_MODULE_ID index)

PLIB_IC_BufferOverflowHasOccurred Function

Checks whether an input capture buffer overflow has occurred.

File

plib_ic.h

C

```
bool PLIB_IC_BufferOverflowHasOccurred(IC_MODULE_ID index);
```

Returns

Boolean

- true - An overflow of the input capture buffer has occurred
- false - An overflow of the input capture buffer has not occurred

Description

This function returns 'true' if an input capture buffer has overflowed and 'false' if the buffer has not overflowed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferOverflowStatus in your application to determine whether this feature is available.

Preconditions

This function only applies when not in Edge Detect mode.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferOverflowHasOccurred(MY_IC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

bool PLIB_IC_BufferOverflowHasOccurred(IC_MODULE_ID index)

PLIB_IC_BufferSizeSelect Function

Sets the input capture buffer size.

File

plib_ic.h

C

```
void PLIB_IC_BufferSizeSelect(IC_MODULE_ID index, IC_BUFFER_SIZE bufSize);
```

Returns

None.

Description

This function sets the input capture buffer size for IC module. The buffer size can be set to 16 bits or 32 bits. The buffer size should be consistent with the timer selected. A 32-bit timer requires a 32-bit buffer and a 16-bit timer requires a 16-bit buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferSize in your application to determine whether this feature is available.

Preconditions

The buffer size should be consistent with the timer selected.

Example

```
#define MY_IC_ID IC_ID_1
// 32-bit timer resource is selected
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

bufSize	Sets the buffer size to 16 bits or 32 bits

Function

void PLIB_IC_BufferSizeSelect(IC_MODULE_ID index, IC_BUFFER_SIZE bufSize)

d) Operation Modes Functions

PLIB_IC_ModeSelect Function

Selects the input capture mode for IC module.

File

plib_ic.h

C

```
void PLIB_IC_ModeSelect(IC_MODULE_ID index, IC_INPUT_CAPTURE_MODES modeSel);
```

Returns

None.

Description

This function selects the input capture mode for the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsCaptureMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
// Every Edge Mode is selected
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
modeSel	Identifies the timer type required

Function

void PLIB_IC_ModeSelect(IC_MODULE_ID index, IC_INPUT_CAPTURE_MODES modeSel)

e) Power-Saving Modes Functions

PLIB_IC_StopInIdleDisable Function

Continues module operation when the device enters Idle mode

File

plib_ic.h

C

```
void PLIB_IC_StopInIdleDisable(IC_MODULE_ID index);
```

Returns

None.

Description

The function continues operation of the IC module when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleDisable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_StopInIdleDisable(IC_MODULE_ID index)

PLIB_IC_StopInIdleEnable Function

Discontinues IC module operation when the device enters Idle mode.

File

plib_ic.h

C

```
void PLIB_IC_StopInIdleEnable(IC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues IC module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleEnable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_StopInIdleEnable(IC_MODULE_ID index)

f) Feature Existence Functions

PLIB_IC_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsAlternateClock(IC_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_AlternateClockEnable
- PLIB_IC_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsAlternateClock(IC_MODULE_ID index)

PLIB_IC_ExistsBufferIsEmptyStatus Function

Identifies whether the BufferIsEmptyStatus feature exists on the IC module

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferIsEmptyStatus(IC_MODULE_ID index);
```

Returns

- true - The BufferIsEmptyStatus feature is supported on the device
- false - The BufferIsEmptyStatus feature is not supported on the device

Description

This function identifies whether the BufferIsEmptyStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferIsEmptyStatus(IC_MODULE_ID index)

PLIB_IC_ExistsBufferOverflowStatus Function

Identifies whether the BufferOverflowStatus feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferOverflowStatus(IC_MODULE_ID index);
```

Returns

- true - The BufferOverflowStatus feature is supported on the device
- false - The BufferOverflowStatus feature is not supported on the device

Description

This function identifies whether the BufferOverflowStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferOverflowHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferOverflowStatus(IC_MODULE_ID index)

PLIB_IC_ExistsBufferSize Function

Identifies whether the BufferSize feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferSize(IC_MODULE_ID index);
```

Returns

- true - The BufferSize feature is supported on the device
- false - The BufferSize feature is not supported on the device

Description

This function identifies whether the BufferSize feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferSize(IC_MODULE_ID index)

PLIB_IC_ExistsBufferValue Function

Identifies whether the BufferValue feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferValue(IC_MODULE_ID index);
```

Returns

- true - The BufferValue feature is supported on the device
- false - The BufferValue feature is not supported on the device

Description

This function identifies whether the BufferValue feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_Buffer32BitGet
- PLIB_IC_Buffer16BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferValue(IC_MODULE_ID index)

PLIB_IC_ExistsCaptureMode Function

Identifies whether the CaptureMode feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsCaptureMode(IC_MODULE_ID index);
```

Returns

- true - The CaptureMode feature is supported on the device
- false - The CaptureMode feature is not supported on the device

Description

This function identifies whether the CaptureMode feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_ModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsCaptureMode(IC_MODULE_ID index)

PLIB_IC_ExistsEdgeCapture Function

Identifies whether the EdgeCapture feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEdgeCapture(IC_MODULE_ID index);
```

Returns

- true - The EdgeCapture feature is supported on the device
- false - The EdgeCapture feature is not supported on the device

Description

This function identifies whether the EdgeCapture feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_FirstCaptureEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEdgeCapture(IC_MODULE_ID index)

PLIB_IC_ExistsEnable Function

Identifies whether the EnableControl feature exists on the IC module

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEnable(IC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_Enable
- PLIB_IC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEnable(IC_MODULE_ID index)

PLIB_IC_ExistsEventsPerInterruptSelect Function

Identifies whether the EventsPerInterruptSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEventsPerInterruptSelect(IC_MODULE_ID index);
```

Returns

- true - The EventsPerInterruptSelect feature is supported on the device
- false - The EventsPerInterruptSelect feature is not supported on the device

Description

This function identifies whether the EventsPerInterruptSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_EventsPerInterruptSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEventsPerInterruptSelect(IC_MODULE_ID index)

PLIB_IC_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsStopInIdle(IC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_StopInIdleEnable
- PLIB_IC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsStopInIdle(IC_MODULE_ID index)

PLIB_IC_ExistsTimerSelect Function

Identifies whether the TimerSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsTimerSelect(IC_MODULE_ID index);
```

Returns

- true - The TimerSelect feature is supported on the device
- false - The TimerSelect feature is not supported on the device

Description

This function identifies whether the TimerSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_TimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsTimerSelect(IC_MODULE_ID index)

g) Data Types and Constants

IC_BUFFER_SIZE Enumeration

Selects the IC Buffer size.

File

plib_ic_help.h

C

```
typedef enum {
  IC_BUFFER_SIZE_32BIT,
  IC_BUFFER_SIZE_16BIT
} IC_BUFFER_SIZE;
```

Members

Members	Description
IC_BUFFER_SIZE_32BIT	32-bit timer source select
IC_BUFFER_SIZE_16BIT	16-bit timer source select

Description

IC Buffer Size

The IC buffer can be set to 16 bits or 32 bits. The buffer size is set by the PLIB_IC_BufferSizeSelect function.

IC_EDGE_TYPES Enumeration

Lists the options to select a rising or falling edge for input capture.

File

plib_ic_help.h

C

```
typedef enum {
  IC_EDGE_FALLING,
  IC_EDGE_RISING
} IC_EDGE_TYPES;
```

Members

Members	Description
IC_EDGE_FALLING	Select Falling Edge
IC_EDGE_RISING	Select Rising Edge

Description

IC Edge Types Select

This enumeration lists the options to select a rising edge input capture or a falling edge input capture. The selection is made by the PLIB_IC_FirstCaptureEdgeSelect function.

IC_EVENTS_PER_INTERRUPT Enumeration

Lists the number of input capture events for an interrupt to occur.

File

plib_ic_help.h

C

```
typedef enum {
  IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT
} IC_EVENTS_PER_INTERRUPT;
```

Members

Members	Description
IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT	Interrupt on every 4th capture event
IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT	Interrupt on every 3rd capture event
IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT	Interrupt on every 2nd capture event
IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT	Interrupt on every capture event

Description

IC Capture Events Per Interrupt Select

This enumeration lists the number of events that should occur before an interrupt is generated. An interrupt can occur on every capture event, and every second, third, or fourth capture event.

IC_INPUT_CAPTURE_MODES Enumeration

Lists all of the input capture modes for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_INPUT_CAPTURE_INTERRUPT_MODE,
  IC_INPUT_CAPTURE_EVERY_EDGE_MODE,
  IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE,
  IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE,
  IC_INPUT_CAPTURE_RISING_EDGE_MODE,
  IC_INPUT_CAPTURE_FALLING_EDGE_MODE,
  IC_INPUT_CAPTURE_EDGE_DETECT_MODE,
  IC_INPUT_CAPTURE_DISABLE_MODE
} IC_INPUT_CAPTURE_MODES;
```

Members

Members	Description
IC_INPUT_CAPTURE_INTERRUPT_MODE	Interrupt only mode: Rising edge on input triggers an interrupt. This mode is used only when device is in Sleep/Idle mode
IC_INPUT_CAPTURE_EVERY_EDGE_MODE	Every Edge Capture mode: The first edge of the input signal is specified via PLIB_IC_RisingEdgeCaptureSet() or PLIB_IC_FallingEdgeCaptureSet() routines. Subsequently, timer count value is captured on every rising and falling of the input signal
IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 16th rising edge of input signal
IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 4th rising edge of input signal
IC_INPUT_CAPTURE_RISING_EDGE_MODE	Rising Edge mode: Timer count value is captured on every rising edge of input signal
IC_INPUT_CAPTURE_FALLING_EDGE_MODE	Falling Edge mode: Timer count value is captured on every falling edge of input signal
IC_INPUT_CAPTURE_EDGE_DETECT_MODE	Edge Detect mode: Timer count value is captured on every rising and falling of the input signal Interrupt control bits are ignored and an interrupt event is generated for every capture. Overflow status is not updated.
IC_INPUT_CAPTURE_DISABLE_MODE	Capture module is disabled.Input signal is ignored and no capture events or interrupts are generated

Description

IC Input Capture Mode Select

This enumeration lists all of the input capture modes for IC module. The capture mode is set via the PLIB_IC_ModeSelect function.

IC_MODULE_ID Enumeration

File

plib_ic_help.h

C

```
typedef enum {
```

```
  IC_ID_1,
  IC_ID_2,
  IC_ID_3,
  IC_ID_4,
  IC_ID_5,
  IC_ID_6,
  IC_ID_7,
  IC_ID_8,
  IC_ID_9,
  IC_NUMBER_OF_MODULES
} IC_MODULE_ID;
```

Members

Members	Description
IC_NUMBER_OF_MODULES	The total number of modules available.

Description

IC_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller.

Refer to the specific device data sheet to determine the correct number of modules defined for the device.

IC_TIMERS Enumeration

Lists the different 16-bit timers for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_TIMER_TMR3,
  IC_TIMER_TMR2
} IC_TIMERS;
```

Members

Members	Description
IC_TIMER_TMR3	Timer3 select
IC_TIMER_TMR2	Timer2 select

Description

IC 16-bit Timer Select

This enumeration lists the different 16-bit timers for the IC time base when the IC module is configured with a 16-bit timer resource.

IC_ALT_TIMERS Enumeration

Lists the different 16-bit alternate timers for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_ALT_TIMER_TMR3,
  IC_ALT_TIMER_TMR2,
  IC_ALT_TIMER_TMR5,
  IC_ALT_TIMER_TMR4,
  IC_ALT_TIMER_TMR7,
  IC_ALT_TIMER_TMR6
} IC_ALT_TIMERS;
```

Members

Members	Description
IC_ALT_TIMER_TMR3	Select Timer3 as the alternate clock source for IC module

IC_ALT_TIMER_TMR2	Select Timer2 as the alternate clock source for IC module
IC_ALT_TIMER_TMR5	Select Timer5 as the alternate clock source for IC module
IC_ALT_TIMER_TMR4	Select Timer4 as the alternate clock source for IC module
IC_ALT_TIMER_TMR7	Select Timer7 as the alternate clock source for IC module
IC_ALT_TIMER_TMR6	Select Timer6 as the alternate clock source for IC module

Description

IC 16-bit alternate Timer Select

This enumeration lists the different 16-bit timers for the IC time base when the IC module is configured with a 16-bit alternate timer resource.

Files

Files

Name	Description
plib_ic.h	This file contains the interface definition for the Input Capture (IC) peripheral library.
plib_ic_help.h	This is file plib_ic_help.h.

Description

This section lists the source and header files used by the library.

plib_ic.h

This file contains the interface definition for the Input Capture (IC) peripheral library.

Functions

	Name	Description
	PLIB_IC_AlternateClockDisable	Selects Timer2 and Timer3, instead of the alternate clock source.
	PLIB_IC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_IC_AlternateTimerSelect	Selects an alternate timer as a clock source for the IC module.
	PLIB_IC_Buffer16BitGet	Obtains the 16-bit input capture buffer value.
	PLIB_IC_Buffer32BitGet	Obtains the 32-bit input capture buffer value.
	PLIB_IC_BufferIsEmpty	Checks whether the input capture buffer is empty.
	PLIB_IC_BufferOverflowHasOccurred	Checks whether an input capture buffer overflow has occurred.
	PLIB_IC_BufferSizeSelect	Sets the input capture buffer size.
	PLIB_IC_Disable	Disables the IC module.
	PLIB_IC_Enable	Enables the IC module.
	PLIB_IC_EventsPerInterruptSelect	Selects the number of capture events before an interrupt is issued.
	PLIB_IC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the IC module.
	PLIB_IC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the IC module.
	PLIB_IC_ExistsBufferIsEmptyStatus	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
	PLIB_IC_ExistsBufferOverflowStatus	Identifies whether the BufferOverflowStatus feature exists on the IC module.
	PLIB_IC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the IC module.
	PLIB_IC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the IC module.
	PLIB_IC_ExistsCaptureMode	Identifies whether the CaptureMode feature exists on the IC module.
	PLIB_IC_ExistsEdgeCapture	Identifies whether the EdgeCapture feature exists on the IC module.
	PLIB_IC_ExistsEnable	Identifies whether the EnableControl feature exists on the IC module
	PLIB_IC_ExistsEventsPerInterruptSelect	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
	PLIB_IC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the IC module.
	PLIB_IC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the IC module.
	PLIB_IC_FirstCaptureEdgeSelect	Captures the timer count value at the first selected edge of input signal.
	PLIB_IC_ModeSelect	Selects the input capture mode for IC module.
	PLIB_IC_StopInIdleDisable	Continues module operation when the device enters Idle mode
	PLIB_IC_StopInIdleEnable	Discontinues IC module operation when the device enters Idle mode.
	PLIB_IC_TimerSelect	Selects the clock source for the IC module.

Description

IC Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Input Capture (IC) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_ic.h

Company

Microchip Technology Inc.

plib_ic_help.h

Enumerations

	Name	Description
	IC_ALT_TIMERS	Lists the different 16-bit alternate timers for the IC module.
	IC_BUFFER_SIZE	Selects the IC Buffer size.
	IC_EDGE_TYPES	Lists the options to select a rising or falling edge for input capture.
	IC_EVENTS_PER_INTERRUPT	Lists the number of input capture events for an interrupt to occur.
	IC_INPUT_CAPTURE_MODES	Lists all of the input capture modes for the IC module.
	IC_MODULE_ID	IC_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller. Refer to the specific device data sheet to determine the correct number of modules defined for the device.
	IC_TIMERS	Lists the different 16-bit timers for the IC module.

Description

This is file plib_ic_help.h.

Interrupt Peripheral Library

This section describes the Interrupt Peripheral Library.

Introduction

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Interrupt Controller is a key component of a microcontroller. The Interrupt Controller reduces the numerous peripheral interrupt request signals to a single interrupt request signal to the microcontroller core.

The key features present on an Interrupt Controller are:

- **Interrupt source configuration**: Provides the ability to enable, disable, and prioritize a source that may interrupt the core
- **Interrupt status flags:** Identify the source of an interrupt request or exception, usually generated by a peripheral

The interrupt will be sent to the core, only if the source is enabled to generate an interrupt and interrupts have been enabled (if supported). Sources will be prioritized so that the highest priority interrupt will be sent to the core if multiple sources cause interrupts at the same time.

Many interrupt controllers also include, other optional features:

- **Global enable configuration**: Controls the generation of any interrupt to the core from the Interrupt Controller
- **Choice of edge transition**: Controls the edge transition (high-to-low or low-to-high) on which the interrupts are generated
- **Priority configuration**: Associates a programmable priority with the interrupt source

Interrupts can be handled inside the core in the following ways:

- **Vectored**: Each interrupt has an associated interrupt handler routine
- **Non-vectored**: All interrupts have a single (shared) interrupt handler routine. The software determines the interrupt source based on the interrupt status-flags register.
- **Combination of vectored and non-vectored**: Interrupts combined in groups. Each group has a single common interrupt vector or handler, but there are multiple groups (i.e., multiple vectors). Within a specific vector group, software determines which one in the group sources caused the interrupt.

A microcontroller can support one, or more than one of these modes.

Note: Please refer to the specific device data sheet to determine availability of these features for your device.

Using the Library

This topic describes the basic architecture of the Interrupt Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_int.h`

The interface to the Interrupt Peripheral Library is defined in the `plib_int.h` header file, which is included in the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Interrupt Peripheral Library must include `peripheral.h`.

Library File:

The Interrupt Peripheral Library (`.a`) file is installed with MPLAB Harmony.

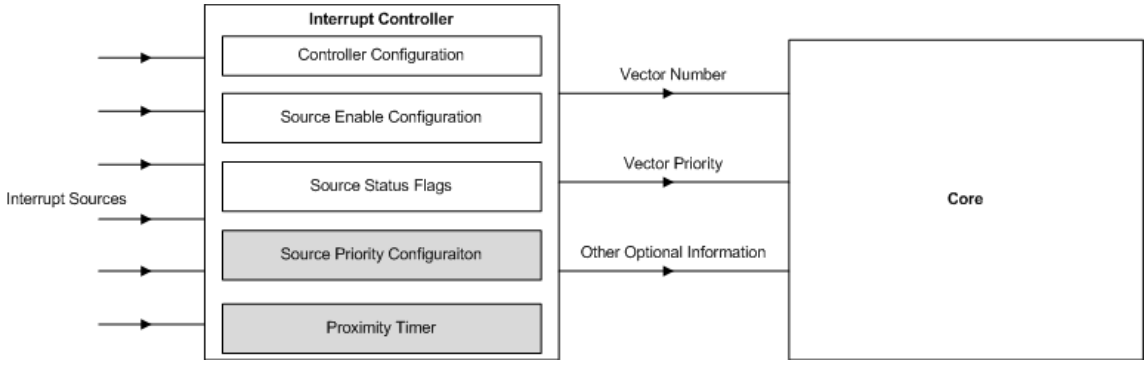
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model



The **Interrupt Controller** receives requests from multiple **Interrupt Sources**. The **Interrupt Controller Configuration** controls the overall operation of the interrupt controller. Not all microcontrollers support all the following features listed:

- Interrupt nesting configuration
- Edge selection of the external interrupts
- Vector mode selection
- Priority level configuration
- Current CPU interrupt and priority status information

Source Enable Configuration enables or disables a particular source. On some microcontrollers other CPU exceptions or traps can also be controlled. **Source Status Flags** give the status of the interrupt request from a specific interrupt source.

Other optional features include the **Vector Priority Configuration**, which is available on some micro controllers and the **Proximity Timer**. Priority configuration allows a priority level to be associated with each interrupt vector. The proximity timer creates a temporal window in which a group of interrupts of the same, or lower, priority will be held off. This provides an opportunity to queue these interrupt requests and process them using tail-chaining of multiple IRQs and allows higher priority interrupts to pass through during the hold-off window.

Traps and Exceptions

Some of the things that can interrupt the normal flow of instruction execution of the CPU core are not actual interrupt sources. These are things such as a division-by-zero error or attempting to address unimplemented memory. These errors are either internal to the CPU core itself or are so catastrophic that they are non-maskable and are trapped separately. This library refers to them in general as "traps" and there are interface elements provided to allow software to handle them.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Interrupt module.

Library Interface Section	Description
General Configuration Functions	Provides setup, configuration, and status interface routines for the overall operation of the Interrupt Controller module.
Interrupt Source Control Functions	Provides setup and status routines for: • Setting up the source enable configuration • Setting up the vector priority configuration • Querying a source's interrupt status flag
Other Status and Control Functions	Provides setup and status routines for: • Global interrupt enable status • Vector number and priority of the current ISR • Proximity timer • Trap handling
Feature Existence Functions	Determine whether certain features are available.

How the Library Works

This section provides information on initialization and interrupt source setup and handling.

Description

Initialization

The steps that are required to initialize the Interrupt Controller module vary for different microcontrollers.

Note: Refer to the specific device data sheet for the correct initialization sequence for your device.

Interrupt Source Setup

An interrupt Source needs to be enabled in order for it to be able to generate a interrupt to the core. Use the function PLIB_INT_SourceEnable to enable the interrupt source. INT_SOURCE is a list of the possible interrupt sources

On some microcontrollers, each interrupt vector can be associated with a priority. Use the function PLIB_INT_VectorPrioritySet to set the interrupt vector priority. INT_VECTOR is a list of the possible interrupt vectors and INT_PRIORITY_LEVEL is the list of possible priority levels.

On some microcontrollers, each interrupt vector can be associated with a sub-priority. Use the function PLIB_INT_VectorSubPrioritySet to set the interrupt source sub-priority. INT_SUBPRIORITY_LEVEL is a list of possible sub-priority levels.

On most microcontrollers, it will be necessary to enable interrupt generation to the core by calling the PLIB_INT_Enable function.

Interrupt Handling

The status of the interrupt indicates whether or not the interrupt request was generated by the source. The status of the trap source can be obtained using the function PLIB_INT_TrapSourceFlagGet and the status of the interrupt source can be obtained using the function PLIB_INT_SourceFlagGet. The interrupt status flags can be used to indicate the interrupt status even if the interrupt is not enabled for the source. When the interrupt is not enabled for the source, no interrupt is generated to the core even though the interrupt controller still recognizes that the source requested an interrupt. INT_TRAP_SOURCE is a list of possible trap events, and INT_SOURCE is a list of possible interrupt sources.

Before the Interrupt Service Routine (ISR) ends, the status of the interrupt needs to be cleared to avoid entering the ISR repeatedly for the same interrupt event. A trap source status can be cleared using the function PLIB_INT_TrapSourceFlagClear, and an interrupt source status can be cleared using the function PLIB_INT_SourceFlagClear. INT_TRAP_SOURCE is a list of possible trap events, and INT_SOURCE is a list of possible interrupt sources.

Configuring the Library

The library is configured for the supported Interrupt module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SetState	Restores the status of CPU interrupts based on the value passed into the function.
	PLIB_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	PLIB_INT_VariableVectorOffsetSet	Sets the offset specific to an interrupt source number.
	PLIB_INT_SoftwareNMITrigger	Triggers software NMI.

b) Interrupt Source Control Functions

	Name	Description
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.

	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

c) Other Status and Control Functions

	Name	Description
	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.
	PLIB_INT_GetStateAndDisable	Disables the generation of interrupts to the CPU.
	PLIB_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	PLIB_INT_VariableVectorOffsetGet	Gets the offset specific to an interrupt source number.

d) Feature Existence Functions

	Name	Description
	PLIB_INT_ExistsCPUCurrentPriorityLevel	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.
	PLIB_INT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Interrupt module.
	PLIB_INT_ExistsExternalINTEdgeSelect	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.
	PLIB_INT_ExistsINTCPUPriority	Identifies whether the INTCPUPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsINTCPUVector	Identifies whether the INTCPUVector feature exists on the Interrupt module.
	PLIB_INT_ExistsProximityTimerControl	Identifies whether the ProximityTimerControl feature exists on the Interrupts module.
	PLIB_INT_ExistsProximityTimerEnable	Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.
	PLIB_INT_ExistsSingleVectorShadowSet	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceControl	Identifies whether the SourceControl feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceFlag	Identifies whether the SourceFlag feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorPriority	Identifies whether the VectorPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt module.
	PLIB_INT_ExistsShadowRegisterAssign	Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.
	PLIB_INT_ExistsVariableOffset	Identifies whether the VariableOffset feature exists on the Interrupt module.
	PLIB_INT_ExistsSoftwareNMI	Identifies whether the SoftwareNMI feature exists on the Interrupt module.

e) Data Types and Constants

	Name	Description
	INT_EXTERNAL_SOURCES	Lists the possible external sources that can cause an interrupt to occur.
	INT_PRIORITY_LEVEL	Lists the possible interrupt priority levels.
	INT_SOURCE	Lists the possible sources that can cause an interrupt to occur.
	INT_SUBPRIORITY_LEVEL	Lists the possible interrupt sub priority levels.
	INT_VECTOR	Lists the possible interrupt vectors.
	INT_STATE_GLOBAL	Data type defining the global interrupt state.
	INT_SHADOW_REGISTER	Lists the possible shadow register sets.
	INT_VECTOR_SPACING	Lists the possible interrupt vector spacing.

Description

This section describes the Application Programming Interface (API) functions of the Interrupt Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_INT_Disable Function

Disables the generation of interrupts to the CPU.

File

plib_int.h

C

```
void PLIB_INT_Disable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables the generation of interrupts to the CPU.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_Disable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_Disable (INT_MODULE_ID index)

PLIB_INT_Enable Function

Enables the generation of interrupts to the CPU.

File

plib_int.h

C

```
void PLIB_INT_Enable(INT_MODULE_ID index);
```

Returns

None.

Description

This function enables the generation of interrupts to the CPU.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_Enable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_Enable (INT_MODULE_ID index)

PLIB_INT_ExternalFallingEdgeSelect Function

Selects the falling edge as the edge polarity of the external interrupt.

File

plib_int.h

C

```
void PLIB_INT_ExternalFallingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
```

Returns

None.

Description

This function selects the falling edge as the edge polarity of the external interrupt.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsExternalINTEdgeSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ExternalFallingEdgeSelect( INT_ID_0,
                                    (INT_EXTERNAL_INT_SOURCE0 |
                                    INT_EXTERNAL_INT_SOURCE1) );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One or more of the values from INT_EXTERNAL_SOURCES. Values can be combined using a bitwise "OR" operation.

Function

void PLIB_INT_ExternalFallingEdgeSelect (INT_MODULE_ID index, INT_EXTERNAL_SOURCES source)

PLIB_INT_ExternalRisingEdgeSelect Function

Selects the rising edge as the edge polarity of the external interrupt.

File

plib_int.h

C

```
void PLIB_INT_ExternalRisingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
```

Returns

None.

Description

This function selects the rising edge as the edge polarity of the external interrupt.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsExternalINTEdgeSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ExternalRisingEdgeSelect( INT_ID_0,
                                   ( INT_EXTERNAL_INT_SOURCE0 |
                                       INT_EXTERNAL_INT_SOURCE1 ) );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One or more of the values from INT_EXTERNAL_SOURCES. Values can be combined using a bitwise "OR" operation.

Function

void PLIB_INT_ExternalRisingEdgeSelect (INT_MODULE_ID index,
INT_EXTERNAL_SOURCES source)

PLIB_INT_IsEnabled Function

Identifies if interrupts are currently enabled or disabled at the top level.

File

plib_int.h

C

```
bool PLIB_INT_IsEnabled(INT_MODULE_ID index);
```

Returns

- true - If the interrupts are currently enabled
- false - If the interrupts are currently disabled

Description

This function identifies if interrupts are enabled or disabled at the top level.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
status = PLIB_INT_IsEnabled( INT_ID_0 );
```

Function

bool PLIB_INT_IsEnabled (INT_MODULE_ID index)

PLIB_INT_MultiVectorSelect Function

Configures the Interrupt Controller for Multiple Vector mode.

File

plib_int.h

C

```
void PLIB_INT_MultiVectorSelect(INT_MODULE_ID index);
```

Returns

None.

Description

This function configures the Interrupt Controller for Multiple Vector mode. Interrupt requests will serviced at the calculated vector addresses. The CPU vectors to the unique address for each vector number.

Remarks

While the user can, during run-time, reconfigure the interrupt controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_MultiVectorSelect( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_MultiVectorSelect(INT_MODULE_ID index)

PLIB_INT_PriorityGet Function

Returns the priority level of the latest interrupt presented to the CPU.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_PriorityGet(INT_MODULE_ID index);
```

Returns

One of the possible values from INT_PRIORITY_LEVEL.

Description

This function returns the priority level of the latest interrupt presented to the CPU.

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function PLIB_INT_SingleVectorSelect.

This function implements an operation of the INTCPUPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsINTCPUPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL level = PLIB_INT_RequestedPriorityLevelGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_PRIORITY_LEVEL PLIB_INT_PriorityGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerDisable Function

Disables the interrupt temporal-proximity timer.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerDisable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables the interrupt temporal-proximity timer.

Remarks

This function implements an operation of the ProximityTimerEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerEnable function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ProximityTimerDisable(INT_PRIORITY_LEVEL3);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_ProximityTimerDisable(INT_MODULE_ID index)

PLIB_INT_ProximityTimerEnable Function

Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerEnable(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

This function enables the interrupt temporal-proximity timer and selects the priority levels that start the timer. One of the possible values from PLIB_INT_PRIORITY_LEVEL can be selected. Interrupts of that priority and lower start the temporal proximity timer.

Remarks

This function implements an operation of the ProximityTimerEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerEnable function in your application to determine whether this feature is available.

Choosing INT_PRIORITY_0 disables the proximity timer (exactly the same as if PLIB_INT_ProximityTimerDisable had been called.

Preconditions

None.

Example

```
//Interrupts of group priority 3 or lower start the temporal proximity timer
PLIB_INT_ProximityTimerEnable( INT_ID_0, INT_PRIORITY_LEVEL3 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
priority	One of possible values from INT_PRIORITY_LEVEL.

Function

void PLIB_INT_ProximityTimerEnable (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority)

PLIB_INT_SingleVectorSelect Function

Configures the Interrupt Controller for Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorSelect(INT_MODULE_ID index);
```

Returns

None.

Description

The function configures the Interrupt Controller for Single Vector mode. All interrupt requests will serviced at one vector address. The CPU vectors to the same address for all interrupt sources.

Remarks

While the user can, during run-time, reconfigure the Interrupt Controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorSelect( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorSelect (INT_MODULE_ID index)

PLIB_INT_SingleVectorShadowSetDisable Function

Disables the Shadow Register Set in Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorShadowSetDisable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSingleVectorShadowSet function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorShadowSetDisable (INT_MODULE_ID index)

PLIB_INT_SingleVectorShadowSetEnable Function

Enables the Shadow Register Set in Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorShadowSetEnable(INT_MODULE_ID index);
```

Returns

None.

Description

This function enables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSingleVectorShadowSet function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorShadowSetEnable (INT_MODULE_ID index)

PLIB_INT_SetState Function

Restores the status of CPU interrupts based on the value passed into the function.

File

plib_int.h

C

```
void PLIB_INT_SetState(INT_MODULE_ID index, INT_STATE_GLOBAL interrupt_state);
```

Returns

None.

Description

This function will use the value passed in, to set the state of global interrupts.

Remarks

This function should be paired with the use of PLIB_INT_GetStateAndDisable. The value returned from PLIB_INT_GetStateAndDisable should be passed into this function.

Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_STATE_GLOBAL interrupt_value;

interrupt_value = PLIB_INT_GetStateAndDisable( INT_ID_0 );

PLIB_INT_SetState(INT_ID_0, interrupt_value);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
interrupt_state	value returned from previous call to PLIB_INT_GetStateAndDisable.

Function

void PLIB_INT_SetState (INT_MODULE_ID index, INT_STATE_GLOBAL interrupt_state)

PLIB_INT_ShadowRegisterAssign Function

Assigns a shadow register set for an interrupt priority level.

File

plib_int.h

C

```
void PLIB_INT_ShadowRegisterAssign(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority, INT_SHADOW_REGISTER
shadowRegister);
```

Returns

None.

Description

The function assigns a shadow register set for an interrupt priority level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsShadowRegisterAssign function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ShadowRegisterAssign( INT_ID_0, INT_PRIORITY_LEVEL5, INT_SHADOW_REGISTER_5 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all devices that have only one Interrupt module).
priority	Interrupt priority level for the shadow register set that is to be assigned.
shadowRegister	Shadow register set number.

Function

void PLIB_INT_ShadowRegisterAssign (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority,
INT_SHADOW_REGISTER shadowRegister)

PLIB_INT_VariableVectorOffsetSet Function

Sets the offset specific to an interrupt source number.

File

plib_int.h

C

```
void PLIB_INT_VariableVectorOffsetSet(INT_MODULE_ID index, INT_VECTOR vector, uint32_t offset);
```

Returns

None.

Description

The function sets the offset specific to an interrupt source number.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVariableOffset function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Set 200 bytes offset
PLIB_INT_VariableVectorOffsetSet ( INT_ID_0, INT_VECTOR_USB1, 200 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	Interrupt source, one of the possible value from INT_VECTOR enum.
offset	Offset in number of bytes.

Function

void PLIB_INT_VariableVectorOffsetSet (INT_MODULE_ID index, INT_VECTOR source, uint32_t offset)

PLIB_INT_SoftwareNMITrigger Function

Triggers software NMI.

File

plib_int.h

C

```
void PLIB_INT_SoftwareNMITrigger(INT_MODULE_ID index);
```

Returns

None.

Description

This function triggers the software NMI.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_INT_ExistsSoftwareNMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SoftwareNMITrigger( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SoftwareNMITrigger(INT_MODULE_ID index)

b) Interrupt Source Control Functions

PLIB_INT_SourceDisable Function

Disables the interrupt source

File

plib_int.h

C

```
void PLIB_INT_SourceDisable(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function disables the given interrupt source.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceDisable(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceDisable (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceEnable Function

Enables the interrupt source.

File

plib_int.h

C

```
void PLIB_INT_SourceEnable(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function enables the interrupt source. The interrupt flag is set when the interrupt request is sampled. The pending interrupt request will not cause further processing if the interrupt is not enabled using this function or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceEnable(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceEnable (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagClear Function

Clears the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
void PLIB_INT_SourceFlagClear(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function clears the status of the interrupt flag for the selected source. The flag is set when the interrupt request is identified. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function PLIB_INT_InterruptSourceEnable or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0,INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceFlagClear (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagGet Function

Returns the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
bool PLIB_INT_SourceFlagGet(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

- true - If the interrupt request is recognized for the source
- false - If the interrupt request is not recognized for the source

Description

This function returns the status of the interrupt flag for the selected source. The flag is set when the interrupt request is recognized. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function PLIB_INT_InterruptSourceEnable or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
```

```
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

bool PLIB_INT_SourceFlagGet (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagSet Function

Sets the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
void PLIB_INT_SourceFlagSet(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function sets the status of the interrupt flag for the selected source. This function will not be used during normal operation of the system. It is used to generate test interrupts for debug and testing purposes.

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceFlagSet(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceFlagSet (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceIsEnabled Function

Gets the enable state of the interrupt source.

File

plib_int.h

C

```
bool PLIB_INT_SourceIsEnabled(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

- true - If the interrupt source is enabled
- false - If the interrupt source is disabled

Description

This function gets the enable state of the interrupt source.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceIsEnabled(INT_ID_0,INT_SOURCE_TIMER_1) != true)
{
    PLIB_INT_SourceEnable(INT_ID_0,INT_SOURCE_TIMER_1);
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

bool PLIB_INT_SourceIsEnabled (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_VectorPriorityGet Function

Gets the priority of the interrupt vector.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_VectorPriorityGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

- priority - One of the possible values from INT_PRIORITY_LEVEL

Description

Gets the priority of the interrupt vector. The priority is one of the possible values from INT_PRIORITY_LEVEL

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL level;
level = PLIB_INT_VectorPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR

Function

PLIB_INT_PRIORITY_LEVEL INT_VectorPriorityGet (INT_MODULE_ID index, INT_VECTOR vector)

PLIB_INT_VectorPrioritySet Function

Sets the priority of the interrupt vector.

File

plib_int.h

C

```
void PLIB_INT_VectorPrioritySet(INT_MODULE_ID index, INT_VECTOR vector, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

Sets the priority of the interrupt vector. The priority is one of the possible values from INT_PRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_PRIORITY_LEVEL4);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR
priority	One of the possible values from INT_PRIORITY_LEVEL

Function

void PLIB_INT_VectorPrioritySet(INT_MODULE_ID index,
INT_VECTOR vector,
INT_PRIORITY_LEVEL priority)

PLIB_INT_VectorSubPriorityGet Function

Gets the sub-priority of the interrupt vector.

File

plib_int.h

C

```
INT_SUBPRIORITY_LEVEL PLIB_INT_VectorSubPriorityGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

- priority - One of the possible values from INT_SUBPRIORITY_LEVEL

Description

This function gets the sub-priority of the interrupt vector. The priority is one of the possible values from INT_SUBPRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_SUBPRIORITY_LEVEL level;
level = PLIB_INT_VectorSubPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR

Function

INT_SUBPRIORITY_LEVEL PLIB_INT_VectorSubPriorityGet (INT_MODULE_ID index,
INT_VECTOR vector)

PLIB_INT_VectorSubPrioritySet Function

Sets the sub-priority of the interrupt vector.

File

plib_int.h

C

```
void PLIB_INT_VectorSubPrioritySet(INT_MODULE_ID index, INT_VECTOR vector, INT_SUBPRIORITY_LEVEL
subPriority);
```

Returns

None.

Description

This function sets the sub priority of the interrupt vector. The sub-priority is one of the possible values from INT_SUBPRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_SUBPRIORITY_LEVEL1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR
subPriority	One of the possible values from INT_SUBPRIORITY_LEVEL

Function

void PLIB_INT_VectorSubPrioritySet (INT_MODULE_ID index,
INT_VECTOR vector,
INT_SUBPRIORITY_LEVEL subPriority)

c) Other Status and Control Functions

PLIB_INT_CPUCurrentPriorityLevelGet Function

Gets the interrupt priority level at which the CPU is currently operating.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_CPUCurrentPriorityLevelGet(INT_MODULE_ID index);
```

Returns

- priority - The current interrupt priority of the CPU. Range (0 - 15)

Description

This function gets the interrupt priority level at which the CPU is currently operating.

Remarks

User interrupts are disabled when the CPU priority is more than 7.

This function implements an operation of the CPUCurrentPriorityLevel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsCPUCurrentPriorityLevel function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL currentCPUPriority;
currentCPUPriority = PLIB_INT_CPUCurrentPriorityLevelGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_PRIORITY_LEVEL PLIB_INT_CPUCurrentPriorityLevelGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerGet Function

Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.

File

plib_int.h

C

```
uint32_t PLIB_INT_ProximityTimerGet(INT_MODULE_ID index);
```

Returns

Timer reload value.

Description

This function returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t timer;

timer = PLIB_INT_ProximityTimerGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

uint32_t PLIB_INT_ProximityTimerGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerSet Function

Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerSet(INT_MODULE_ID index, uint32_t timerreloadvalue);
```

Returns

None.

Description

This function sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ProximityTimerSet(INT_ID_0, 0x12345678);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
timerReloadValue	Timer reload value.

Function

void PLIB_INT_ProximityTimerSet (INT_MODULE_ID index, uint32_t timerReloadValue)

PLIB_INT_VectorGet Function

Returns the interrupt vector that is presented to the CPU.

File

plib_int.h

C

```
INT_VECTOR PLIB_INT_VectorGet(INT_MODULE_ID index);
```

Returns

One of the possible values from INT_VECTOR.

Description

This function returns the interrupt vector that is presented to the CPU.

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function PLIB_INT_MultiVectorDisable.

This function implements an operation of the INTCPUVector feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsINTCPUVector function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_VECTOR level = PLIB_INT_VectorGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_VECTOR PLIB_INT_VectorGet (INT_MODULE_ID index)

PLIB_INT_GetStateAndDisable Function

Disables the generation of interrupts to the CPU.

File

plib_int.h

C

```
INT_STATE_GLOBAL PLIB_INT_GetStateAndDisable(INT_MODULE_ID index);
```

Returns

Value of CP0 Status register before interrupts were disabled globally.

Description

This function disables the generation of interrupts to the CPU.

Remarks

This function should be paired with the use of PLIB_INT_SetState. The value returned from this function should be passed into PLIB_INT_SetState.

Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_STATE_GLOBAL interrupt_value;

interrupt_value = PLIB_INT_GetStateAndDisable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_STATE_GLOBAL PLIB_INT_GetStateAndDisable (INT_MODULE_ID index)

PLIB_INT_ShadowRegisterGet Function

Gets the shadow register set assigned for an interrupt priority level.

File

plib_int.h

C

```
INT_SHADOW_REGISTER PLIB_INT_ShadowRegisterGet(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

The function gets the shadow register set assigned for an interrupt priority level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsShadowRegisterAssign function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_SHADOW_REGISTER shadowReg;

shadowReg = PLIB_INT_ShadowRegisterGet( INT_ID_0, INT_PRIORITY_LEVEL5 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
priority	Interrupt priority level for the shadow register set that is to be assigned.

Function

INT_SHADOW_REGISTER PLIB_INT_ShadowRegisterGet (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority)

PLIB_INT_VariableVectorOffsetGet Function

Gets the offset specific to an interrupt source number.

File

plib_int.h

C

```
uint32_t PLIB_INT_VariableVectorOffsetGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

Offset value.

Description

The function gets the offset specific to an interrupt source number.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVariableOffset function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t offset;

offset = PLIB_INT_VariableVectorOffsetGet ( INT_ID_0, INT_VECTOR_USB1 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	Interrupt source, one of the possible value from INT_VECTOR enum.

Function

uint32_t PLIB_INT_VariableVectorOffsetGet (INT_MODULE_ID index, INT_VECTOR vector)

d) Feature Existence Functions

PLIB_INT_ExistsCPUCurrentPriorityLevel Function

Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsCPUCurrentPriorityLevel(INT_MODULE_ID index);
```

Returns

- true - The CPUCurrentPriorityLevel feature is supported on the device
- false - The CPUCurrentPriorityLevel feature is not supported on the device

Description

This function identifies whether the CPUCurrentPriorityLevel feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_CPUCurrentPriorityLevelGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsCPUCurrentPriorityLevel(INT_MODULE_ID index)

PLIB_INT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsEnableControl(INT_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_Enable
- PLIB_INT_Disable
- PLIB_INT_IsEnabled
- PLIB_INT_SetState
- PLIB_INT_GetStateAndDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsEnableControl(INT_MODULE_ID index)

PLIB_INT_ExistsExternalINTEdgeSelect Function

Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsExternalINTEdgeSelect(INT_MODULE_ID index);
```

Returns

- true - The ExternalINTEdgeSelect feature is supported on the device
- false - The ExternalINTEdgeSelect feature is not supported on the device

Description

This function identifies whether the ExternalINTEdgeSelect feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ExternalRisingEdgeSelect
- PLIB_INT_ExternalFallingEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsExternalINTEdgeSelect(INT_MODULE_ID index)

PLIB_INT_ExistsINTCPUPriority Function

Identifies whether the INTCPUPriority feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsINTCPUPriority(INT_MODULE_ID index);
```

Returns

- true - The INTCPUPriority feature is supported on the device
- false - The INTCPUPriority feature is not supported on the device

Description

This function identifies whether the INTCPUPriority feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_PriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsINTCPUPriority(INT_MODULE_ID index)

PLIB_INT_ExistsINTCPUVector Function

Identifies whether the INTCPUVector feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsINTCPUVector(INT_MODULE_ID index);
```

Returns

- true - The INTCPUVector feature is supported on the device
- false - The INTCPUVector feature is not supported on the device

Description

This function identifies whether the INTCPUVector feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_VectorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsINTCPUVector(INT_MODULE_ID index)

PLIB_INT_ExistsProximityTimerControl Function

Identifies whether the ProximityTimerControl feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsProximityTimerControl(INT_MODULE_ID index);
```

Returns

- true - The ProximityTimerControl feature is supported on the device
- false - The ProximityTimerControl feature is not supported on the device

Description

This function identifies whether the ProximityTimerControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ProximityTimerSet
- PLIB_INT_ProximityTimerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsProximityTimerControl(INT_MODULE_ID index)

PLIB_INT_ExistsProximityTimerEnable Function

Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsProximityTimerEnable(INT_MODULE_ID index);
```

Returns

- true - The ProximityTimerEnable feature is supported on the device
- false - The ProximityTimerEnable feature is not supported on the device

Description

This function identifies whether the ProximityTimerEnable feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ProximityTimerEnable
- PLIB_INT_ProximityTimerDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsProximityTimerEnable(INT_MODULE_ID index)

PLIB_INT_ExistsSingleVectorShadowSet Function

Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSingleVectorShadowSet(INT_MODULE_ID index);
```

Returns

- true - The SingleVectorShadowSet feature is supported on the device
- false - The SingleVectorShadowSet feature is not supported on the device

Description

This function identifies whether the SingleVectorShadowSet feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SingleVectorShadowSetDisable
- PLIB_INT_SingleVectorShadowSetEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSingleVectorShadowSet(INT_MODULE_ID index)

PLIB_INT_ExistsSourceControl Function

Identifies whether the SourceControl feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSourceControl(INT_MODULE_ID index);
```

Returns

- true - The SourceControl feature is supported on the device
- false - The SourceControl feature is not supported on the device

Description

This function identifies whether the SourceControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SourceEnable
- PLIB_INT_SourceDisable
- PLIB_INT_SourceIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSourceControl(INT_MODULE_ID index)

PLIB_INT_ExistsSourceFlag Function

Identifies whether the SourceFlag feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSourceFlag(INT_MODULE_ID index);
```

Returns

- true - The SourceFlag feature is supported on the device
- false - The SourceFlag feature is not supported on the device

Description

This function identifies whether the SourceFlag feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SourceFlagGet
- PLIB_INT_SourceFlagSet
- PLIB_INT_SourceFlagClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSourceFlag(INT_MODULE_ID index)

PLIB_INT_ExistsVectorPriority Function

Identifies whether the VectorPriority feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVectorPriority(INT_MODULE_ID index);
```

Returns

- true - The VectorPriority feature is supported on the device
- false - The VectorPriority feature is not supported on the device

Description

This function identifies whether the VectorPriority feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_VectorPrioritySet
- PLIB_INT_VectorPriorityGet
- PLIB_INT_VectorSubPrioritySet
- PLIB_INT_VectorSubPriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVectorPriority(INT_MODULE_ID index)

PLIB_INT_ExistsVectorSelect Function

Identifies whether the VectorSelect feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVectorSelect(INT_MODULE_ID index);
```

Returns

- true - The VectorSelect feature is supported on the device
- false - The VectorSelect feature is not supported on the device

Description

This function identifies whether the VectorSelect feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_MultiVectorSelect
- PLIB_INT_SingleVectorSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVectorSelect(INT_MODULE_ID index)

PLIB_INT_ExistsShadowRegisterAssign Function

Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsShadowRegisterAssign(INT_MODULE_ID index);
```

Returns

- true - The ShadowRegisterAssign feature is supported on the device
- false - The ShadowRegisterAssign feature is not supported on the device

Description

This function identifies whether the ShadowRegisterAssign feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ShadowRegisterAssign
- PLIB_INT_ShadowRegisterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsShadowRegisterAssign(INT_MODULE_ID index)

PLIB_INT_ExistsVariableOffset Function

Identifies whether the VariableOffset feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVariableOffset(INT_MODULE_ID index);
```

Returns

- true - The VariableOffset feature is supported on the device
- false - The VariableOffset feature is not supported on the device

Description

This function identifies whether the VariableOffset feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_VariableOffsetSet
- PLIB_INT_VariableOffsetGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVariableOffset(INT_MODULE_ID index)

PLIB_INT_ExistsSoftwareNMI Function

Identifies whether the SoftwareNMI feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSoftwareNMI(INT_MODULE_ID index);
```

Returns

- true - The SoftwareNMI feature is supported on the device
- false - The SoftwareNMI feature is not supported on the device

Description

This function identifies whether the SoftwareNMI feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_SoftwareNMITrigger

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSoftwareNMI(INT_MODULE_ID index)

e) Data Types and Constants

INT_EXTERNAL_SOURCES Enumeration

Lists the possible external sources that can cause an interrupt to occur.

File

plib_int_help.h

C

```
typedef enum {
  INT_EXTERNAL_INT_SOURCE0,
  INT_EXTERNAL_INT_SOURCE1,
  INT_EXTERNAL_INT_SOURCE2,
  INT_EXTERNAL_INT_SOURCE3,
  INT_EXTERNAL_INT_SOURCE4
} INT_EXTERNAL_SOURCES;
```

Members

Members	Description
INT_EXTERNAL_INT_SOURCE0	External Interrupt Source 0

INT_EXTERNAL_INT_SOURCE1	External Interrupt Source 1
INT_EXTERNAL_INT_SOURCE2	External Interrupt Source 2
INT_EXTERNAL_INT_SOURCE3	External Interrupt Source 3
INT_EXTERNAL_INT_SOURCE4	External Interrupt Source 4

Description

External Interrupt Sources

This enumeration lists the possible external sources that can cause an interrupt to occur.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_PRIORITY_LEVEL Enumeration

Lists the possible interrupt priority levels.

File

plib_int_help.h

C

```
typedef enum {
  INT_PRIORITY_LEVEL0,
  INT_PRIORITY_LEVEL1,
  INT_PRIORITY_LEVEL2,
  INT_PRIORITY_LEVEL3,
  INT_PRIORITY_LEVEL4,
  INT_PRIORITY_LEVEL5,
  INT_PRIORITY_LEVEL6,
  INT_PRIORITY_LEVEL7
} INT_PRIORITY_LEVEL;
```

Members

Members	Description
INT_PRIORITY_LEVEL0	Disabled
INT_PRIORITY_LEVEL1	Priority 1
INT_PRIORITY_LEVEL2	Priority 2
INT_PRIORITY_LEVEL3	Priority 3
INT_PRIORITY_LEVEL4	Priority 4
INT_PRIORITY_LEVEL5	Priority 5
INT_PRIORITY_LEVEL6	Priority 6
INT_PRIORITY_LEVEL7	Priority 7

Description

Priority Level

This enumeration lists the possible interrupt priority levels that can be selected for each source.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_SOURCE Enumeration

Lists the possible sources that can cause an interrupt to occur.

File

plib_int_help.h

C

```
typedef enum {
  INT_SOURCE_SOFTWARE_0,
  INT_SOURCE_SOFTWARE_1,
  INT_SOURCE_EXTERNAL_0,
  INT_SOURCE_EXTERNAL_1,
```

```
    INT_SOURCE_EXTERNAL_2,
    INT_SOURCE_EXTERNAL_3,
    INT_SOURCE_EXTERNAL_4,
    INT_SOURCE_TIMER_CORE,
    INT_SOURCE_TIMER_1,
    INT_SOURCE_TIMER_2,
    INT_SOURCE_TIMER_3,
    INT_SOURCE_TIMER_4,
    INT_SOURCE_TIMER_5,
    INT_SOURCE_INPUT_CAPTURE_1,
    INT_SOURCE_INPUT_CAPTURE_1_ERROR,
    INT_SOURCE_INPUT_CAPTURE_2,
    INT_SOURCE_INPUT_CAPTURE_2_ERROR,
    INT_SOURCE_INPUT_CAPTURE_3,
    INT_SOURCE_INPUT_CAPTURE_3_ERROR,
    INT_SOURCE_INPUT_CAPTURE_4,
    INT_SOURCE_INPUT_CAPTURE_4_ERROR,
    INT_SOURCE_INPUT_CAPTURE_5,
    INT_SOURCE_INPUT_CAPTURE_5_ERROR,
    INT_SOURCE_OUTPUT_COMPARE_1,
    INT_SOURCE_OUTPUT_COMPARE_2,
    INT_SOURCE_OUTPUT_COMPARE_3,
    INT_SOURCE_OUTPUT_COMPARE_4,
    INT_SOURCE_OUTPUT_COMPARE_5,
    INT_SOURCE_SPI_1_ERROR,
    INT_SOURCE_SPI_1_RECEIVE,
    INT_SOURCE_SPI_1_TRANSMIT,
    INT_SOURCE_SPI_2_ERROR,
    INT_SOURCE_SPI_2_RECEIVE,
    INT_SOURCE_SPI_2_TRANSMIT,
    INT_SOURCE_SPI_3_ERROR,
    INT_SOURCE_SPI_3_RECEIVE,
    INT_SOURCE_SPI_3_TRANSMIT,
    INT_SOURCE_SPI_4_ERROR,
    INT_SOURCE_SPI_4_RECEIVE,
    INT_SOURCE_SPI_4_TRANSMIT,
    INT_SOURCE_USART_1_ERROR,
    INT_SOURCE_USART_1_RECEIVE,
    INT_SOURCE_USART_1_TRANSMIT,
    INT_SOURCE_USART_2_ERROR,
    INT_SOURCE_USART_2_RECEIVE,
    INT_SOURCE_USART_2_TRANSMIT,
    INT_SOURCE_USART_3_ERROR,
    INT_SOURCE_USART_3_RECEIVE,
    INT_SOURCE_USART_3_TRANSMIT,
    INT_SOURCE_USART_4_ERROR,
    INT_SOURCE_USART_4_RECEIVE,
    INT_SOURCE_USART_4_TRANSMIT,
    INT_SOURCE_USART_5_ERROR,
    INT_SOURCE_USART_5_RECEIVE,
    INT_SOURCE_USART_5_TRANSMIT,
    INT_SOURCE_USART_6_ERROR,
    INT_SOURCE_USART_6_RECEIVE,
    INT_SOURCE_USART_6_TRANSMIT,
    INT_SOURCE_I2C_1_ERROR,
    INT_SOURCE_I2C_1_SLAVE,
    INT_SOURCE_I2C_1_MASTER,
    INT_SOURCE_I2C_2_ERROR,
    INT_SOURCE_I2C_2_SLAVE,
    INT_SOURCE_I2C_2_MASTER,
    INT_SOURCE_I2C_3_ERROR,
    INT_SOURCE_I2C_3_SLAVE,
    INT_SOURCE_I2C_3_MASTER,
    INT_SOURCE_I2C_4_ERROR,
    INT_SOURCE_I2C_4_SLAVE,
    INT_SOURCE_I2C_4_MASTER,
    INT_SOURCE_I2C_5_ERROR,
    INT_SOURCE_I2C_5_SLAVE,
    INT_SOURCE_I2C_5_MASTER,
    INT_SOURCE_CHANGE_NOTICE,
    INT_SOURCE_CHANGE_NOTICE_A,
    INT_SOURCE_CHANGE_NOTICE_B,
    INT_SOURCE_CHANGE_NOTICE_C,
    INT_SOURCE_CHANGE_NOTICE_D,
    INT_SOURCE_CHANGE_NOTICE_E,
```

```
    INT_SOURCE_CHANGE_NOTICE_F,
    INT_SOURCE_CHANGE_NOTICE_G,
    INT_SOURCE_CHANGE_NOTICE_H,
    INT_SOURCE_CHANGE_NOTICE_J,
    INT_SOURCE_CHANGE_NOTICE_K,
    INT_SOURCE_ADC_1,
    INT_SOURCE_ADC_1_DC1,
    INT_SOURCE_ADC_1_DC2,
    INT_SOURCE_ADC_1_DC3,
    INT_SOURCE_ADC_1_DC4,
    INT_SOURCE_ADC_1_DC5,
    INT_SOURCE_ADC_1_DC6,
    INT_SOURCE_ADC_1_DF1,
    INT_SOURCE_ADC_1_DF2,
    INT_SOURCE_ADC_1_DF3,
    INT_SOURCE_ADC_1_DF4,
    INT_SOURCE_ADC_1_DF5,
    INT_SOURCE_ADC_1_DF6,
    INT_SOURCE_ADC_1_DATA0,
    INT_SOURCE_ADC_1_DATA1,
    INT_SOURCE_ADC_1_DATA2,
    INT_SOURCE_ADC_1_DATA3,
    INT_SOURCE_ADC_1_DATA4,
    INT_SOURCE_ADC_1_DATA5,
    INT_SOURCE_ADC_1_DATA6,
    INT_SOURCE_ADC_1_DATA7,
    INT_SOURCE_ADC_1_DATA8,
    INT_SOURCE_ADC_1_DATA9,
    INT_SOURCE_ADC_1_DATA10,
    INT_SOURCE_ADC_1_DATA11,
    INT_SOURCE_ADC_1_DATA12,
    INT_SOURCE_ADC_1_DATA13,
    INT_SOURCE_ADC_1_DATA14,
    INT_SOURCE_ADC_1_DATA15,
    INT_SOURCE_ADC_1_DATA16,
    INT_SOURCE_ADC_1_DATA17,
    INT_SOURCE_ADC_1_DATA18,
    INT_SOURCE_ADC_1_DATA19,
    INT_SOURCE_ADC_1_DATA20,
    INT_SOURCE_ADC_1_DATA21,
    INT_SOURCE_ADC_1_DATA22,
    INT_SOURCE_ADC_1_DATA23,
    INT_SOURCE_ADC_1_DATA24,
    INT_SOURCE_ADC_1_DATA25,
    INT_SOURCE_ADC_1_DATA26,
    INT_SOURCE_ADC_1_DATA27,
    INT_SOURCE_ADC_1_DATA28,
    INT_SOURCE_ADC_1_DATA29,
    INT_SOURCE_ADC_1_DATA30,
    INT_SOURCE_ADC_1_DATA31,
    INT_SOURCE_ADC_1_DATA32,
    INT_SOURCE_ADC_1_DATA33,
    INT_SOURCE_ADC_1_DATA34,
    INT_SOURCE_ADC_1_DATA35,
    INT_SOURCE_ADC_1_DATA36,
    INT_SOURCE_ADC_1_DATA37,
    INT_SOURCE_ADC_1_DATA38,
    INT_SOURCE_ADC_1_DATA39,
    INT_SOURCE_ADC_1_DATA40,
    INT_SOURCE_ADC_1_DATA41,
    INT_SOURCE_ADC_1_DATA42,
    INT_SOURCE_ADC_1_DATA43,
    INT_SOURCE_ADC_1_DATA44,
    INT_SOURCE_ADC_END_OF_SCAN,
    INT_SOURCE_ADC_ANALOG_CIRCUIT_READY,
    INT_SOURCE_ADC_UPDATE_READY,
    INT_SOURCE_ADC_GROUP,
    INT_SOURCE_ADC_0_EARLY,
    INT_SOURCE_ADC_1_EARLY,
    INT_SOURCE_ADC_2_EARLY,
    INT_SOURCE_ADC_3_EARLY,
    INT_SOURCE_ADC_4_EARLY,
    INT_SOURCE_ADC_7_EARLY,
    INT_SOURCE_ADC_0_WARM,
    INT_SOURCE_ADC_1_WARM,
```

```
    INT_SOURCE_ADC_2_WARM,
    INT_SOURCE_ADC_3_WARM,
    INT_SOURCE_ADC_4_WARM,
    INT_SOURCE_ADC_7_WARM,
    INT_SOURCE_PARALLEL_PORT,
    INT_SOURCE_PARALLEL_PORT_ERROR,
    INT_SOURCE_COMPARATOR_1,
    INT_SOURCE_COMPARATOR_2,
    INT_SOURCE_CLOCK_MONITOR,
    INT_SOURCE_RTCC,
    INT_SOURCE_DMA_0,
    INT_SOURCE_DMA_1,
    INT_SOURCE_DMA_2,
    INT_SOURCE_DMA_3,
    INT_SOURCE_DMA_4,
    INT_SOURCE_DMA_5,
    INT_SOURCE_DMA_6,
    INT_SOURCE_DMA_7,
    INT_SOURCE_FLASH_CONTROL,
    INT_SOURCE_USB_1,
    INT_SOURCE_CAN_1,
    INT_SOURCE_CAN_2,
    INT_SOURCE_ETH_1,
    INT_SOURCE_ADC_FAULT,
    INT_SOURCE_CRYPTO,
    INT_SOURCE_SPI_5_RECEIVE,
    INT_SOURCE_SPI_5_TRANSMIT,
    INT_SOURCE_SPI_6_ERROR,
    INT_SOURCE_SPI_6_RECEIVE,
    INT_SOURCE_SPI_6_TRANSMIT
} INT_SOURCE;
```

Members

Members	Description
INT_SOURCE_SOFTWARE_0	Software interrupt 0
INT_SOURCE_SOFTWARE_1	Software interrupt 1
INT_SOURCE_EXTERNAL_0	External interrupt 0
INT_SOURCE_EXTERNAL_1	External interrupt 1
INT_SOURCE_EXTERNAL_2	External interrupt 2
INT_SOURCE_EXTERNAL_3	External interrupt 3
INT_SOURCE_EXTERNAL_4	External interrupt 4
INT_SOURCE_TIMER_CORE	Core timer interrupt
INT_SOURCE_TIMER_1	Timer 1 interrupt
INT_SOURCE_TIMER_2	Timer 2 interrupt
INT_SOURCE_TIMER_3	Timer 3 interrupt
INT_SOURCE_TIMER_4	Timer 4 interrupt
INT_SOURCE_TIMER_5	Timer 5 interrupt
INT_SOURCE_INPUT_CAPTURE_1	Input Capture 1 interrupt
INT_SOURCE_INPUT_CAPTURE_1_ERROR	Input Capture 1 Error interrupt
INT_SOURCE_INPUT_CAPTURE_2	Input Capture 2 interrupt
INT_SOURCE_INPUT_CAPTURE_2_ERROR	Input Capture 2 Error interrupt
INT_SOURCE_INPUT_CAPTURE_3	Input Capture 3 interrupt
INT_SOURCE_INPUT_CAPTURE_3_ERROR	Input Capture 3 Error interrupt
INT_SOURCE_INPUT_CAPTURE_4	Input Capture 4 interrupt
INT_SOURCE_INPUT_CAPTURE_4_ERROR	Input Capture 4 Error interrupt
INT_SOURCE_INPUT_CAPTURE_5	Input Capture 5 interrupt
INT_SOURCE_INPUT_CAPTURE_5_ERROR	Input Capture 5 Error interrupt
INT_SOURCE_OUTPUT_COMPARE_1	Output Compare 1 interrupt
INT_SOURCE_OUTPUT_COMPARE_2	Output Compare 2 interrupt
INT_SOURCE_OUTPUT_COMPARE_3	Output Compare 3 interrupt
INT_SOURCE_OUTPUT_COMPARE_4	Output Compare 4 interrupt
INT_SOURCE_OUTPUT_COMPARE_5	Output Compare 5 interrupt
INT_SOURCE_SPI_1_ERROR	SPI1 Error interrupt

INT_SOURCE_SPI_1_RECEIVE	SPI1 Receive Done interrupt
INT_SOURCE_SPI_1_TRANSMIT	SPI1 Transmit Done interrupt
INT_SOURCE_SPI_2_ERROR	SPI2 Error interrupt
INT_SOURCE_SPI_2_RECEIVE	SPI2 Receive Done interrupt
INT_SOURCE_SPI_2_TRANSMIT	SPI2 Transmit Done interrupt
INT_SOURCE_SPI_3_ERROR	SPI3 Error interrupt
INT_SOURCE_SPI_3_RECEIVE	SPI3 Receive Done interrupt
INT_SOURCE_SPI_3_TRANSMIT	SPI3 Transmit Done interrupt
INT_SOURCE_SPI_4_ERROR	SPI4 Error interrupt
INT_SOURCE_SPI_4_RECEIVE	SPI4 Receive Done interrupt
INT_SOURCE_SPI_4_TRANSMIT	SPI4 Transmit Done interrupt
INT_SOURCE_USART_1_ERROR	UART1 Error interrupt
INT_SOURCE_USART_1_RECEIVE	UART1 Receive interrupt
INT_SOURCE_USART_1_TRANSMIT	UART1 Transmit interrupt
INT_SOURCE_USART_2_ERROR	UART2 Error interrupt
INT_SOURCE_USART_2_RECEIVE	UART2 Receive interrupt
INT_SOURCE_USART_2_TRANSMIT	UART2 Transmit interrupt
INT_SOURCE_USART_3_ERROR	UART3 Error interrupt
INT_SOURCE_USART_3_RECEIVE	UART3 Receive interrupt
INT_SOURCE_USART_3_TRANSMIT	UART3 Transmit interrupt
INT_SOURCE_USART_4_ERROR	UART4 Error interrupt
INT_SOURCE_USART_4_RECEIVE	UART4 Receive interrupt
INT_SOURCE_USART_4_TRANSMIT	UART4 Transmit interrupt
INT_SOURCE_USART_5_ERROR	UART5 Error interrupt
INT_SOURCE_USART_5_RECEIVE	UART5 Receive interrupt
INT_SOURCE_USART_5_TRANSMIT	UART5 Transmit interrupt
INT_SOURCE_USART_6_ERROR	UART6 Error interrupt
INT_SOURCE_USART_6_RECEIVE	UART6 Receive interrupt
INT_SOURCE_USART_6_TRANSMIT	UART6 Transmit interrupt
INT_SOURCE_I2C_1_ERROR	I2C1 Bus Error Event interrupt
INT_SOURCE_I2C_1_SLAVE	I2C1 Slave Event interrupt
INT_SOURCE_I2C_1_MASTER	I2C1 Master Event interrupt
INT_SOURCE_I2C_2_ERROR	I2C2 Bus Error Event interrupt
INT_SOURCE_I2C_2_SLAVE	I2C2 Slave Event interrupt
INT_SOURCE_I2C_2_MASTER	I2C2 Master Event interrupt
INT_SOURCE_I2C_3_ERROR	I2C3 Bus Error Event interrupt
INT_SOURCE_I2C_3_SLAVE	I2C3 Slave Event interrupt
INT_SOURCE_I2C_3_MASTER	I2C3 Master Event interrupt
INT_SOURCE_I2C_4_ERROR	I2C4 Bus Error Event interrupt
INT_SOURCE_I2C_4_SLAVE	I2C4 Slave Event interrupt
INT_SOURCE_I2C_4_MASTER	I2C4 Master Event interrupt
INT_SOURCE_I2C_5_ERROR	I2C5 Bus Error Event interrupt
INT_SOURCE_I2C_5_SLAVE	I2C5 Slave Event interrupt
INT_SOURCE_I2C_5_MASTER	I2C5 Master Event interrupt
INT_SOURCE_CHANGE_NOTICE	Input Change Notice interrupt
INT_SOURCE_CHANGE_NOTICE_A	Input Change Notice Channel A interrupt
INT_SOURCE_CHANGE_NOTICE_B	Input Change Notice Channel B interrupt
INT_SOURCE_CHANGE_NOTICE_C	Input Change Notice Channel C interrupt
INT_SOURCE_CHANGE_NOTICE_D	Input Change Notice Channel D interrupt
INT_SOURCE_CHANGE_NOTICE_E	Input Change Notice Channel E interrupt
INT_SOURCE_CHANGE_NOTICE_F	Input Change Notice Channel F interrupt
INT_SOURCE_CHANGE_NOTICE_G	Input Change Notice Channel G interrupt
INT_SOURCE_CHANGE_NOTICE_H	Input Change Notice Channel H interrupt
INT_SOURCE_CHANGE_NOTICE_J	Input Change Notice Channel J interrupt
INT_SOURCE_CHANGE_NOTICE_K	Input Change Notice Channel K interrupt

INT_SOURCE_ADC_1	Analog-to-Digital Converter 1 interrupt
INT_SOURCE_ADC_1_DC1	ADC Digital Comparator 1
INT_SOURCE_ADC_1_DC2	ADC Digital Comparator 2
INT_SOURCE_ADC_1_DC3	ADC Digital Comparator 3
INT_SOURCE_ADC_1_DC4	ADC Digital Comparator 4
INT_SOURCE_ADC_1_DC5	ADC Digital Comparator 5
INT_SOURCE_ADC_1_DC6	ADC Digital Comparator 6
INT_SOURCE_ADC_1_DF1	ADC Digital Filter 1
INT_SOURCE_ADC_1_DF2	ADC Digital Filter 2
INT_SOURCE_ADC_1_DF3	ADC Digital Filter 3
INT_SOURCE_ADC_1_DF4	ADC Digital Filter 4
INT_SOURCE_ADC_1_DF5	ADC Digital Filter 5
INT_SOURCE_ADC_1_DF6	ADC Digital Filter 6
INT_SOURCE_ADC_1_DATA0	ADC Data 0
INT_SOURCE_ADC_1_DATA1	ADC Data 1
INT_SOURCE_ADC_1_DATA2	ADC Data 2
INT_SOURCE_ADC_1_DATA3	ADC Data 3
INT_SOURCE_ADC_1_DATA4	ADC Data 4
INT_SOURCE_ADC_1_DATA5	ADC Data 5
INT_SOURCE_ADC_1_DATA6	ADC Data 6
INT_SOURCE_ADC_1_DATA7	ADC Data 7
INT_SOURCE_ADC_1_DATA8	ADC Data 8
INT_SOURCE_ADC_1_DATA9	ADC Data 9
INT_SOURCE_ADC_1_DATA10	ADC Data 10
INT_SOURCE_ADC_1_DATA11	ADC Data 11
INT_SOURCE_ADC_1_DATA12	ADC Data 12
INT_SOURCE_ADC_1_DATA13	ADC Data 13
INT_SOURCE_ADC_1_DATA14	ADC Data 14
INT_SOURCE_ADC_1_DATA15	ADC Data 15
INT_SOURCE_ADC_1_DATA16	ADC Data 16
INT_SOURCE_ADC_1_DATA17	ADC Data 17
INT_SOURCE_ADC_1_DATA18	ADC Data 18
INT_SOURCE_ADC_1_DATA19	ADC Data 19
INT_SOURCE_ADC_1_DATA20	ADC Data 20
INT_SOURCE_ADC_1_DATA21	ADC Data 21
INT_SOURCE_ADC_1_DATA22	ADC Data 22
INT_SOURCE_ADC_1_DATA23	ADC Data 23
INT_SOURCE_ADC_1_DATA24	ADC Data 24
INT_SOURCE_ADC_1_DATA25	ADC Data 25
INT_SOURCE_ADC_1_DATA26	ADC Data 26
INT_SOURCE_ADC_1_DATA27	ADC Data 27
INT_SOURCE_ADC_1_DATA28	ADC Data 28
INT_SOURCE_ADC_1_DATA29	ADC Data 29
INT_SOURCE_ADC_1_DATA30	ADC Data 30
INT_SOURCE_ADC_1_DATA31	ADC Data 31
INT_SOURCE_ADC_1_DATA32	ADC Data 32
INT_SOURCE_ADC_1_DATA33	ADC Data 33
INT_SOURCE_ADC_1_DATA34	ADC Data 34
INT_SOURCE_ADC_1_DATA35	ADC Data 35
INT_SOURCE_ADC_1_DATA36	ADC Data 36
INT_SOURCE_ADC_1_DATA37	ADC Data 37
INT_SOURCE_ADC_1_DATA38	ADC Data 38
INT_SOURCE_ADC_1_DATA39	ADC Data 39
INT_SOURCE_ADC_1_DATA40	ADC Data 40
INT_SOURCE_ADC_1_DATA41	ADC Data 41

INT_SOURCE_ADC_1_DATA42	ADC Data 42
INT_SOURCE_ADC_1_DATA43	ADC Data 43
INT_SOURCE_ADC_1_DATA44	ADC Data 44
INT_SOURCE_ADC_END_OF_SCAN	ADC End of Scan
INT_SOURCE_ADC_ANALOG_CIRCUIT_READY	ADC Analog Circuit Ready
INT_SOURCE_ADC_UPDATE_READY	ADC Update Ready
INT_SOURCE_ADC_GROUP	ADC Group
INT_SOURCE_ADC_0_EARLY	ADC0 Early Interrupt
INT_SOURCE_ADC_1_EARLY	ADC1 Early Interrupt
INT_SOURCE_ADC_2_EARLY	ADC2 Early Interrupt
INT_SOURCE_ADC_3_EARLY	ADC3 Early Interrupt
INT_SOURCE_ADC_4_EARLY	ADC4 Early Interrupt
INT_SOURCE_ADC_7_EARLY	ADC7 Early Interrupt
INT_SOURCE_ADC_0_WARM	ADC0 Warm Interrupt
INT_SOURCE_ADC_1_WARM	ADC1 Warm Interrupt
INT_SOURCE_ADC_2_WARM	ADC2 Warm Interrupt
INT_SOURCE_ADC_3_WARM	ADC3 Warm Interrupt
INT_SOURCE_ADC_4_WARM	ADC4 Warm Interrupt
INT_SOURCE_ADC_7_WARM	ADC7 Warm Interrupt
INT_SOURCE_PARALLEL_PORT	Parallel Master Port interrupt
INT_SOURCE_PARALLEL_PORT_ERROR	Parallel Master Port Error interrupt
INT_SOURCE_COMPARATOR_1	Comparator 1 interrupt
INT_SOURCE_COMPARATOR_2	Comparator 2 interrupt
INT_SOURCE_CLOCK_MONITOR	Fail-Safe Clock Monitor interrupt
INT_SOURCE_RTCC	Real-Time Clock and Calender interrupt
INT_SOURCE_DMA_0	Direct Memory Access Channel 0 interrupt
INT_SOURCE_DMA_1	Direct Memory Access Channel 1 interrupt
INT_SOURCE_DMA_2	Direct Memory Access Channel 2 interrupt
INT_SOURCE_DMA_3	Direct Memory Access Channel 3 interrupt
INT_SOURCE_DMA_4	Direct Memory Access Channel 4 interrupt
INT_SOURCE_DMA_5	Direct Memory Access Channel 5 interrupt
INT_SOURCE_DMA_6	Direct Memory Access Channel 6 interrupt
INT_SOURCE_DMA_7	Direct Memory Access Channel 7 interrupt
INT_SOURCE_FLASH_CONTROL	Flash Control Event interrupt
INT_SOURCE_USB_1	Universal Serial Bus 1 interrupt
INT_SOURCE_CAN_1	Controller Area Network 1 interrupt
INT_SOURCE_CAN_2	Controller Area Network 2 interrupt
INT_SOURCE_ETH_1	Ethernet interrupt
INT_SOURCE_ADC_FAULT	ADC Fault interrupt
INT_SOURCE_CRYPTO	Crypto interrupt
INT_SOURCE_SPI_5_RECEIVE	SPI 5 Receive Done interrupt
INT_SOURCE_SPI_5_TRANSMIT	SPI 5 Transmit Done interrupt
INT_SOURCE_SPI_6_ERROR	SPI 6 Error interrupt
INT_SOURCE_SPI_6_RECEIVE	SPI 6 Receive Done interrupt
INT_SOURCE_SPI_6_TRANSMIT	SPI 6 Transmit Done interrupt

Description

Interrupt Source

This enumeration lists the possible sources that can cause an interrupt to occur.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_SUBPRIORITY_LEVEL Enumeration

Lists the possible interrupt sub priority levels.

File

plib_int_help.h

C

```
typedef enum {
  INT_SUBPRIORITY_LEVEL0,
  INT_SUBPRIORITY_LEVEL1,
  INT_SUBPRIORITY_LEVEL2,
  INT_SUBPRIORITY_LEVEL3
} INT_SUBPRIORITY_LEVEL;
```

Members

Members	Description
INT_SUBPRIORITY_LEVEL0	Sub Priority 0
INT_SUBPRIORITY_LEVEL1	Sub Priority 1
INT_SUBPRIORITY_LEVEL2	Sub Priority 2
INT_SUBPRIORITY_LEVEL3	Sub Priority 3

Description

Sub Priority Level

This enumeration lists the possible interrupt sub priority levels that can be selected for each source.

Remarks

This enumeration (INT_SUBPRIORITY_LEVEL) is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_VECTOR Enumeration

Lists the possible interrupt vectors.

File

plib_int_help.h

C

```
typedef enum {
  INT_VECTOR_CT,
  INT_VECTOR_CS0,
  INT_VECTOR_CS1,
  INT_VECTOR_INT0,
  INT_VECTOR_T1,
  INT_VECTOR_IC1,
  INT_VECTOR_OC1,
  INT_VECTOR_INT1,
  INT_VECTOR_T2,
  INT_VECTOR_IC2,
  INT_VECTOR_OC2,
  INT_VECTOR_INT2,
  INT_VECTOR_T3,
  INT_VECTOR_IC3,
  INT_VECTOR_OC3,
  INT_VECTOR_INT3,
  INT_VECTOR_T4,
  INT_VECTOR_IC4,
  INT_VECTOR_OC4,
  INT_VECTOR_INT4,
  INT_VECTOR_T5,
  INT_VECTOR_IC5,
  INT_VECTOR_OC5,
  INT_VECTOR_SPI1,
  INT_VECTOR_UART1,
  INT_VECTOR_SPI3,
  INT_VECTOR_I2C3,
  INT_VECTOR_I2C1,
  INT_VECTOR_CN,
  INT_VECTOR_AD1,
  INT_VECTOR_PMP,
  INT_VECTOR_CMP1,
  INT_VECTOR_CMP2,
```

```
    INT_VECTOR_UART3,
    INT_VECTOR_SPI2,
    INT_VECTOR_I2C4,
    INT_VECTOR_UART2,
    INT_VECTOR_SPI4,
    INT_VECTOR_I2C5,
    INT_VECTOR_I2C2,
    INT_VECTOR_FSCM,
    INT_VECTOR_RTCC,
    INT_VECTOR_DMA0,
    INT_VECTOR_DMA1,
    INT_VECTOR_DMA2,
    INT_VECTOR_DMA3,
    INT_VECTOR_DMA4,
    INT_VECTOR_DMA5,
    INT_VECTOR_DMA6,
    INT_VECTOR_DMA7,
    INT_VECTOR_FCE,
    INT_VECTOR_USB,
    INT_VECTOR_CAN1,
    INT_VECTOR_CAN2,
    INT_VECTOR_ETH,
    INT_VECTOR_UART4,
    INT_VECTOR_UART6,
    INT_VECTOR_UART5,
    INT_VECTOR_ADC1,
    INT_VECTOR_ADC1_DC1,
    INT_VECTOR_ADC1_DC2,
    INT_VECTOR_ADC1_DC3,
    INT_VECTOR_ADC1_DC4,
    INT_VECTOR_ADC1_DC5,
    INT_VECTOR_ADC1_DC6,
    INT_VECTOR_ADC1_DF1,
    INT_VECTOR_ADC1_DF2,
    INT_VECTOR_ADC1_DF3,
    INT_VECTOR_ADC1_DF4,
    INT_VECTOR_ADC1_DF5,
    INT_VECTOR_ADC1_DF6,
    INT_VECTOR_ADC1_DATA0,
    INT_VECTOR_ADC1_DATA1,
    INT_VECTOR_ADC1_DATA2,
    INT_VECTOR_ADC1_DATA3,
    INT_VECTOR_ADC1_DATA4,
    INT_VECTOR_ADC1_DATA5,
    INT_VECTOR_ADC1_DATA6,
    INT_VECTOR_ADC1_DATA7,
    INT_VECTOR_ADC1_DATA8,
    INT_VECTOR_ADC1_DATA9,
    INT_VECTOR_ADC1_DATA10,
    INT_VECTOR_ADC1_DATA11,
    INT_VECTOR_ADC1_DATA12,
    INT_VECTOR_ADC1_DATA13,
    INT_VECTOR_ADC1_DATA14,
    INT_VECTOR_ADC1_DATA15,
    INT_VECTOR_ADC1_DATA16,
    INT_VECTOR_ADC1_DATA17,
    INT_VECTOR_ADC1_DATA18,
    INT_VECTOR_ADC1_DATA19,
    INT_VECTOR_ADC1_DATA20,
    INT_VECTOR_ADC1_DATA21,
    INT_VECTOR_ADC1_DATA22,
    INT_VECTOR_ADC1_DATA23,
    INT_VECTOR_ADC1_DATA24,
    INT_VECTOR_ADC1_DATA25,
    INT_VECTOR_ADC1_DATA26,
    INT_VECTOR_ADC1_DATA27,
    INT_VECTOR_ADC1_DATA28,
    INT_VECTOR_ADC1_DATA29,
    INT_VECTOR_ADC1_DATA30,
    INT_VECTOR_ADC1_DATA31,
    INT_VECTOR_ADC1_DATA32,
    INT_VECTOR_ADC1_DATA33,
    INT_VECTOR_ADC1_DATA34,
    INT_VECTOR_ADC1_DATA35,
    INT_VECTOR_ADC1_DATA36,
```

```
    INT_VECTOR_ADC1_DATA37,
    INT_VECTOR_ADC1_DATA38,
    INT_VECTOR_ADC1_DATA39,
    INT_VECTOR_ADC1_DATA40,
    INT_VECTOR_ADC1_DATA41,
    INT_VECTOR_ADC1_DATA42,
    INT_VECTOR_ADC1_DATA43,
    INT_VECTOR_ADC1_DATA44,
    INT_VECTOR_ADC_END_OF_SCAN,
    INT_VECTOR_ADC_ANALOG_CIRCUIT_READY,
    INT_VECTOR_ADC_UPDATE_READY,
    INT_VECTOR_ADC_GROUP,
    INT_VECTOR_ADC_0_EARLY,
    INT_VECTOR_ADC_1_EARLY,
    INT_VECTOR_ADC_2_EARLY,
    INT_VECTOR_ADC_3_EARLY,
    INT_VECTOR_ADC_4_EARLY,
    INT_VECTOR_ADC_7_EARLY,
    INT_VECTOR_ADC_0_WARM,
    INT_VECTOR_ADC_1_WARM,
    INT_VECTOR_ADC_2_WARM,
    INT_VECTOR_ADC_3_WARM,
    INT_VECTOR_ADC_4_WARM,
    INT_VECTOR_ADC_7_WARM,
    INT_VECTOR_ADC_FAULT,
    INT_VECTOR_CRYPTO,
    INT_VECTOR_I2C2_BUS,
    INT_VECTOR_I2C2_SLAVE,
    INT_VECTOR_I2C2_MASTER,
    INT_VECTOR_SPI5_FAULT,
    INT_VECTOR_SPI5_RX,
    INT_VECTOR_SPI5_TX,
    INT_VECTOR_SPI6_FAULT,
    INT_VECTOR_SPI6_RX,
    INT_VECTOR_SPI6_TX,
    INT_VECTOR_CHANGE_NOTICE_A,
    INT_VECTOR_CHANGE_NOTICE_B,
    INT_VECTOR_CHANGE_NOTICE_C,
    INT_VECTOR_CHANGE_NOTICE_D,
    INT_VECTOR_CHANGE_NOTICE_E,
    INT_VECTOR_CHANGE_NOTICE_F,
    INT_VECTOR_CHANGE_NOTICE_G,
    INT_VECTOR_CHANGE_NOTICE_H,
    INT_VECTOR_CHANGE_NOTICE_J,
    INT_VECTOR_CHANGE_NOTICE_K
} INT_VECTOR;
```

Members

Members	Description
INT_VECTOR_CT	Core Timer Interrupt
INT_VECTOR_CS0	Core Software Interrupt 0
INT_VECTOR_CS1	Core Software Interrupt 1
INT_VECTOR_INT0	External Interrupt 0
INT_VECTOR_T1	Timer 1 Interrupt
INT_VECTOR_IC1	Input Capture 1 Interrupt
INT_VECTOR_OC1	Output Compare 1 Interrupt
INT_VECTOR_INT1	External Interrupt 1
INT_VECTOR_T2	Timer 2 Interrupt
INT_VECTOR_IC2	Input Capture 2 Interrupt
INT_VECTOR_OC2	Output Compare 2 Interrupt
INT_VECTOR_INT2	External Interrupt 2
INT_VECTOR_T3	Timer 3 Interrupt
INT_VECTOR_IC3	Input Capture 3 Interrupt
INT_VECTOR_OC3	Output Compare 3 Interrupt
INT_VECTOR_INT3	External Interrupt 3
INT_VECTOR_T4	Timer 4 Interrupt
INT_VECTOR_IC4	Input Capture 4 Interrupt

INT_VECTOR_OC4	Output Compare 4 Interrupt
INT_VECTOR_INT4	External Interrupt 4
INT_VECTOR_T5	Timer 5 Interrupt
INT_VECTOR_IC5	Input Capture 5 Interrupt
INT_VECTOR_OC5	Output Compare 5 Interrupt
INT_VECTOR_SPI1	SPI 1 Interrupt
INT_VECTOR_UART1	UART 1 Interrupt
INT_VECTOR_SPI3	SPI 3 Interrupt
INT_VECTOR_I2C3	I2C 3 Interrupt
INT_VECTOR_I2C1	I2C 1 Interrupt
INT_VECTOR_CN	Change Notification Interrupt
INT_VECTOR_AD1	ADC Interrupt
INT_VECTOR_PMP	PMP Interrupt
INT_VECTOR_CMP1	Comparator 1 Interrupt
INT_VECTOR_CMP2	Comparator 2 Interrupt
INT_VECTOR_UART3	UART 3 Interrupt
INT_VECTOR_SPI2	SPI 2 Interrupt
INT_VECTOR_I2C4	I2C 4 Interrupt
INT_VECTOR_UART2	UART 2 Interrupt
INT_VECTOR_SPI4	SPI 4 Interrupt
INT_VECTOR_I2C5	I2C 5 Interrupt
INT_VECTOR_I2C2	I2C 2 Interrupt
INT_VECTOR_FSCM	Fail Safe Clock Monitor Interrupt
INT_VECTOR_RTCC	RTCC Interrupt
INT_VECTOR_DMA0	DMA 0 Interrupt
INT_VECTOR_DMA1	DMA 1 Interrupt
INT_VECTOR_DMA2	DMA 2 Interrupt
INT_VECTOR_DMA3	DMA 3 Interrupt
INT_VECTOR_DMA4	DMA 4 Interrupt
INT_VECTOR_DMA5	DMA 5 Interrupt
INT_VECTOR_DMA6	DMA 6 Interrupt
INT_VECTOR_DMA7	DMA 7 Interrupt
INT_VECTOR_FCE	Flash Control Event Interrupt
INT_VECTOR_USB	USB Interrupt
INT_VECTOR_CAN1	CAN 1 Interrupt
INT_VECTOR_CAN2	CAN 2 Interrupt
INT_VECTOR_ETH	Ethernet Interrupt
INT_VECTOR_UART4	UART 4 Interrupt
INT_VECTOR_UART6	UART 6 Interrupt
INT_VECTOR_UART5	UART 5 Interrupt
INT_VECTOR_ADC1	Analog-to-Digital Converter 1 interrupt
INT_VECTOR_ADC1_DC1	ADC Digital Comparator 1
INT_VECTOR_ADC1_DC2	ADC Digital Comparator 2
INT_VECTOR_ADC1_DC3	ADC Digital Comparator 3
INT_VECTOR_ADC1_DC4	ADC Digital Comparator 4
INT_VECTOR_ADC1_DC5	ADC Digital Comparator 5
INT_VECTOR_ADC1_DC6	ADC Digital Comparator 6
INT_VECTOR_ADC1_DF1	ADC Digital Filter 1
INT_VECTOR_ADC1_DF2	ADC Digital Filter 2
INT_VECTOR_ADC1_DF3	ADC Digital Filter 3
INT_VECTOR_ADC1_DF4	ADC Digital Filter 4
INT_VECTOR_ADC1_DF5	ADC Digital Filter 5
INT_VECTOR_ADC1_DF6	ADC Digital Filter 6
INT_VECTOR_ADC1_DATA0	ADC Data 0
INT_VECTOR_ADC1_DATA1	ADC Data 1

INT_VECTOR_ADC1_DATA2	ADC Data 2
INT_VECTOR_ADC1_DATA3	ADC Data 3
INT_VECTOR_ADC1_DATA4	ADC Data 4
INT_VECTOR_ADC1_DATA5	ADC Data 5
INT_VECTOR_ADC1_DATA6	ADC Data 6
INT_VECTOR_ADC1_DATA7	ADC Data 7
INT_VECTOR_ADC1_DATA8	ADC Data 8
INT_VECTOR_ADC1_DATA9	ADC Data 9
INT_VECTOR_ADC1_DATA10	ADC Data 10
INT_VECTOR_ADC1_DATA11	ADC Data 11
INT_VECTOR_ADC1_DATA12	ADC Data 12
INT_VECTOR_ADC1_DATA13	ADC Data 13
INT_VECTOR_ADC1_DATA14	ADC Data 14
INT_VECTOR_ADC1_DATA15	ADC Data 15
INT_VECTOR_ADC1_DATA16	ADC Data 16
INT_VECTOR_ADC1_DATA17	ADC Data 17
INT_VECTOR_ADC1_DATA18	ADC Data 18
INT_VECTOR_ADC1_DATA19	ADC Data 19
INT_VECTOR_ADC1_DATA20	ADC Data 20
INT_VECTOR_ADC1_DATA21	ADC Data 21
INT_VECTOR_ADC1_DATA22	ADC Data 22
INT_VECTOR_ADC1_DATA23	ADC Data 23
INT_VECTOR_ADC1_DATA24	ADC Data 24
INT_VECTOR_ADC1_DATA25	ADC Data 25
INT_VECTOR_ADC1_DATA26	ADC Data 26
INT_VECTOR_ADC1_DATA27	ADC Data 27
INT_VECTOR_ADC1_DATA28	ADC Data 28
INT_VECTOR_ADC1_DATA29	ADC Data 29
INT_VECTOR_ADC1_DATA30	ADC Data 30
INT_VECTOR_ADC1_DATA31	ADC Data 31
INT_VECTOR_ADC1_DATA32	ADC Data 32
INT_VECTOR_ADC1_DATA33	ADC Data 33
INT_VECTOR_ADC1_DATA34	ADC Data 34
INT_VECTOR_ADC1_DATA35	ADC Data 35
INT_VECTOR_ADC1_DATA36	ADC Data 36
INT_VECTOR_ADC1_DATA37	ADC Data 37
INT_VECTOR_ADC1_DATA38	ADC Data 38
INT_VECTOR_ADC1_DATA39	ADC Data 39
INT_VECTOR_ADC1_DATA40	ADC Data 40
INT_VECTOR_ADC1_DATA41	ADC Data 41
INT_VECTOR_ADC1_DATA42	ADC Data 42
INT_VECTOR_ADC1_DATA43	ADC Data 43
INT_VECTOR_ADC1_DATA44	ADC Data 44
INT_VECTOR_ADC_END_OF_SCAN	ADC End of Scan
INT_VECTOR_ADC_ANALOG_CIRCUIT_READY	ADC Analog Circuit Ready
INT_VECTOR_ADC_UPDATE_READY	ADC Update Ready
INT_VECTOR_ADC_GROUP	ADC Group
INT_VECTOR_ADC_0_EARLY	ADC0 Early Interrupt
INT_VECTOR_ADC_1_EARLY	ADC1 Early Interrupt
INT_VECTOR_ADC_2_EARLY	ADC2 Early Interrupt
INT_VECTOR_ADC_3_EARLY	ADC3 Early Interrupt
INT_VECTOR_ADC_4_EARLY	ADC4 Early Interrupt
INT_VECTOR_ADC_7_EARLY	ADC7 Early Interrupt
INT_VECTOR_ADC_0_WARM	ADC0 Warm Interrupt
INT_VECTOR_ADC_1_WARM	ADC1 Warm Interrupt

INT_VECTOR_ADC_2_WARM	ADC2 Warm Interrupt
INT_VECTOR_ADC_3_WARM	ADC3 Warm Interrupt
INT_VECTOR_ADC_4_WARM	ADC4 Warm Interrupt
INT_VECTOR_ADC_7_WARM	ADC7 Warm Interrupt
INT_VECTOR_ADC_FAULT	ADC Fault interrupt
INT_VECTOR_CRYPTO	Crypto interrupt
INT_VECTOR_I2C2_BUS	I2C 2 Bus Interrupt
INT_VECTOR_I2C2_SLAVE	I2C 2 Bus Interrupt
INT_VECTOR_I2C2_MASTER	I2C 2 Bus Interrupt
INT_VECTOR_SPI5_FAULT	SPI 5 Error interrupt
INT_VECTOR_SPI5_RX	SPI 5 Receive Done interrupt
INT_VECTOR_SPI5_TX	SPI 5 Transmit Done interrupt
INT_VECTOR_SPI6_FAULT	SPI 6 Error interrupt
INT_VECTOR_SPI6_RX	SPI 6 Receive Done interrupt
INT_VECTOR_SPI6_TX	SPI 6 Transmit Done interrupt
INT_VECTOR_CHANGE_NOTICE_A	Input Change Notice Channel A interrupt
INT_VECTOR_CHANGE_NOTICE_B	Input Change Notice Channel B interrupt
INT_VECTOR_CHANGE_NOTICE_C	Input Change Notice Channel C interrupt
INT_VECTOR_CHANGE_NOTICE_D	Input Change Notice Channel D interrupt
INT_VECTOR_CHANGE_NOTICE_E	Input Change Notice Channel E interrupt
INT_VECTOR_CHANGE_NOTICE_F	Input Change Notice Channel F interrupt
INT_VECTOR_CHANGE_NOTICE_G	Input Change Notice Channel G interrupt
INT_VECTOR_CHANGE_NOTICE_H	Input Change Notice Channel H interrupt
INT_VECTOR_CHANGE_NOTICE_J	Input Change Notice Channel J interrupt
INT_VECTOR_CHANGE_NOTICE_K	Input Change Notice Channel K interrupt

Description

Interrupt Vectors

This enumeration lists the possible interrupt vectors.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

INT_STATE_GLOBAL Type

Data type defining the global interrupt state.

File

plib_int.h

C

```
typedef uint32_t INT_STATE_GLOBAL;
```

Description

INT global state type definition

This data type is used for interrupt enable and disable functions.

Remarks

None.

INT_SHADOW_REGISTER Enumeration

Lists the possible shadow register sets.

File

plib_int_help.h

C

```
typedef enum {
  INT_SHADOW_REGISTER_0,
  INT_SHADOW_REGISTER_1,
  INT_SHADOW_REGISTER_2,
  INT_SHADOW_REGISTER_3,
  INT_SHADOW_REGISTER_4,
  INT_SHADOW_REGISTER_5,
  INT_SHADOW_REGISTER_6,
  INT_SHADOW_REGISTER_7
} INT_SHADOW_REGISTER;
```

Members

Members	Description
INT_SHADOW_REGISTER_0	Shadow register set 0
INT_SHADOW_REGISTER_1	Shadow register set 1
INT_SHADOW_REGISTER_2	Shadow register set 2
INT_SHADOW_REGISTER_3	Shadow register set 3
INT_SHADOW_REGISTER_4	Shadow register set 4
INT_SHADOW_REGISTER_5	Shadow register set 5
INT_SHADOW_REGISTER_6	Shadow register set 6
INT_SHADOW_REGISTER_7	Shadow register set 7

Description

Shadow Register Sets

This enumeration lists the possible shadow register sets.

Remarks

This feature may not be available on all the devices. Refer to the specific device header files for availability.

INT_VECTOR_SPACING Enumeration

Lists the possible interrupt vector spacing.

File

plib_int_help.h

C

```
typedef enum {
  INT_VECTOR_SPACING_0,
  INT_VECTOR_SPACING_8,
  INT_VECTOR_SPACING_16,
  INT_VECTOR_SPACING_32,
  INT_VECTOR_SPACING_64,
  INT_VECTOR_SPACING_128,
  INT_VECTOR_SPACING_256,
  INT_VECTOR_SPACING_512
} INT_VECTOR_SPACING;
```

Members

Members	Description
INT_VECTOR_SPACING_0	0 Byte Vector Spacing
INT_VECTOR_SPACING_8	8 Byte Vector Spacing
INT_VECTOR_SPACING_16	16 Byte Vector Spacing
INT_VECTOR_SPACING_32	32 Byte Vector Spacing
INT_VECTOR_SPACING_64	64 Byte Vector Spacing
INT_VECTOR_SPACING_128	128 Byte Vector Spacing
INT_VECTOR_SPACING_256	256 Byte Vector Spacing
INT_VECTOR_SPACING_512	512 Byte Vector Spacing

Description

Interrupt Vector Spacing

This enumeration lists the possible interrupt vector spacing, which can be provided between two interrupt vectors.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

Files

Files

Name	Description
plib_int.h	Defines the Interrupt Peripheral Library interface
plib_int_help.h	

Description

This section lists the source and header files used by the library.

plib_int.h

Defines the Interrupt Peripheral Library interface

Functions

	Name	Description
	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
	PLIB_INT_ExistsCPUCurrentPriorityLevel	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.
	PLIB_INT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Interrupt module.
	PLIB_INT_ExistsExternalINTEdgeSelect	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.
	PLIB_INT_ExistsINTCPUPriority	Identifies whether the INTCPUPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsINTCPUVector	Identifies whether the INTCPUVector feature exists on the Interrupt module.
	PLIB_INT_ExistsProximityTimerControl	Identifies whether the ProximityTimerControl feature exists on the Interrupts module.
	PLIB_INT_ExistsProximityTimerEnable	Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.
	PLIB_INT_ExistsShadowRegisterAssign	Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.
	PLIB_INT_ExistsSingleVectorShadowSet	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.
	PLIB_INT_ExistsSoftwareNMI	Identifies whether the SoftwareNMI feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceControl	Identifies whether the SourceControl feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceFlag	Identifies whether the SourceFlag feature exists on the Interrupt module.
	PLIB_INT_ExistsVariableOffset	Identifies whether the VariableOffset feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorPriority	Identifies whether the VectorPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt module.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_GetStateAndDisable	Disables the generation of interrupts to the CPU.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
	PLIB_INT_SetState	Restores the status of CPU interrupts based on the value passed into the function.
	PLIB_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	PLIB_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.

	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SoftwareNMITrigger	Triggers software NMI.
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_VariableVectorOffsetGet	Gets the offset specific to an interrupt source number.
	PLIB_INT_VariableVectorOffsetSet	Sets the offset specific to an interrupt source number.
	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.
	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

Types

	Name	Description
	INT_STATE_GLOBAL	Data type defining the global interrupt state.

Description

Interrupt Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Interrupt Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Interrupt Controller module.

File Name

plib_int.h

Company

Microchip Technology Inc.

plib_int_help.h

Enumerations

	Name	Description
	INT_EXTERNAL_SOURCES	Lists the possible external sources that can cause an interrupt to occur.
	INT_PRIORITY_LEVEL	Lists the possible interrupt priority levels.
	INT_SHADOW_REGISTER	Lists the possible shadow register sets.
	INT_SOURCE	Lists the possible sources that can cause an interrupt to occur.
	INT_SUBPRIORITY_LEVEL	Lists the possible interrupt sub priority levels.
	INT_VECTOR	Lists the possible interrupt vectors.
	INT_VECTOR_SPACING	Lists the possible interrupt vector spacing.

Section

Interrupt Controller Definitions - Processor-Specific Enumerations

****

****

This section defines Interrupt Controller enumerations that are
processor-specific.

NVM Peripheral Library

This section describes the Non-volatile Memory (NVM) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Non-Volatile Memory (NVM)component (Flash and EEPROM) on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

Flash Program Memory

The Flash program memory is readable, writeable and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete thus restricting memory access and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for Flash program memory modification.

- Run-Time Self Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

Note: This topic covers the RTSP techniques.

EEPROM Memory

The EEPROM memory is the data storage memory.

On device power-up, power is disconnected from the EEPROM and the EEPROM will be in its lowest power mode. When it is enabled, power is applied to the EEPROM, but it will be ready for the access only after a power-up delay. Once ready for access, all read and write operations are executed using the Data EEPROM registers.

Write/Read/Erase operations are possible in words.

A bulk erase may be performed to erase the full EEPROM.

Using the Library

This topic describes the basic architecture of the NVM Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_nvm.h`

The interface to the NVM library is defined in the `plib_nvm.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the NVM library must include `peripheral.h`.

Library File:

The NVM peripheral library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

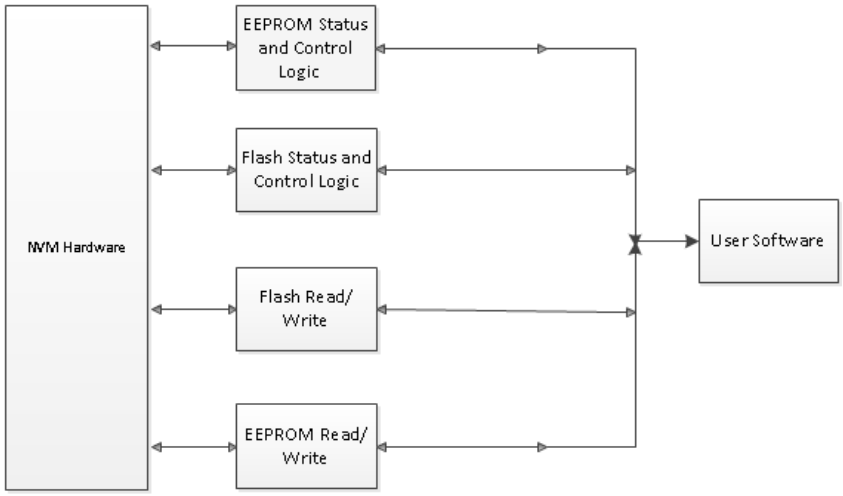
This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Abstraction Model

The NVM Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

NVM Software Abstraction Block Diagram



The **Flash Status and Control** logic ensures that the Flash memory is configured appropriately for modification. It also provides the status of the different operations in progress as well as errors, if any. It also decides if the operation is for Flash program memory or the special device Configuration registers.

The **Flash Read/Write** block ensures that the user data is written to/read from the program memory holding latches. It provides a layer of abstraction over the sequence of processor specific instructions which are required to access this data. Depending on the processor type, either block or Word programming options are available.

The **EEPROM Status and Control** logic ensures that the EEPROM is configured appropriately for access and also provides the status of the different operations in progress, as well as errors, if any.

The **EEPROM Read/Write** block ensures that the user data is written to/read from the data EEPROM. All of the EEPROM operations are word programmable with the exception of the bulk erase, which erases the entire EEPROM.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the NVM module

Library Interface Section	Description
Flash Memory Configuration and Status Functions	Provides Flash memory setup, configuration, and status control interface routines.
Flash Memory Functions	Provides Flash memory interface routines.
Other Flash Memory Functions	Provides additional functions for Boot page and Flash memory.
EEPROM Configuration and Status Functions	Provides EEPROM setup, configuration, and status control interface routines.
EEPROM Operation Functions	Provides interface routines for EEPROM operations.
Feature Existence Functions	Determine whether or not certain features are available.

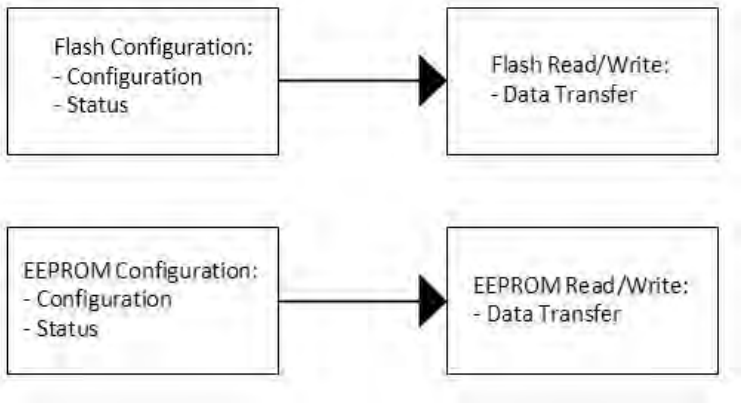
How the Library Works

The usage model for this library is explained in the following sections.

Description

Each of the blocks in the diagram correspond to the library interface section.

Usage Model



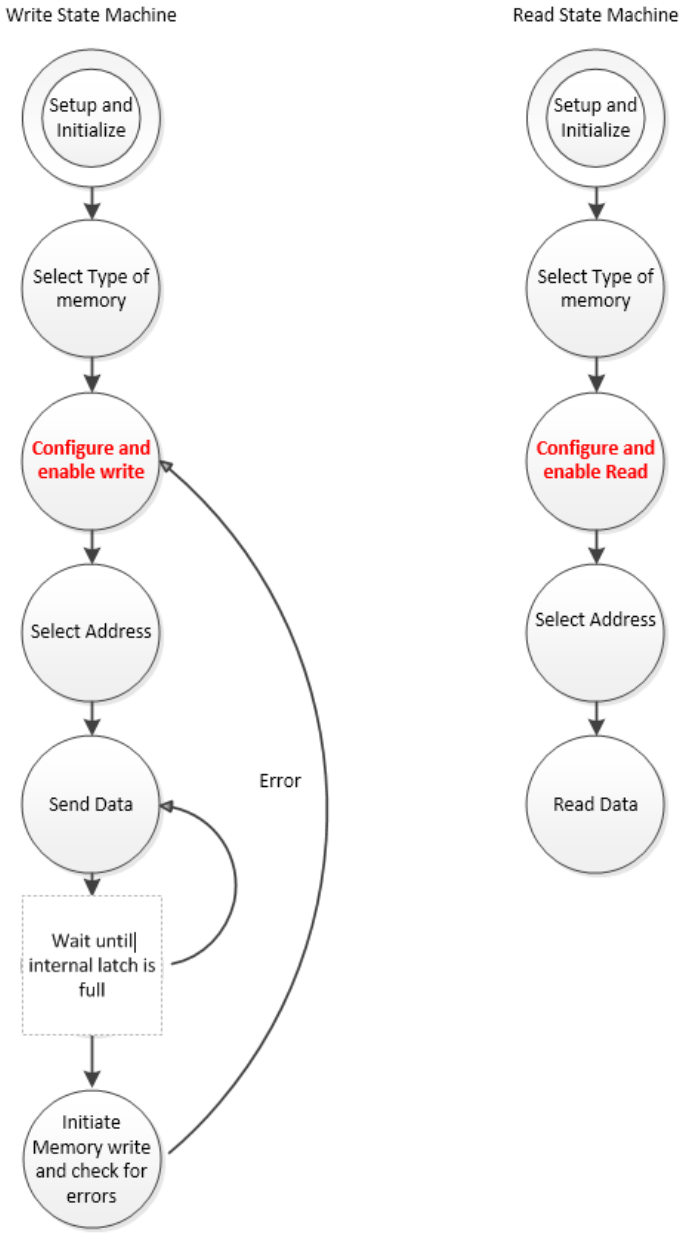
State Machine

This topic describes the NVM Write and Read state machine.

Description

The following state machine is for the NVM Read and Write during the normal operation. This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

NVM Write and Read State Machine



State	Associated Function
Setup and Initialization	Refer to mode of NVM for detailed instructions of setup.
Select Type of memory	Once the NVM has been appropriately set up and initialized, the state machine waits for the application to select the type of memory to be accessed, either Memory region (Flash/EEPROM) or the corresponding configuration registers.
Configure and enable write/read	Configures the registers and enables the write/read operation.
Select Address	The application provides the write/read address from which to access memory.
Send Data/Read Data	When the application is polling it can fill data/read data into the internal holding registers and initiate write via PLIB_NVM_FlashWriteStart(NVM_ID_0) or PLIB_NVM_EEPROMWriteStart(NVM_ID_0).
Read Error/Write Status	The error/write status is available to the application through the PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0) or PLIB_NVM_EEPROMOperationHasCompleted(NVM_ID_0) functions and other status/error APIs.

Note: Refer to the mode used for the NVM for the setup and initialization steps.

Flash Operations

This topic describes Flash operations.

Description

Note: Flash program memory operations vary across microcontrollers. Please refer to the **"Flash Controller"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

Flash Read Operation Setup

Do the following set up Flash read operation:

1. Provide the Flash program memory from which the data memory has to be read via the API PLIB_NVM_FlashRead function. This address is then updated in the respective address registers or the Table Pointers depending on the device.
2. Ensure the alignment of the address as specified by the specific device data sheet.
3. The size of the data read depends upon the microcontroller in use. For details, please refer to the specific device data sheet.

Example: Flash Program Memory Read

```
// Determine the address from which to read
#define MY_FLASH_BASE_ADDRESS      0xA0007862 // Or any address in Flash to read

uint32_t    address = MY_FLASH_BASE_ADDRESS;
// The Data Available at the memory location
size_t    DataToBeRead;

// where, MY_NVM_INSTANCE - is a specific instance of the hardware peripheral.
// where, address         - The address in the memory from which to read.
DataToBeRead = PLIB_NVM_FlashRead(NVM_ID_0, address);
```

EEPROM Operations

This topic describes EEPROM operations.

Description

Note: EEPROM operations vary across microcontrollers. Please refer to the **"Data EEPROM"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. Refer to **"EEPROM Configuration Write Operation Setup"** in Flash Operations for more information.

EEPROM Read Operation Setup

Prior to reading the Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

Do the following to execute a Data EEPROM read operation:

1. Load the Data EEPROM address to be read. The address must be on a 32-bit boundary and the last two bits must be '00'.
2. Select the read operation command.
3. Enable for read access.
4. Start the read operation.
5. Wait until the read cycle is complete.
6. Read the data from the intermediate register.

Example: EEPROM Read

```
if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
{
    // There should be no ongoing operation.
    if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
    {
        // Set address on 32-bit boundary, last two bits must be '00'
        uint32_t ADDR = 0x0100;
        // Load Data EEPROM read command
        PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_WORD_READ_OPERATION);
        // Load address and check if it is valid
        if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
        {
```

```
            // Enable access for read
            PLIB_NVM_EEPROMReadEnable( NVM_ID_0 );
            // Start the read operation
            PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
            // Wait until read is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
            // Read the data
            *ee_data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
        }
    }
}
```

EEPROM Write Operation Setup

Prior to writing to the Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

Do the following to execute a Data EEPROM write operation:

1. Load the Data EEPROM address where data has to be stored. The address must be on a 32-bit boundary so that the last two bits must be '00'.
2. Select the write operation command.
3. Enable for write access.
4. Write data into the intermediate register.
5. Execute the Data EEPROM unlock sequence.
6. Start the write operation.
7. Wait until the write cycle is complete.

Example: EEPROM Write

```
if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
{
    // There should be no ongoing operation.
    if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
    {
        // Set address on 32-bit boundary, last two bits must be '00'
        uint32_t ADDR = 0x0100;
        //Keys to unlock EEPROM
        uint32_t key1 = 0xEDB7;
        uint32_t key2 = 0x1248;
        // Load Data EEPROM write command
        PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_WORD_WRITE_OPERATION);
        // Load address and check if it is valid
        if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
        {
            // Enable access to write
            PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
            // Load data to intermediate register
            PLIB_NVM_EEPROMDataToWrite ( NVM_ID_0 , ee_data ) ;
            // Unlock the EEPROM
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
            // Start the write operation
            PLIB_NVM_EEPROMWriteStart( NVM_MODULE_ID index );
            // Wait until write is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
        }
    }
}
```

EEPROM Forced Word Erase Operation Setup

Under normal conditions, there is no need to attempt a Forced Word Erase. The Data EEPROM has internal logic which automatically manages all read, erase, and write command sequences.

If a verification error occurs during a write to the Data EEPROM, the user can attempt a Forced Word Erase operation to recover the Data EEPROM word storage location.

The Word Erase operation is similar to the write operation.

Example: EEPROM Word Erase

```
if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
{
   // There should be no ongoing operation.
    if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
```

```
    {
        // Set address on 32-bit boundary, last two bits must be '00'
        uint32_t ADDR = 0x0100;
        //Keys to unlock EEPROM
        uint32_t key1 = 0xEDB7;
        uint32_t key2 = 0x1248;
        // Load Data EEPROM forced word erase command
        PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_FORCED_WORD_ERASE_OPERATION);
        // Load address and check if it is valid
        if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
        {
            // Enable access to write
            PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
            // Unlock the EEPROM
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
            // Start the write operation
            PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
            // Wait until write is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
    }
}
```

EEPROM Bulk Erase Operation Setup

Prior to erasing the entire Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

The Bulk Erase operation is similar to the write operation except the EEPROM operation is selected for Bulk erase and the data or address does not need to be loaded.

Example: EEPROM Bulk Erase

```
    if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
    {
        // There should be no ongoing operation.
        if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
        {
            //Keys to unlock EEPROM
            uint32_t key1 = 0xEDB7;
            uint32_t key2 = 0x1248;
            // Load Data EEPROM forced word erase command
            PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_ERASE_ALL_OPERATION);
            // Enable access to write
            PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
            // Unlock the EEPROM
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
            // Start the write operation
            PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
            // Wait until write is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
        }
    }
```

EEPROM Configuration Write Operation Setup

Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. This is done through the Configuration Write operation. The configuration values to be written to the Data EEPROM Controller are stored in the DEVEE1 through DEVEE8 registers. Eight consecutive word write cycles should be performed with the EEPROM Configuration Write mode operation selected.

Miscellaneous Functions

This topic describes miscellaneous features.

Description

Note: Features vary across microcontrollers. Please refer to the **"Flash Controller"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

This section will detail out a few niche and processor-specific APIs provided by this library.

Stop in Idle Mode

The PLIB_NVM_EEPROMStopInIdleEnable and PLIB_NVM_EEPROMStopInIdleDisable functions cater to this functionality. These functions ensure that EEPROM operations are discontinued or continued in Idle mode.

Low Voltage Detection

The PLIB_NVM_LowVoltageIsDetected and PLIB_NVM_LowVoltageEventIsActive functions support low-voltage error detection and triggering of a Flash memory low-voltage event.

Long Write/Erase Cycle Detection

The PLIB_NVM_EEPROMNextWriteCycleIsLong function allows the application to know whether the next EEPROM Write/Erase cycle is long.

Configuring the Library

The library is configured for the supported NVM module when the processor is chosen in the MPLAB X IDE.

Building the Library

This section list the files that are available in the `src` folder of the NVM driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

Library Interface

a) Flash Memory Configuration and Status Functions

	Name	Description
	PLIB_NVM_LowVoltageEventIsActive	Provides low voltage detection status.
	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status.
	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash memory.
	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash memory.
	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash memory.

b) Flash Memory Operation Functions

	Name	Description
	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.
	PLIB_NVM_FlashAddressToModify	Modifies the Flash memory address.
	PLIB_NVM_FlashEraseStart	Performs erase operation on the selected Flash memory area.
	PLIB_NVM_FlashProvideData	Provides the data to be written into Flash memory.
	PLIB_NVM_FlashProvideQuadData	Provides the quad data to be written into Flash memory.
	PLIB_NVM_FlashRead	Read the specified address of Flash memory.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteStart	Performs a write operation on the Flash memory row selected.
	PLIB_NVM_FlashWriteCycleHasCompleted	Provides the status of the Flash write cycle.
	PLIB_NVM_WriteOperationHasTerminated	Provides the status of the Flash write operation or sequence.

c) Other Flash Memory Functions

	Name	Description
	PLIB_NVM_FlashSwapLockSelect	Selects the kind of Flash swap lock required.
	PLIB_NVM_FlashSwapLockStatusGet	Get the status of Swap lock bits.
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Sets the address below which physical memory will be write protected.
	PLIB_NVM_BootPageWriteProtectionDisable	Write protection for selected boot page is disabled.
	PLIB_NVM_BootPageWriteProtectionEnable	Locks the selected boot page.
	PLIB_NVM_IsBootMemoryLocked	Provides lock status of boot page write-protect bits.
	PLIB_NVM_IsBootPageWriteProtected	Provides write protection status for boot memory page.
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides lock status of Program Flash Write-Protect register.
	PLIB_NVM_LockBootMemory	Locks the boot write-protect bits.
	PLIB_NVM_LockProgramFlashMemory	Lock the Program Flash write-protect register.
	PLIB_NVM_ProgramFlashBank1LowerRegion	Maps Flash Bank 1 to the lower mapped region.
	PLIB_NVM_ProgramFlashBank2LowerRegion	Maps the bank 2 to lower mapped region.

	PLIB_NVM_ProgramFlashBank2IsLowerRegion	Gives the status of Program Flash Bank mapping.
	PLIB_NVM_BootFlashBank1LowerRegion	Maps Boot Flash Bank 1 to lower mapped region.
	PLIB_NVM_BootFlashBank2IsLowerRegion	Gives the status of Boot Flash Bank mapping.
	PLIB_NVM_BootFlashBank2LowerRegion	Maps Boot Flash Bank 2 to the lower mapped region.

d) EEPROM Configuration and Status Functions

	Name	Description
	PLIB_NVM_EEPROMEnable	Enables the EEPROM memory.
	PLIB_NVM_EEPROMDisable	Disables the EEPROM memory.
	PLIB_NVM_EEPROMIsReady	Provides the availability status of the EEPROM.
	PLIB_NVM_EEPROMStopInIdleEnable	Discontinues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleDisable	Continues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleIsEnabled	Returns Stop in Idle mode status of the EEPROM operation.
	PLIB_NVM_EEPROMOperationSelect	Selects the operation to be performed on the EEPROM.
	PLIB_NVM_EEPROMWriteEnable	Allows write or erase operation to the EEPROM.
	PLIB_NVM_EEPROMReadEnable	Allows read operation to the EEPROM.
	PLIB_NVM_EEPROMWriteIsEnabled	Returns EEPROM Write permission status.
	PLIB_NVM_EEPROMReadIsEnabled	Returns EEPROM read permission status.
	PLIB_NVM_EEPROMNextWriteCycleIsLong	Informs whether the next write or erase cycle of the EEPROM is long.
	PLIB_NVM_EEPROMErrorGet	Returns the EEPROM operation error.
	PLIB_NVM_EEPROMErrorClear	Clears the EEPROM operation error.

e) EEPROM Operation Functions

	Name	Description
	PLIB_NVM_EEPROMAddress	EEPROM address where operation has to be performed.
	PLIB_NVM_EEPROMDataToWrite	Accepts the data to be written into the EEPROM.
	PLIB_NVM_EEPROMKeySequenceWrite	Write mandatory KEY sequence to unlock the EEPROM write or erase protection.
	PLIB_NVM_EEPROMEraseStart	Initiates EEPROM erase cycle.
	PLIB_NVM_EEPROMWriteStart	Initiates a EEPROM write cycle.
	PLIB_NVM_EEPROMReadStart	Initiates a EEPROM read cycle.
	PLIB_NVM_EEPROMOperationHasCompleted	Provides the status of the EEPROM write or erase or read cycle.
	PLIB_NVM_EEPROMRead	Read the EEPROM data.
	PLIB_NVM_EEPROMOperationAbort	Aborts the current EEPROM operation.

f) Feature Existence Functions

	Name	Description
	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module.
	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.
	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module.
	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module.
	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module.
	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module.
	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module.
	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module.
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module.

	PLIB_NVM_ExistsBootFlashBankRegion	Identifies whether the BootFlashBankRegion feature exists on the NVM module.
	PLIB_NVM_ExistsSwapLockControl	Identifies whether the SwapLockControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMAddressControl	Identifies whether the EEPROMAddressControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMDataControl	Identifies whether the EEPROMDataControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableControl	Identifies whether the EEPROMEnableControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableOperationControl	Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMErrorStatus	Identifies whether the EEPROMErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMKeySequence	Identifies whether the EEPROMKeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMLongWriteStatus	Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationAbortControl	Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationModeControl	Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStartOperationControl	Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStopInIdleControl	Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.

g) Data Types and Constants

	Name	Description
	NVM_BOOT_MEMORY_AREA	Lists the different possible boot memory region.
	NVM_BOOT_MEMORY_PAGE	Lists the different NVM boot memory pages.
	NVM_OPERATION_MODE	Lists the different Flash operation modes.
	NVM_MODULE_ID	Possible instances of the NVM module.
	NVM_FLASH_SWAP_LOCK_TYPE	Lists the possible type of Flash swap lock.
	EEPROM_ERROR	Lists the different EEPROM operation errors.
	EEPROM_OPERATION_MODE	Lists the different EEPROM operation modes.

Description

This section describes the Application Programming Interface (API) functions of the NVM Peripheral Library.

Refer to each section for a detailed description.

a) Flash Memory Configuration and Status Functions

PLIB_NVM_LowVoltageEventIsActive Function

Provides low voltage detection status.

File

plib_nvm.h

C

```
bool PLIB_NVM_LowVoltageEventIsActive(NVM_MODULE_ID index);
```

Returns

- 1 - Low Voltage Event is active
- 0 - Low Voltage Event is not active

Description

This function provides detection of low voltage event, if any.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLowVoltageStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_LowVoltageEventIsActive(NVM_MODULE_ID index)

PLIB_NVM_LowVoltageIsDetected Function

Provides low voltage error detection status.

File

plib_nvm.h

C

```
bool PLIB_NVM_LowVoltageIsDetected(NVM_MODULE_ID index);
```

Returns

- 1 - Low Voltage detection and possible data corruption
- 0 - Voltage Level Acceptable for programming

Description

This function provides detection of low voltage error status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLowVoltageStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_LowVoltageIsDetected(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_LowVoltageIsDetected(NVM_MODULE_ID index)

PLIB_NVM_MemoryModifyEnable Function

Allows write cycles to Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryModifyEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function allows changing the write control (WR) bit and disables writing into the SWAP and NVMOP bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsMemoryModificationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryModifyEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryModifyEnable(NVM_MODULE_ID index)

PLIB_NVM_MemoryModifyInhibit Function

Inhibits write cycles to Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryModifyInhibit(NVM_MODULE_ID index);
```

Returns

None.

Description

This function disables the writing in the write control (WR) bit and enables writing into the SWAP and NVMOP bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsMemoryModificationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryModifyInhibit(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryModifyInhibit(NVM_MODULE_ID index)

PLIB_NVM_MemoryOperationSelect Function

Selects the operation to be performed on Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryOperationSelect(NVM_MODULE_ID index, NVM_OPERATION_MODE operationmode);
```

Returns

None.

Description

This function selects the operation to be performed on Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryOperationSelect(MY_NVM_INSTANCE, NVM_MEMORY_ROW_PROGRAM_OPERATION);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryOperationSelect(NVM_MODULE_ID index,
NVM_OPERATION_TYPE_SELECT operationmode)

b) Flash Memory Operation Functions

PLIB_NVM_DataBlockSourceAddress Function

Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_DataBlockSourceAddress(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory. This is to copy a row of data directly into Flash in one iteration without handling any intermediate holding registers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSourceAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    address = MY_RAM_BASE_ADDRESS;
PLIB_NVM_DataBlockSourceAddress(MY_NVM_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the user data memory from which data will be written

Function

void PLIB_NVM_DataBlockSourceAddress(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashAddressToModify Function

Modifies the Flash memory address.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashAddressToModify(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function takes the address parameter in the argument and loads the address that will be modified by the actual write operation. The write or erase operation following this will write or erase the user data into or from the Flash program memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsAddressModifyControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    address = MY_FLASH_BASE_ADDRESS;
PLIB_NVM_FlashAddressToModify(MY_NVM_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the memory from which data will be written. This address should be a physical address.

Function

void PLIB_NVM_FlashAddressToModify(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashEraseStart Function

Performs erase operation on the selected Flash memory area.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashEraseStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function Performs erase operation on the selected Flash memory region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

- The Address of the page to be erased must be provided using PLIB_NVM_FlashAddressToModify
- Erase Operation should be selected using PLIB_NVM_MemoryOperationSelect
- The module should be configured to access Flash memory using PLIB_NVM_MemoryModifyEnable
- The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence

Example

```
PLIB_NVM_FlashEraseStart(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_FlashEraseStart(NVM_MODULE_ID index)

PLIB_NVM_FlashProvideData Function

Provides the data to be written into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashProvideData(NVM_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function provides the user data to intermediate registers before being written into Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsProvideData in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DataToWrite;
PLIB_NVM_FlashProvideData(MY_NVM_INSTANCE, DataToWrite);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be written

Function

void PLIB_NVM_FlashProvideData(NVM_MODULE_ID index, uint32_t data)

PLIB_NVM_FlashProvideQuadData Function

Provides the quad data to be written into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashProvideQuadData(NVM_MODULE_ID index, uint32_t * data);
```

Returns

None.

Description

This function provides the user quad data to intermediate registers before being written into Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsProvideQuadData in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t  DataToWrite[4];
PLIB_NVM_FlashProvideQuadData(MY_NVM_INSTANCE, &DataToWrite);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Address pointing to the data to be written.

Function

void PLIB_NVM_FlashProvideQuadData(NVM_MODULE_ID index, uint32_t *data);

PLIB_NVM_FlashRead Function

Read the specified address of Flash memory.

File

plib_nvm.h

C

```
uint32_t PLIB_NVM_FlashRead(NVM_MODULE_ID index, uint32_t address);
```

Returns

Data value read at the memory address.

Description

This function takes the address parameter in the argument and loads the read address to the appropriate register. The read operation provides data from the given address.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    DataToBeRead;
```

```
DataToBeRead = PLIB_NVM_FlashRead(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The address in the memory from which to read

Function

uint32_t PLIB_NVM_FlashRead(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashWriteKeySequence Function

Copies the mandatory KEY sequence into the respective registers.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteKeySequence(NVM_MODULE_ID index, uint32_t keysequence);
```

Returns

None.

Description

This function copies the mandatory KEY sequence into the respective registers.

Remarks

Without the KEY sequence write or erase operation will not be executed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsKeySequence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_FlashWriteKeySequence(MY_NVM_INSTANCE, keysequence);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
keysequence	Mandatory KEY sequence depending on the controller type

Function

void PLIB_NVM_FlashWriteKeySequence(NVM_MODULE_ID index, uint32_t keysequence)

PLIB_NVM_FlashWriteStart Function

Performs a write operation on the Flash memory row selected.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function performs a write operation on the Flash memory row selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

- The Address of the page to be written must be provided using PLIB_NVM_FlashAddressToModify
- Erase Operation should be selected using PLIB_NVM_MemoryOperationSelect
- The module should be configured to access Flash memory using PLIB_NVM_MemoryModifyEnable
- The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence

Example

```
PLIB_NVM_FlashWriteStart(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_FlashWriteStart(NVM_MODULE_ID index)

PLIB_NVM_FlashWriteCycleHasCompleted Function

Provides the status of the Flash write cycle.

File

plib_nvm.h

C

```
bool PLIB_NVM_FlashWriteCycleHasCompleted(NVM_MODULE_ID index);
```

Returns

- 0 - Write or erase cycle is incomplete
- 1 - Write or erase cycle has completed

Description

This function provides the status of the Flash write cycle that was initiated by a write or erase operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_FlashWriteCycleHasCompleted(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_FlashWriteCycleHasCompleted(NVM_MODULE_ID index)

PLIB_NVM_WriteOperationHasTerminated Function

Provides the status of the Flash write operation or sequence.

File

plib_nvm.h

C

```
bool PLIB_NVM_WriteOperationHasTerminated(NVM_MODULE_ID index);
```

Returns

- 1 - Write operation prematurely terminated
- 0 - Write operation completed

Description

This function provides the status of the Flash write operation or sequence that was initiated by a write or erase operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_WriteOperationHasTerminated(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_WriteOperationHasTerminated(NVM_MODULE_ID index)

c) Other Flash Memory Functions

PLIB_NVM_FlashSwapLockSelect Function

Selects the kind of Flash swap lock required.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashSwapLockSelect(NVM_MODULE_ID index, NVM_FLASH_SWAP_LOCK_TYPE lockType);
```

Returns

None.

Description

This function allows user to select which swap bits will be writable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSwapLockControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_FlashSwapLockSelect(MY_NVM_INSTANCE, NVM_FLASH_SWAP_UNLOCKED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
lockType	One of the value from NVM_FLASH_SWAP_LOCK_TYPE enum to identify which swap bit will be locked

Function

void PLIB_NVM_FlashSwapLockSelect
(
NVM_MODULE_ID index,
NVM_FLASH_SWAP_LOCK_TYPE lockType
)

PLIB_NVM_FlashSwapLockStatusGet Function

Get the status of Swap lock bits.

File

plib_nvm.h

C

```
NVM_FLASH_SWAP_LOCK_TYPE PLIB_NVM_FlashSwapLockStatusGet(NVM_MODULE_ID index);
```

Returns

- NVM_FLASH_SWAP_LOCK_TYPE - The lock status of Flash swap bits.

Description

This function allows user to get the status of swap lock bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSwapLockControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
NVM_FLASH_SWAP_LOCK_TYPE lockStatus;
lockStatus = PLIB_NVM_FlashSwapLockStatusGet(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.

Function

NVM_FLASH_SWAP_LOCK_TYPE PLIB_NVM_FlashSwapLockStatusGet
(
NVM_MODULE_ID index
)

PLIB_NVM_FlashWriteProtectMemoryAreaRange Function

Sets the address below which physical memory will be write protected.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteProtectMemoryAreaRange(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function sets the address below which physical memory will be protected. Physical memory below address 0x1Dxxxxxx is write protected, where 'xxxxxx' is specified by "address" parameter. When "address" has a value of '0', write protection is disabled for the entire program Flash. If the specified address falls within the page, the entire page and all pages below the current page will be protected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashWPMemoryRangeProvide in your application to determine whether this feature is available.

Preconditions

The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_FlashWriteProtectMemoryAreaRange(MY_NVM_INSTANCE, WRITE_PROTECT_PAGE_ADDRESS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	Flash program write-protect (Page) address

Function

void PLIB_NVM_FlashWriteProtectMemoryAreaRange(NVM_MODULE_ID index, uint32_t address);

PLIB_NVM_BootPageWriteProtectionDisable Function

Write protection for selected boot page is disabled.

File

plib_nvm.h

C

```
void PLIB_NVM_BootPageWriteProtectionDisable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

None.

Description

This function disables Write protection for selected boot page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_BootPageWriteProtectionDisable(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page for which write protection has to be disabled

Function

void PLIB_NVM_BootPageWriteProtectionDisable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_BootPageWriteProtectionEnable Function

Locks the selected boot page.

File

plib_nvm.h

C

```
void PLIB_NVM_BootPageWriteProtectionEnable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

None.

Description

This function locks the selected boot page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_BootPageWriteProtectionEnable(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page which has to be locked

Function

void PLIB_NVM_BootPageWriteProtectionEnable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_IsBootMemoryLocked Function

Provides lock status of boot page write-protect bits.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsBootMemoryLocked(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);
```

Returns

- 1 - Selected boot alias write-protect bits are locked
- 0 - Selected boot alias write-protect bits are not locked

Description

This function provides lock status of boot page write-protect bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockBootSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_IsBootMemoryLocked(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_AREA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
memoryArea	Selects between the Lower or Upper Boot Alias area

Function

bool PLIB_NVM_IsBootMemoryLocked(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);

PLIB_NVM_IsBootPageWriteProtected Function

Provides write protection status for boot memory page.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsBootPageWriteProtected(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

- 1 - Selected boot region is write protected
- 0 - Selected boot region is not write protected

Description

This function provides write protection status for selected boot memory page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_IsBootPageWriteProtected(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page region

Function

bool PLIB_NVM_IsBootPageWriteProtected(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_IsProgramFlashMemoryLocked Function

Provides lock status of Program Flash Write-Protect register.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsProgramFlashMemoryLocked(NVM_MODULE_ID index);
```

Returns

- 1 - Program Flash write-protect register is locked
- 0 - Program Flash write-protect register is not locked

Description

This function provides lock status of Program Flash Write-Protect (NVMPWP) register.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockPFMSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
    status = PLIB_NVM_IsProgramFlashMemoryLocked(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_IsProgramFlashMemoryLocked(NVM_MODULE_ID index);

PLIB_NVM_LockBootMemory Function

Locks the boot write-protect bits.

File

plib_nvm.h

C

```
void PLIB_NVM_LockBootMemory(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);
```

Returns

None.

Description

This function locks the given (lower or upper alias) boot write-protect bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockBootSelect in your application to determine whether this feature is available.

Preconditions

The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_LockBootMemory(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_AREA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
memoryArea	Selects between Lower or Upper Boot Alias area

Function

void PLIB_NVM_LockBootMemory(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);

PLIB_NVM_LockProgramFlashMemory Function

Lock the Program Flash write-protect register.

File

plib_nvm.h

C

```
void PLIB_NVM_LockProgramFlashMemory(NVM_MODULE_ID index);
```

Returns

None.

Description

This function locks the Program Flash Write-Protect register (NVMPWP).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockPFMSelect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_LockProgramFlashMemory(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_LockProgramFlashMemory(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank1LowerRegion Function

Maps Flash Bank 1 to the lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_ProgramFlashBank1LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Program Flash Bank 1 to the lower mapped region and programs Flash Bank 2 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_ProgramFlashBank1LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_ProgramFlashBank1LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank2LowerRegion Function

Maps the bank 2 to lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Program Flash Bank 2 to the lower mapped region and Program Flash Bank 1 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_ProgramFlashBank2LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank2IsLowerRegion Function

Gives the status of Program Flash Bank mapping.

File

plib_nvm.h

C

```
bool PLIB_NVM_ProgramFlashBank2IsLowerRegion(NVM_MODULE_ID index);
```

Returns

- 1 - Program Flash Bank 2 is mapped to the lower mapped region and Program Flash Bank 1 is mapped to the upper mapped region
- 0 - Program Flash Bank 1 is mapped to the lower mapped region and Program Flash Bank 2 is mapped to the upper mapped region

Description

This function tells which Program Flash Bank is mapped to which region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(!PLIB_NVM_ProgramFlashBank2IsLowerRegion(MY_NVM_INSTANCE))
PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_ProgramFlashBank2IsLowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank1LowerRegion Function

Maps Boot Flash Bank 1 to lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_BootFlashBank1LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Boot Flash Bank 1 to the lower mapped region and Boot Flash Bank 2 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_BootFlashBank1LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_BootFlashBank1LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank2IsLowerRegion Function

Gives the status of Boot Flash Bank mapping.

File

plib_nvm.h

C

```
bool PLIB_NVM_BootFlashBank2IsLowerRegion(NVM_MODULE_ID index);
```

Returns

- 1 - Boot Flash Bank 2 is mapped to the lower mapped region and Boot Flash Bank 1 is mapped to the upper mapped region
- 0 - Boot Flash Bank 1 is mapped to the lower mapped region and Boot Flash Bank 2 is mapped to the upper mapped region

Description

This function tells which Boot Flash Bank is mapped to which region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(!PLIB_NVM_BootFlashBank2IsLowerRegion(MY_NVM_INSTANCE))
PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_BootFlashBank2IsLowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank2LowerRegion Function

Maps Boot Flash Bank 2 to the lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Boot Flash Bank 2 to the lower mapped region and Boot Flash Bank 1 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_BootFlashBank2LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);

d) EEPROM Configuration and Status Functions

PLIB_NVM_EEPROMEnable Function

Enables the EEPROM memory.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables the EEPROM memory for access.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMEnable(MY_NVM_INSTANCE);
if(PLIB_NVM_EEPROMIsReady(MY_NVM_INSTANCE))
    {
        //Perform operation
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMDisable Function

Disables the EEPROM memory.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMDisable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function disables EEPROM memory access.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMOperationHasCompleted)
    {
        PLIB_NVM_EEPROMDisable(MY_NVM_INSTANCE);
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMDisable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMIsReady Function

Provides the availability status of the EEPROM.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMIsReady(NVM_MODULE_ID index);
```

Returns

- true - EEPROM is ready to use
- false - EEPROM is not yet ready to use

Description

This function provides the ready status of the EEPROM which was initiated by PLIB_NVM_EEPROMEnable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMEnable(MY_NVM_INSTANCE);
if(PLIB_NVM_EEPROMIsReady(MY_NVM_INSTANCE))
    {
        //Perform operation
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMIsReady(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleEnable Function

Discontinues EEPROM operation when device enters Idle mode.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMStopInIdleEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function discontinues EEPROM operation when device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMStopInIdleEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMStopInIdleEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleDisable Function

Continues EEPROM operation when device enters Idle mode.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMStopInIdleDisable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function continues EEPROM operation when device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMStopInIdleDisable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMStopInIdleDisable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleIsEnabled Function

Returns Stop in Idle mode status of the EEPROM operation.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMStopInIdleIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM discontinues operation when the device enters Idle mode
- false - EEPROM continues operation when the device enters Idle mode

Description

This function returns whether the EEPROM continues or discontinues operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool sidl_status;
sidl_status = PLIB_NVM_EEPROMStopInIdleIsEnabled(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMStopInIdleIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationSelect Function

Selects the operation to be performed on the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMOperationSelect(NVM_MODULE_ID index, EEPROM_OPERATION_MODE mode);
```

Returns

None.

Description

This function selects the operation to be performed on the EEPROM from the set of operations available from the enum EEPROM_OPERATION_MODE.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMOperationModeControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMOperationSelect(MY_NVM_INSTANCE, EEPROM_WORD_READ_OPERATION);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMOperationSelect(NVM_MODULE_ID index , EEPROM_OPERATION_MODE mode)

PLIB_NVM_EEPROMWriteEnable Function

Allows write or erase operation to the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMWriteEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables write or erase operations on the EEPROM and inhibits read.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMWriteEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMWriteEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadEnable Function

Allows read operation to the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMReadEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables read operations on the EEPROM and inhibits write or erase.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMReadEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMReadEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMWriteIsEnabled Function

Returns EEPROM Write permission status.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMWriteIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM write or erase is enabled
- false - EEPROM write or erase is not enabled

Description

This function returns whether or not the EEPROM write or erase.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMWriteIsEnabled(MY_NVM_INSTANCE))
{
    //perform write or erase
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMWriteIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadIsEnabled Function

Returns EEPROM read permission status.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMReadIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM read is enabled.
- false - EEPROM read is not enabled.

Description

This function returns whether or not the EEPROM read is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMReadIsEnabled(MY_NVM_INSTANCE))
{
    //perform read
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMReadIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMNextWriteCycleIsLong Function

Informs whether the next write or erase cycle of the EEPROM is long.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMNextWriteCycleIsLong(NVM_MODULE_ID index);
```

Returns

false - Next write/Erase cycle will not take more time to complete. true - Next write/Erase cycle will take more time to complete.

Description

This function informs whether the next write or erase cycle of the EEPROM is long (i.e., the next write or erase to the EEPROM address (passed by PLIB_NVM_EEPROMAddress) will require more time (~20 ms) than usual).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMLongWriteStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_NVM_EEPROMNextWriteCycleIsLong( MY_NVM_INSTANCE ))
{
    //stays here for ~20ms, better skip to other task for this time.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMNextWriteCycleIsLong(NVM_MODULE_ID index)

PLIB_NVM_EEPROMErrorGet Function

Returns the EEPROM operation error.

File

plib_nvm.h

C

```
EEPROM_ERROR PLIB_NVM_EEPROMErrorGet(NVM_MODULE_ID index);
```

Returns

EEPROM_ERROR.

Description

This function returns the EEPROM operation error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
EEPROM_ERROR error;
error = PLIB_NVM_EEPROMErrorGet(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

EEPROM_ERROR PLIB_NVM_EEPROMErrorGet(NVM_MODULE_ID index)

PLIB_NVM_EEPROMErrorClear Function

Clears the EEPROM operation error.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMErrorClear(NVM_MODULE_ID index);
```

Returns

None.

Description

This function clears the EEPROM operation error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMErrorClear(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMErrorClear(NVM_MODULE_ID index)

e) EEPROM Operation Functions

PLIB_NVM_EEPROMAddress Function

EEPROM address where operation has to be performed.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMAddress(NVM_MODULE_ID index, uint32_t address);
```

Returns

- true - EEPROM address is valid
- false - EEPROM address is invalid, please check if it is in the EEPROM address boundary and last two bits are '00'

Description

This function accepts the EEPROM address upon which to operate. Lower two bits of the address must always be '00'; otherwise, an error will occur when operation is started on EEPROM.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMAddressControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
uint32_t  address = MY_EEPROM_ADDRESS;
// Load address and check if it is valid
if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , address ))
{
    //Perform operation
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The address of the EEPROM from or to which data will be read or written. This address should be a physical address.

Function

bool PLIB_NVM_EEPROMAddress(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_EEPROMDataToWrite Function

Accepts the data to be written into the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMDataToWrite(NVM_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function accepts data to be written into the EEPROM.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMDataControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMDataToWrite(MY_NVM_INSTANCE, DATA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be written

Function

void PLIB_NVM_EEPROMDataToWrite (NVM_MODULE_ID index , uint32_t data)

PLIB_NVM_EEPROMKeySequenceWrite Function

Write mandatory KEY sequence to unlock the EEPROM write or erase protection.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMKeySequenceWrite(NVM_MODULE_ID index, uint32_t keysequence);
```

Returns

None.

Description

Without the KEY sequence write or erase operation will not be executed. Writing the KEY 0xEDB7 followed by writing the another KEY 0x1248 will unlock the EEPROM control register for write or erase operations. Writing any other value will lock the EEPROM control register. The unlock sequence is not necessary for a read operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMKeySequence in your application to determine whether this feature is available.

Preconditions

Interrupts should be disabled when writing the unlock sequence to the EEKEY register to prevent a disruption of the unlock sequence.

Example

```
//EEPROM is locked.
uint32_t key1 = 0xEDB7;
uint32_t key2 = 0x1248;
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
//EEPROM is now unlocked.
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
keysequence	Mandatory KEY sequence depending on the controller type

Function

void PLIB_NVM_EEPROMKeySequenceWrite (NVM_MODULE_ID index , uint32_t keysequence)

PLIB_NVM_EEPROMEraseStart Function

Initiates EEPROM erase cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMEraseStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM erase cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

PLIB_NVM_EEPROMWriteEnable should be called.

Example

```
// Enable write or erase operation
PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
// unlock
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
// Start the erase operation
PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
// Wait until erase is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMEraseStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMWriteStart Function

Initiates a EEPROM write cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMWriteStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM write cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

EEPROM write should be enabled through PLIB_NVM_EEPROMWriteEnable.

Example

```
// Enable write operation
PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
// Provide data
PLIB_NVM_EEPROMDataToWrite ( NVM_ID_0 , ee_data ) ;
// unlock
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
// Start the write operation
PLIB_NVM_EEPROMWriteStart( NVM_MODULE_ID index );
// Wait until write is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMWriteStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadStart Function

Initiates a EEPROM read cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMReadStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM read cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

EEPROM read should be enabled through PLIB_NVM_EEPROMReadEnable.

Example

```
// Enable read operation
PLIB_NVM_EEPROMReadEnable( NVM_ID_0 );
// Start the read cycle
PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
// Wait until read is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
// Read the data
ee_data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMReadStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationHasCompleted Function

Provides the status of the EEPROM write or erase or read cycle.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMOperationHasCompleted(NVM_MODULE_ID index);
```

Returns

- true - Write or erase cycle has completed
- false - Write or erase cycle has not completed

Description

This function provides the status of the EEPROM write or erase or read operation status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Start the erase operation
PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
// Wait until erase is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMOperationHasCompleted(NVM_MODULE_ID index)

PLIB_NVM_EEPROMRead Function

Read the EEPROM data.

File

plib_nvm.h

C

```
uint32_t PLIB_NVM_EEPROMRead(NVM_MODULE_ID index);
```

Returns

32-bit EEPROM data.

Description

This function returns the EERPOM data that is been fetched by performing the EEPROM read operation sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMDataControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation, or may return junk data.

Example

```
uint32_t EEPROM_Data;
// Start the read operation
PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
// Wait until read is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
// Now read the data
EEPROM_Data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_NVM_EEPROMRead(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationAbort Function

Aborts the current EEPROM operation.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMOperationAbort(NVM_MODULE_ID index);
```

Returns

None.

Description

This function aborts the on-going write or erase operation as soon as possible.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMOperationAbortControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMOperationAbort(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMOperationAbort(NVM_MODULE_ID index)

f) Feature Existence Functions

PLIB_NVM_ExistsAddressModifyControl Function

Identifies whether the AddressModifyControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsAddressModifyControl(NVM_MODULE_ID index);
```

Returns

- true - The AddressModifyControl feature is supported on the device
- false - The AddressModifyControl feature is not supported on the device

Description

This function identifies whether the AddressModifyControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashAddressToModify

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsAddressModifyControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsBootPageWriteProtect Function

Identifies whether the BootPageWriteProtect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsBootPageWriteProtect(NVM_MODULE_ID index);
```

Returns

- true - The BootPageWriteProtect feature is supported on the device
- false - The BootPageWriteProtect feature is not supported on the device

Description

This function identifies whether the BootPageWriteProtect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_BootPageWriteProtectionEnable
- PLIB_NVM_BootPageWriteProtectionDisable
- PLIB_NVM_IsBootPageWriteProtected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsBootPageWriteProtect(NVM_MODULE_ID index)

PLIB_NVM_ExistsFlashBankRegionSelect Function

Identifies whether the FlashBankRegionSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsFlashBankRegionSelect(NVM_MODULE_ID index);
```

Returns

- true - The FlashBankRegionSelect feature is supported on the device
- false - The FlashBankRegionSelect feature is not supported on the device

Description

This function identifies whether the FlashBankRegionSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_ProgramFlashBank1LowerRegion
- PLIB_NVM_ProgramFlashBank2LowerRegion

- PLIB_NVM_ProgramFlashBank2IsLowerRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsFlashBankRegionSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsFlashWPMemoryRangeProvide Function

Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsFlashWPMemoryRangeProvide(NVM_MODULE_ID index);
```

Returns

- true - The FlashWPMemoryRangeProvide feature is supported on the device
- false - The FlashWPMemoryRangeProvide feature is not supported on the device

Description

This function identifies whether the FlashWPMemoryRangeProvide feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashWriteProtectMemoryAreaRange

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsFlashWPMemoryRangeProvide(NVM_MODULE_ID index)

PLIB_NVM_ExistsKeySequence Function

Identifies whether the KeySequence feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsKeySequence(NVM_MODULE_ID index);
```

Returns

- true - The KeySequence feature is supported on the device
- false - The KeySequence feature is not supported on the device

Description

This function identifies whether the KeySequence feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashWriteKeySequence

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsKeySequence(NVM_MODULE_ID index)

PLIB_NVM_ExistsLockBootSelect Function

Identifies whether the LockBootSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLockBootSelect(NVM_MODULE_ID index);
```

Returns

- true - The LockBootSelect feature is supported on the device
- false - The LockBootSelect feature is not supported on the device

Description

This function identifies whether the LockBootSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_LockBootMemory
- PLIB_NVM_IsBootMemoryLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLockBootSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsLockPFMSelect Function

Identifies whether the LockPFMSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLockPFMSelect(NVM_MODULE_ID index);
```

Returns

- true - The LockPFMSelect feature is supported on the device
- false - The LockPFMSelect feature is not supported on the device

Description

This function identifies whether the LockPFMSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_LockProgramFlashMemory
- PLIB_NVM_IsProgramFlashMemoryLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLockPFMSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsLowVoltageError Function

Identifies whether the LowVoltageError feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLowVoltageError(NVM_MODULE_ID index);
```

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_LowVoltageIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLowVoltageError(NVM_MODULE_ID index)

PLIB_NVM_ExistsLowVoltageStatus Function

Identifies whether the LowVoltageStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLowVoltageStatus(NVM_MODULE_ID index);
```

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_LowVoltageEventIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLowVoltageStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsMemoryModificationControl Function

Identifies whether the MemoryModificationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsMemoryModificationControl(NVM_MODULE_ID index);
```

Returns

- true - The MemoryModificationControl feature is supported on the device
- false - The MemoryModificationControl feature is not supported on the device

Description

This function identifies whether the MemoryModificationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_MemoryModifyEnable
- PLIB_NVM_MemoryModifyInhibit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsMemoryModificationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsOperationMode(NVM_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_MemoryOperationSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsOperationMode(NVM_MODULE_ID index)

PLIB_NVM_ExistsProvideData Function

Identifies whether the ProvideData feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsProvideData(NVM_MODULE_ID index);
```

Returns

- true - The ProvideData feature is supported on the device
- false - The ProvideData feature is not supported on the device

Description

This function identifies whether the ProvideData feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashProvideData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsProvideData(NVM_MODULE_ID index)

PLIB_NVM_ExistsProvideQuadData Function

Identifies whether the ProvideQuadData feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsProvideQuadData(NVM_MODULE_ID index);
```

Returns

- true - The ProvideQuadData feature is supported on the device
- false - The ProvideQuadData feature is not supported on the device

Description

This function identifies whether the ProvideQuadData feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashProvideQuadData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsProvideQuadData(NVM_MODULE_ID index)

PLIB_NVM_ExistsSourceAddress Function

Identifies whether the SourceAddress feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsSourceAddress(NVM_MODULE_ID index);
```

Returns

- true - The SourceAddress feature is supported on the device
- false - The SourceAddress feature is not supported on the device

Description

This function identifies whether the SourceAddress feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_DataBlockSourceAddress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsSourceAddress(NVM_MODULE_ID index)

PLIB_NVM_ExistsWriteErrorStatus Function

Identifies whether the WriteErrorStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsWriteErrorStatus(NVM_MODULE_ID index);
```

Returns

- true - The WriteErrorStatus feature is supported on the device
- false - The WriteErrorStatus feature is not supported on the device

Description

This function identifies whether the WriteErrorStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_WriteOperationHasTerminated

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsWriteErrorStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsWriteOperation Function

Identifies whether the WriteOperation feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsWriteOperation(NVM_MODULE_ID index);
```

Returns

- true - The WriteOperation feature is supported on the device
- false - The WriteOperation feature is not supported on the device

Description

This function identifies whether the WriteOperation feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_FlashRead
- PLIB_NVM_FlashWriteStart
- PLIB_NVM_FlashEraseStart
- PLIB_NVM_FlashWriteCycleHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsWriteOperation(NVM_MODULE_ID index)

PLIB_NVM_ExistsBootFlashBankRegion Function

Identifies whether the BootFlashBankRegion feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsBootFlashBankRegion(NVM_MODULE_ID index);
```

Returns

- true - The BootFlashBankRegion feature is supported on the device
- false - The BootFlashBankRegion feature is not supported on the device

Description

This function identifies whether the BootFlashBankRegion feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_BootFlashBank1LowerRegion
- PLIB_NVM_BootFlashBank2LowerRegion
- PLIB_NVM_BootFlashBank2IsLowerRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsBootFlashBankRegion(NVM_MODULE_ID index)

PLIB_NVM_ExistsSwapLockControl Function

Identifies whether the SwapLockControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsSwapLockControl(NVM_MODULE_ID index);
```

Returns

- true - The SwapLockControl feature is supported on the device
- false - The SwapLockControl feature is not supported on the device

Description

This function identifies whether the SwapLockControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_FlashSwapLockSelect
- PLIB_NVM_FlashSwapLockStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsSwapLockControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMAddressControl Function

Identifies whether the EEPROMAddressControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMAddressControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMAddressControl feature is supported on the device
- false - The EEPROMAddressControl feature is not supported on the device

Description

This function identifies whether the EEPROMAddressControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMAddress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMAddressControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMDataControl Function

Identifies whether the EEPROMDataControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMDataControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMDataControl feature is supported on the device
- false - The EEPROMDataControl feature is not supported on the device

Description

This function identifies whether the EEPROMDataControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMDataToWrite
- PLIB_NVM_EEPROMRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMDataControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMEnableControl Function

Identifies whether the EEPROMEnableControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMEnableControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMEnableControl feature is supported on the device
- false - The EEPROMEnableControl feature is not supported on the device

Description

This function identifies whether the EEPROMEnableControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMEnable
- PLIB_NVM_EEPROMDisable
- PLIB_NVM_EEPROMIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMEnableControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMEnableOperationControl Function

Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMEnableOperationControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMEnableOperationControl feature is supported on the device
- false - The EEPROMEnableOperationControl feature is not supported on the device

Description

This function identifies whether the EEPROMEnableOperationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMWriteEnable
- PLIB_NVM_EEPROMWriteIsEnabled
- PLIB_NVM_EEPROMReadEnable
- PLIB_NVM_EEPROMReadIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMEnableOperationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMErrorStatus Function

Identifies whether the EEPROMErrorStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMErrorStatus(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMErrorStatus feature is supported on the device
- false - The EEPROMErrorStatus feature is not supported on the device

Description

This function identifies whether the EEPROMErrorStatus feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMErrorGet
- PLIB_NVM_EEPROMErrorClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMErrorStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMKeySequence Function

Identifies whether the EEPROMKeySequence feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMKeySequence(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMKeySequence feature is supported on the device
- false - The EEPROMKeySequence feature is not supported on the device

Description

This function identifies whether the EEPROMKeySequence feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMKeySequenceWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMKeySequence(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMLongWriteStatus Function

Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMLongWriteStatus(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMLongWriteStatus feature is supported on the device
- false - The EEPROMLongWriteStatus feature is not supported on the device

Description

This function identifies whether the EEPROMLongWriteStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMNextWriteCycleIsLong

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMLongWriteStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMOperationAbortControl Function

Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMOperationAbortControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMOperationAbortControl feature is supported on the device
- false - The EEPROMOperationAbortControl feature is not supported on the device

Description

This function identifies whether the EEPROMOperationAbortControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMOperationAbort

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMOperationAbortControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMOperationModeControl Function

Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMOperationModeControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMOperationModeControl feature is supported on the device
- false - The EEPROMOperationModeControl feature is not supported on the device

Description

This function identifies whether the EEPROMOperationModeControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMOperationSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMOperationModeControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMStartOperationControl Function

Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMStartOperationControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMStartOperationControl feature is supported on the device
- false - The EEPROMStartOperationControl feature is not supported on the device

Description

This function identifies whether the EEPROMStartOperationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMReadStart
- PLIB_NVM_EEPROMWriteStart
- PLIB_NVM_EEPROMEraseStart
- PLIB_NVM_EEPROMOperationHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMStartOperationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMStopInIdleControl Function

Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMStopInIdleControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMStopInIdleControl feature is supported on the device
- false - The EEPROMStopInIdleControl feature is not supported on the device

Description

This function identifies whether the EEPROMStopInIdleControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMStopInIdleEnable
- PLIB_NVM_EEPROMStopInIdleDisable
- PLIB_NVM_EEPROMStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMStopInIdleControl(NVM_MODULE_ID index)

g) Data Types and Constants

NVM_BOOT_MEMORY_AREA Enumeration

Lists the different possible boot memory region.

File

plib_nvm_help.h

C

```
typedef enum {
  LOWER_BOOT_ALIAS_AREA,
  UPPER_BOOT_ALIAS_AREA,
  LOWER_BOOT_MEMORY_AREA_NOT_SUPPORTED
} NVM_BOOT_MEMORY_AREA;
```

Members

Members	Description
LOWER_BOOT_ALIAS_AREA	Lower boot alias region
UPPER_BOOT_ALIAS_AREA	Upper boot alias region
LOWER_BOOT_MEMORY_AREA_NOT_SUPPORTED	Lower boot alias region is not supported

Description

NVM Boot Memory Area

This enumeration lists the different possible boot memory region.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_BOOT_MEMORY_PAGE Enumeration

Lists the different NVM boot memory pages.

File

plib_nvm_help.h

C

```
typedef enum {
  LOWER_BOOT_ALIAS_PAGE4,
```

```
  LOWER_BOOT_ALIAS_PAGE3,
  LOWER_BOOT_ALIAS_PAGE2,
  LOWER_BOOT_ALIAS_PAGE1,
  LOWER_BOOT_ALIAS_PAGE0,
  UPPER_BOOT_ALIAS_PAGE4,
  UPPER_BOOT_ALIAS_PAGE3,
  UPPER_BOOT_ALIAS_PAGE2,
  UPPER_BOOT_ALIAS_PAGE1,
  UPPER_BOOT_ALIAS_PAGE0,
  LOWER_BOOT_MEMORY_PAGE_NOT_SUPPORTED
} NVM_BOOT_MEMORY_PAGE;
```

Members

Members	Description
LOWER_BOOT_ALIAS_PAGE4	Lower boot alias page 4
LOWER_BOOT_ALIAS_PAGE3	Lower boot alias page 3
LOWER_BOOT_ALIAS_PAGE2	Lower boot alias page 2
LOWER_BOOT_ALIAS_PAGE1	Lower boot alias page 1
LOWER_BOOT_ALIAS_PAGE0	Lower boot alias page 0
UPPER_BOOT_ALIAS_PAGE4	Upper boot alias page 4
UPPER_BOOT_ALIAS_PAGE3	Upper boot alias page 3
UPPER_BOOT_ALIAS_PAGE2	Upper boot alias page 2
UPPER_BOOT_ALIAS_PAGE1	Upper boot alias page 1
UPPER_BOOT_ALIAS_PAGE0	Upper boot alias page 0
LOWER_BOOT_MEMORY_PAGE_NOT_SUPPORTED	Lower boot memory page not supported

Description

NVM Boot Memory Page

This enumeration lists the different NVM boot memory page details.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_OPERATION_MODE Enumeration

Lists the different Flash operation modes.

File

plib_nvm_help.h

C

```
typedef enum {
  WORD_PROGRAM_OPERATION,
  ROW_PROGRAM_OPERATION,
  PAGE_ERASE_OPERATION,
  FLASH_ERASE_OPERATION,
  UPPER_FLASH_REGION_ERASE_OPERATION,
  LOWER_FLASH_REGION_ERASE_OPERATION,
  QUAD_WORD_PROGRAM_OPERATION,
  NO_OPERATION
} NVM_OPERATION_MODE;
```

Members

Members	Description
WORD_PROGRAM_OPERATION	Word Program Operation
ROW_PROGRAM_OPERATION	Row Program Operation
PAGE_ERASE_OPERATION	Page Erase Operation
FLASH_ERASE_OPERATION	Flash Erase Operation
UPPER_FLASH_REGION_ERASE_OPERATION	Upper Flash Region Erase Operation
LOWER_FLASH_REGION_ERASE_OPERATION	Lower Flash Region Erase Operation
QUAD_WORD_PROGRAM_OPERATION	Quad Word Program Operation
NO_OPERATION	No Operation

Description

NVM Operation Mode

This enumeration lists the different Flash operation modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_MODULE_ID Enumeration

Possible instances of the NVM module.

File

plib_nvm_help.h

C

```
typedef enum {
  NVM_ID_0,
  NVM_NUMBER_OF_MODULES
} NVM_MODULE_ID;
```

Members

Members	Description
NVM_ID_0	NVM Module 0 ID
NVM_NUMBER_OF_MODULES	Number of available NVM modules.

Description

NVM Module ID

This data type defines the possible instances of the NVM module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_FLASH_SWAP_LOCK_TYPE Enumeration

Lists the possible type of Flash swap lock.

File

plib_nvm_help.h

C

```
typedef enum {
  NVM_FLASH_SWAP_UNLOCKED = 0x0,
  NVM_FLASH_SWAP_LOCKED = 0x1,
  NVM_FLASH_SWAP_LOCKED_UNTIL_RESET = 0x3
} NVM_FLASH_SWAP_LOCK_TYPE;
```

Members

Members	Description
NVM_FLASH_SWAP_UNLOCKED = 0x0	Program Flash Bank and Boot Flash Bank region swapping is Not locked
NVM_FLASH_SWAP_LOCKED = 0x1	Program Flash Bank and Boot Flash Bank region swapping is locked
NVM_FLASH_SWAP_LOCKED_UNTIL_RESET = 0x3	Program Flash Bank and Boot Flash Bank region swapping is locked until device is reset

Description

NVM Flash Swap Lock

This enumeration lists the possible type of Flash swap lock.

Remarks

This feature may not be available on all the devices. Refer to the specific device header files for availability.

EEPROM_ERROR Enumeration

Lists the different EEPROM operation errors.

File

plib_nvm_help.h

C

```
typedef enum {
  NO_ERROR,
  VERIFY_ERROR,
  INVALID_OPERATION,
  BOR_ERROR
} EEPROM_ERROR;
```

Members

Members	Description
NO_ERROR	No error has occurred
VERIFY_ERROR	Error occurred during verification
INVALID_OPERATION	Operation selected and executed are not the same
BOR_ERROR	BOR has suspended EEPROM operation

Description

EEPROM operation error

This enumeration lists the different EEPROM operation errors.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

EEPROM_OPERATION_MODE Enumeration

Lists the different EEPROM operation modes.

File

plib_nvm_help.h

C

```
typedef enum {
  EEPROM_WORD_READ_OPERATION,
  EEPROM_WORD_WRITE_OPERATION,
  EEPROM_FORCED_WORD_ERASE_OPERATION,
  EEPROM_ERASE_ALL_OPERATION,
  EEPROM_CONFIG_WRITE_OPERATION
} EEPROM_OPERATION_MODE;
```

Members

Members	Description
EEPROM_WORD_READ_OPERATION	Word read operation
EEPROM_WORD_WRITE_OPERATION	Word write operation
EEPROM_FORCED_WORD_ERASE_OPERATION	Page erase operation. Under normal conditions, there is no need to attempt a Forced Word Erase. The Data EEPROM has internal logic which automatically manages all read/erase/write command sequences. Software execution of the Forced Word Erase operation should be used in response to write verification error, VERIFY_ERROR
EEPROM_ERASE_ALL_OPERATION	Bulk erase operation. Unlike the read command, the bulk erase command cannot be aborted by software. During an erase cycle, any software attempt to write to the EECON register will be ignored
EEPROM_CONFIG_WRITE_OPERATION	Write to configuration registers operation. Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. This is done through the Configuration Write operation

Description

EEPROM Operation Mode

This enumeration lists the different EEPROM operation modes.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

Files

Files

Name	Description
plib_nvm.h	NVM PLIB Interface Header for NVM common definitions.
plib_nvm_help.h	Defines the NVM Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_nvm.h

NVM PLIB Interface Header for NVM common definitions.

Functions

	Name	Description
	PLIB_NVM_BootFlashBank1LowerRegion	Maps Boot Flash Bank 1 to lower mapped region.
	PLIB_NVM_BootFlashBank2IsLowerRegion	Gives the status of Boot Flash Bank mapping.
	PLIB_NVM_BootFlashBank2LowerRegion	Maps Boot Flash Bank 2 to the lower mapped region.
	PLIB_NVM_BootPageWriteProtectionDisable	Write protection for selected boot page is disabled.
	PLIB_NVM_BootPageWriteProtectionEnable	Locks the selected boot page.
	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.
	PLIB_NVM_EEPROMAddress	EEPROM address where operation has to be performed.
	PLIB_NVM_EEPROMDataToWrite	Accepts the data to be written into the EEPROM.
	PLIB_NVM_EEPROMDisable	Disables the EEPROM memory.
	PLIB_NVM_EEPROMEnable	Enables the EEPROM memory.
	PLIB_NVM_EEPROMEraseStart	Initiates EEPROM erase cycle.
	PLIB_NVM_EEPROMErrorClear	Clears the EEPROM operation error.
	PLIB_NVM_EEPROMErrorGet	Returns the EEPROM operation error.
	PLIB_NVM_EEPROMIsReady	Provides the availability status of the EEPROM.
	PLIB_NVM_EEPROMKeySequenceWrite	Write mandatory KEY sequence to unlock the EEPROM write or erase protection.
	PLIB_NVM_EEPROMNextWriteCycleIsLong	Informs whether the next write or erase cycle of the EEPROM is long.
	PLIB_NVM_EEPROMOperationAbort	Aborts the current EEPROM operation.
	PLIB_NVM_EEPROMOperationHasCompleted	Provides the status of the EEPROM write or erase or read cycle.
	PLIB_NVM_EEPROMOperationSelect	Selects the operation to be performed on the EEPROM.
	PLIB_NVM_EEPROMRead	Read the EEPROM data.
	PLIB_NVM_EEPROMReadEnable	Allows read operation to the EEPROM.
	PLIB_NVM_EEPROMReadIsEnabled	Returns EEPROM read permission status.
	PLIB_NVM_EEPROMReadStart	Initiates a EEPROM read cycle.
	PLIB_NVM_EEPROMStopInIdleDisable	Continues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleEnable	Discontinues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleIsEnabled	Returns Stop in Idle mode status of the EEPROM operation.
	PLIB_NVM_EEPROMWriteEnable	Allows write or erase operation to the EEPROM.
	PLIB_NVM_EEPROMWriteIsEnabled	Returns EEPROM Write permission status.
	PLIB_NVM_EEPROMWriteStart	Initiates a EEPROM write cycle.
	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module.
	PLIB_NVM_ExistsBootFlashBankRegion	Identifies whether the BootFlashBankRegion feature exists on the NVM module.

	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMAddressControl	Identifies whether the EEPROMAddressControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMDataControl	Identifies whether the EEPROMDataControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableControl	Identifies whether the EEPROMEnableControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableOperationControl	Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMErrorStatus	Identifies whether the EEPROMErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMKeySequence	Identifies whether the EEPROMKeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMLongWriteStatus	Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationAbortControl	Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationModeControl	Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStartOperationControl	Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStopInIdleControl	Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.
	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.
	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module.
	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module.
	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module.
	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module.
	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module.
	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module.
	PLIB_NVM_ExistsSwapLockControl	Identifies whether the SwapLockControl feature exists on the NVM module.
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module.
	PLIB_NVM_FlashAddressToModify	Modifies the Flash memory address.
	PLIB_NVM_FlashEraseStart	Performs erase operation on the selected Flash memory area.
	PLIB_NVM_FlashProvideData	Provides the data to be written into Flash memory.
	PLIB_NVM_FlashProvideQuadData	Provides the quad data to be written into Flash memory.
	PLIB_NVM_FlashRead	Read the specified address of Flash memory.
	PLIB_NVM_FlashSwapLockSelect	Selects the kind of Flash swap lock required.
	PLIB_NVM_FlashSwapLockStatusGet	Get the status of Swap lock bits.
	PLIB_NVM_FlashWriteCycleHasCompleted	Provides the status of the Flash write cycle.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Sets the address below which physical memory will be write protected.
	PLIB_NVM_FlashWriteStart	Performs a write operation on the Flash memory row selected.
	PLIB_NVM_IsBootMemoryLocked	Provides lock status of boot page write-protect bits.
	PLIB_NVM_IsBootPageWriteProtected	Provides write protection status for boot memory page.
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides lock status of Program Flash Write-Protect register.
	PLIB_NVM_LockBootMemory	Locks the boot write-protect bits.
	PLIB_NVM_LockProgramFlashMemory	Lock the Program Flash write-protect register.
	PLIB_NVM_LowVoltageEventIsActive	Provides low voltage detection status.

	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status.
	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash memory.
	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash memory.
	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash memory.
	PLIB_NVM_ProgramFlashBank1LowerRegion	Maps Flash Bank 1 to the lower mapped region.
	PLIB_NVM_ProgramFlashBank2IsLowerRegion	Gives the status of Program Flash Bank mapping.
	PLIB_NVM_ProgramFlashBank2LowerRegion	Maps the bank 2 to lower mapped region.
	PLIB_NVM_WriteOperationHasTerminated	Provides the status of the Flash write operation or sequence.

Description

Non-Volatile Memory (NVM) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the NVM PLIB for all families of Microchip microcontrollers..The definitions in this file are common to NVM peripheral.

File Name

plib_nvm.h

Company

Microchip Technology Inc.

plib_nvm_help.h

Defines the NVM Peripheral Library data types.

Enumerations

	Name	Description
	EEPROM_ERROR	Lists the different EEPROM operation errors.
	EEPROM_OPERATION_MODE	Lists the different EEPROM operation modes.
	NVM_BOOT_MEMORY_AREA	Lists the different possible boot memory region.
	NVM_BOOT_MEMORY_PAGE	Lists the different NVM boot memory pages.
	NVM_FLASH_SWAP_LOCK_TYPE	Lists the possible type of Flash swap lock.
	NVM_MODULE_ID	Possible instances of the NVM module.
	NVM_OPERATION_MODE	Lists the different Flash operation modes.

Description

NVM Peripheral Library Constants Header

This header file contains the definitions of the data types and constants that make up the interface to the NVM Peripheral Library for Microchip microcontrollers. The definitions in this file are for NVM module.

File Name

plib_nvm_help.h

Company

Microchip Technology Inc.

Output Compare Peripheral Library

This section describes the Output Compare Peripheral Library.

Introduction

This library provides a low-level abstraction of the Output Compare Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Output Compare module is primarily used to generate a single pulse or a series of pulses in response to selected time base events.

Output Compare module operation essentially requires a timer and one or two compare values. Users have an option to select either a 32-bit timer or a 16-bit timer. The timer can be programmed to count at a desired frequency and count up to a desired period value. Details regarding timer operation and setup can be found in the Timer Peripheral Library. The Output Compare module compares the value of the timer with the compare values depending on the selected operating mode. When a compare match occurs between a timer value and compare values, the Output Compare module outputs a change of state on its output pins in accordance with the chosen Output Compare compare mode logic. Either a single pulse or a sequence of pulses may be generated.

Some of the key operating modes of the OC module are:

- Single Compare Set High/Low modes: In these compare modes, a compare match between the timer and the buffer (primary compare value) sets the output High/Low. The output remains at the same state after compare match event unless the module is disabled or the mode is changed.
- Single Compare Toggle mode: Output toggles state at every compare match event between the timer and the buffer (primary compare value)
- Dual compare Single/Continuous Pulse modes: These modes require two compare values. Leading edge of output pulse is generated during compare match of the incrementing timer and the buffer (primary compare value). Trailing edge of output pulse results when a compare match occurs between the incrementing timer and the pulse width value (secondary compare value). The output may be a single pulse or a sequence of pulses.
- Pulse-Width Modulation (PWM) modes: In this mode, output goes high when a compare match occurs between timer and the pulse width value (duty cycle). The output is reset back to its initial state when the timer resets after attaining its maximum count.

The Output Compare module also provides programmable interrupt generation on a compare match event. In PWM mode, hardware-based Fault detection and automatic output disable features are provided.

Figure 1 shows a block diagram of the Output Compare module. A compare match between the timer value and the compare values generates a pulse at the output.

Figure 1: Output Compare Module Block Diagram

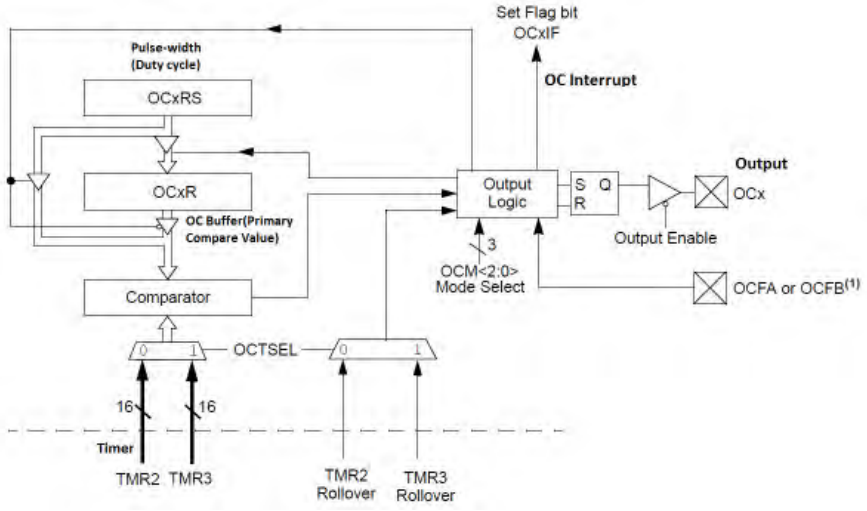
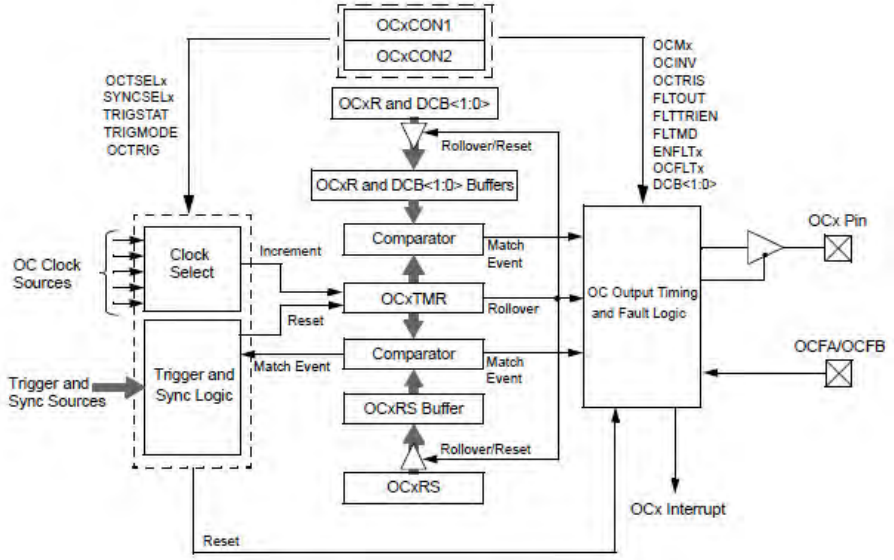


Figure 2 shows a block diagram of the Output Compare module with dedicated timers, present on some devices. It facilitates the use of multiple Output Compare modules operating synchronously or the use of an asynchronous trigger to generate a pulse.

Figure 2: Output Compare Module Block Diagram with Dedicated Timers



Using the Library

This topic describes the basic architecture of the Output Compare Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_oc.h`

The interface to the Output Compare Peripheral library is defined in the `plib_oc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Output Compare Peripheral Library must include `peripheral.h`.

Library File:

The Output Compare Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

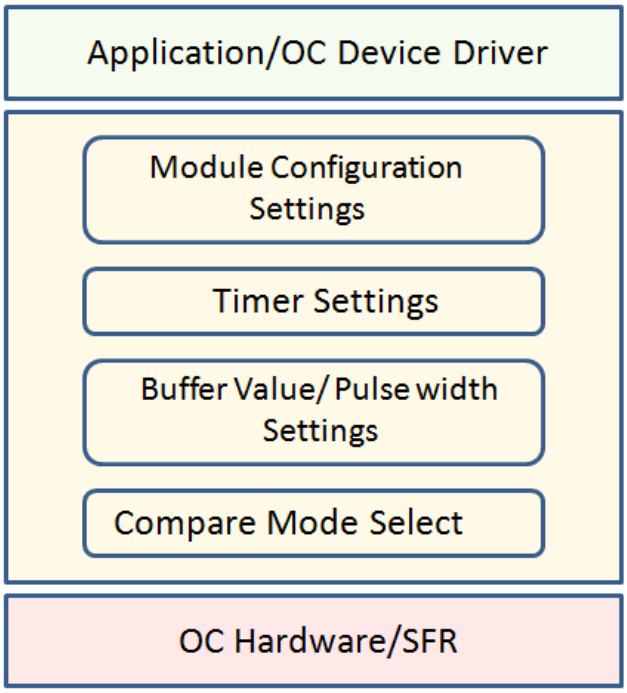
This library provides the low-level abstraction of the Output Compare module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available Output Compare modules on the device. Refer to the **"Output Compare"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Output Compare module on your device.

Description

The Output Compare Peripheral Library is used to simplify low-level access to the Output Compare module without having the need to directly interact the module's registers. The function names are generic and lead to easier access to the Output Compare module on different devices.

Output Compare module Software Abstraction Block Diagram



The Output Compare module can be described from a software standpoint as having functions to configure the module itself, to select the timer, set the buffer and pulse width values, and select the mode of operation. The desired nature of the output pulse is thus obtained.

A 16-bit or a 32-bit timer can be selected for the output compare time base. In some devices, the Output Compare module can be synchronized to an external input source. The Output Compare module can also operate asynchronously, based on a trigger by an input source. The Output Compare module entails the use of some functions dealing with Timer set up. These details can be found in Timer Peripheral Library.

Functions have been provided to set buffer values and pulse-width values. Pulse-width values are used only in Dual compare and PWM modes. In PWM mode, the buffer value provides the initial duty cycle for the first time period, while all later time cycles employ the 'pulse-width' value as the duty cycle value.

Compare modes:

- Single Compare Level Mode: Sets the output of Output Compare module at either 'High' or 'Low' at a compare match event
- Single Compare Toggle Mode: Toggles the output of OC module at each compare match event. This mode will produce continuous pulses.
- Dual Compare Mode: Output Compare module output is driven 'High' on compare match with buffer value and driven 'Low' on a compare match with the pulse-width value. The output can be either a single pulse or a continuous stream of pulses.
- PWM Mode: In Edge-aligned PWM mode, the Output Compare module output is driven 'High' whenever the timer resets, and is driven 'Low' on a compare match with the pulse-width value. Faults may or may not be enabled. In Center-aligned PWM mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Output Compare module.

Library Interface Section	Description
General Setup Functions	Provides setup, configuration and status control interface routines.
Compare Mode Functions	Provides Compare mode interface routines.
Timer Functions	Provides Timer interface routines.
Power-Saving Modes Functions	Provides Power-Saving modes interface routines.
Faults Functions	Provides Faults interface routines.
Feature Existence Functions	Provides interface routines that determine whether or not certain features are available.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General Setup: These routines deal with general setup of Output Compare module
- Compare/PWM Mode: These routines facilitate the selection of a Compare/PWM mode and setting up the compare values
- Timer: These functions deal with configuring the timer, selection of the clock source, and setting up a synchronous or asynchronous mode of operation for the OC module
- Power-Saving Modes:
  - Sleep Mode: The Output Compare module output is driven to the same state as it was in prior to the device entering Sleep mode. In PWM Fault mode, Fault detection is active. A Fault forces output of the OC module to tri-state or to a fixed predetermined state.
  - Idle Mode: The Output Compare module can be configured to suspend its operations or continue its operations. If the Output Compare module suspends its operations, it has the same behavior as it does in Sleep mode.
- Faults: These functions are used to select Fault inputs and identify occurrences of Faults when the Output Compare module functions in PWM mode
- Exists: These functions notify whether or not a particular feature is present on a device

The following sections provide examples that depict the use of interface functions to perform general tasks such as initialization and set up of the Output Compare module and setting up the Output Compare module in different compare modes.

Single Compare Set High Mode

The Single Compare Set High mode sets the output of the Output Compare module 'High' at a compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Set High mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Set High
mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the
API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);


/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_SET_HIGH_SINGLE_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/* Configure interrupts associated with Output Compare module. Refer to
Timer Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

Single Compare Toggle Mode

The Single Compare Toggle mode toggles the output of the Output Compare module at each compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Toggle mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Toggle mode */
```

```
/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for
the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_TOGGLE_CONTINOUS_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set Buffer Value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

Dual Compare Continuous Mode

In this mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value. A continuous stream of pulses is generated. This section illustrates the steps required to configure the Output Compare module in Dual Compare Continuous Pulse mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Dual Compare Continuous Pulse mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for
the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer(primary compare) value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width(secondary compare) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

PWM Mode with Enabled Faults

This section illustrates the steps required to configure the Output Compare module in PWM mode.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

Example:

This section illustrates the steps required to configure the Output Compare module in PWM mode with Fault protection. This mode can be selected using PLIB_OC_ModeSelect or PLIB_OC_FaultInputSelect.

```
/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode. PWM with fault protection mode is selected ,
   fault selection is preset in the hardware*/

PLIB_OC_ModeSelect(MY_OC_ID, OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION );
/*or use PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_PRESET) to achieve the same*/

/* Set buffer size to 16-bits. Refer to Timer Peripheral Library for the API */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer (initial duty cycle) Value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width (Duty cycle) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable OC module */
PLIB_OC_Enable(MY_OC_ID);

/* Check for PWM Fault */
while(!PLIB_OC_FaultHasOccurred(MY_OC_ID))
{
    /* If no PWM fault, continue normal operation*/
}
```

Configuring the Library

The library is configured for the supported Output Compare module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_OC_Disable	Disable the Output Compare module.
	PLIB_OC_Enable	Enables the Output Compare module.
	PLIB_OC_IsEnabled	Checks whether the Output Compare module is enabled or not.

b) Compare Mode Functions

	Name	Description
	PLIB_OC_ModeSelect	Selects the compare mode for the Output Compare module.
	PLIB_OC_Buffer16BitSet	Sets a 16-bit primary compare value for compare operations.
	PLIB_OC_Buffer32BitSet	Sets a 32-bit primary compare value for compare operations.
	PLIB_OC_BufferSizeSelect	Sets the buffer size and pulse width to 16-bits or 32-bits.
	PLIB_OC_PulseWidth16BitSet	Sets a 16-bit pulse width for Output Compare module output.
	PLIB_OC_PulseWidth32BitSet	Sets a 32-bit pulse width for Output Compare module output.

c) Timer Functions

	Name	Description
	PLIB_OC_TimerSelect	Selects a clock source for the Output Compare module.
	PLIB_OC_AlternateClockDisable	Selects Timer 2/3, instead of the alternate clock source.
	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_OC_AlternateTimerSelect	Selects an alternate timer as a clock source for the Output Compare module.

d) Power-Saving Modes Functions

	Name	Description
	PLIB_OC_StopInIdleDisable	Output Compare module continues operating when the device enters Idle mode.
	PLIB_OC_StopInIdleEnable	Discontinues Output Compare module operation when the device enters Idle mode.

e) Faults Functions

	Name	Description
	PLIB_OC_FaultHasOccurred	Checks if a PWM fault has occurred.
	PLIB_OC_FaultInputSelect	Enables/Disables the Fault input for the Output Compare PWM mode.

f) Feature Existence Functions

	Name	Description
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the Output Compare module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the Output Compare module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the Output Compare module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Output Compare module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.

g) Data Types and Constants

	Name	Description
	OC_16BIT_TIMERS	Lists different 16 bit time bases for Output Compare module.
	OC_BUFFER_SIZE	Lists different buffer sizes for compare value.
	OC_COMPARE_MODES	Lists the different compare modes for the Output Compare module.
	OC_FAULTS	Lists different fault sources for Output Compare module
	OC_MODULE_ID	OC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.
	OC_ALT_TIMERS	Lists the different 16-bit alternate timers for the Output Compare module.

Description

This section describes the Application Programming Interface (API) functions of the Output Compare Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_OC_Disable Function

Disable the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_Disable(OC_MODULE_ID index);
```

Returns

None.

Description

This function disables the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Disable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the Output Compare module

Function

void PLIB_OC_Disable(OC_MODULE_ID index)

PLIB_OC_Enable Function

Enables the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_Enable(OC_MODULE_ID index);
```

Returns

None.

Description

This function enables the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_OC_ID OC_ID_1

//Do all the other configurations before enabling.

PLIB_OC_Enable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_Enable(OC_MODULE_ID index)

PLIB_OC_IsEnabled Function

Checks whether the Output Compare module is enabled or not.

File

plib_oc.h

C

```
bool PLIB_OC_IsEnabled(OC_MODULE_ID index);
```

Returns

Boolean

- true - The Output Compare module is enabled
- false - The Output Compare module is not enabled

Description

The function returns the enable status of the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

if(PLIB_OC_IsEnabled(MY_OC_ID))
{
//Take respective actions
}
else
{
//Take respective actions
}
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

bool PLIB_OC_IsEnabled(OC_MODULE_ID index)

b) Compare Mode Functions

PLIB_OC_ModeSelect Function

Selects the compare mode for the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_ModeSelect(OC_MODULE_ID index, OC_COMPARE_MODES cmpMode);
```

Returns

None.

Description

This function selects the compare mode for the Output Compare module.

Remarks

If PLIB_OC_FaultInputSelect is called after PLIB_OC_ModeSelect, the mode selected by the PLIB_OC_ModeSelect function will be overwritten as the PLIB_OC_FaultInputSelect function selects the PWM mode with or without Fault protection.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsCompareModeSelect in your application to determine whether this feature is available.

Preconditions

The Output Compare module must be turned off before a new mode is selected. The Output Compare module is turned off through the PLIB_OC_ModeSelect(MY_OC_ID,OC_COMPARE_TURN_OFF_MODE) function. Refer to the enumeration description for information on different modes and preconditions.

Example

```
#define MY_OC_ID OC_ID_1
// Dual compare continuous pulse mode is selected
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
cmpMode	Identifies the compare mode for Output Compare module

Function

void PLIB_OC_ModeSelect(OC_MODULE_ID index, OC_COMPARE_MODES cmpMode)

PLIB_OC_Buffer16BitSet Function

Sets a 16-bit primary compare value for compare operations.

File

plib_oc.h

C

```
void PLIB_OC_Buffer16BitSet(OC_MODULE_ID index, uint16_t val16Bit);
```

Returns

None.

Description

This function sets a 16-bit primary compare value for compare operations in all modes except PWM modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 16-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
val16Bit	Sets a 16-bit primary compare value

Function

void PLIB_OC_Buffer16BitSet(OC_MODULE_ID index, uint16_t val16Bit)

PLIB_OC_Buffer32BitSet Function

Sets a 32-bit primary compare value for compare operations.

File

plib_oc.h

C

```
void PLIB_OC_Buffer32BitSet(OC_MODULE_ID index, uint32_t val32Bit);
```

Returns

None.

Description

This function sets a 32-bit primary compare value for compare operations in all modes except PWM modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 32-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer32BitSet(MY_OC_ID, 0x000000FF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
val32Bit	Sets a 32-bit primary compare value

Function

void PLIB_OC_Buffer32BitSet(OC_MODULE_ID index, uint32_t val32Bit)

PLIB_OC_BufferSizeSelect Function

Sets the buffer size and pulse width to 16-bits or 32-bits.

File

plib_oc.h

C

```
void PLIB_OC_BufferSizeSelect(OC_MODULE_ID index, OC_BUFFER_SIZE size);
```

Returns

None.

Description

This function sets the size of the buffer and pulse width to 16-bits or 32-bits. The choice is made based on whether a 16-bit timer or a 32-bit timer is selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferSize in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1
// Buffer size and pulse width size are set to 32-bits
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_32BIT);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
size	Identifies the size of compare value

Function

void PLIB_OC_BufferSizeSelect(OC_MODULE_ID index, OC_BUFFER_SIZE size)

PLIB_OC_PulseWidth16BitSet Function

Sets a 16-bit pulse width for Output Compare module output.

File

plib_oc.h

C

```
void PLIB_OC_PulseWidth16BitSet(OC_MODULE_ID index, uint16_t pulseWidth);
```

Returns

None.

Description

This function sets a 16-bit pulse width for the Output Compare module in dual compare modes. A dual compare mode can be selected using the PLIB_OC_ModeSelect function. Secondary compare match event (pulse width) decides the trailing (falling) edge of the Output Compare module output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsPulseWidth in your application to determine whether this feature is available.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 16-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
pulseWidth	Pulse width value

Function

void PLIB_OC_PulseWidth16BitSet(OC_MODULE_ID index,uint16_t pulseWidth)

PLIB_OC_PulseWidth32BitSet Function

Sets a 32-bit pulse width for Output Compare module output.

File

plib_oc.h

C

```
void PLIB_OC_PulseWidth32BitSet(OC_MODULE_ID index, uint32_t pulseWidth);
```

Returns

None.

Description

This function sets a 32-bit pulse width for Output Compare module in dual compare modes. A dual compare mode can be selected using PLIB_OC_ModeSelect function. Secondary compare match event (pulse width) decides the trailing (falling) edge of the Output Compare module output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsPulseWidth in your application to determine whether this feature is available.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 32-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PulseWidth32BitSet(MY_OC_ID, 0x00000FFF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
pulseWidth	Pulse width value

Function

void PLIB_OC_PulseWidth32BitSet(OC_MODULE_ID index,uint32_t pulseWidth)

c) Timer Functions

PLIB_OC_TimerSelect Function

Selects a clock source for the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_TimerSelect(OC_MODULE_ID index, OC_16BIT_TIMERS tmr);
```

Returns

None.

Description

This function selects a clock source for the Output Compare module if the 16-bit time base is set.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set.

Example

```
#define MY_OC_ID OC_ID_1
// 16-bit Timer2 is selected
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
tmr	Identifies the timer

Function

void PLIB_OC_TimerSelect(OC_MODULE_ID index, OC_16BIT_TIMERS tmr)

PLIB_OC_AlternateClockDisable Function

Selects Timer 2/3, instead of the alternate clock source.

File

plib_oc.h

C

```
void PLIB_OC_AlternateClockDisable(OC_MODULE_ID index);
```

Returns

None.

Description

This function selects Timer2/Timer3, instead of the alternate clock source.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the OC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockDisable( MY_OC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_AlternateClockDisable(OC_MODULE_ID index)

PLIB_OC_AlternateClockEnable Function

Selects the alternate clock source.

File

plib_oc.h

C

```
void PLIB_OC_AlternateClockEnable(OC_MODULE_ID index);
```

Returns

None.

Description

This function selects the alternate clock source instead of Timer2/Timer3.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the OC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockEnable( MY_OC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_AlternateClockEnable(OC_MODULE_ID index)

PLIB_OC_AlternateTimerSelect Function

Selects an alternate timer as a clock source for the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_AlternateTimerSelect(OC_MODULE_ID index, OC_ALT_TIMERS tmr);
```

Returns

- true - Alternate timer selected successfully
- false - Alternate timer selection failure, select an appropriate alternate timer for the Output Compare module index

Description

This function selects an alternate timer as a clock source for the Output Compare module.

- OC_ID_1,OC_ID_2,OC_ID_3: Can use Timer4 or Timer5 as alternate timers
- OC_ID_4,OC_ID_5,OC_ID_6: Can use Timer2 or Timer3 as alternate timers
- OC_ID_7,OC_ID_8,OC_ID_9: Can use Timer6 or Timer7 as alternate timers

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsAlternateTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set. The PLIB_OC_AlternateClockEnable function should be called for the Output Compare module to enable the alternate clock selection.

Example

```
#define MY_OC_ID   OC_ID_1
bool result;

//Enabling alternate timer selection
PLIB_OC_AlternateClockEnable( MY_OC_ID );

// 16-bit Timer4 is selected as the clock source for Output Compare module 1
result = PLIB_OC_AlternateTimerSelect(MY_OC_ID, OC_ALT_TIMER_TMR4);

if(false == result)
{
    // Selected alternate timer does not available for the desired Output
    // Compare module.
    // Select the appropriate alternate timer.
}
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
tmr	Identifies the alternate timer

Function

bool PLIB_OC_AlternateTimerSelect(OC_MODULE_ID index, OC_ALT_TIMERS tmr)

d) Power-Saving Modes Functions

PLIB_OC_StopInIdleDisable Function

Output Compare module continues operating when the device enters Idle mode.

File

plib_oc.h

C

```
void PLIB_OC_StopInIdleDisable(OC_MODULE_ID index);
```

Returns

None.

Description

The function continues Output Compare module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleDisable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_StopInIdleDisable(OC_MODULE_ID index)

PLIB_OC_StopInIdleEnable Function

Discontinues Output Compare module operation when the device enters Idle mode.

File

plib_oc.h

C

```
void PLIB_OC_StopInIdleEnable(OC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues Output Compare module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleEnable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_StopInIdleEnable(OC_MODULE_ID index)

e) Faults Functions

PLIB_OC_FaultHasOccurred Function

Checks if a PWM fault has occurred.

File

plib_oc.h

C

```
bool PLIB_OC_FaultHasOccurred(OC_MODULE_ID index);
```

Returns

- true - PWM Fault has occurred
- false - No PWM fault has occurred

Description

This function returns 'true' if a PWM Fault has occurred and 'false' if no Fault condition exists.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsFaultStatus in your application to determine whether this feature is available.

Preconditions

This function should be used only in Edge or Center-Aligned PWM mode set by the PLIB_OC_ModeSelect() function.

Example

```
#define MY_OC_ID OC_ID_1
if(!PLIB_OC_FaultHasOccurred(MY_OC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

bool PLIB_OC_FaultHasOccurred(OC_MODULE_ID index)

PLIB_OC_FaultInputSelect Function

Enables/Disables the Fault input for the Output Compare PWM mode.

File

plib_oc.h

C

```
void PLIB_OC_FaultInputSelect(OC_MODULE_ID index, OC_FAULTS flt);
```

Returns

None.

Description

This function enables/disables the Fault input for the Output Compare PWM mode.

If some other mode was selected using PLIB_OC_ModeSelect, the mode selected by PLIB_OC_ModeSelect will be overwritten as PLIB_OC_FaultInputSelect selects PWM mode with/without Fault protection.

Fault input is valid if the fault pin is enabled in the hardware. If a logic '0' is detected on the OCFA/OCFB pin, the selected PWM output pin(s) are placed in the tri-state. The user may elect to provide a pull-down or pull-up resistor on the PWM pin to provide for a desired state if a Fault condition occurs. The shutdown of the PWM output is immediate and is not tied to the device clock source. Fault occurrence can be detected by calling the function PLIB_OC_FaultHasOccurred. The Output Compare module will be disabled until the following conditions are met: • The external Fault condition has been removed • The PWM mode is re-enabled

Remarks

This function selects the PWM mode of the Output Compare module with Fault protection or without Fault protection.

These modes can be selected using PLIB_OC_ModeSelect also.

If any other Output Compare mode is selected prior to this function, that mode will be overwritten as this feature is available for PWM mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsFaultInput in your application to determine whether this feature is available.

Preconditions

Fault pin: OCFA for OC_ID_1 to OC_ID_4 , OCFB for OC_ID_5 in MX devices. OCFA for OC_ID_1 to OC_ID_3 and OC_ID_7 to OC_ID_9 , OCFB for OC_ID_4 to OC_ID_6 in MZ devices. should be enabled in the hardware if enabling the fault input, that is if selecting the OC_FAULT_PRESET.

Example

```
#define MY_OC_ID OC_ID_1
// Fault pin is enabled in the hardware
// This function selects PWM with fault protection mode for MY_OC_ID instance.
PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_PRESET);
// Output Compare fault input is now enabled for Output Compare Module
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
flt	Identifies the Output Compare module Fault input

Function

void PLIB_OC_FaultInputSelect(OC_MODULE_ID index, OC_FAULTS flt)

f) Feature Existence Functions

PLIB_OC_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsAlternateClock(OC_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_AlternateClockEnable
- PLIB_OC_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OC_ExistsAlternateClock(OC_MODULE_ID index)

PLIB_OC_ExistsBufferSize Function

Identifies whether the BufferSize feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsBufferSize(OC_MODULE_ID index);
```

Returns

- true - If the BufferSize feature is supported on the device
- false - If the BufferSize feature is not supported on the device

Description

This function identifies whether the BufferSize feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_BufferSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsBufferSize(OC_MODULE_ID index)

PLIB_OC_ExistsBufferValue Function

Identifies whether the BufferValue feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsBufferValue(OC_MODULE_ID index);
```

Returns

- true - If the BufferValue feature is supported on the device
- false - If the BufferValue feature is not supported on the device

Description

This function identifies whether the BufferValue feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_Buffer32BitSet
- PLIB_OC_Buffer16BitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsBufferValue(OC_MODULE_ID index)

PLIB_OC_ExistsCompareModeSelect Function

Identifies whether the CompareModeSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsCompareModeSelect(OC_MODULE_ID index);
```

Returns

- true - If the CompareModeSelect feature is supported on the device
- false - If the CompareModeSelect feature is not supported on the device

Description

This function identifies whether the CompareModeSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_ModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsCompareModeSelect(OC_MODULE_ID index)

PLIB_OC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsEnableControl(OC_MODULE_ID index);
```

Returns

- true - If the EnableControl feature is supported on the device
- false - If the EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_Enable
- PLIB_OC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsEnableControl(OC_MODULE_ID index)

PLIB_OC_ExistsFaultInput Function

Identifies whether the FaultInput feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsFaultInput(OC_MODULE_ID index);
```

Returns

- true - If the FaultInput feature is supported on the device
- false - If the FaultInput feature is not supported on the device

Description

This function identifies whether the FaultInput feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_FaultInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsFaultInput(OC_MODULE_ID index)

PLIB_OC_ExistsFaultStatus Function

Identifies whether the FaultStatus feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsFaultStatus(OC_MODULE_ID index);
```

Returns

- true - If the FaultStatus feature is supported on the device
- false - If the FaultStatus feature is not supported on the device

Description

This function identifies whether the FaultStatus feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_FaultHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsFaultStatus(OC_MODULE_ID index)

PLIB_OC_ExistsPulseWidth Function

Identifies whether the PulseWidth feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsPulseWidth(OC_MODULE_ID index);
```

Returns

- true - If the PulseWidth feature is supported on the device
- false - If the PulseWidth feature is not supported on the device

Description

This function identifies whether the PulseWidth feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_PulseWidth32BitSet
- PLIB_OC_PulseWidth16BitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsPulseWidth(OC_MODULE_ID index)

PLIB_OC_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsStopInIdle(OC_MODULE_ID index);
```

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_StopInIdleEnable
- PLIB_OC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsStopInIdle(OC_MODULE_ID index)

PLIB_OC_ExistsTimerSelect Function

Identifies whether the TimerSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsTimerSelect(OC_MODULE_ID index);
```

Returns

- true - If the TimerSelect feature is supported on the device
- false - If the TimerSelect feature is not supported on the device

Description

This function identifies whether the TimerSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_TimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsTimerSelect(OC_MODULE_ID index)

PLIB_OC_ExistsAlternateTimerSelect Function

Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsAlternateTimerSelect(OC_MODULE_ID index);
```

Returns

- true - If the AlternateTimerSelect feature is supported on the device
- false - If the AlternateTimerSelect feature is not supported on the device

Description

This function identifies whether the AlternateTimerSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_AlternateTimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsAlternateTimerSelect(OC_MODULE_ID index)

g) Data Types and Constants

OC_16BIT_TIMERS Enumeration

Lists different 16 bit time bases for Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_TIMER_16BIT_TMR2,
  OC_TIMER_16BIT_TMR3
} OC_16BIT_TIMERS;
```

Members

Members	Description
OC_TIMER_16BIT_TMR2	Clock source of Timer2 is the clock source of Output Compare module
OC_TIMER_16BIT_TMR3	Clock source of Timer3 is the clock source of Output Compare module

Description

Output Compare 16-bit Timer Select

This enumeration lists different 16 bit time bases for Output Compare module. The time base is set by the PLIB_OC_TimerSelect function.

OC_BUFFER_SIZE Enumeration

Lists different buffer sizes for compare value.

File

plib_oc_help.h

C

```
typedef enum {
  OC_BUFFER_SIZE_16BIT,
  OC_BUFFER_SIZE_32BIT
} OC_BUFFER_SIZE;
```

Members

Members	Description
OC_BUFFER_SIZE_16BIT	Buffer size is 16-bits. Duty cycle and compare values will have 16-bit values.
OC_BUFFER_SIZE_32BIT	Buffer size is 32-bits. Duty cycle and compare values will have 32-bit values.

Description

Output Compare Buffer Size

This enumeration lists different buffer sizes for compare value and duty cycle value.

OC_COMPARE_MODES Enumeration

Lists the different compare modes for the Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION,
  OC_COMPARE_PWM_MODE_WITHOUT_FAULT_PROTECTION,
  OC_COMPARE_PWM_EDGE_ALIGNED_MODE,
  OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE,
  OC_DUAL_COMPARE_SINGLE_PULSE_MODE,
```

```
  OC_TOGGLE_CONTINUOUS_PULSE_MODE,
  OC_SET_LOW_SINGLE_PULSE_MODE,
  OC_SET_HIGH_SINGLE_PULSE_MODE,
  OC_COMPARE_TURN_OFF_MODE
} OC_COMPARE_MODES;
```

Members

Members	Description
OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION	Output Compare module output is PWM signal and is fault protected if fault protection pin is enabled. Fault protection is valid if the fault pin is enabled in the hardware. Fault pin: OCFA for OC_ID_1 to OC_ID_4 , OCFB for OC_ID_5 in MX devices. OCFA for OC_ID_1 to OC_ID_3 and OC_ID_7 to OC_ID_9 , OCFB for OC_ID_4 to OC_ID_6 in PIC32MZ devices. If a logic â€˜0â€™ is detected on the OCFA/OCFB pin, the selected PWM output pin(s) are placed in the tri-state. The user may elect to provide a pull-down or pull-up resistor on the PWM pin to provide for a desired state if a Fault condition occurs. The shutdown of the PWM output is immediate and is not tied to the device clock source. Fault occurrence can be detected by calling the function PLIB_OC_FaultHasOccurred. The Output Compare will be disabled until the following conditions are met: 1. The external Fault condition has been removed 2. The PWM mode is re-enabled
OC_COMPARE_PWM_MODE_WITHOUT_FAULT_PROTECTION	Output Compare module output is PWM signal and is not fault protected
OC_COMPARE_PWM_EDGE_ALIGNED_MODE	This element is obsolete and it will be removed from next release
OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE	Dual Compare, Continuous Pulse mode: Output Compare module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. A continuous stream of pulses is generated unless the compare mode is changed or the module is disabled. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, Output Compare module output stays high.
OC_DUAL_COMPARE_SINGLE_PULSE_MODE	Dual Compare, Single Pulse mode: Output Compare module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, Output Compare module output stays high until a mode change is made or module is disabled
OC_TOGGLE_CONTINUOUS_PULSE_MODE	Single Compare Toggle mode: Output Compare module output is initialized to Low. Output Compare module output toggles at every compare match event with primary compare value with a single peripheral bus clock cycle delay. This scheme generates a square wave with 50% duty cycle. An interrupt is generated each time the output toggles.
OC_SET_LOW_SINGLE_PULSE_MODE	Single Compare Set Low mode: A compare match event with primary compare value will set the output of Output Compare module 'Low' with a single peripheral bus clock cycle delay. Output stays Low unless Output Compare module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. Output Compare module output is initially forced High.
OC_SET_HIGH_SINGLE_PULSE_MODE	Single Compare Set High mode: A compare match event with primary compare value will set the output of Output Compare module 'High' with a single peripheral bus clock cycle delay. Output stays High unless Output Compare module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. Output Compare module output is initially forced Low.
OC_COMPARE_TURN_OFF_MODE	Turn OFF mode: Output Compare module is disabled but still draws current. This mode is used to temporarily turn OFF the Output Compare module before a new compare mode is selected

Description

Output Compare Compare Modes

This enumeration lists different compare modes for Output Compare module. The compare mode is set by the PLIB_OC_ModeSelect function.

OC_FAULTS Enumeration

Lists different fault sources for Output Compare module

File

plib_oc_help.h

C

```
typedef enum {
  OC_FAULT_PRESET,
  OC_FAULT_DISABLE
} OC_FAULTS;
```

Members

Members	Description
OC_FAULT_PRESET	Enable Fault protection. PWM operation with fault protection.
OC_FAULT_DISABLE	Disable Fault protection. PWM operation without faults.

Description

Output Compare Fault Select

This enumeration lists different fault sources for Output Compare module. The fault source is selected by the PLIB_OC_FaultInputSelect function.

OC_MODULE_ID Enumeration

File

plib_oc_help.h

C

```
typedef enum {
  OC_ID_1,
  OC_ID_2,
  OC_ID_3,
  OC_ID_4,
  OC_ID_5,
  OC_ID_6,
  OC_ID_7,
  OC_ID_8,
  OC_ID_9,
  OC_NUMBER_OF_MODULES
} OC_MODULE_ID;
```

Members

Members	Description
OC_NUMBER_OF_MODULES	The total number of modules available.

Description

OC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.

OC_ALT_TIMERS Enumeration

Lists the different 16-bit alternate timers for the Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_ALT_TIMER_TMR2,
  OC_ALT_TIMER_TMR3,
  OC_ALT_TIMER_TMR4,
  OC_ALT_TIMER_TMR5,
  OC_ALT_TIMER_TMR6,
  OC_ALT_TIMER_TMR7
} OC_ALT_TIMERS;
```

Members

Members	Description
OC_ALT_TIMER_TMR2	Select Timer2 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR3	Select Timer3 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR4	Select Timer4 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR5	Select Timer5 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR6	Select Timer6 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR7	Select Timer7 as the alternate clock source for Output Compare module

Description

Output Compare 16-bit alternate Timer Select

This enumeration lists the different 16-bit timers for the Output Compare time base when the Output Compare module is configured with a 16-bit alternate timer resource.

Files

Files

Name	Description
plib_oc.h	This file contains the interface definition for the Output Compare Peripheral Library.
plib_oc_help.h	This is file plib_oc_help.h.

Description

This section lists the source and header files used by the library.

plib_oc.h

This file contains the interface definition for the Output Compare Peripheral Library.

Functions

	Name	Description
	PLIB_OC_AlternateClockDisable	Selects Timer 2/3, instead of the alternate clock source.
	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_OC_AlternateTimerSelect	Selects an alternate timer as a clock source for the Output Compare module.
	PLIB_OC_Buffer16BitSet	Sets a 16-bit primary compare value for compare operations.
	PLIB_OC_Buffer32BitSet	Sets a 32-bit primary compare value for compare operations.
	PLIB_OC_BufferSizeSelect	Sets the buffer size and pulse width to 16-bits or 32-bits.
	PLIB_OC_Disable	Disable the Output Compare module.
	PLIB_OC_Enable	Enables the Output Compare module.
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the Output Compare module.
	PLIB_OC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the Output Compare module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the Output Compare module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the Output Compare module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Output Compare module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the Output Compare module.
	PLIB_OC_FaultHasOccurred	Checks if a PWM fault has occurred.
	PLIB_OC_FaultInputSelect	Enables/Disables the Fault input for the Output Compare PWM mode.
	PLIB_OC_IsEnabled	Checks whether the Output Compare module is enabled or not.
	PLIB_OC_ModeSelect	Selects the compare mode for the Output Compare module.
	PLIB_OC_PulseWidth16BitSet	Sets a 16-bit pulse width for Output Compare module output.
	PLIB_OC_PulseWidth32BitSet	Sets a 32-bit pulse width for Output Compare module output.

	PLIB_OC_StopInIdleDisable	Output Compare module continues operating when the device enters Idle mode.
	PLIB_OC_StopInIdleEnable	Discontinues Output Compare module operation when the device enters Idle mode.
	PLIB_OC_TimerSelect	Selects a clock source for the Output Compare module.

Description

Output Compare Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Output Compare module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_oc.h

Company

Microchip Technology Inc.

plib_oc_help.h

Enumerations

	Name	Description
	OC_16BIT_TIMERS	Lists different 16 bit time bases for Output Compare module.
	OC_ALT_TIMERS	Lists the different 16-bit alternate timers for the Output Compare module.
	OC_BUFFER_SIZE	Lists different buffer sizes for compare value.
	OC_COMPARE_MODES	Lists the different compare modes for the Output Compare module.
	OC_FAULTS	Lists different fault sources for Output Compare module
	OC_MODULE_ID	OC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.

Description

This is file plib_oc_help.h.

Oscillator Peripheral Library

This section describes the Oscillator Peripheral Library.

Introduction

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Oscillator is the heart of the microcontroller and provides the clock on which the core and the peripherals run.

For all of the oscillators present on a Microchip microcontroller, two kinds of configurations exist:

- Through Configuration bits
- At execution time

The first is through 'Configuration bits', which is a one-time configuration done during the programming of the device. These one-time configurations are programmed in the code memory.

The other is 'Execution time' configuration, which deals with features that are allowed to be changed during run-time. This peripheral library provides functions which deal only with the 'execution time' configurable features of the Oscillator module.

Certain Oscillator module features can only be selected through 'Configuration bits'. However, most of the features can be altered during run-time by using the functions provided in this library.

Using the Library

This topic describes the basic architecture of the Oscillator Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_osc.h`

The interface to the Oscillator Peripheral Library is defined in the `plib_osc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Oscillator Peripheral Library must include `peripheral.h`.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

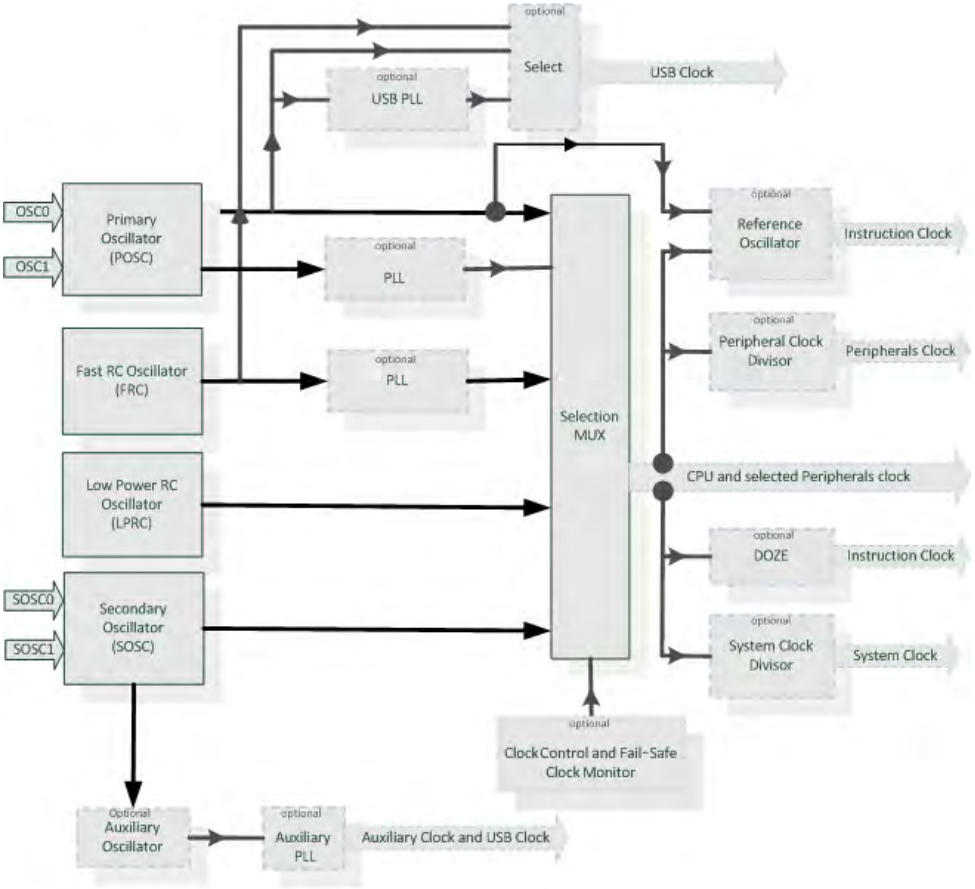
Hardware Abstraction Model

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

The Oscillator hardware model is shown in the following figure. The blocks represented in dashed lines may not be present in all microcontrollers. Refer to the **"Oscillator"** chapter in the specific device data sheet to determine availability.



To understand the Oscillator module and how each feature is mapped in this library, it is important to understand the following terminology.

Clock Source

A clock source is hardware that generates oscillations, which may be internal or external.

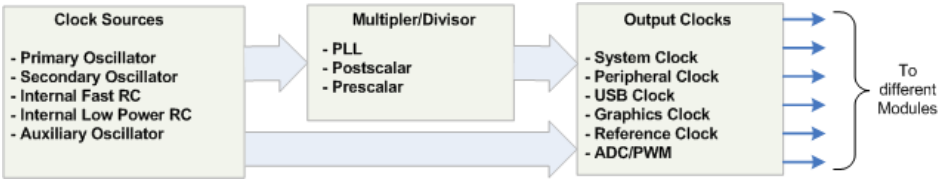
Divisor and Multiplier/PLL

These are hardware modules that can scale the clock. The rate at which the scaling is done may be fixed or configurable.

Clock Outputs

Clock outputs means output lines from the Oscillator module, which may route to different modules of the device or to the CPU (i.e., the system clock).

The following diagram provides a simplified explanation and the relationship between the previously mentioned terms. In most cases, there are multiple clock source options available for each of the clock outputs. However, not all the clock sources are available for all the output clocks. Scaling is an optional feature in most of the cases.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Output Compare module.

Library Interface Section	Description
General Setup Functions	Provides General Configurations • Operation on a `WAIT` instruction • System clock source selection
Primary Oscillator Setup Functions	Provides Configuration Routines for the Primary Oscillator: • Primary Oscillator Sleep Enable

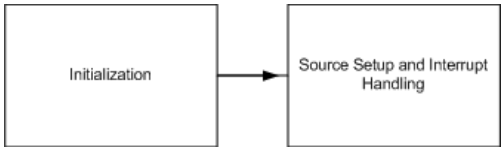
Secondary Oscillator Setup Functions	Provides Configuration Routines for the Secondary Oscillator: • Secondary Oscillator Enable/Disable • Secondary Oscillator Ready Indicator
Auxiliary Oscillator Setup Functions	Provides Configuration Routines for the Auxiliary Oscillator Configuration: • Reference Clock Source for Auxiliary Clock • Auxiliary Oscillator Mode Selection
Reference Oscillator Setup Functions	Provides Configuration Routines for the Reference Oscillator Configuration: • Reference Oscillator Output Enable/Disable • Select Reference Oscillator Output Divider • Reference Oscillator Clock Source Selection • Configure Reference Oscillator in Sleep Mode
Fast RC (FRC) Oscillator Setup Functions	Provides Configuration Routines for FRC Oscillator: • FRC Oscillator clock divider selection • FRC Oscillator tuning
Oscillator Switch Setup Functions	Provides Configuration Routines for Oscillator Switch and New Oscillator Selection: • Initiate an oscillator switch • Select the new oscillator • Get the current oscillator selection bits
Doze Setup Functions	Provides Configuration Routines for Doze mode Configuration • Enable/Disable Doze mode • Select DOZE (CPU Peripheral Clock Ratio) bits
USB and Display Clock Setup Functions	Provides Configuration Routines for USB and Display Clock Configuration: • Select FRC as USB Clock • Select Graphics Controller Clock • Display Module Clock Divider Selection
PLL Setup Functions	Provides Configuration Routines for PLL (Including USB and Auxiliary PLL) • PLL Multiplier • PLL Output Divider • 96 MHz PLL Enable for USB/Graphics Clock • Auxiliary PLL Input Divider • Auxiliary PLL Output Divider • Auxiliary PLL Multiplier • Auxiliary PLL Enable/Disable • PLL and USB PLL Lock Status
Peripheral Clock Setup Functions	Provides Configuration Routines for the Peripheral Clock • Set Peripheral Block Clock Divisor
Clock Fail Monitoring Functions	Provides Routines for Clock Fail Monitoring • Clock Fail Detect Status
Feature Existence Functions	Provides interface routines that determine whether certain features are available.

How the Library Works

The library can be used to control the Oscillator module. The usage model is explained in this section.

Description

The following diagram shows the major components of the usage model:



Oscillator Selection and Switching

Note: The device Configuration options change with the microcontroller family. Refer to the **"Oscillator"** chapter in the specific device data sheet for possible device configurations.

On devices that allow it, the oscillator selection can be overridden at run-time using the Oscillator Peripheral Library, as shown in the following code example:

```
//Do the configuration bit settings
OSC_SYS_TYPE currOsc;
OSC_SYS_TYPE newOsc=OSC_PRIMARY;

//get the current oscillator
currOsc =  PLIB_OSC_CurrentSysClockGet(OSC_ID_0);

//check if the current oscillator is same as new oscillator
if(currOsc != newOsc)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    //set the new oscillator request to Fast RC oscillator
    PLIB_OSC_SysClockSelect ( OSC_ID_0, newOsc );
}
```

Example: Oscillator Selection Change During Run-time

In the event the clock switch did not complete, the clock switch logic can be reset by calling the following API:

```
bool oscSwitch_st;

oscSwitch_st = PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0);

if(!oscSwitch_st)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
}
```

Example: Oscillator Switch Abort

This will abandon the clock switch process, stop and reset the oscillator start-up timer (if applicable), and stop the PLL (if applicable). The clock switch process can be aborted at any time. A clock switch that is already in progress can also be aborted by performing a second clock switch.

Clock Sources

This section explains the available clock sources and their setup.

The following clock sources are available in Microchip microcontrollers:

- Primary Oscillator (Posc)
- Secondary Oscillator (Sosc)
- Internal Fast RC (FRC) Oscillator
- Internal Low-Power RC (LPRC) Oscillator

Primary Oscillator (Posc)

The Primary Oscillator has several operating modes (refer to the **"Oscillator"** chapter in the specific device data sheet for exact operating modes). The selection of the operating mode is done using the device Configuration registers during device programming. During run-time this can be changed using the oscillator switch option. However, changing the operating mode from High Gain (HC) to External Clock (EC) or External Resonator(XT) is not possible.

If a PLL is available, the PLL input divider can only be programmed during device programming. However, the PLL output divider and the PLL multiplier can be changed during run-time using the PLL functions described in the PLL section.

Secondary Oscillator (Sosc)

The optional Secondary Oscillator is designed specifically for low-power operation with an external crystal. The Secondary Oscillator is enabled in hardware by the device Configuration bits. Once it is enabled in hardware, it can be switched on during software run-time.

```
bool oscSecondary_st;

oscSecondary_st = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
if(oscSecondary_st)
{
   // Unlock the Oscillator Registers
   PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

   PLIB_OSC_SecondaryDisable(OSC_ID_0);
}
```

Internal Fast RC (FRC) Oscillator

The FRC Oscillator is a fast user-trimmable internal RC oscillator with a user-selectable input divider, PLL multiplier, and output divider. See the **"Oscillator"** chapter in the specific device data sheet for more information on the FRC oscillator.

Once the FRC oscillator is selected using the device Configuration registers or an oscillator switch is initiated selecting the FRC as a new oscillator, the FRC Oscillator PLL divider can be changed during run-time.

```
OSC_FRC_DIV FRCDivisor;

//Get the current divider value for FRC
FRCDivisor = PLIB_OSC_FRCDivisorGet(OSC_ID_0);

if(FRCDivisor != OSC_FRC_DIV_4)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
}
```

Internal Low-Power RC (LPRC) Oscillator

The LPRC Oscillator is separate from the FRC Oscillator. It oscillates at a nominal frequency of 31.25 kHz (this value is device-dependent). The LPRC Oscillator can act as the clock source for the Power-up Timer (PWRT), Watchdog Timer (WDT), Fail-Safe Clock Monitor (FSCM), and PLL reference circuits. It can also be used to provide a low-frequency clock source option for the device in those applications where power consumption is critical and timing accuracy is not required.

The LPRC remains enabled after power on in the following conditions:

- Fail-safe clock monitoring is enabled
- Watchdog Timer is enabled
- LPRC is selected as the system clock

PLL Configuration

```
OSC_SYSPLL_OUT_DIV PLLOutDiv;
OSC_SYSPLL_MULTIPLIER pll_multiply;

// Unlock the Oscillator Registers
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
if(pll_multiply != 15)
{
    PLIB_OSC_SysPLLMultiplierSelect(OSC_ID_0, 15);
}
PLLOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);

if(PLLOutDiv != OSC_SYSPLL_OUT_DIV_8)
{
    PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_8);
}
```

Fail-Safe Clock Monitor

Fail-Safe Clock Monitor (FSCM)

The Fail-Safe Clock Monitor (FSCM) is designed to allow continued device operation if the current oscillator fails. It is intended for use with the Primary Oscillator (Posc) and automatically switches to the FRC oscillator if a Posc failure is detected. The switch to the FRC oscillator allows continued device operation and the ability to retry the Posc or to execute the appropriate for a clock failure.

The FSCM can be enabled in the device configuration during the programming of the device. During run-time, the clock failure status can be obtained as follows:

```
bool clockFail;

//Returns true if clock failure is detected.
clockFail = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

Internal FRC Oscillator Tuning

Tuning the Oscillator

Oscillator tuning will help compensate for temperature effects on the FRC frequency over a wide range of temperatures. The tuning step size is an approximation, the application is supposed to try different values to achieve the best result. In some of the devices there are different tuning modes available.

Direct Number Method

FRC tuning is based on the number specified by the PLIB_OSC_FRCTuningSelect function.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_USING_NUMBER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);
```

Sequential Dithering

To get the sequential dithering working, the client needs to set eight sequencer values. Value 0 is set using the PLIB_OSC_FRCTuningSelect function. The other seven values (values 1 through 7) are set using the PLIB_OSC_FRCTuningSequenceValueSet function.

```
// Unlock the Oscillator Registers
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_SEQ_DITHER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_1,
        OSC_FRC_TUNE_MINUS_2_25_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_2,
        OSC_FRC_TUNE_MINUS_1_5_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_3,
        OSC_FRC_TUNE_MINUS_0_375_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_4,
        OSC_FRC_TUNE_PLUS_0_43_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_5,
        OSC_FRC_TUNE_PLUS_1_29_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_6,
        OSC_FRC_TUNE_PLUS_2_54_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_7,
        OSC_FRC_TUNE_MINUS_3_PERCENT);

//Configure PWM (period, pulse width and turn on the module)
```

Pseudo-Random Number

Select the tuning mode. Then, configure the PWM module and set the period and pulse width. The Oscillator system generates a 4-bit number based on a pseudo-random number generation algorithm. It then uses this value to tune the FRC oscillator. The module will generate different frequencies corresponding to the generated pseudo-random numbers every eighth PWM cycle.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_PSEUDO_RANDOM);

//15 bit Linear Feedback shift register value
PLIB_OSC_LinearFeedbackShiftRegSet(OSC_ID_0, 0x7FFF);

//Configure PWM (period, pulse width and turn on the module)
```

Configuring the Library

The library is configured for the supported Oscillator module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_OSC_OnWaitActionGet	Gets the configured operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_OnWaitActionSet	Selects the operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_SlewDisable	Disables the selected type of slewing.
	PLIB_OSC_SlewDivisorStepGet	Get the slew divisor maximum step.

	PLIB_OSC_SlewDivisorStepSelect	Selects division steps used while slewing.
	PLIB_OSC_SlewEnable	Enables the selected type of slewing.
	PLIB_OSC_SlewIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_SleepToStartupClockGet	Returns the clock used for the duration when the device wakes from sleep and the clock ready.
	PLIB_OSC_SleepToStartupClockSelect	Selects the clock duration for when the device wakes from sleep and the clock is ready.
	PLIB_OSC_DreamModeDisable	Disables the dream mode.
	PLIB_OSC_DreamModeEnable	Enables the dream mode.
	PLIB_OSC_DreamModeStatus	gets the status of the dream mode.

b) Primary Oscillator Setup Functions

	Name	Description
	PLIB_OSC_ClockIsReady	Get the ready status of clock.
	PLIB_OSC_ClockSlewingIsActive	Returns the status of clock slewing.
	PLIB_OSC_SystemClockDivisorGet	Get the system clock divisor value.
	PLIB_OSC_SystemClockDivisorSelect	Selects system clock divisor.

c) Secondary Oscillator Setup Functions

	Name	Description
	PLIB_OSC_SecondaryDisable	Disables the Secondary Oscillator.
	PLIB_OSC_SecondaryEnable	Enables the Secondary Oscillator.
	PLIB_OSC_SecondaryIsEnabled	Returns 'true' if the Secondary Oscillator is enabled.
	PLIB_OSC_SecondaryIsReady	Returns 'true' if the Secondary Oscillator is ready.

d) Reference Oscillator Setup Functions

	Name	Description
	PLIB_OSC_ReferenceOscBaseClockSelect	Sets the base clock for the reference oscillator.
	PLIB_OSC_ReferenceOscDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOscDivisorValueSet	Selects the reference oscillator divisor value.
	PLIB_OSC_ReferenceOscEnable	Enables the reference oscillator.
	PLIB_OSC_ReferenceOscIsEnabled	Gets the enable status of the reference oscillator output.
	PLIB_OSC_ReferenceOscSourceChangeIsActive	Returns 'true' if a reference oscillator source change request is active.
	PLIB_OSC_ReferenceOscStopInIdleDisable	Enables the reference oscillator in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleEnable	Configures the reference oscillator to stop operating in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_ReferenceOscStopInSleepDisable	Enables the reference oscillator in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepEnable	Configures the reference oscillator to stop operating in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepIsEnabled	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	PLIB_OSC_ReferenceOscSwitchIsComplete	Returns 'true' if the reference oscillator base clock switching is complete.
	PLIB_OSC_ReferenceOscTrimSet	Sets the reference oscillator divisor trim value.
	PLIB_OSC_ReferenceOutputDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOutputEnable	Enables the reference oscillator output.
	PLIB_OSC_ReferenceOutputIsEnabled	Returns 'true' if the reference oscillator output is enabled.

e) Fast RC Oscillator Setup Functions

	Name	Description
	PLIB_OSC_FRCDivisorGet	Gets the FRC clock divisor.
	PLIB_OSC_FRCDivisorSelect	Sets the FRC clock divisor to the specified value.
	PLIB_OSC_FRCTuningSelect	Sets the FRC tuning value.

f) Oscillator Switch Setup Functions

	Name	Description
	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.
	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.

g) USB and Display Clock Setup Functions

	Name	Description
	PLIB_OSC_UsbClockSourceGet	Gets the USB module clock source.
	PLIB_OSC_UsbClockSourceSelect	Sets the USB module clock source.

h) PLL Setup Functions

	Name	Description
	PLIB_OSC_PLLClockIsLocked	Gets the lock status for the clock and PLL selections.
	PLIB_OSC_PLLClockLock	Locks the clock and PLL selections.
	PLIB_OSC_PLLClockUnlock	Unlocks the clock and PLL selections.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the selected PLL module is locked.
	PLIB_OSC_SysPLLFrequencyRangeGet	Gets the frequency range for the PLL module.
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceGet	Gets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_BTPLLClockOutDisable	Disables the Bluetooth PLL Clock Output.
	PLIB_OSC_BTPLLClockOutEnable	Enables the Bluetooth PLL clock Ouput.
	PLIB_OSC_BTPLLClockOutStatus	gets the status of the Bluetooth PLL clock Output.
	PLIB_OSC_BTPLLFrequencyRangeGet	Gets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLFrequencyRangeSet	Sets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceGet	Gets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceSet	Sets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputDivisorGet	Gets the input divisor for the Bluetooth PLL.
	PLIB_OSC_BTPLLInputDivisorSet	Sets the input divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_BTPLLMultiplierGet	Gets the Bluetooth PLL multiplier.
	PLIB_OSC_BTPLLMultiplierSelect	Sets the Bluetooth PLL multiplier to the specified value.
	PLIB_OSC_BTPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_BTPLLOutputDivisorSet	Sets the output divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_ForceSPLLLockDisable	Disables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockEnable	Enables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockStatus	gets the status of the force PLL Lock bit of the specified PLL.
	PLIB_OSC_PLLBypassDisable	Disables the PLL Bypass.
	PLIB_OSC_PLLBypassEnable	Enables the PLL Bypass.
	PLIB_OSC_PLLBypassStatus	gets the status of the PLL Bypass.
	PLIB_OSC_UPLLFrequencyRangeGet	Gets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLFrequencyRangeSet	Sets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLInputDivisorGet	Gets the input divisor for the USB PLL.
	PLIB_OSC_UPLLInputDivisorSet	Sets the input divider for the USB PLL to the specified value.
	PLIB_OSC_UPLLMultiplierGet	Gets the USB PLL multiplier.
	PLIB_OSC_UPLLMultiplierSelect	Sets the USB PLL multiplier to the specified value.
	PLIB_OSC_UPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_UPLLOutputDivisorSet	Sets the output divider for the USB PLL to the specified value.
	PLIB_OSC_ResetPLLAssert	Asserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLDeassert	Deasserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLStatus	gets the status of the PLL reset bit for the specified PLL.

i) Peripheral Clock Setup Functions

	Name	Description
	PLIB_OSC_PBClockDivisorGet	Gets the peripheral bus clock divisor.
	PLIB_OSC_PBClockDivisorIsReady	Checks whether the peripheral bus clock divisor is ready to be written.
	PLIB_OSC_PBClockDivisorSet	Sets the peripheral bus clock divisor to the specified value.
	PLIB_OSC_PBOutputClockDisable	Disables the peripheral bus output clock.
	PLIB_OSC_PBOutputClockEnable	Enables the peripheral bus output clock
	PLIB_OSC_PBOutputClockIsEnabled	Checks whether or not the peripheral bus clock output is enabled.

j) Clock Functions

	Name	Description
	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.
	PLIB_OSC_ClockStart	Starts the specified clock source.
	PLIB_OSC_ClockStop	Stops the specified clock source.
	PLIB_OSC_ClockStopStatus	returns the status of clock stop bit for the specified clock source.

k) Feature Existence Functions

	Name	Description
	PLIB_OSC_ExistsClockFail	Identifies whether the ClockFail feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCDivisor	Identifies whether the FRCDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCTuning	Identifies whether the FRCTuning feature exists on the Oscillator module.
	PLIB_OSC_ExistsOnWaitAction	Identifies whether the OnWaitAction feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscCurrentGet	Identifies whether the OscCurrentGet feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSelect	Identifies whether the OscSelect feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSwitchInit	Identifies whether the OscSwitchInit feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockDivisor	Identifies whether the PBClockDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockOutputEnable	Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockReady	Identifies whether the PBClockReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLClockLock	Identifies whether the PLLClockLock feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLLockStatus	Identifies whether the PLLLockStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscBaseClock	Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChange	Identifies whether the ReferenceOscChange feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChangeActive	Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscDivisor	Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscEnable	Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInIdleEnable	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInSleep	Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscTrim	Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOutputEnable	Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryEnable	Identifies whether the SecondaryEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryReady	Identifies whether the SecondaryReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLFrequencyRange	Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputClockSource	Identifies whether the PLLInputClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputDivisor	Identifies whether the PLLInputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLMultiplier	Identifies whether the PLLMultiplier feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLOutputDivisor	Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.

	PLIB_OSC_ExistsUsbClockSource	Identifies whether the UsbClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockReadyStatus	Identifies whether the ClockReadyStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockSlewingStatus	Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsSleepToStartupClock	Identifies whether the SleepToStartupClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewDivisorStepControl	Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewEnableControl	Identifies whether the SlewEnableControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSystemClockDivisorControl	Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsBTPLLClockOut	Identifies whether the BTPLLClockOut feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLFrequencyRange	Identifies whether the BTPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputClockSource	Identifies whether the BTPLLInputClockSource feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputDivisor	Identifies whether the BTPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLMultiplier	Identifies whether the BTPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLOutputDivisor	Identifies whether the BTPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsClockDiagStatus	Identifies whether the ClockDiagStatus feature exists on the OSC module
	PLIB_OSC_ExistsDreamModeControl	Identifies whether the DreamModeControl feature exists on the OSC module
	PLIB_OSC_ExistsForceLock	Identifies whether the ForceLock feature exists on the OSC module
	PLIB_OSC_ExistsPLLBypass	Identifies whether the SPLLBypass feature exists on the OSC module
	PLIB_OSC_ExistsResetPLL	Identifies whether the ResetPLL feature exists on the OSC module
	PLIB_OSC_ExistsUPLLFrequencyRange	Identifies whether the UPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsUPLLInputDivisor	Identifies whether the UPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUPLLMultiplier	Identifies whether the UPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsUPLLOutputDivisor	Identifies whether the UPLLOutputDivisor feature exists on the OSC module

l) Data Types and Constants

	Name	Description
	OSC_PB_CLOCK_DIV_TYPE	Type of the oscillator PB Clock divisor value.
	OSC_REF_DIVISOR_TYPE	Reference oscillator divisor type.
	OSC_REFERENCE_MAX_DIV	Defines the reference clock output divisor maximum value.
	OSC_SYSPLL_MULTIPLIER_TYPE	Type of the oscillator system PLL multiplier value.
	OSC_FRC_DIV	Lists the possible Fast RC (FRC) Oscillator divider values.
	OSC_MODULE_ID	Possible instances of the OSC module.
	OSC_OPERATION_ON_WAIT	Lists the possible base clock values for the reference oscillator.
	OSC_PERIPHERAL_BUS	Lists the possible Peripheral buses.
	OSC_PLL_SELECT	Lists the PLLs in the Oscillator module.
	OSC_REFERENCE	Lists the possible reference oscillator.
	OSC_SYS_TYPE	Lists the possible oscillator type values.
	OSC_SYSPLL_FREQ_RANGE	Lists the possible PLL frequency range.
	OSC_SYSPLL_IN_CLK_SOURCE	Lists the possible input clock source for PLL module.
	OSC_SYSPLL_OUT_DIV	Lists the possible PLL output divider values.
	OSC_USBCLOCK_SOURCE	Lists the possible USB clock sources.
	OSC_CLOCK_ID	Lists the clock sources for which ready status can be checked.
	OSC_CLOCK_SLEW_TYPE	Lists the possible type of clock slewing.
	OSC_SLEEP_TO_STARTUP_CLK_TYPE	Lists the possible clock sources for sleep to start-up period.

Description

This section describes the Application Programming Interface (API) functions of the Oscillator Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_OSC_OnWaitActionGet Function

Gets the configured operation to be performed when a WAIT instruction is executed.

File

plib_osc.h

C

```
OSC_OPERATION_ON_WAIT PLIB_OSC_OnWaitActionGet(OSC_MODULE_ID index);
```

Returns

On a WAIT action, one of the possible values of OSC_OPERATION_ON_WAIT.

Description

This function gets the configured operation that is to be performed when a WAIT instruction is executed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOnWaitAction in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_OnWaitActionGet(OSC_ID_0) == OSC_ON_WAIT_SLEEP)
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_OPERATION_ON_WAIT PLIB_OSC_OnWaitActionGet (OSC_MODULE_ID index)

PLIB_OSC_OnWaitActionSet Function

Selects the operation to be performed when a WAIT instruction is executed.

File

plib_osc.h

C

```
void PLIB_OSC_OnWaitActionSet(OSC_MODULE_ID index, OSC_OPERATION_ON_WAIT onWait);
```

Returns

None.

Description

This function selects the operation to be performed when a WAIT instruction is executed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOnWaitAction in your application to determine whether this feature is available.

If this function is not called, the device will enter Idle mode on execution of a WAIT instruction.

Preconditions

None.

Example

```
PLIB_OSC_OnWaitActionSet(OSC_ID_0, OSC_ON_WAIT_SLEEP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
onWait	Operation to be performed when a WAIT instruction is executed. One of the possible values of OSC_OPERATION_ON_WAIT.

Function

void PLIB_OSC_OnWaitActionSet (OSC_MODULE_ID index,
OSC_OPERATION_ON_WAIT onWait)

PLIB_OSC_SlewDisable Function

Disables the selected type of slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewDisable(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

None.

Description

This function disables slewing to an upward or downward frequency based on the selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewDisable(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE

Function

void PLIB_OSC_SlewDisable (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SlewDivisorStepGet Function

Get the slew divisor maximum step.

File

plib_osc.h

C

```
uint32_t PLIB_OSC_SlewDivisorStepGet(OSC_MODULE_ID index);
```

Returns

- slewSteps - Number of steps in which slewing is occurring

Description

This function returns the number of division steps used when slewing during a frequency change.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewDivisorStepControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t slewSteps;
slewSteps = PLIB_OSC_SlewDivisorStepGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint32_t PLIB_OSC_SlewDivisorStepGet

(

OSC_MODULE_ID index

)

PLIB_OSC_SlewDivisorStepSelect Function

Selects division steps used while slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewDivisorStepSelect(OSC_MODULE_ID index, uint32_t slewSteps);
```

Returns

None.

Description

This API selects number of division steps to be used when slewing during a frequency change. If the number of steps chosen is 0, no divisor will be used while slewing. If the number of steps chosen is 1, slewing will start with divide by 2, and then no divisor. Similarly, if the number of steps chosen is 2, slewing will start with divisor 4. Then,after a few seconds the divisor will be 2, and then after a few more seconds no divisor will be used and so on. Therefore, the largest step would be equal to 2^(number of steps), and then the step size will reduce by half every few seconds.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewDivisorStepControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewDivisorStepSelect(OSC_ID_0, OSC_SLEW_DIVISOR_STEP_MAX_32);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewSteps	Number of steps in which slewing is desired

Function

void PLIB_OSC_SlewDivisorStepSelect

(
OSC_MODULE_ID index,
uint32_t slewSteps
)

PLIB_OSC_SlewEnable Function

Enables the selected type of slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewEnable(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

None.

Description

This function enables slewing to an upward or downward frequency based on the selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewEnable(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE

Function

void PLIB_OSC_SlewEnable (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SlewIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Idle mode.

File

plib_osc.h

C

```
bool PLIB_OSC_SlewIsEnabled(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

- true - The selected type of Slewing is enabled
- false - The selected type of Slewing is disabled

Description

This function returns 'true' if the reference oscillator is disabled in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool slewStatus;
slewStatus = PLIB_OSC_SlewIsEnabled(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE.

Function

bool PLIB_OSC_SlewIsEnabled (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SleepToStartupClockGet Function

Returns the clock used for the duration when the device wakes from sleep and the clock ready.

File

plib_osc.h

C

```
OSC_SLEEP_TO_STARTUP_CLK_TYPE PLIB_OSC_SleepToStartupClockGet(OSC_MODULE_ID index);
```

Returns

- startupOsc - One of the possible values from OSC_SLEEP_TO_STARTUP_CLK_TYPE

Description

When a device enters Sleep mode, the source of the system clock usually stops and takes some time after waking up to start running at full speed. This function is to return the clock set for that duration in which the actual clock source is starting up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSleepToStartupClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SLEEP_TO_STARTUP_CLK_TYPE startupClk;
startupClk = PLIB_OSC_SleepToStartupClockGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SLEEP_TO_STARTUP_CLK_TYPE PLIB_OSC_SleepToStartupClockGet
(
OSC_MODULE_ID index
)

PLIB_OSC_SleepToStartupClockSelect Function

Selects the clock duration for when the device wakes from sleep and the clock is ready.

File

plib_osc.h

C

```
void PLIB_OSC_SleepToStartupClockSelect(OSC_MODULE_ID index, OSC_SLEEP_TO_STARTUP_CLK_TYPE startupOsc);
```

Returns

None.

Description

When a device enters Sleep mode, the source of the system clock usually stops and takes some time after waking up to start running at full speed. This function allows the user to select a clock for that duration in which the actual clock source is starting up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSleepToStartupClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SleepToStartupClockSelect(OSC_ID_0, OSC_SLEEP_TO_STARTUP_CLK_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
startupOsc	One of the possible values from OSC_SLEEP_TO_STARTUP_CLK_TYPE

Function

void PLIB_OSC_SleepToStartupClockSelect

(

OSC_MODULE_ID index,

OSC_SLEEP_TO_STARTUP_CLK_TYPE startupOsc

)

PLIB_OSC_DreamModeDisable Function

Disables the dream mode.

File

plib_osc.h

C

```
void PLIB_OSC_DreamModeDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to disable the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_DreamModeDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_DreamModeDisable (OSC_MODULE_ID index)

PLIB_OSC_DreamModeEnable Function

Enables the dream mode.

File

plib_osc.h

C

```
void PLIB_OSC_DreamModeEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to enable the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_DreamModeEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_DreamModeEnable (OSC_MODULE_ID index)

PLIB_OSC_DreamModeStatus Function

gets the status of the dream mode.

File

plib_osc.h

C

```
bool PLIB_OSC_DreamModeStatus(OSC_MODULE_ID index);
```

Returns

Status of dream mode bit.

Description

This function is used to get the status of the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool drmMode = PLIB_OSC_DreamModeStatus(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_DreamModeStatus (OSC_MODULE_ID index)

b) Primary Oscillator Setup Functions

PLIB_OSC_ClockIsReady Function

Get the ready status of clock.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockIsReady(OSC_MODULE_ID index, OSC_CLOCK_ID clk);
```

Returns

- true - Indicates that the selected clock is running and is stable
- false - Indicates that the selected clock is either turned off or is still warming up

Description

This function returns the ready status of selected clock.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockReadyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool clkReady;
clkReady = PLIB_OSC_ClockIsReady(OSC_ID_0, OSC_CLOCK_FAST_RC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module.
clk	One of the values of OSC_CLOCK_ID.

Function

bool PLIB_OSC_ClockIsReady (OSC_MODULE_ID index, OSC_CLOCK_ID clk)

PLIB_OSC_ClockSlewingIsActive Function

Returns the status of clock slewing.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockSlewingIsActive(OSC_MODULE_ID index);
```

Returns

- true - The clock frequency is being actively slewed to the new frequency.
- false - The clock switch has reached its final value

Description

This function returns the current status of clock switching slewing.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockSlewingStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_ClockSlewingIsActive(OSC_ID_0))
{
    //wait for clock switch to complete
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module.

Function

bool PLIB_OSC_ClockSlewingIsActive (OSC_MODULE_ID index)

PLIB_OSC_SystemClockDivisorGet Function

Get the system clock divisor value.

File

plib_osc.h

C

```
uint32_t PLIB_OSC_SystemClockDivisorGet(OSC_MODULE_ID index);
```

Returns

- systemClkDivisor - The value by which system clock is divided.

Description

This function returns the divisor set for system clock.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSystemClockDivisorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t systemClkDivisor;
systemClkDivisor = PLIB_OSC_SystemClockDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint32_t PLIB_OSC_SystemClockDivisorGet
(
OSC_MODULE_ID index
)

PLIB_OSC_SystemClockDivisorSelect Function

Selects system clock divisor.

File

plib_osc.h

C

```
void PLIB_OSC_SystemClockDivisorSelect(OSC_MODULE_ID index, uint32_t systemClkDivisor);
```

Returns

None.

Description

This function selects the system clock divisor.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSystemClockDivisorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SystemClockDivisorSelect(OSC_ID_0, OSC_SYSTEM_CLOCK_DIVIDED_BY_10);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
systemClkDivisor	The value by which the system clock is to be divided

Function

void PLIB_OSC_SystemClockDivisorSelect
(
OSC_MODULE_ID index,
uint32_t systemClkDivisor
)

c) Secondary Oscillator Setup Functions

PLIB_OSC_SecondaryDisable Function

Disables the Secondary Oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SecondaryDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function disables the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SecondaryDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_SecondaryDisable (OSC_MODULE_ID index);

PLIB_OSC_SecondaryEnable Function

Enables the Secondary Oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SecondaryEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function enables the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SecondaryEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_SecondaryEnable (OSC_MODULE_ID index)

PLIB_OSC_SecondaryIsEnabled Function

Returns 'true' if the Secondary Oscillator is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_SecondaryIsEnabled(OSC_MODULE_ID index);
```

Returns

- true - The Secondary Oscillator is enabled
- false - The Secondary Oscillator is disabled

Description

This function returns 'true' if the Secondary Oscillator is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool secOscEnable;

secOscEnable = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_SecondaryIsEnabled (OSC_MODULE_ID index);

PLIB_OSC_SecondaryIsReady Function

Returns 'true' if the Secondary Oscillator is ready.

File

plib_osc.h

C

```
bool PLIB_OSC_SecondaryIsReady(OSC_MODULE_ID index);
```

Returns

- true - Indicates that the Secondary Oscillator is running and is stable
- false - Indicates that the Secondary Oscillator is either turned off or is still warming up

Description

This function returns the ready status of the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryReady in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool secOscReady;

secOscReady = PLIB_OSC_SecondaryIsReady(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_SecondaryIsReady (OSC_MODULE_ID index)

d) Reference Oscillator Setup Functions

PLIB_OSC_ReferenceOscBaseClockSelect Function

Sets the base clock for the reference oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscBaseClockSelect(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_BASECLOCK refOscBaseClock);
```

Returns

None.

Description

This function sets the base clock for the reference oscillator. There are multiple clock sources by which the user can configure the module to output to the pin. Users can check the accuracy of the clock by probing the pin. Use the PLIB_OSC_ReferenceOscDivisorValueSet function to divide the clock if it is a very high value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscBaseClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscBaseClk	One of the possible values of OSC_REF_BASECLOCK

Function

void PLIB_OSC_ReferenceOscBaseClockSelect (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc, OSC_REF_BASECLOCK refOscBaseClock)

PLIB_OSC_ReferenceOscDisable Function

Disables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function disables output from the reference oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscDivisorValueSet Function

Selects the reference oscillator divisor value.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscDivisorValueSet(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_DIVISOR_TYPE refOscDivValue);
```

Returns

None.

Description

This function selects the reference oscillator divisor value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscDivisor in your application to determine whether this feature is available.

The value entered may not be the actual divisor. Please refer to the specific device data sheet to determine the actual divisor corresponding to the value entered.

Preconditions

None.

Example

```
// Select the clock source.
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);

PLIB_OSC_ReferenceOscDivisorValueSet(OSC_ID_0, OSC_REFERENCE_1, 128);

PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscDivValue	Value for Reference Oscillator Divisor (RODIV) field. If it is '0', the divider is not used.

Function

void PLIB_OSC_ReferenceOscDivisorValueSet (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_DIVISOR_TYPE refOscDivValue)

PLIB_OSC_ReferenceOscEnable Function

Enables the reference oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

If the device has a reference clock output enable control, calling this function may not give the reference clock output. Use the PLIB_OSC_ReferenceOutputEnable function to enable the output.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscIsEnabled Function

Gets the enable status of the reference oscillator output.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is enabled
- false - The reference oscillator is disabled

Description

This function gets the enable status of the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_OSC_ReferenceOscIsEnabled(OSC_ID_0, OSC_REFERENCE_1))
{
//Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscSourceChangeIsActive Function

Returns 'true' if a reference oscillator source change request is active.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscSourceChangeIsActive(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator change request is active
- false - The reference oscillator change request is not active

Description

This function returns 'true' if the reference oscillator source change is in progress. The software is not allowed to give a new source change request.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscChangeActive in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (!PLIB_OSC_ReferenceOscSourceChangeIsActive(OSC_ID_0, OSC_REFERENCE_1))
{
    //Allowed to change the reference clock source
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscSourceChangeIsActive (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleDisable Function

Enables the reference oscillator in Idle mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInIdleDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator in Idle mode. The reference oscillator continues to run in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

The default state of the device is the reference oscillator is enabled in Idle mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOsctopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInIdleDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleEnable Function

Configures the reference oscillator to stop operating in Idle mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInIdleEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function configures the reference oscillator to stop operating in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator enabled in Idle mode. Therefore, calling this function is necessary only if the PLIB_OSC_ReferenceOscStopInIdleDisable function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscStopInIdleEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInIdleEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Idle mode.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscStopInIdleIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is disabled in Idle mode
- false - The reference oscillator is enabled in Idle mode

Description

This function returns 'true' if the reference oscillator is disabled in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOscStopInIdleIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscStopInIdleIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepDisable Function

Enables the reference oscillator in Sleep mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInSleepDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator in Sleep mode. The reference oscillator continues to run in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

The reference clock output will be stopped in Sleep mode if the base clock selected is 'System Clock' or 'Peripheral Clock' regardless of this function.

The default state of the device is for the reference oscillator to be enabled in Sleep mode. Therefore, calling this function is necessary only if the PLIB_OSC_ReferenceOscStopInSleepEnable function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOsctopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInSleepDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepEnable Function

Configures the reference oscillator to stop operating in Sleep mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInSleepEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function configures the reference oscillator to stop operating in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

The default state of the device is for the reference oscillator to be enabled in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscStopInSleepEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInSleepEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Sleep mode.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscStopInSleepIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is disabled in Sleep mode
- false - The reference oscillator is enabled in Sleep mode

Description

This function returns 'true' if the reference oscillator is disabled in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscSleep;

refOscSleep = PLIB_OSC_ReferenceOscStopInSleepIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscStopInSleepIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscSwitchIsComplete Function

Returns 'true' if the reference oscillator base clock switching is complete.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscSwitchIsComplete(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference clock base clock switching is complete
- false - The reference clock base clock switching is not complete; switching is not started

Description

This function returns 'true' if the reference oscillator base clock switching is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscChange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOscSwitchIsComplete(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscSwitchIsComplete (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscTrimSet Function

Sets the reference oscillator divisor trim value.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscTrimSet(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc, OSC_REF_TRIM_TYPE
trimValue);
```

Returns

None.

Description

This function selects the reference oscillator divisor trim value. The value selected divided by OSC_REF_TRIM_MAX_VALUE will be added to the oscillator divisor value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscTrim in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscTrimSet(OSC_ID_0, OSC_REFERENCE_1, 50);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
trimValue	Reference oscillator trim value

Function

void PLIB_OSC_ReferenceOscTrimSet (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_TRIM_TYPE trimValue)

PLIB_OSC_ReferenceOutputDisable Function

Disables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOutputDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function disables the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOutputDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOutputDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOutputEnable Function

Enables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOutputEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOutputEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOutputEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOutputIsEnabled Function

Returns 'true' if the reference oscillator output is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOutputIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator output is enabled
- false - The reference oscillator output is disabled

Description

This function returns 'true' if the reference oscillator output is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOutputIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOutputIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

e) Fast RC Oscillator Setup Functions

PLIB_OSC_FRCDivisorGet Function

Gets the FRC clock divisor.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_FRCDivisorGet(OSC_MODULE_ID index);
```

Returns

The FRC divisor value.

Description

This function gets the FRC clock divisor. The value returned will be direct number and not enum equivalent.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t divisorFRC;

divisorFRC = PLIB_OSC_FRCDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_FRCDivisorGet(OSC_MODULE_ID index)

PLIB_OSC_FRCDivisorSelect Function

Sets the FRC clock divisor to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_FRCDivisorSelect(OSC_MODULE_ID index, OSC_FRC_DIV divisorFRC);
```

Returns

None.

Description

This function sets the FRC clock divisor to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
divisorFRC	One of the possible values from OSC_FRC_DIV

Function

void PLIB_OSC_FRCDivisorSelect (OSC_MODULE_ID index,
OSC_FRC_DIV divisorFRC)

PLIB_OSC_FRCTuningSelect Function

Sets the FRC tuning value.

File

plib_osc.h

C

```
void PLIB_OSC_FRCTuningSelect(OSC_MODULE_ID index, OSC_FRC_TUNE_TYPE tuningValue);
```

Returns

None.

Description

This function tunes the FRC oscillator to the value specified. The application is supposed to try different values and find the best one. If the device has different tuning modes, this function will be used differently. See the example provided for the PLIB_OSC_FRCTuningSequenceValueSet function for details.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCTuning in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x05);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
tuningValue	Tuning value. One of the possible values from OSC_FRC_TUNE_TYPE.

Function

void PLIB_OSC_FRCTuningSelect (OSC_MODULE_ID index,
OSC_FRC_TUNE_TYPE tuningValue)

f) Oscillator Switch Setup Functions

PLIB_OSC_ClockSwitchingAbort Function

Aborts an oscillator switch.

File

plib_osc.h

C

```
void PLIB_OSC_ClockSwitchingAbort(OSC_MODULE_ID index);
```

Returns

None.

Description

This function aborts the oscillator switch to the selection specified by the new oscillator selection bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSwitchInit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_ClockSwitchingAbort (OSC_MODULE_ID index)

PLIB_OSC_ClockSwitchingIsComplete Function

Gets the oscillator switch progress status.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockSwitchingIsComplete(OSC_MODULE_ID index);
```

Returns

- true - The oscillator switch is complete
- false - The oscillator switch is in progress

Description

This function gets the status of the oscillator switch progress.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSwitchInit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);

while(!PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0));
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_ClockSwitchingIsComplete(OSC_MODULE_ID index);

PLIB_OSC_CurrentSysClockGet Function

Gets the current oscillator selected.

File

plib_osc.h

C

```
OSC_SYS_TYPE PLIB_OSC_CurrentSysClockGet(OSC_MODULE_ID index);
```

Returns

One of the possible values from OSC_SYS_TYPE.

Description

This function gets the current oscillator. If the application has not changed the oscillator selection, this will be same as the oscillator selected through the Configuration bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscCurrentGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYS_TYPE oscCurrent;

oscCurrent = PLIB_OSC_CurrentSysClockGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYS_TYPE PLIB_OSC_CurrentSysClockGet (OSC_MODULE_ID index)

PLIB_OSC_SysClockSelect Function

Selects the new oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SysClockSelect(OSC_MODULE_ID index, OSC_SYS_TYPE newOsc);
```

Returns

None.

Description

This function selects the new oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSelect in your application to determine whether this feature is available.

This function adds the necessary delay (NOP instructions) after switching the oscillator. Therefore, the user need not add any delay as specified in the device data sheet.

Preconditions

None.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
newOsc	One of the possible values from OSC_SYS_TYPE

Function

void PLIB_OSC_SysClockSelect (OSC_MODULE_ID index,
OSC_SYS_TYPE newOsc)

g) USB and Display Clock Setup Functions

PLIB_OSC_UsbClockSourceGet Function

Gets the USB module clock source.

File

plib_osc.h

C

```
OSC_USBCLOCK_SOURCE PLIB_OSC_UsbClockSourceGet(OSC_MODULE_ID index);
```

Returns

USB module clock source. One of the possible values from OSC_USBCLOCK_SOURCE.

Description

This function gets the USB module clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUsbClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (SYS_OSC_USBCLK_FRC == PLIB_OSC_UsbClockSourceGet(OSC_ID_0))
{
//Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_USBCLOCK_SOURCE PLIB_OSC_UsbClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_UsbClockSourceSelect Function

Sets the USB module clock source.

File

plib_osc.h

C

```
void PLIB_OSC_UsbClockSourceSelect(OSC_MODULE_ID index, OSC_USBCLOCK_SOURCE usbClock);
```

Returns

None.

Description

This function sets the USB module clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUsbClockSource in your application to determine whether this feature is available.

Before placing the USB module in Suspend mode, use this function to enable the FRC clock.

Preconditions

None.

Example

```
PLIB_OSC_UsbClockSourceSelect(OSC_ID_0, SYS_OSC_USBCLK_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
usbClock	Select the USB module clock source. One of the possible values from OSC_USBCLOCK_SOURCE.

Function

void PLIB_OSC_UsbClockSourceSelect (OSC_MODULE_ID index,
OSC_USBCLOCK_SOURCE usbClock)

h) PLL Setup Functions

PLIB_OSC_PLLClockIsLocked Function

Gets the lock status for the clock and PLL selections.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLClockIsLocked(OSC_MODULE_ID index);
```

Returns

- true - The clock and PLL selections are locked
- false - The clock and PLL selections are not locked

Description

This function gets the lock status for the clock and PLL selections.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

Preconditions

None.

Example

```
bool clockPLL_st;

clockPLL_st = PLIB_OSC_PLLClockIsLocked(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_PLLClockIsLocked (OSC_MODULE_ID index)

PLIB_OSC_PLLClockLock Function

Locks the clock and PLL selections.

File

plib_osc.h

C

```
void PLIB_OSC_PLLClockLock(OSC_MODULE_ID index);
```

Returns

None.

Description

This function locks the clock and PLL selections.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

The data given by this function is only valid if clock switching and monitoring are enabled. Otherwise Clock and PLL selections are never locked and may be modified.

Preconditions

The Fail-Safe Clock Monitor (FSCM) should be enabled.

Example

```
PLIB_OSC_PLLClockLock(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_PLLClockLock (OSC_MODULE_ID index)

PLIB_OSC_PLLClockUnlock Function

Unlocks the clock and PLL selections.

File

plib_osc.h

C

```
void PLIB_OSC_PLLClockUnlock(OSC_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the clock and PLL selection so that the clock and PLL may be modified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLClockUnlock(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_PLLClockUnlock (OSC_MODULE_ID index)

PLIB_OSC_PLLIsLocked Function

Returns 'true' if the selected PLL module is locked.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLIsLocked(OSC_MODULE_ID index, OSC_PLL_SELECT pllselect);
```

Returns

- true - The PLL module is in lock or the PLL module start-up timer is satisfied
- false - The PLL module is out of lock, the PLL start-up timer is running, or the PLL module is disabled

Description

This function returns the lock status of the selected PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLLockStatus in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

Preconditions

None.

Example

```
bool lockPLL_status;

lockPLL_status = PLIB_OSC_PLLIsLocked(OSC_ID_0, OSC_PLL_SYSTEM);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllselect	Selects the PLL module

Function

bool PLIB_OSC_PLLIsLocked (OSC_MODULE_ID index, OSC_PLL_SELECT pllselect)

PLIB_OSC_SysPLLFrequencyRangeGet Function

Gets the frequency range for the PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_FREQ_RANGE PLIB_OSC_SysPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_SYSPLL_FREQ_RANGE

Description

This function returns the frequency range set for the PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_SysPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_FREQ_RANGE PLIB_OSC_SysPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLFrequencyRangeSet Function

Sets the frequency range for the PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLFrequencyRangeSet(OSC_ID_0, OSC_SYSPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_SYSPLL_FREQ_RANGE

Function

void PLIB_OSC_SysPLLFrequencyRangeSet (OSC_MODULE_ID index,
OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_SysPLLInputClockSourceGet Function

Gets the input clock source for the PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_SysPLLInputClockSourceGet(OSC_MODULE_ID index);
```

Returns

- PLLInClockSource - One of the possible values from OSC_SYSPLL_IN_CLK_SOURCE

Description

This function returns the input clock source for the PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource;
PLLInClockSource = PLIB_OSC_SysPLLInputClockSourceGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_SysPLLInputClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLInputClockSourceSet Function

Sets the input clock source for the PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLInputClockSourceSet(OSC_MODULE_ID index, OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource);
```

Returns

None.

Description

This function sets the input clock source for the PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLInputClockSourceSet(OSC_ID_0, OSC_SYSPLL_IN_CLK_SOURCE_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInClockSource	One of the possible values from OSC_SYSPLL_IN_CLK_SOURCE

Function

void PLIB_OSC_SysPLLInputClockSourceSet (OSC_MODULE_ID index,
OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource)

PLIB_OSC_SysPLLInputDivisorGet Function

Gets the input divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_SysPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The System PLL input divisor as a number.

Description

This function returns the input divisor for the PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;
```

```
PLLInDiv = PLIB_OSC_SysPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_SysPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLMultiplierGet Function

Gets the PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_SysPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of PB clock divisor of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'PLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;

pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_SysPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLMultiplierSelect Function

Sets the PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLMultiplier in your application to determine whether this feature is available.

Use the PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'PLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values of PB clock divisor of type OSC_PLL_MUL_TYPE

Function

void PLIB_OSC_SysPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_SysPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_SysPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

System PLL output divisor value.

Description

This function returns the output divisor for the System PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_SysPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLOutputDivisorSet Function

Sets the output divider for the PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_SYSPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_SYSPLL_OUT_DIV

Function

void PLIB_OSC_SysPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_SYSPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_SysPLLInputDivisorSet Function

Sets the input divider for the PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	System PLL input divisor value

Function

void PLIB_OSC_SysPLLInputDivisorSet (OSC_MODULE_ID index,
uint16_t PLLInDiv)

PLIB_OSC_BTPLLClockOutDisable Function

Disables the Bluetooth PLL Clock Output.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLClockOutDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to disable the Bluetooth PLL clock output to SoC pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLClockOutDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_BTPLLClockOutDisable (OSC_MODULE_ID index)

PLIB_OSC_BTPLLClockOutEnable Function

Enables the Bluetooth PLL clock Ouput.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLClockOutEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to enable the Bluetooth PLL clock Output to the SoC pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLClockOutEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_BTPLLClockOutEnable (OSC_MODULE_ID index)

PLIB_OSC_BTPLLClockOutStatus Function

gets the status of the Bluetooth PLL clock Output.

File

plib_osc.h

C

```
bool PLIB_OSC_BTPLLClockOutStatus(OSC_MODULE_ID index);
```

Returns

Status of Bluetooth PLL clock output.

Description

This function is used to get the status of the Bluetooth PLL clock Output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool btpllClkOut = PLIB_OSC_BTPLLClockOutStatus(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_BTPLLClockOutStatus (OSC_MODULE_ID index)

PLIB_OSC_BTPLLFrequencyRangeGet Function

Gets the frequency range for the Bluetooth PLL module.

File

plib_osc.h

C

```
OSC_BTPLL_FREQ_RANGE PLIB_OSC_BTPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_BTPLL_FREQ_RANGE

Description

This function returns the frequency range set for the Bluetooth PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_BTPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_BTPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_BTPLL_FREQ_RANGE PLIB_OSC_BTPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLFrequencyRangeSet Function

Sets the frequency range for the Bluetooth PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_BTPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the Bluetooth PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLFrequencyRangeSet(OSC_ID_0, OSC_BTPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_BTPLL_FREQ_RANGE

Function

void PLIB_OSC_BTPLLFrequencyRangeSet (OSC_MODULE_ID index,
OSC_BTPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_BTPLLInputClockSourceGet Function

Gets the input clock source for the Bluetooth PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_BTPLLInputClockSourceGet(OSC_MODULE_ID index);
```

Returns

- PLLInClockSource - One of the possible values from OSC_BTPLL_IN_CLK_SOURCE

Description

This function returns the input clock source for the Bluetooth PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource;
PLLInClockSource = PLIB_OSC_BTPLLInputClockSourceGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_BTPLL_IN_CLK_SOURCE PLIB_OSC_BTPLLInputClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLInputClockSourceSet Function

Sets the input clock source for the Bluetooth PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLInputClockSourceSet(OSC_MODULE_ID index, OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource);
```

Returns

None.

Description

This function sets the input clock source for the Bluetooth PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLInputClockSourceSet(OSC_ID_0, OSC_BTPLL_IN_CLK_SOURCE_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInClockSource	One of the possible values from OSC_BTPLL_IN_CLK_SOURCE

Function

void PLIB_OSC_BTPLLInputClockSourceSet (OSC_MODULE_ID index,
OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource)

PLIB_OSC_BTPLLInputDivisorGet Function

Gets the input divisor for the Bluetooth PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_BTPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The Bluetooth PLL input divisor as a number.

Description

This function returns the input divisor for the Bluetooth PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;

PLLInDiv = PLIB_OSC_BTPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_BTPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLInputDivisorSet Function

Sets the input divider for the Bluetooth PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the Bluetooth PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	USB PLL input divisor value

Function

void PLIB_OSC_BTPLLInputDivisorSet (OSC_MODULE_ID index, uint16_t PLLInDiv)

PLIB_OSC_BTPLLMultiplierGet Function

Gets the Bluetooth PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_BTPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of Bluetooth Multiplier of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the Bluetooth PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'BTPLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;

pll_multiply = PLIB_OSC_BTPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_BTPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLMultiplierSelect Function

Sets the Bluetooth PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the Bluetooth PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLMultiplier in your application to determine whether this feature is available.

Use the USB PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'BTPLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values between 0 and 255

Function

void PLIB_OSC_BTPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_BTPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_BTPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

Bluetooth PLL output divisor value.

Description

This function returns the output divisor for the Bluetooth PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_BTPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_BTPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLOutputDivisorSet Function

Sets the output divider for the Bluetooth PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_BTPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the Bluetooth PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLOutputDivisorSelect(OSC_ID_0, OSC_BTPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_BTPLL_OUT_DIV

Function

void PLIB_OSC_BTPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_BTPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_ForceSPLLLockDisable Function

Disables the Force PLL Lock feature for specified PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ForceSPLLLockDisable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to disable the Force PLL Lock feature for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ForceSPLLLockDisable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which force locked to be disabled

Function

void PLIB_OSC_ForceSPLLLockDisable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ForceSPLLLockEnable Function

Enables the Force PLL Lock feature for specified PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ForceSPLLLockEnable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to enable the Force PLL Lock feature for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ForceSPLLLockEnable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL to be force locked

Function

void PLIB_OSC_ForceSPLLLockEnable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ForceSPLLLockStatus Function

gets the status of the force PLL Lock bit of the specified PLL.

File

plib_osc.h

C

```
bool PLIB_OSC_ForceSPLLLockStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of Force PLL Lock Status for the specified PLL.

Description

This function is used to get the status of the force PLL Lock bit of the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool spllFLock = PLIB_OSC_ForceSPLLLockStatus(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which force lock status to be read.

Function

bool PLIB_OSC_ForceSPLLLockStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassDisable Function

Disables the PLL Bypass.

File

plib_osc.h

C

```
void PLIB_OSC_PLLBypassDisable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to disable the PLL Bypass for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLBypassDisable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which bypass has to be disabled

Function

void PLIB_OSC_SPLLBypassDisable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassEnable Function

Enables the PLL Bypass.

File

plib_osc.h

C

```
void PLIB_OSC_PLLBypassEnable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to enable the PLL Bypass for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLBypassEnable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which bypass has to be enabled

Function

void PLIB_OSC_PLLBypassEnable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassStatus Function

gets the status of the PLL Bypass.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLBypassStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of SPLL Bypass bit.

Description

This function is used to get the status of the PLL Bypass bit for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool spllBypass = PLIB_OSC_SPLLBypassStatus(OSC_ID_0. OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL whose Bypass status has to be read

Function

bool PLIB_OSC_SPLLBypassStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_UPLLFrequencyRangeGet Function

Gets the frequency range for the USB PLL module.

File

plib_osc.h

C

```
OSC_UPLL_FREQ_RANGE PLIB_OSC_UPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_UPLL_FREQ_RANGE

Description

This function returns the frequency range set for the USB PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_UPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_UPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_UPLL_FREQ_RANGE PLIB_OSC_UPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLFrequencyRangeSet Function

Sets the frequency range for the USB PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_UPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the USB PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLFrequencyRangeSet(OSC_ID_0, OSC_UPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_UPLL_FREQ_RANGE

Function

void PLIB_OSC_UPLLFrequencyRangeSet (OSC_MODULE_ID index,
OSC_UPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_UPLLInputDivisorGet Function

Gets the input divisor for the USB PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_UPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The USB PLL input divisor as a number.

Description

This function returns the input divisor for the USB PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;

PLLInDiv = PLIB_OSC_UPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_UPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLInputDivisorSet Function

Sets the input divider for the USB PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the USB PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	USB PLL input divisor value

Function

void PLIB_OSC_UPLLInputDivisorSet (OSC_MODULE_ID index,

uint16_t PLLInDiv)

PLIB_OSC_UPLLMultiplierGet Function

Gets the USB PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_UPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of USB Multiplier of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the USB PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'UPLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;

pll_multiply = PLIB_OSC_UPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_UPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLMultiplierSelect Function

Sets the USB PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the USB PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLMultiplier in your application to determine whether this feature is available.

Use the USB PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'UPLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_UPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values between 0 and 255

Function

void PLIB_OSC_UPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_UPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_UPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

USB PLL output divisor value.

Description

This function returns the output divisor for the USB PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_UPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_UPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLOutputDivisorSet Function

Sets the output divider for the USB PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_UPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the USB PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLOutputDivisorSelect(OSC_ID_0, OSC_UPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_UPLL_OUT_DIV

Function

void PLIB_OSC_UPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_UPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_ResetPLLAssert Function

Asserts the PLL reset for selected PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ResetPLLAssert(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to assert the PLL reset for the selected PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ResetPLLAssert(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which reset to be asserted

Function

void PLIB_OSC_ResetPLLAssert (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ResetPLLDeassert Function

Deasserts the PLL reset for selected PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ResetPLLDeassert(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to deassert the PLL reset for the selected PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ResetPLLDeassert(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which reset to be deasserted

Function

void PLIB_OSC_ResetPLLDeassert (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ResetPLLStatus Function

gets the status of the PLL reset bit for the specified PLL.

File

plib_osc.h

C

```
bool PLIB_OSC_ResetPLLStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of PLL reset bit for the specified PLL.

Description

This function is used to get the status of the PLL reset bit for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool upllReset = PLIB_OSC_ResetPLLStatus(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL whose reset status to be checked

Function

bool PLIB_OSC_ResetPLLStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

i) Peripheral Clock Setup Functions

PLIB_OSC_PBClockDivisorGet Function

Gets the peripheral bus clock divisor.

File

plib_osc.h

C

```
OSC_PB_CLOCK_DIV_TYPE PLIB_OSC_PBClockDivisorGet(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS
peripheralBusNumber);
```

Returns

One of the possible values of PB clock divisor of type OSC_PB_CLOCK_DIV_TYPE.

Description

This function returns the peripheral bus clock divisor.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockDivisor in your application to determine whether this feature is available.

Actual Divisor value will be returned, NOT the 'PBDIV' field value. Refer the data sheet of the device for the detail.

Preconditions

None.

Example

```
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv;

peripheralBusClkDiv = PLIB_OSC_PBClockDivisorGet(OSC_ID_0,
                                                 OSC_PERIPHERAL_BUS_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

peripheralBusNumber	Identifies the desired peripheral bus

Function

OSC_PB_CLOCK_DIV_TYPE PLIB_OSC_PBClockDivisorGet (OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBClockDivisorIsReady Function

Checks whether the peripheral bus clock divisor is ready to be written.

File

plib_osc.h

C

```
bool PLIB_OSC_PBClockDivisorIsReady(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

- true - The peripheral bus clock divisor can be written
- false - The peripheral bus clock divisor cannot be written

Description

This function checks whether the peripheral bus clock divisor is ready to be written.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockReady in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_PBClockDivisorIsReady(OSC_ID_0, OSC_PERIPHERAL_BUS_1))
{
    PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1,
                                0x01);
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

bool PLIB_OSC_PBClockDivisorIsReady(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBClockDivisorSet Function

Sets the peripheral bus clock divisor to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_PBClockDivisorSet(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber,
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv);
```

Returns

None.

Description

This function sets the peripheral bus clock divisor to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockDivisor in your application to determine whether this feature is available.

Use PB Divider value directly for the parameter 'peripheralBusClkDiv', and NOT the value of the 'PBDIV' field. Use of the PB Divider value is not supported by the selected device, and therefore, library behavior is undefined. Refer the the specific device data sheet for information.

Preconditions

Peripheral bus clock divisor should be ready to be written.

Example

```
PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1, 0x01);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus
peripheralBusClkDiv	One of the possible values of PB clock divisor of type OSC_PB_CLOCK_DIV_TYPE

Function

void PLIB_OSC_PBClockDivisorSet(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber,
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv)

PLIB_OSC_PBOutputClockDisable Function

Disables the peripheral bus output clock.

File

plib_osc.h

C

```
void PLIB_OSC_PBOutputClockDisable(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

None

Description

This function disables the peripheral bus output clock.

Remarks

The clock for peripheral bus 1 cannot be turned off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PBOutputClockDisable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

void PLIB_OSC_PBOutputClockDisable(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBOutputClockEnable Function

Enables the peripheral bus output clock

File

plib_osc.h

C

```
void PLIB_OSC_PBOutputClockEnable(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

None

Description

This function enables the peripheral bus output clock

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PBOutputClockEnable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

void PLIB_OSC_PBOutputClockEnable(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBOutputClockIsEnabled Function

Checks whether or not the peripheral bus clock output is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_PBOutputClockIsEnabled(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

- true - The peripheral bus clock output is enabled
- false - The peripheral bus clock output is disabled

Description

This function checks whether or not the peripheral bus clock output is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_PBOutputClockIsEnabled(OSC_ID_0, OSC_PERIPHERAL_BUS_2))
```

```
{
    PLIB_OSC_PBOutputClockDisable(OSC_ID_0, OSC_PERIPHERAL_BUS_2);
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

bool PLIB_OSC_PBOutputClockIsEnabled(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

j) Clock Functions

PLIB_OSC_ClockHasFailed Function

Returns 'true' if the clock fails.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockHasFailed(OSC_MODULE_ID index);
```

Returns

- true - The FSCM detected a clock failure
- false - The no clock failure has been detected

Description

This function returns 'true' if the clock fails. Monitors the Fail-Safe Clock Monitor (FSCM).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockFail in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool clockStatus;
clockStatus = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_ClockHasFailed (OSC_MODULE_ID index);

PLIB_OSC_ClockStart Function

Starts the specified clock source.

File

plib_osc.h

C

```
void PLIB_OSC_ClockStart(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

None.

Description

This function is used to start a specified clock source if it has already been stopped.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockStart(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock to be started

Function

void PLIB_OSC_ClockStart (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

PLIB_OSC_ClockStop Function

Stops the specified clock source.

File

plib_osc.h

C

```
void PLIB_OSC_ClockStop(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

None.

Description

This function is used to stop a specified clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockStop(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock to be stopped

Function

void PLIB_OSC_ClockStop (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

PLIB_OSC_ClockStopStatus Function

returns the status of clock stop bit for the specified clock source.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockStopStatus(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

Status of the clock stop bit for a specified clock source.

Description

This function is used to get the status of clock stop bit for the specified clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool PoscStat = PLIB_OSC_ClockStart(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock source whose status to be checked

Function

bool PLIB_OSC_ClockStopStatus (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

k) Feature Existence Functions

PLIB_OSC_ExistsClockFail Function

Identifies whether the ClockFail feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockFail(OSC_MODULE_ID index);
```

Returns

- true - If the ClockFail feature is supported on the device
- false - If the ClockFail feature is not supported on the device

Description

This function identifies whether the ClockFail feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockHasFailed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockFail(OSC_MODULE_ID index)

PLIB_OSC_ExistsFRCDivisor Function

Identifies whether the FRCDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsFRCDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the FRCDivisor feature is supported on the device
- false - If the FRCDivisor feature is not supported on the device

Description

This function identifies whether the FRCDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_FRCDivisorSelect
- PLIB_OSC_FRCDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsFRCDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsFRCTuning Function

Identifies whether the FRCTuning feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsFRCTuning(OSC_MODULE_ID index);
```

Returns

- true - If the FRCTuning feature is supported on the device
- false - If the FRCTuning feature is not supported on the device

Description

This function identifies whether the FRCTuning feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_FRCTuningSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsFRCTuning(OSC_MODULE_ID index)

PLIB_OSC_ExistsOnWaitAction Function

Identifies whether the OnWaitAction feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOnWaitAction(OSC_MODULE_ID index);
```

Returns

- true - If the OnWaitAction feature is supported on the device
- false - If the OnWaitAction feature is not supported on the device

Description

This function identifies whether the OnWaitAction feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_OnWaitActionSet
- PLIB_OSC_OnWaitActionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOnWaitAction(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscCurrentGet Function

Identifies whether the OscCurrentGet feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscCurrentGet(OSC_MODULE_ID index);
```

Returns

- true - If the OscCurrentGet feature is supported on the device
- false - If the OscCurrentGet feature is not supported on the device

Description

This function identifies whether the OscCurrentGet feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_CurrentSysClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscCurrentGet(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscSelect Function

Identifies whether the OscSelect feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscSelect(OSC_MODULE_ID index);
```

Returns

- true - If the OscSelect feature is supported on the device
- false - If the OscSelect feature is not supported on the device

Description

This function identifies whether the OscSelect feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscSelect(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscSwitchInit Function

Identifies whether the OscSwitchInit feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscSwitchInit(OSC_MODULE_ID index);
```

Returns

- true - If the OscSwitchInit feature is supported on the device
- false - If the OscSwitchInit feature is not supported on the device

Description

This function identifies whether the OscSwitchInit feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockSwitchingAbort
- PLIB_OSC_ClockSwitchingIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscSwitchInit(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockDivisor Function

Identifies whether the PBClockDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockDivisor feature is supported on the device
- false - If the PBClockDivisor feature is not supported on the device

Description

This function identifies whether the PBClockDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBClockDivisorGet
- PLIB_OSC_PBClockDivisorSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockOutputEnable Function

Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockOutputEnable(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockOutputEnable feature is supported on the device
- false - If the PBClockOutputEnable feature is not supported on the device

Description

This function identifies whether the PBClockOutputEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBOutputClockEnable
- PLIB_OSC_PBOutputClockDisable
- PLIB_OSC_PBOutputClockIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockOutputEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockReady Function

Identifies whether the PBClockReady feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockReady(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockReady feature is supported on the device
- false - If the PBClockReady feature is not supported on the device

Description

This function identifies whether the PBClockReady feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBClockDivisorIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockReady(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLClockLock Function

Identifies whether the PLLClockLock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLClockLock(OSC_MODULE_ID index);
```

Returns

- true - If the PLLClockLock feature is supported on the device
- false - If the PLLClockLock feature is not supported on the device

Description

This function identifies whether the PLLClockLock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLClockLock
- PLIB_OSC_PLLClockUnlock
- PLIB_OSC_PLLClockIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLClockLock(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLLockStatus Function

Identifies whether the PLLLockStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLLockStatus(OSC_MODULE_ID index);
```

Returns

- true - If the PLLLockStatus feature is supported on the device
- false - If the PLLLockStatus feature is not supported on the device

Description

This function identifies whether the PLLLockStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLLockStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscBaseClock Function

Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscBaseClock(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscBaseClock feature is supported on the device
- false - If the ReferenceOscBaseClock feature is not supported on the device

Description

This function identifies whether the ReferenceOscBaseClock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscBaseClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscBaseClock(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscChange Function

Identifies whether the ReferenceOscChange feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscChange(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscChange feature is supported on the device
- false - If the ReferenceOscChange feature is not supported on the device

Description

This function identifies whether the ReferenceOscChange feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscSwitchIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscChange(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscChangeActive Function

Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscChangeActive(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscChangeActive feature is supported on the device
- false - If the ReferenceOscChangeActive feature is not supported on the device

Description

This function identifies whether the ReferenceOscChangeActive feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscSourceChangeIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscChangeActive(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscDivisor Function

Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscDivisor feature is supported on the device
- false - If the ReferenceOscDivisor feature is not supported on the device

Description

This function identifies whether the ReferenceOscDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscDivisorValueSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscEnable Function

Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscEnable feature is supported on the device
- false - If the ReferenceOscEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOscEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscEnable
- PLIB_OSC_ReferenceOscDisable
- PLIB_OSC_ReferenceOscIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscStopInIdleEnable Function

Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscStopInIdleEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscStopInIdleEnable feature is supported on the device
- false - If the ReferenceOscStopInIdleEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOscStopInIdleEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscStopInIdleEnable
- PLIB_OSC_ReferenceOscStopInIdleDisable
- PLIB_OSC_ReferenceOscStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscStopInIdleEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscStopInSleep Function

Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscStopInSleep(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscStopInSleep feature is supported on the device
- false - If the ReferenceOscStopInSleep feature is not supported on the device

Description

This function identifies whether the ReferenceOscStopInSleep feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscStopInSleepEnable
- PLIB_OSC_ReferenceOscStopInSleepDisable
- PLIB_OSC_ReferenceOscStopInSleepIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscStopInSleep(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscTrim Function

Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscTrim(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscTrim feature is supported on the device
- false - If the ReferenceOscTrim feature is not supported on the device

Description

This function identifies whether the ReferenceOscTrim feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscTrimSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscTrim(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOutputEnable Function

Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOutputEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOutputEnable feature is supported on the device
- false - If the ReferenceOutputEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOutputEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOutputEnable
- PLIB_OSC_ReferenceOutputDisable
- PLIB_OSC_ReferenceOutputIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOutputEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsSecondaryEnable Function

Identifies whether the SecondaryEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSecondaryEnable(OSC_MODULE_ID index);
```

Returns

- true - If the SecondaryEnable feature is supported on the device
- false - If the SecondaryEnable feature is not supported on the device

Description

This function identifies whether the SecondaryEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SecondaryEnable
- PLIB_OSC_SecondaryDisable
- PLIB_OSC_SecondaryIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSecondaryEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsSecondaryReady Function

Identifies whether the SecondaryReady feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSecondaryReady(OSC_MODULE_ID index);
```

Returns

- true - If the SecondaryReady feature is supported on the device
- false - If the SecondaryReady feature is not supported on the device

Description

This function identifies whether the SecondaryReady feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SecondaryIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSecondaryReady(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLFrequencyRange Function

Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - If the PLLFrequencyRange feature is supported on the device
- false - If the PLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the PLLFrequencyRange feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLFrequencyRangeSet
- PLIB_OSC_SysPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLInputClockSource Function

Identifies whether the PLLInputClockSource feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLInputClockSource(OSC_MODULE_ID index);
```

Returns

- true - If the PLLInputClockSource feature is supported on the device
- false - If the PLLInputClockSource feature is not supported on the device

Description

This function identifies whether the PLLInputClockSource feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLInputClockSourceSet
- PLIB_OSC_SysPLLInputClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLInputClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLInputDivisor Function

Identifies whether the PLLInputDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PLLInputDivisor feature is supported on the device
- false - If the PLLInputDivisor feature is not supported on the device

Description

This function identifies whether the PLLInputDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLInputDivisorSet
- PLIB_OSC_SysPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLMultiplier Function

Identifies whether the PLLMultiplier feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - If the PLLMultiplier feature is supported on the device
- false - If the PLLMultiplier feature is not supported on the device

Description

This function identifies whether the PLLMultiplier feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLMultiplierSelect
- PLIB_OSC_SysPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLOutputDivisor Function

Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PLLOutputDivisor feature is supported on the device
- false - If the PLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the PLLOutputDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLOutputDivisorSet
- PLIB_OSC_SysPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLOutputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsUsbClockSource Function

Identifies whether the UsbClockSource feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUsbClockSource(OSC_MODULE_ID index);
```

Returns

- true - If the UsbClockSource feature is supported on the device
- false - If the UsbClockSource feature is not supported on the device

Description

This function identifies whether the UsbClockSource feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UsbClockSourceSelect
- PLIB_OSC_UsbClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUsbClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockReadyStatus Function

Identifies whether the ClockReadyStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockReadyStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockReadyStatus feature is supported on the device
- false - The ClockReadyStatus feature is not supported on the device

Description

This function identifies whether the ClockReadyStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockReadyStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockSlewingStatus Function

Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockSlewingStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockSlewingStatus feature is supported on the device
- false - The ClockSlewingStatus feature is not supported on the device

Description

This function identifies whether the ClockSlewingStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockSlewingIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockSlewingStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsSleepToStartupClock Function

Identifies whether the SleepToStartupClock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSleepToStartupClock(OSC_MODULE_ID index);
```

Returns

- true - The SleepToStartupClock feature is supported on the device
- false - The SleepToStartupClock feature is not supported on the device

Description

This function identifies whether the SleepToStartupClock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SleepToStartupClockSelect
- PLIB_OSC_SleepToStartupClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSleepToStartupClock(OSC_MODULE_ID index)

PLIB_OSC_ExistsSlewDivisorStepControl Function

Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSlewDivisorStepControl(OSC_MODULE_ID index);
```

Returns

- true - The SlewDivisorStepControl feature is supported on the device
- false - The SlewDivisorStepControl feature is not supported on the device

Description

This function identifies whether the SlewDivisorStepControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SlewDivisorStepSelect
- PLIB_OSC_SlewDivisorStepGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSlewDivisorStepControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsSlewEnableControl Function

Identifies whether the SlewEnableControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSlewEnableControl(OSC_MODULE_ID index);
```

Returns

- true - The SlewEnableControl feature is supported on the device
- false - The SlewEnableControl feature is not supported on the device

Description

This function identifies whether the SlewEnableControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SlewEnable
- PLIB_OSC_SlewDisable
- PLIB_OSC_SlewIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSlewEnableControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsSystemClockDivisorControl Function

Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSystemClockDivisorControl(OSC_MODULE_ID index);
```

Returns

- true - The SystemClockDivisorControl feature is supported on the device
- false - The SystemClockDivisorControl feature is not supported on the device

Description

This function identifies whether the SystemClockDivisorControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SystemClockDivisorSelect
- PLIB_OSC_SystemClockDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSystemClockDivisorControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLClockOut Function

Identifies whether the BTPLLClockOut feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLClockOut(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLClockOut feature is supported on the device
- false - The BTPLLClockOut feature is not supported on the device

Description

This function identifies whether the BTPLLClockOut feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLClockOutEnable
- PLIB_OSC_BTPLLClockOutDisable
- PLIB_OSC_BTPLLClockOutStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLClockOut(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLFrequencyRange Function

Identifies whether the BTPLLFrequencyRange feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLFrequencyRange feature is supported on the device
- false - The BTPLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the BTPLLFrequencyRange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLFrequencyRangeSet
- PLIB_OSC_BTPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLInputClockSource Function

Identifies whether the BTPLLInputClockSource feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLInputClockSource(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLInputClockSource feature is supported on the device
- false - The BTPLLInputClockSource feature is not supported on the device

Description

This function identifies whether the BTPLLInputClockSource feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLInputClockSourceSet
- PLIB_OSC_BTPLLInputClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLInputClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLInputDivisor Function

Identifies whether the BTPLLInputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLInputDivisor feature is supported on the device
- false - The BTPLLInputDivisor feature is not supported on the device

Description

This function identifies whether the BTPLLInputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLInputDivisorSet
- PLIB_OSC_BTPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLMultiplier Function

Identifies whether the BTPLLMultiplier feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLMultiplier feature is supported on the device
- false - The BTPLLMultiplier feature is not supported on the device

Description

This function identifies whether the BTPLLMultiplier feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLMultiplierSelect
- PLIB_OSC_BTPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLOutputDivisor Function

Identifies whether the BTPLLOutputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLOutputDivisor feature is supported on the device
- false - The BTPLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the BTPLLOutputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLOutputDivisorSet
- PLIB_OSC_BTPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLOutputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockDiagStatus Function

Identifies whether the ClockDiagStatus feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockDiagStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockDiagStatus feature is supported on the device
- false - The ClockDiagStatus feature is not supported on the device

Description

This function identifies whether the ClockDiagStatus feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockStop
- PLIB_OSC_ClockStart
- PLIB_OSC_ClockStopStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockDiagStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsDreamModeControl Function

Identifies whether the DreamModeControl feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsDreamModeControl(OSC_MODULE_ID index);
```

Returns

- true - The DreamModeControl feature is supported on the device
- false - The DreamModeControl feature is not supported on the device

Description

This function identifies whether the DreamModeControl feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_DreamModeEnable
- PLIB_OSC_DreamModeDisable

- PLIB_OSC_DreamModeStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsDreamModeControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsForceLock Function

Identifies whether the ForceLock feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsForceLock(OSC_MODULE_ID index);
```

Returns

- true - The ForceLock feature is supported on the device
- false - The ForceLock feature is not supported on the device

Description

This function identifies whether the ForceLock feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ForceSPLLLockEnable
- PLIB_OSC_ForceSPLLLockDisable
- PLIB_OSC_ForceSPLLLockStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsForceLock(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLBypass Function

Identifies whether the SPLLBypass feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLBypass(OSC_MODULE_ID index);
```

Returns

- true - The PLLBypass feature is supported on the device

- false - The PLLBypass feature is not supported on the device

Description

This function identifies whether the PLLBypass feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLBypassEnable
- PLIB_OSC_PLLBypassDisable
- PLIB_OSC_PLLBypassStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLBypass(OSC_MODULE_ID index)

PLIB_OSC_ExistsResetPLL Function

Identifies whether the ResetPLL feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsResetPLL(OSC_MODULE_ID index);
```

Returns

- true - The ResetPLL feature is supported on the device
- false - The ResetPLL feature is not supported on the device

Description

This function identifies whether the ResetPLL feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ResetPLLAssert
- PLIB_OSC_ResetPLLDeassert
- PLIB_OSC_ResetPLLStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsResetPLL(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLFrequencyRange Function

Identifies whether the UPLLFrequencyRange feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - The UPLLFrequencyRange feature is supported on the device
- false - The UPLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the UPLLFrequencyRange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLFrequencyRangeSet
- PLIB_OSC_UPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLInputDivisor Function

Identifies whether the UPLLInputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The UPLLInputDivisor feature is supported on the device
- false - The UPLLInputDivisor feature is not supported on the device

Description

This function identifies whether the UPLLInputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLInputDivisorSet
- PLIB_OSC_UPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLMultiplier Function

Identifies whether the UPLLMultiplier feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - The UPLLMultiplier feature is supported on the device
- false - The UPLLMultiplier feature is not supported on the device

Description

This function identifies whether the UPLLMultiplier feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLMultiplierSelect
- PLIB_OSC_UPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLOutputDivisor Function

Identifies whether the UPLLOutputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The UPLLOutputDivisor feature is supported on the device
- false - The UPLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the UPLLOutputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLOutputDivisorSet
- PLIB_OSC_UPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLOutputDivisor(OSC_MODULE_ID index)

I) Data Types and Constants

OSC_PB_CLOCK_DIV_TYPE Macro

Type of the oscillator PB Clock divisor value.

File

plib_osc_help.h

C

```
#define OSC_PB_CLOCK_DIV_TYPE SFR_TYPE
```

Description

Oscillator Peripheral Bus Clock Divisor Value Type

This macro defines the type of the oscillator PB Clock divisor value.

Remarks

None.

OSC_REF_DIVISOR_TYPE Macro

Reference oscillator divisor type.

File

plib_osc_help.h

C

```
#define OSC_REF_DIVISOR_TYPE SFR_TYPE
```

Description

Reference oscillator divisor type

This macro defines the reference oscillator divisor type. Please refer to the specific device data sheet to determine the possible values of this type.

Remarks

None.

OSC_REFERENCE_MAX_DIV Macro

Defines the reference clock output divisor maximum value.

File

plib_osc_help.h

C

```
#define OSC_REFERENCE_MAX_DIV 65534
```

Description

Reference clock output divisor maximum

This macro defines the reference clock output divisor maximum value.

Remarks

None.

OSC_SYSPLL_MULTIPLIER_TYPE Macro

Type of the oscillator system PLL multiplier value.

File

plib_osc_help.h

C

```
#define OSC_SYSPLL_MULTIPLIER_TYPE SFR_TYPE
```

Description

Oscillator System PLL Multiplier Value Type

This macro defines the type of the oscillator system PLL multiplier value.

Remarks

None.

OSC_FRC_DIV Enumeration

Lists the possible Fast RC (FRC) Oscillator divider values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_FRC_DIV_1,
  OSC_FRC_DIV_2,
  OSC_FRC_DIV_4,
  OSC_FRC_DIV_8,
  OSC_FRC_DIV_16,
  OSC_FRC_DIV_32,
  OSC_FRC_DIV_64,
  OSC_FRC_DIV_256
} OSC_FRC_DIV;
```

Members

Members	Description
OSC_FRC_DIV_1	Fast RC Oscillator output Divide by 1
OSC_FRC_DIV_2	Fast RC Oscillator output Divide by 2
OSC_FRC_DIV_4	Fast RC Oscillator output Divide by 4
OSC_FRC_DIV_8	Fast RC Oscillator output Divide by 8
OSC_FRC_DIV_16	Fast RC Oscillator output Divide by 16
OSC_FRC_DIV_32	Fast RC Oscillator output Divide by 32
OSC_FRC_DIV_64	Fast RC Oscillator output Divide by 64
OSC_FRC_DIV_256	Fast RC Oscillator output Divide by 256

Description

FRC divider.

This enumeration lists the possible FRC Oscillator divider values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_MODULE_ID Enumeration

Possible instances of the OSC module.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_ID_0
} OSC_MODULE_ID;
```

Members

Members	Description
OSC_ID_0	first instance of the OSC

Description

OSC module ID

This data type defines the possible instances of the Oscillator module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_OPERATION_ON_WAIT Enumeration

Lists the possible base clock values for the reference oscillator.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_REF_BASECLOCK_SYSCLK,
  OSC_REF_BASECLOCK_PBCLK,
  OSC_REF_BASECLOCK_PRIMARY,
  OSC_REF_BASECLOCK_FRC,
  OSC_REF_BASECLOCK_LPRC,
  OSC_REF_BASECLOCK_SOSC,
  OSC_REF_BASECLOCK_USBCLK,
  OSC_REF_BASECLOCK_SYSPLLOUT,
  OSC_ON_WAIT_IDLE,
  OSC_ON_WAIT_SLEEP
} OSC_OPERATION_ON_WAIT;
```

Members

Members	Description
OSC_REF_BASECLOCK_SYSCLK	The base clock for reference clock is System Clock
OSC_REF_BASECLOCK_PBCLK	The base clock for reference clock is Peripheral Clock
OSC_REF_BASECLOCK_PRIMARY	The base clock for reference clock is Primary Oscillator
OSC_REF_BASECLOCK_FRC	The base clock for reference clock is FRC Oscillator
OSC_REF_BASECLOCK_LPRC	The base clock for reference clock is internal Low-Power RC
OSC_REF_BASECLOCK_SOSC	The base clock for reference clock is Secondary Oscillator
OSC_REF_BASECLOCK_USBCLK	The base clock for reference clock is USB Clock
OSC_REF_BASECLOCK_SYSPLLOUT	The base clock for reference clock is System PLL output
OSC_ON_WAIT_IDLE	Idle on Wait Instruction
OSC_ON_WAIT_SLEEP	Idle on Sleep Instruction

Description

Base clock for reference oscillator.

This enumeration lists the possible base clock values for the reference oscillator.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_PERIPHERAL_BUS Enumeration

Lists the possible Peripheral buses.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_PERIPHERAL_BUS_1,
  OSC_PERIPHERAL_BUS_2,
  OSC_PERIPHERAL_BUS_3,
  OSC_PERIPHERAL_BUS_4,
  OSC_PERIPHERAL_BUS_5,
  OSC_PERIPHERAL_BUS_6,
  OSC_PERIPHERAL_BUS_7,
  OSC_PERIPHERAL_BUS_8
} OSC_PERIPHERAL_BUS;
```

Members

Members	Description
OSC_PERIPHERAL_BUS_1	Peripheral bus 1
OSC_PERIPHERAL_BUS_2	Peripheral bus 2
OSC_PERIPHERAL_BUS_3	Peripheral bus 3
OSC_PERIPHERAL_BUS_4	Peripheral bus 4
OSC_PERIPHERAL_BUS_5	Peripheral bus 5
OSC_PERIPHERAL_BUS_6	Peripheral bus 6
OSC_PERIPHERAL_BUS_7	Peripheral bus 7
OSC_PERIPHERAL_BUS_8	Peripheral bus 8

Description

Peripheral Buses

This enumeration lists the possible peripheral buses available in the device

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_PLL_SELECT Enumeration

Lists the PLLs in the Oscillator module.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_PLL_SYSTEM,
  OSC_PLL_USB
} OSC_PLL_SELECT;
```

Members

Members	Description
OSC_PLL_SYSTEM	Select system PLL
OSC_PLL_USB	Select USB PLL

Description

Oscillator PLL selection.

This enumeration lists the available PLLs in the Oscillator module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_REFERENCE Enumeration

Lists the possible reference oscillator.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_REFERENCE_1,
  OSC_REFERENCE_2,
  OSC_REFERENCE_3,
  OSC_REFERENCE_4
} OSC_REFERENCE;
```

Members

Members	Description
OSC_REFERENCE_1	Reference Oscillator 1
OSC_REFERENCE_2	Reference Oscillator 2
OSC_REFERENCE_3	Reference Oscillator 3
OSC_REFERENCE_4	Reference Oscillator 4

Description

Reference Oscillators

This enumeration lists the possible reference oscillator instances available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYS_TYPE Enumeration

Lists the possible oscillator type values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_FRC,
  OSC_FRC_WITH_PLL,
  OSC_PRIMARY,
  OSC_PRIMARY_WITH_PLL,
  OSC_SECONDARY,
  OSC_LPRC,
  OSC_FRC_DIV_BY_16,
  OSC_FRC_BY_FRCDIV
} OSC_SYS_TYPE;
```

Members

Members	Description
OSC_FRC	Fast RC Oscillator. This includes the 8 MHz oscillator
OSC_FRC_WITH_PLL	Fast RC Oscillator with PLL
OSC_PRIMARY	Primary Oscillator
OSC_PRIMARY_WITH_PLL	Fast RC Oscillator with PLL and PostScaler. Includes the 8MHz oscillator
OSC_SECONDARY	Secondary Oscillator
OSC_LPRC	Low Power RC Oscillator
OSC_FRC_DIV_BY_16	Low-Power Fast RC Oscillator with PostScaler
OSC_FRC_BY_FRCDIV	Fast RC Oscillator divided by FRCDIV bits

Description

Oscillator type.

This enumeration lists the possible oscillator type values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_FREQ_RANGE Enumeration

Lists the possible PLL frequency range.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SYSPLL_FREQ_RANGE_BYPASS,
  OSC_SYSPLL_FREQ_RANGE_5M_TO_10M,
  OSC_SYSPLL_FREQ_RANGE_8M_TO_16M,
  OSC_SYSPLL_FREQ_RANGE_13M_TO_26M,
  OSC_SYSPLL_FREQ_RANGE_21M_TO_42M,
  OSC_SYSPLL_FREQ_RANGE_34M_TO_68M
} OSC_SYSPLL_FREQ_RANGE;
```

Members

Members	Description
OSC_SYSPLL_FREQ_RANGE_BYPASS	PLL freq range bypass
OSC_SYSPLL_FREQ_RANGE_5M_TO_10M	PLL frequency range is 5 MHz to 10 MHz
OSC_SYSPLL_FREQ_RANGE_8M_TO_16M	PLL frequency range is 8 MHz to 16 MHz
OSC_SYSPLL_FREQ_RANGE_13M_TO_26M	PLL frequency range is 13 MHz to 26 MHz
OSC_SYSPLL_FREQ_RANGE_21M_TO_42M	PLL frequency range is 21 MHz to 42 MHz
OSC_SYSPLL_FREQ_RANGE_34M_TO_68M	PLL frequency range is 34 MHz to 68 MHz

Description

System PLL Frequency Range

This enumeration lists the possible frequency range for PLL module available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_IN_CLK_SOURCE Enumeration

Lists the possible input clock source for PLL module.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SYSPLL_IN_CLK_SOURCE_FRC,
  OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY
} OSC_SYSPLL_IN_CLK_SOURCE;
```

Members

Members	Description
OSC_SYSPLL_IN_CLK_SOURCE_FRC	FRC is input clock source for PLL module
OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY	Primary clock is the input clock source for PLL module

Description

System PLL Input Clock Source

This enumeration lists the possible input clock source for PLL module available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_OUT_DIV Enumeration

Lists the possible PLL output divider values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SYSPLL_OUT_DIV_1,
  OSC_SYSPLL_OUT_DIV_2,
  OSC_SYSPLL_OUT_DIV_4,
  OSC_SYSPLL_OUT_DIV_8,
  OSC_SYSPLL_OUT_DIV_16,
  OSC_SYSPLL_OUT_DIV_32,
  OSC_SYSPLL_OUT_DIV_64,
  OSC_SYSPLL_OUT_DIV_256
} OSC_SYSPLL_OUT_DIV;
```

Members

Members	Description
OSC_SYSPLL_OUT_DIV_1	Divide by 1
OSC_SYSPLL_OUT_DIV_2	Divide by 2
OSC_SYSPLL_OUT_DIV_4	Divide by 4
OSC_SYSPLL_OUT_DIV_8	Divide by 8
OSC_SYSPLL_OUT_DIV_16	Divide by 16
OSC_SYSPLL_OUT_DIV_32	Divide by 32
OSC_SYSPLL_OUT_DIV_64	Divide by 64
OSC_SYSPLL_OUT_DIV_256	Divide by 256

Description

PLL Output divider.

This enumeration lists the possible PLL output divider values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_USBCLOCK_SOURCE Enumeration

Lists the possible USB clock sources.

File

plib_osc_help.h

C

```
typedef enum {
  SYS_OSC_USBCLK_PRIMARY,
  SYS_OSC_USBCLK_FRC
} OSC_USBCLOCK_SOURCE;
```

Members

Members	Description
SYS_OSC_USBCLK_PRIMARY	USB clock source is primary oscillator
SYS_OSC_USBCLK_FRC	USB clock source isFRC oscillator

Description

USB clock sources.

This enumeration lists the the possible USB clock sources.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_CLOCK_ID Enumeration

Lists the clock sources for which ready status can be checked.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_CLOCK_FAST_RC,
  OSC_CLOCK_PRIMARY_OSC,
  OSC_CLOCK_SECONDARY_OSC,
  OSC_CLOCK_LOW_POWER_RC,
  OSC_CLOCK_SYSTEM_PLL
} OSC_CLOCK_ID;
```

Members

Members	Description
OSC_CLOCK_FAST_RC	Fast RC Oscillator
OSC_CLOCK_PRIMARY_OSC	Primary Oscillator
OSC_CLOCK_SECONDARY_OSC	Secondary Oscillator
OSC_CLOCK_LOW_POWER_RC	Low Power RC Oscillator
OSC_CLOCK_SYSTEM_PLL	System PLL

Description

OSC Clock ID

This enumeration lists all the possible clock sources for which the ready status can be checked.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

OSC_CLOCK_SLEW_TYPE Enumeration

Lists the possible type of clock slewing.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_CLOCK_SLEW_DOWNWARD,
  OSC_CLOCK_SLEW_UPWARD
} OSC_CLOCK_SLEW_TYPE;
```

Members

Members	Description
OSC_CLOCK_SLEW_DOWNWARD	Slewing to a lower clock frequency
OSC_CLOCK_SLEW_UPWARD	Slewing to a higher clock frequency

Description

OSC Clock Slew Rate

This enumeration lists the possible type of clock slewing.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

OSC_SLEEP_TO_STARTUP_CLK_TYPE Enumeration

Lists the possible clock sources for sleep to start-up period.

File

plib_osc_help.h

C

```
typedef enum {
```

```
    OSC_SLEEP_TO_STARTUP_CLK_FRC,
    OSC_SLEEP_TO_STARTUP_NO_ADDITIONAL_CLK
} OSC_SLEEP_TO_STARTUP_CLK_TYPE;
```

Members

Members	Description
OSC_SLEEP_TO_STARTUP_CLK_FRC	FRC will be used after wakeup from sleep until selected clock is ready
OSC_SLEEP_TO_STARTUP_NO_ADDITIONAL_CLK	No additional clock will be used after wakeup from sleep, selected clock will be used directly once it is ready

Description

Sleep to Speed Startup Clock

This enumeration lists the possible clock sources that can be used from the time when the device wakes from sleep until the actual clock source is ready to be used.

Remarks

This feature may not be available in all the devices. Check device specific header files for availability.

Files

Files

Name	Description
plib_osc.h	Defines the Oscillator (OSC) Peripheral Library interface.
plib_osc_help.h	Defines the Oscillator Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_osc.h

Defines the Oscillator (OSC) Peripheral Library interface.

Functions

	Name	Description
	PLIB_OSC_BTPLLClockOutDisable	Disables the Bluetooth PLL Clock Output.
	PLIB_OSC_BTPLLClockOutEnable	Enables the Bluetooth PLL clock Ouput.
	PLIB_OSC_BTPLLClockOutStatus	gets the status of the Bluetooth PLL clock Output.
	PLIB_OSC_BTPLLFrequencyRangeGet	Gets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLFrequencyRangeSet	Sets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceGet	Gets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceSet	Sets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputDivisorGet	Gets the input divisor for the Bluetooth PLL.
	PLIB_OSC_BTPLLInputDivisorSet	Sets the input divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_BTPLLMultiplierGet	Gets the Bluetooth PLL multiplier.
	PLIB_OSC_BTPLLMultiplierSelect	Sets the Bluetooth PLL multiplier to the specified value.
	PLIB_OSC_BTPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_BTPLLOutputDivisorSet	Sets the output divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.
	PLIB_OSC_ClockIsReady	Get the ready status of clock.
	PLIB_OSC_ClockSlewingIsActive	Returns the status of clock slewing.
	PLIB_OSC_ClockStart	Starts the specified clock source.
	PLIB_OSC_ClockStop	Stops the specified clock source.
	PLIB_OSC_ClockStopStatus	returns the status of clock stop bit for the specified clock source.
	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.
	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
	PLIB_OSC_DreamModeDisable	Disables the dream mode.

	PLIB_OSC_DreamModeEnable	Enables the dream mode.
	PLIB_OSC_DreamModeStatus	gets the status of the dream mode.
	PLIB_OSC_ExistsBTPLLClockOut	Identifies whether the BTPLLClockOut feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLFrequencyRange	Identifies whether the BTPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputClockSource	Identifies whether the BTPLLInputClockSource feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputDivisor	Identifies whether the BTPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLMultiplier	Identifies whether the BTPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLOutputDivisor	Identifies whether the BTPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsClockDiagStatus	Identifies whether the ClockDiagStatus feature exists on the OSC module
	PLIB_OSC_ExistsClockFail	Identifies whether the ClockFail feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockReadyStatus	Identifies whether the ClockReadyStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockSlewingStatus	Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsDreamModeControl	Identifies whether the DreamModeControl feature exists on the OSC module
	PLIB_OSC_ExistsForceLock	Identifies whether the ForceLock feature exists on the OSC module
	PLIB_OSC_ExistsFRCDivisor	Identifies whether the FRCDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCTuning	Identifies whether the FRCTuning feature exists on the Oscillator module.
	PLIB_OSC_ExistsOnWaitAction	Identifies whether the OnWaitAction feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscCurrentGet	Identifies whether the OscCurrentGet feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSelect	Identifies whether the OscSelect feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSwitchInit	Identifies whether the OscSwitchInit feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockDivisor	Identifies whether the PBClockDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockOutputEnable	Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockReady	Identifies whether the PBClockReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLBypass	Identifies whether the SPLLBypass feature exists on the OSC module
	PLIB_OSC_ExistsPLLClockLock	Identifies whether the PLLClockLock feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLLockStatus	Identifies whether the PLLLockStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscBaseClock	Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChange	Identifies whether the ReferenceOscChange feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChangeActive	Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscDivisor	Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscEnable	Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInIdleEnable	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInSleep	Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscTrim	Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOutputEnable	Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsResetPLL	Identifies whether the ResetPLL feature exists on the OSC module
	PLIB_OSC_ExistsSecondaryEnable	Identifies whether the SecondaryEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryReady	Identifies whether the SecondaryReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsSleepToStartupClock	Identifies whether the SleepToStartupClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewDivisorStepControl	Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewEnableControl	Identifies whether the SlewEnableControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLFrequencyRange	Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.

	PLIB_OSC_ExistsSysPLLInputClockSource	Identifies whether the PLLInputClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputDivisor	Identifies whether the PLLInputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLMultiplier	Identifies whether the PLLMultiplier feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLOutputDivisor	Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSystemClockDivisorControl	Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsUPLLFrequencyRange	Identifies whether the UPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsUPLLInputDivisor	Identifies whether the UPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUPLLMultiplier	Identifies whether the UPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsUPLLOutputDivisor	Identifies whether the UPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUsbClockSource	Identifies whether the UsbClockSource feature exists on the Oscillator module.
	PLIB_OSC_ForceSPLLLockDisable	Disables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockEnable	Enables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockStatus	gets the status of the force PLL Lock bit of the specified PLL.
	PLIB_OSC_FRCDivisorGet	Gets the FRC clock divisor.
	PLIB_OSC_FRCDivisorSelect	Sets the FRC clock divisor to the specified value.
	PLIB_OSC_FRCTuningSelect	Sets the FRC tuning value.
	PLIB_OSC_OnWaitActionGet	Gets the configured operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_OnWaitActionSet	Selects the operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_PBClockDivisorGet	Gets the peripheral bus clock divisor.
	PLIB_OSC_PBClockDivisorIsReady	Checks whether the peripheral bus clock divisor is ready to be written.
	PLIB_OSC_PBClockDivisorSet	Sets the peripheral bus clock divisor to the specified value.
	PLIB_OSC_PBOutputClockDisable	Disables the peripheral bus output clock.
	PLIB_OSC_PBOutputClockEnable	Enables the peripheral bus output clock
	PLIB_OSC_PBOutputClockIsEnabled	Checks whether or not the peripheral bus clock output is enabled.
	PLIB_OSC_PLLBypassDisable	Disables the PLL Bypass.
	PLIB_OSC_PLLBypassEnable	Enables the PLL Bypass.
	PLIB_OSC_PLLBypassStatus	gets the status of the PLL Bypass.
	PLIB_OSC_PLLClockIsLocked	Gets the lock status for the clock and PLL selections.
	PLIB_OSC_PLLClockLock	Locks the clock and PLL selections.
	PLIB_OSC_PLLClockUnlock	Unlocks the clock and PLL selections.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the selected PLL module is locked.
	PLIB_OSC_ReferenceOscBaseClockSelect	Sets the base clock for the reference oscillator.
	PLIB_OSC_ReferenceOscDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOscDivisorValueSet	Selects the reference oscillator divisor value.
	PLIB_OSC_ReferenceOscEnable	Enables the reference oscillator.
	PLIB_OSC_ReferenceOscIsEnabled	Gets the enable status of the reference oscillator output.
	PLIB_OSC_ReferenceOscSourceChangeIsActive	Returns 'true' if a reference oscillator source change request is active.
	PLIB_OSC_ReferenceOscStopInIdleDisable	Enables the reference oscillator in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleEnable	Configures the reference oscillator to stop operating in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_ReferenceOscStopInSleepDisable	Enables the reference oscillator in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepEnable	Configures the reference oscillator to stop operating in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepIsEnabled	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	PLIB_OSC_ReferenceOscSwitchIsComplete	Returns 'true' if the reference oscillator base clock switching is complete.
	PLIB_OSC_ReferenceOscTrimSet	Sets the reference oscillator divisor trim value.
	PLIB_OSC_ReferenceOutputDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOutputEnable	Enables the reference oscillator output.
	PLIB_OSC_ReferenceOutputIsEnabled	Returns 'true' if the reference oscillator output is enabled.
	PLIB_OSC_ResetPLLAssert	Asserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLDeassert	Deasserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLStatus	gets the status of the PLL reset bit for the specified PLL.
	PLIB_OSC_SecondaryDisable	Disables the Secondary Oscillator.
	PLIB_OSC_SecondaryEnable	Enables the Secondary Oscillator.

	PLIB_OSC_SecondaryIsEnabled	Returns 'true' if the Secondary Oscillator is enabled.
	PLIB_OSC_SecondaryIsReady	Returns 'true' if the Secondary Oscillator is ready.
	PLIB_OSC_SleepToStartupClockGet	Returns the clock used for the duration when the device wakes from sleep and the clock ready.
	PLIB_OSC_SleepToStartupClockSelect	Selects the clock duration for when the device wakes from sleep and the clock is ready.
	PLIB_OSC_SlewDisable	Disables the selected type of slewing.
	PLIB_OSC_SlewDivisorStepGet	Get the slew divisor maximum step.
	PLIB_OSC_SlewDivisorStepSelect	Selects division steps used while slewing.
	PLIB_OSC_SlewEnable	Enables the selected type of slewing.
	PLIB_OSC_SlewIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.
	PLIB_OSC_SysPLLFrequencyRangeGet	Gets the frequency range for the PLL module.
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceGet	Gets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.
	PLIB_OSC_SystemClockDivisorGet	Get the system clock divisor value.
	PLIB_OSC_SystemClockDivisorSelect	Selects system clock divisor.
	PLIB_OSC_UPLLFrequencyRangeGet	Gets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLFrequencyRangeSet	Sets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLInputDivisorGet	Gets the input divisor for the USB PLL.
	PLIB_OSC_UPLLInputDivisorSet	Sets the input divider for the USB PLL to the specified value.
	PLIB_OSC_UPLLMultiplierGet	Gets the USB PLL multiplier.
	PLIB_OSC_UPLLMultiplierSelect	Sets the USB PLL multiplier to the specified value.
	PLIB_OSC_UPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_UPLLOutputDivisorSet	Sets the output divider for the USB PLL to the specified value.
	PLIB_OSC_UsbClockSourceGet	Gets the USB module clock source.
	PLIB_OSC_UsbClockSourceSelect	Sets the USB module clock source.

Description

Oscillator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Oscillator (OSC) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Oscillator module.

File Name

plib_osc.h

Company

Microchip Technology Inc.

plib_osc_help.h

Defines the Oscillator Peripheral Library data types.

Enumerations

	Name	Description
	OSC_CLOCK_ID	Lists the clock sources for which ready status can be checked.
	OSC_CLOCK_SLEW_TYPE	Lists the possible type of clock slewing.
	OSC_FRC_DIV	Lists the possible Fast RC (FRC) Oscillator divider values.
	OSC_MODULE_ID	Possible instances of the OSC module.
	OSC_OPERATION_ON_WAIT	Lists the possible base clock values for the reference oscillator.

	OSC_PERIPHERAL_BUS	Lists the possible Peripheral buses.
	OSC_PLL_SELECT	Lists the PLLs in the Oscillator module.
	OSC_REFERENCE	Lists the possible reference oscillator.
	OSC_SLEEP_TO_STARTUP_CLK_TYPE	Lists the possible clock sources for sleep to start-up period.
	OSC_SYS_TYPE	Lists the possible oscillator type values.
	OSC_SYSPLL_FREQ_RANGE	Lists the possible PLL frequency range.
	OSC_SYSPLL_IN_CLK_SOURCE	Lists the possible input clock source for PLL module.
	OSC_SYSPLL_OUT_DIV	Lists the possible PLL output divider values.
	OSC_USBCLOCK_SOURCE	Lists the possible USB clock sources.

Macros

	Name	Description
	OSC_PB_CLOCK_DIV_TYPE	Type of the oscillator PB Clock divisor value.
	OSC_REF_DIVISOR_TYPE	Reference oscillator divisor type.
	OSC_REFERENCE_MAX_DIV	Defines the reference clock output divisor maximum value.
	OSC_SYSPLL_MULTIPLIER_TYPE	Type of the oscillator system PLL multiplier value.

Description

Oscillator Peripheral Library Interface Header

This header file contains the definitions of the data types and constants that make up the interface to the Oscillator Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Oscillator module.

File Name

plib_osc_help.h

Company

Microchip Technology Inc.

PMP Peripheral Library

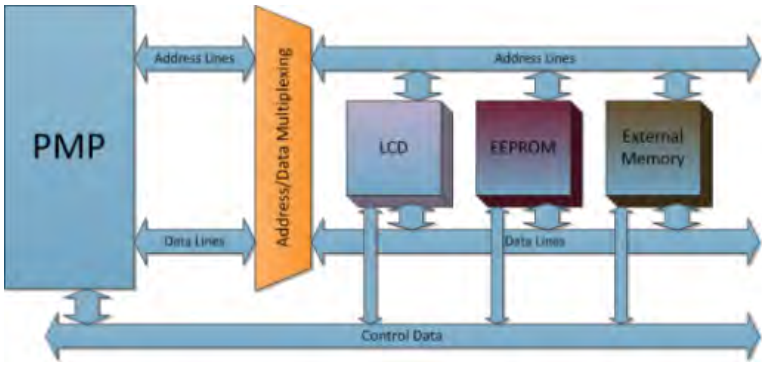
This section describes the Parallel Master Port (PMP) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The following diagram shows a block diagram of the PMP module and how it interacts with other peripherals.



Using the Library

This topic describes the basic architecture of the PMP Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_pmp.h`

The interface to the PMP Peripheral library is defined in the `plib_pmp.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the PMP Peripheral Library must include `peripheral.h`.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

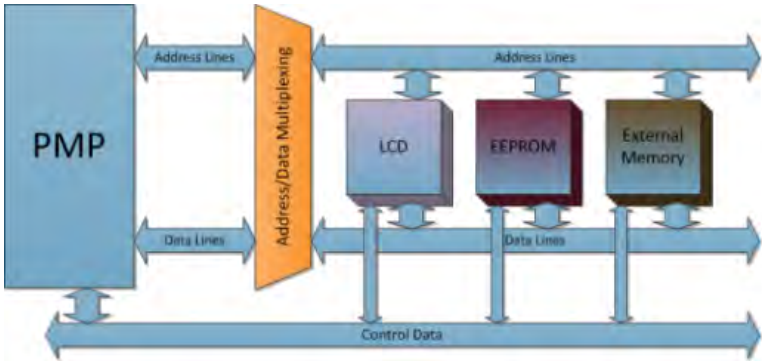
Hardware Abstraction Model

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.

PMP Hardware Abstraction Model Diagram



The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the PMP module

Library Interface Section	Description
General Initialization Functions	Provides configuration for: • Interrupt mode • Address Incrementing mode • Stop-in-Idle operation • Operating mode • Multiplexing mode • Chip Select mode
Wait States Initialization Functions	Provides configuration routines to tune the wait states of the PMP module.
Data Read and Write Functions	Provides functions for receiving the data from another module when operating in Master or Slave mode. Also provides routines for transmitting data from the module when operating in Master or Slave mode.
Address Handling	Provides functions to set the PMP module to point to the desired addresses.
Port Configuration	Provides functions for configuring the PMP pins either for PMP purpose or general purpose.
Polarity Configuration	Provides routines for configuring the PMP pins polarity either as active-high or active-low.
Error Status/Control Functions	Provides functions and status for error handling (either as a master or a slave).
General Status Functions	Provides status of the PMP module.

Note: The enhanced feature APIs are not available on all devices. Please refer to the specific device data sheet to determine the availability of these enhanced features.

How the Library Works

Provides information on how the library works.

Description

Before enabling the PMP module, the caller must initialize basic parameters such as wait states timings, and interrupt mode features (see the Initialization section).

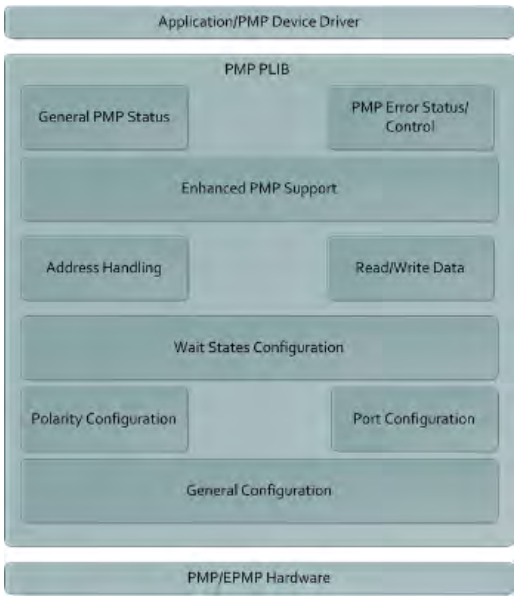
After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see the Operating as a Slave section). A slave device only transfers data on the bus when it has been addressed by a master. If the module is used to start a transfer, it is operating as a master. A master addresses a slave and controls the transfer of data by providing the clock (see the Operating as a Master section).

Some operations in the PMP peripheral library initiate actions on the PMP bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. The State Machine section describes the different states that can cause an interrupt and show what causes the transition from one state to another. The Handling Errors section describes the various errors that can occur and how to deal with them.

Usage Model

The following diagram shows the high level organization of the PMP Peripheral Library usage model. The items in the diagram correspond to the groupings in the Library Interface section.

PMP Library Usage Model



Initialization

This section describes the general initialization of the PMP module.

Description

The important configurations to look for are as follows.

Operating Mode

Depending on the device, the PMP module can function in Master modes 1 or 2, buffered/enhanced slave mode or the legacy slave mode. To set the PMP module in the appropriate mode, use PLIB_PMP_OperationModeSelect.

Multiplexing Mode

Depending on the application requirement or hardware arrangement, the data lines can be used to multiplex the addressing information. Various multiplexing modes supported are listed by PMP_MUX_MODE. Use PLIB_PMP_MultiplexModeSelect to select the appropriate multiplexing mode.

Chip Select Mode

As needed, the Chip Select lines can be made to function as Chip Select or as address lines. Use PLIB_PMP_ChipSelectFunctionSelect to select the required functionality of Chip Select lines.

Interrupt Mode

PMP generates interrupts based on the option selected for the interrupt mode. Interrupts can be enabled, disabled, generated at the end of a read or write cycle, or when other events occur. Use the PLIB_PMP_InterruptModeSelect to select these options.

Address Increment Mode

After every read/write cycle, the PMP module can be configured to automatically increment or decrement the address it is accessing. Use PLIB_PMP_AddressIncrementModeSelect to select this option.

To initialize the PMP module, perform the following sequence:

1. Select the desired initialization options using the General Initialization Functions (see the Library Interface section) to select the desired operation of the following features:
   - Operating mode
   - Multiplexing mode
   - Chip Select mode
   - Interrupt mode
   - Address Incrementing mode
   - Stop-in-Idle operation
2. Program the wait states of the PMP module. Refer to Wait States Initialization Functions in the Library Interface section for an example.
3. If operating in Master mode, set the desired address (8-bit in this case, the size may vary depending on the application) using PLIB_PMP_AddressSet.
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and error) PMP interrupts.

Note: Refer to the **"Interrupt Controller"** chapter in the specific device data sheet for details.

5. Enable the module for operation using PLIB_PMP_Enable.

Example

```
// Configure General PMP Options
// Select the PMP operation mode
PLIB_PMP_OperationModeSelect( PMP_ID_0, PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT  );
// Set the multiplexing to none. Address and data lines are not multiplexed.
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
// Select the function of chip-select lines
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
// Select the interrupt mode
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
// disable the auto increment/decrement of address after each read/write
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
// Enable stop in Idle mode.
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
// Set Desired Wait State Values
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_FOUR );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
// Set the appropriate address (MASTER only)
PLIB_PMP_AddressSet ( PMP_ID_0, 0x12 );

// Optional:  Clear and enable interrupts before enabling the PMP module.
// Enable the module
PLIB_PMP_Enable( PMP_ID_0 );
```

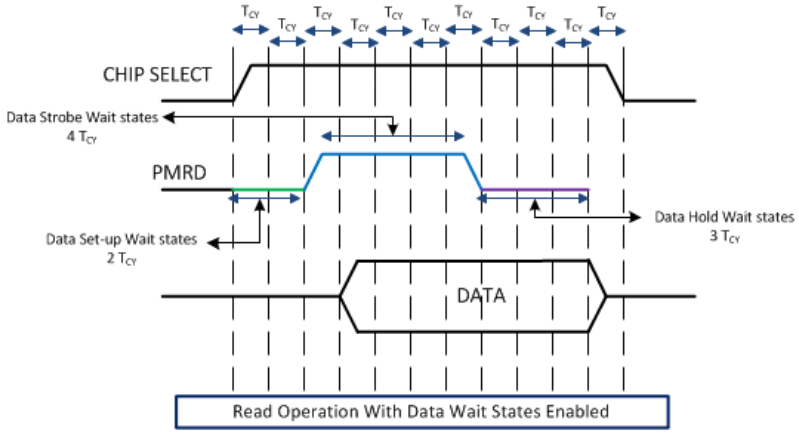
Note: Refer to the **"Interrupt Controller"** chapter in the specific device data sheet for details on how to clear and enable the PMP module interrupts.

Wait States Initialization

This section describes set up and initialization of the wait states for the PMP module.

Description

In Master mode, the user can control the duration of the read, write and address cycles by configuring the module Wait states. One Wait state period is equivalent to one peripheral bus clock cycles. The following figure shows an example of a Master Read operation using Wait states.



Wait States

Wait states can be added to the beginning, middle and end of any read or write cycle using the corresponding bits in the PMP Configuration. The wait states differ depending on which PMP mode you are in, but setting the wait states is the same per mode and device, as shown in the following example.

Initializing Wait States

The following sequence can be used to set up the wait states for the PMP Master mode.

Preconditions:

None.

Process:

1. Set the data hold time using PLIB_PMP_WaitStatesDataHoldSelect.
2. Set the strobe wait time using PLIB_PMP_WaitStatesStrobeSelect.
3. Set the data wait time using PLIB_PMP_WaitStatesDataSetUpSelect.

Example

```
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_ONE );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
```

Operating as a Master

This section describes how to set up the PMP module to operate as a master.

Description

The PMP module, while acting as a master, waits for its input or output buffers to be read or written. Once the buffers have been manipulated, the appropriate action takes place.

A PMP bus transfer always begins with reading or writing to the appropriate PMP input/output buffers. A read from a PMP input buffer performs a PMP read. A write to a PMP output buffer performs a PMP write. The address pins associated with the transfer will have the value that is inside the address register of the PMP module. These steps are summarized as follows.

Summary of Steps

- Set up the proper PMP address
- Send or read data bytes

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Sending a Data Byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module
- The proper control signals are configured for PMP operation
- The PMP module is configured for the desired Master mode
- The PMP module is enabled

Process:

1. Ensure that the PMP module is not busy by using PLIB_PMP_PortIsBusy.
2. Write the output data buffer using PLIB_PMP_MasterSend.

Example:

```
uint8_t data = 'a';
// Set the destination address to be written
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see if the PMP is busy, and then send the byte
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
  {
    // Send the data
    PLIB_PMP_MasterSend( PMP_ID_0, data );
  }
```

Receiving a Data byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module
- The proper control signal are configured for PMP operation

Process:

1. Ensure that the PMP module is not busy by using PLIB_PMP_PortIsBusy.
2. Write the output data buffer using PLIB_PMP_MasterReceive.

Example:

```
uint8_t data;
// Set the source address
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see the PMP is not busy, and then read the data
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
  {
    // receive the data
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
  }
```

Operating as a Slave

This section describes how to set up the PMP module to operate as a slave.

Description

The PMP module while acting as a slave, waits for read or write strobes to occur. Once the buffers have been manipulated, the appropriate action takes place.

Summary of Steps

- Check to see if data is available to be read
- Prepare output buffer for slave read

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Reading Available Data

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be set in the PMP module
- The proper control signals are initialized for PMP operation

Process:

1. Check to see if data is available using PLIB_PMP_InputBufferXIsFull.
2. Read the available data using PLIB_PMP_InputBufferXByteReceive.

Example:

```
uint8_t data;
// Check to see if data is available, and then receive the byte
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 1))
  {
    // Receive the data from buffer one.
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
  }
```

Preparing Output Buffer for Slave Write

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

The proper control signals are initialized for PMP operation.

Process:

1. Ensure that the output buffer to be written is empty by using PLIB_PMP_OutputBufferXIsEmpty.
2. Write the output data buffer using PLIB_PMP_OutputBufferXByteSend.

Example:

```
 uint8_t data;
// Check to see if output buffer is available, and then send the byte
if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1))
  {
    // Fill the output buffer-1 with the data to be sent
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
  }
```

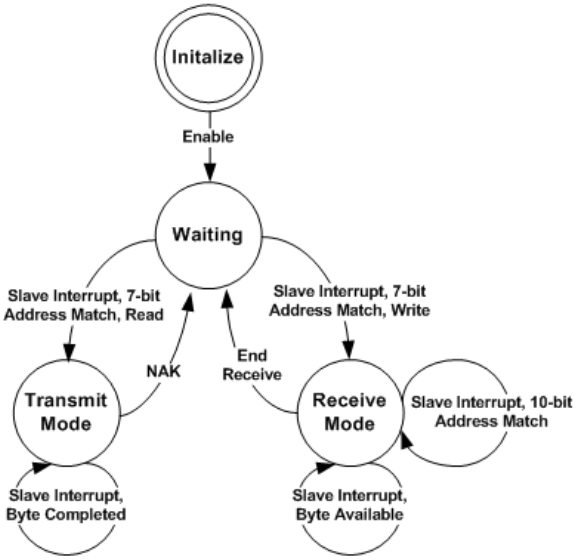
State Machine

The PMP state machine can be used in either a polled or an interrupt-driven manner. However, in both cases, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

Description

The PMP module has one basic interrupt that is triggered when data has been written or read. In Buffered Slave mode, this interrupt can be set to wait until an 'X' amount of buffers are written or read. This interrupt must be cleared in software, but can also be used in DMA transactions to trigger the next DMA transfer.

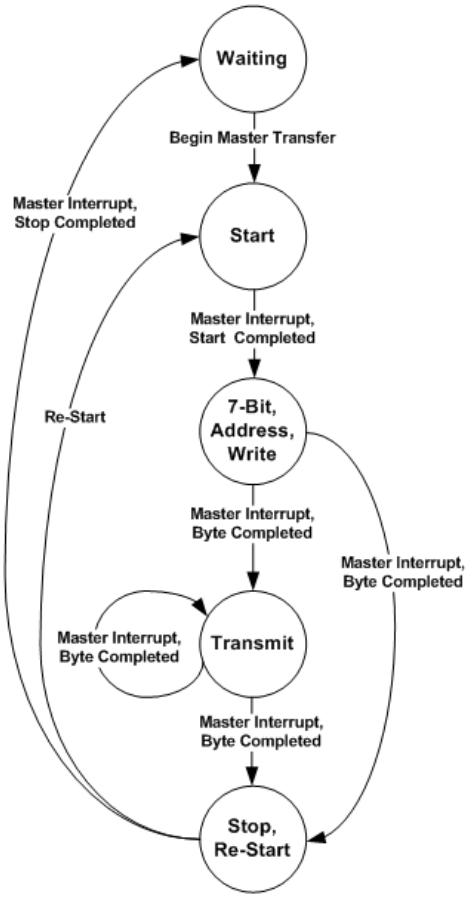
The PMP module will not start generating interrupts (setting the PMP interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described in subsequent sections. Software has to respond appropriately once the interrupt has occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described in the following sections for the state machine diagram to which they refer.



Slave Mode State Machine

Slave-Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if a read/write strobe occurs, the appropriate read/write occurs, which in turn sets the interrupt.



Master Mode State Machine

Master-Mode State Transitions

In Master mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if an input or output register is manipulated, the appropriate read/write happens, which in turn sets the interrupt. The PMP module can also be configured to have read/write strobes indicating whether a read/write has completed.

Handling Errors

There are two basic types of errors that can occur during various slave PMP operations:

- Input Buffer Overrun
- Output Buffer Underflow

This section provides information on handling these types of errors.

Description

Handling Input Buffer Overrun Errors

An input buffer overflow error occurs when the software is not reading or clearing the input buffers fast enough for the master PMP device attached to it. When this occurs, the master write is not allowed and the "input buffer overflow" status bit is set. This can be identified by calling PLIB_PMP_InputOverflowHasOccurred. Additional attempts to write to the input buffer will not be allowed until the overflow error is cleared by calling PLIB_PMP_InputOverflowClear. This type of error should be either avoided by having proper communication techniques with the PMP Master device or checked using PLIB_PMP_InputBuffersAreFull, which informs the controller that no buffers have data in them (overflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

Handling Output Buffer Underflow Errors

An output buffer underflow error occurs when the software is not writing to the output buffers fast enough for the master PMP device attached to it. When this occurs, the master read is not allowed and the "output buffer underflow" status bit is set. This can be identified by calling PLIB_PMP_OutputUnderflowHasOccurred. Additional attempts to read from the output buffer will not be allowed until the overflow error is cleared by calling PLIB_PMP_OutputUnderflowClear. This sort of error should be either avoided by having proper communication techniques with the PMP Master device, or checked using PLIB_PMP_OutputBuffersAreEmpty, which informs the controller that buffers have data in them (underflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

Other Features

This section provides information on additional features that may be available. These features are not available on all devices. Please refer to the **"Parallel Master Port (PMP)"** chapter in the specific device data sheet to determine which features are supported by your device.

Description

Operation During Sleep Mode

When the device enters Sleep mode, the system clock is disabled. The consequences of Sleep mode depend on which mode the PMP module is configured in at the time that Sleep mode is invoked.

Master

If the device enters Sleep mode while the PMP module is operating in Master mode, PMP operation will be suspended in its current state until clock execution resumes. As this may cause unexpected control pin timings, users should avoid invoking Sleep mode when continuous use of the module is needed.

Slave

While the PMP module is inactive, but enabled for any Slave mode operation, any read or write operations occurring at that time will be able to complete without the use of the microcontroller clock. Once the operation is completed, the module will issue an interrupt.

This interrupt may be used to wake the device from Sleep or Idle modes, depending on the configuration and capabilities of the device. Refer to the specific device data sheet or the related peripheral section in the reference manual to determine the capabilities of your device.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution. PMP operation during Idle mode can be controlled using the PLIB_PMP_StopInIdleEnable or the PLIB_PMP_StopInIdleDisable. By default, the PMP module continues to operate in Idle mode and provide full functionality.

Configuring the Library

The library is configured for the supported PMP module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Initialization Functions

	Name	Description
	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.
	PLIB_PMP_DualModeReadAddressGet	Gets the current read address of the PMP module.

	PLIB_PMP_DualModeReadAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeWriteAddressGet	Gets the current write address of the PMP module.
	PLIB_PMP_DualModeWriteAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeMasterReceive	Receives the data in the Master Dual mode.
	PLIB_PMP_DualBufferEnable	Enables PMP dual Read/Write buffer.
	PLIB_PMP_DualBufferDisable	Disables the specific PMP module.
	PLIB_PMP_DualModeMasterSend	Sends the specified data in Dual Master mode.

b) Enhanced General Initialization Functions

	Name	Description
	PLIB_PMP_ReadChipSelectXDisable	Configures the Read Chip Select.
	PLIB_PMP_ReadChipSelectXEnable	Configures the Read Chip Select.
	PLIB_PMP_WriteChipSelectXDisable	Configures the Write Chip Select.
	PLIB_PMP_WriteChipSelectXEnable	Configures the Write Chip Select.

c) General Status Functions

	Name	Description
	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_DualBufferIsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_PortIsBusy	Identifies if the (Master mode) PMP port is busy.
	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.
	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.

d) Error Status/Control Functions

	Name	Description
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.

e) Data Read and Write Functions

	Name	Description
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_MasterReceive	Receives the data in Master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.
	PLIB_PMP_ReadCycleIsStarted	Checks whether or not the read cycle on PMP bus is enabled.
	PLIB_PMP_ReadCycleStart	Starts a read cycle on the PMP bus.

f) Wait States Initialization Functions

	Name	Description
	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.

g) Address Handling Functions

	Name	Description
	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.

h) Port Configuration Functions

	Name	Description
	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

i) Polarity Configuration Functions

	Name	Description
	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.

j) Feature Existence Functions

	Name	Description
	PLIB_PMP_ExistsAddressControl	Identifies whether the AddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchPolarity	Identifies whether the AddressLatchPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchStrobePortControl	Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressPortPinControl	Identifies whether the AddressPortPinControl feature exists on the PMP module.
	PLIB_PMP_ExistsBufferOverFlow	Identifies whether the BufferOverFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferRead	Identifies whether the BufferRead feature exists on the PMP module.
	PLIB_PMP_ExistsBufferType	Identifies whether the BufferType feature exists on the PMP module.
	PLIB_PMP_ExistsBufferUnderFlow	Identifies whether the BufferUnderFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferWrite	Identifies whether the BufferWrite feature exists on the PMP module.
	PLIB_PMP_ExistsBusyStatus	Identifies whether the BusyStatus feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectoperation	Identifies whether the ChipSelectoperation feature exists on the PMP module.
	PLIB_PMP_ExistsChipXPolarity	Identifies whether the ChipXPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsCSXActiveStatus	Identifies whether the CSXActiveStatus feature exists on the PMP module.
	PLIB_PMP_ExistsDataHoldWaitStates	Identifies whether the DataHoldWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataSetUpWaitStates	Identifies whether the DataSetUpWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataStrobeWaitStates	Identifies whether the DataStrobeWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataTransferSize	Identifies whether the DataTransferSize feature exists on the PMP module.
	PLIB_PMP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the PMP module.
	PLIB_PMP_ExistsIncrementMode	Identifies whether the IncrementMode feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferFull	Identifies whether the InputBufferFull feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferXStatus	Identifies whether the InputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsInterruptMode	Identifies whether the InterruptMode feature exists on the PMP module.
	PLIB_PMP_ExistsMasterRXTX	Identifies whether the MasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsMUXModeSelect	Identifies whether the MUXModeSelect feature exists on the PMP module.
	PLIB_PMP_ExistsOperationMode	Identifies whether the OperationMode feature exists on the PMP module.
	PLIB_PMP_ExistsOutPutBufferEmpty	Identifies whether the OutPutBufferEmpty feature exists on the PMP module.
	PLIB_PMP_ExistsOutputBufferXStatus	Identifies whether the OutputBufferXStatus feature exists on the PMP module.

	PLIB_PMP_ExistsReadWritePolarity	Identifies whether the ReadWritePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsReadWriteStrobePortControl	Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveRX	Identifies whether the SlaveRX feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveTX	Identifies whether the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies whether the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies whether the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies whether the WriteEnablePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualBufferControl	Identifies whether the DualBufferControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeMasterRXTX	Identifies whether the DualModeMasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeReadAddressControl	Identifies whether the DualModeReadAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeWriteAddressControl	Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsReadChipSelectEnable	Identifies whether the ReadChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsWriteChipSelectEnable	Identifies whether the WriteChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsStartReadControl	Identifies whether the StartReadControl feature exists on the PMP module.

k) Data Types and Constants

	Name	Description
	PMP_ACK_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES	PMP hold after address latch strobe wait states configuration.
	PMP_ADDRESS_LATCH	Address Latch Strobe configuration.
	PMP_ADDRESS_LATCH_WAIT_STATES	PMP address latch strobe wait states configuration.
	PMP_ADDRESS_PORT	Defines the different address lines that are available for configuration.
	PMP_ALTERNATE_MASTER_WAIT_STATES	PMP alternate master wait states.
	PMP_CHIP_SELECT	PMP Chip Select data type.
	PMP_CHIPSELECT_FUNCTION	Defines different functions available for the Chip Select lines multiplexed with address lines.
	PMP_DATA_HOLD_STATES	PMP Data Hold after strobe wait state.
	PMP_DATA_LENGTH	Possible data lengths handled by the PMP module.
	PMP_DATA_SIZE	PMP data size.
	PMP_DATA_WAIT_STATES	PMP data setup time configuration.
	PMP_INCREMENT_MODE	PMP address incrementing configuration.
	PMP_INPUT_BUFFER_TYPE	PMP Input Buffers type.
	PMP_INTERRUPT_MODE	PMP interrupt request mode data type.
	PMP_MASTER_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_MODULE_ID	Possible instances of the PMP module.
	PMP_MUX_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_OPERATION_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_PMBE_PORT	Defines the available Byte Enable ports.
	PMP_POLARITY_LEVEL	Possible polarity levels for the PMP pins.
	PMP_STROBE_WAIT_STATES	PMP strobe signal wait time configuration.

Description

This section describes the Application Programming Interface (API) functions of the PMP Peripheral Library.

Refer to each section for a detailed description.

a) General Initialization Functions

PLIB_PMP_AddressIncrementModeGet Function

Gets the increment mode being used with the address of the PMP module.

File

plib_pmp.h

C

```
PMP_INCREMENT_MODE PLIB_PMP_AddressIncrementModeGet(PMP_MODULE_ID index);
```

Returns

PMP_INCREMENT_MODE - One of the possible values from PMP_INCREMENT_MODE

Description

This function gets the pins used by the PMP module. Refer to the enumeration PMP_INCREMENT_MODE for the possible settings.

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsIncrementMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_INCREMENT_MODE curAddressIncMode;
curAddressIncMode = PLIB_PMP_AddressIncrementModeGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_INCREMENT_MODE PLIB_PMP_AddressIncrementModeGet (PMP_MODULE_ID index)

PLIB_PMP_AddressIncrementModeSelect Function

Configures the increment mode being used with the address of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressIncrementModeSelect(PMP_MODULE_ID index, PMP_INCREMENT_MODE incrementMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_INCREMENT_MODE for the possible settings.

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsIncrementMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
incrementMode	One of the possible values from PMP_INCREMENT_MODE

Function

void PLIB_PMP_AddressIncrementModeSelect(PMP_MODULE_ID index,
PMP_INCREMENT_MODE incrementMode)

PLIB_PMP_AddressLatchStrobeDisable Function

Disables the specific address latch strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchStrobeDisable(PMP_MODULE_ID index, PMP_ADDRESS_LATCH latch);
```

Returns

None.

Description

This function disables the PMP module with a specific address latch strobe depending on which strobes are not needed.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be disabled

Function

void PLIB_PMP_AddressLatchStrobeDisable (PMP_MODULE_ID index,
PMP_ADDRESS_LATCH latch)

PLIB_PMP_AddressLatchStrobeEnable Function

Enables the specific address latch strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchStrobeEnable(PMP_MODULE_ID index, PMP_ADDRESS_LATCH latch);
```

Returns

None.

Description

This function enables the PMP module with a specific address latch strobe depending on which strobes are needed.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be enabled

Function

void PLIB_PMP_AddressLatchStrobeEnable (PMP_MODULE_ID index,
PMP_ADDRESS_LATCH latch)

PLIB_PMP_ChipSelectFunctionSelect Function

Defines the functionality of the pins intended to be used as Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectFunctionSelect(PMP_MODULE_ID index, PMP_CHIPSELECT_FUNCTION chipselfunct);
```

Returns

None.

Description

This function selects the PMCS1/PMCS2 as either Chip Select or as address lines depending on the way these bits are programmed.

Remarks

This function implements an operation of the ChipSelectoperation feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectoperation in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselfunct	One of the possible values from PMP_CHIPSELECT_FUNCTION

Function

void PLIB_PMP_ChipSelectFunctionSelect(PMP_MODULE_ID index,
PMP_CHIPSELECT_FUNCTION chipselfunct)

PLIB_PMP_ChipSelectXDisable Function

Configures the Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ChipSelectXEnable Function

Configures the Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ChipSelectXIsActive Function

Gets the current status of the specified Chip Select.

File

plib_pmp.h

C

```
bool PLIB_PMP_ChipSelectXIsActive(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

- true - Chip Select is active
- false - Chip Select is not active

Description

This function returns the current status of the specified Chip Select.

Remarks

This function implements an operation of the CSXActiveStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsCSXActiveStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
if(PLIB_PMP_ChipSelectXIsActive( PMP_ID_0, chipSelect ))
{
  //Do something useful
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for the Chip Select to be checked

Function

bool PLIB_PMP_ChipSelectXIsActive (PMP_MODULE_ID index,

PMP_CHIP_SELECT chipSelect)

PLIB_PMP_DataSizeSelect Function

Enables data transfer size for the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_DataSizeSelect(PMP_MODULE_ID index, PMP_DATA_SIZE size);
```

Returns

None.

Description

This function enables 4-bit, 8-bit, or 16-bit data transfer for the PMP module.

Remarks

This function implements an operation of the DataTransferSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataTransferSize in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_DataSizeSelect( PMP_ID_0, PMP_DATA_SIZE_8_BITS );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
size	Identifier for the data size to be used

Function

void PLIB_PMP_DataSizeSelect (PMP_MODULE_ID index,
PMP_DATA_SIZE size)

PLIB_PMP_Disable Function

Disables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_Disable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific PMP module.

Remarks

The default state after any reset for a PMP module is the disable state. If the PMP is disabled, all the related pins are in control of the general purpose I/O logic.

Disabling the PMP module resets the buffers to empty states. Any data characters in the buffers are lost. All error and status bits are also reset.

Disabling the PMP while the PMP is active, will abort all pending transmissions and receptions. Re-enabling the PMP will restart the module in the same configuration.

When disabled, the PMP power consumption is minimal.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_Disable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_Disable (PMP_MODULE_ID index)

PLIB_PMP_Enable Function

Enables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_Enable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific PMP module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_Enable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_Enable (PMP_MODULE_ID index)

PLIB_PMP_InputBufferTypeSelect Function

Selects the input buffer based on the input passed.

File

plib_pmp.h

C

```
void PLIB_PMP_InputBufferTypeSelect(PMP_MODULE_ID index, PMP_INPUT_BUFFER_TYPE inputBuffer);
```

Returns

None.

Description

This function selects the input buffer based on the input passed. Either TTL or Schmitt Trigger input buffers are selected.

Remarks

This function implements an operation of the BufferType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_InputBufferTypeSelect( PMP_ID_0, PMP_INPUT_BUFFER_TTL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
inputBuffer	One of the possible input buffer types

Function

void PLIB_PMP_InputBufferTypeSelect (PMP_MODULE_ID index,
PMP_INPUT_BUFFER_TYPE inputBuffer)

PLIB_PMP_InterruptModeGet Function

Gets the current configured interrupt mode being used with the PMP module.

File

plib_pmp.h

C

```
PMP_INTERRUPT_MODE PLIB_PMP_InterruptModeGet(PMP_MODULE_ID index);
```

Returns

One of the possible values from PMP_INTERRUPT_MODE.

Description

This function gets the current configured interrupt mode being used with the PMP module.

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInterruptMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_INTERRUPT_MODE currentIntMode;
currentIntMode = PLIB_PMP_InterruptModeGet ( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_INTERRUPT_MODE PLIB_PMP_InterruptModeGet (PMP_MODULE_ID index)

PLIB_PMP_InterruptModeSelect Function

Configures the interrupt request mode being used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_InterruptModeSelect(PMP_MODULE_ID index, PMP_INTERRUPT_MODE interruptMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_INTERRUPT_MODE for the possible settings.

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInterruptMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
interruptMode	One of the possible values from PMP_INTERRUPT_MODE

Function

void PLIB_PMP_InterruptModeSelect(PMP_MODULE_ID index,
PMP_INTERRUPT_MODE interruptMode)

PLIB_PMP_MultiplexModeGet Function

Gets the current multiplexing mode configured between the address and data of the PMP module.

File

plib_pmp.h

C

```
PMP_MUX_MODE PLIB_PMP_MultiplexModeGet(PMP_MODULE_ID index);
```

Returns

index - Identifier for the device instance to be addressed PMP_MUX_MODE - One of the possible values from PMP_MUX_MODE

Description

This function gets the current multiplexing mode configured between the address and data of the PMP module.

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMUXModeSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_MUX_MODE currentMuxMode;
currentMuxMode = PLIB_PMP_MultiplexModeGet( PMP_ID_0 );
```

Function

PMP_MUX_MODE PLIB_PMP_MultiplexModeGet(PMP_MODULE_ID index)

PLIB_PMP_MultiplexModeSelect Function

Configures the multiplexing between the address and data of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_MultiplexModeSelect(PMP_MODULE_ID index, PMP_MUX_MODE multiplexMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_MUX_MODE for the possible settings.

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMUXModeSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
multiplexMode	One of the possible values from the PMP_MUX_MODE

Function

void PLIB_PMP_MultiplexModeSelect(PMP_MODULE_ID index,
PMP_MUX_MODE multiplexMode)

PLIB_PMP_OperationModeGet Function

Gets the current operation mode of the PMP module.

File

plib_pmp.h

C

```
PMP_OPERATION_MODE PLIB_PMP_OperationModeGet(PMP_MODULE_ID index);
```

Returns

PMP_OPERATION_MODE - One of the possible values from PMP_OPERATION_MODE

Description

This function gets the current operation mode of the PMP module.

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOperationMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_OPERATION_MODE curOpMode;
curOpMode = PLIB_PMP_OperationModeGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_OPERATION_MODE PLIB_PMP_OperationModeGet (PMP_MODULE_ID index)

PLIB_PMP_OperationModeSelect Function

Configures the operation mode of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_OperationModeSelect(PMP_MODULE_ID index, PMP_OPERATION_MODE operationMode);
```

Returns

None.

Description

This function configures operation mode of the PMP module. Refer to the enumeration PMP_OPERATION_MODE for the possible settings.

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOperationMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_OperationModeSelect( PMP_ID_0,
                              PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
operationMode	One of the possible values from PMP_OPERATION_MODE

Function

void PLIB_PMP_OperationModeSelect(PMP_MODULE_ID index,
PMP_OPERATION_MODE operationMode)

PLIB_PMP_StopInIdleDisable Function

Enables the PMP module to continue operation in Idle mode.

File

plib_pmp.h

C

```
void PLIB_PMP_StopInIdleDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the stop in idle flag. The PMP module continues operation in Idle mode.

Remarks

By default, the PMP module will continue operation in Idle mode.

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStopInIdleControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_StopInIdleDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_StopInIdleDisable (PMP_MODULE_ID index)

PLIB_PMP_StopInIdleEnable Function

Discontinues PMP module operation when the device enters Idle mode.

File

plib_pmp.h

C

```
void PLIB_PMP_StopInIdleEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the PMP module to discontinue operation when the device enters Idle mode.

Remarks

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStopInIdleControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_StopInIdleEnable (PMP_MODULE_ID index)

PLIB_PMP_DualModeReadAddressGet Function

Gets the current read address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_DualModeReadAddressGet(PMP_MODULE_ID index);
```

Returns

- readAddress - Device address from where PMP read will occur

Description

This function gets the current read address of the PMP module.

Remarks

This function implements an operation of the DualModeReadAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeReadAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint32_t readAddress;
readAddress = PLIB_PMP_DualModeReadAddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_DualModeReadAddressGet (PMP_MODULE_ID index)

PLIB_PMP_DualModeReadAddressSet Function

Sets the address to be written in Dual mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeReadAddressSet(PMP_MODULE_ID index, uint32_t readAddress);
```

Returns

None.

Description

This function sets the address to be written to the specified value in Dual mode.

Remarks

This function implements an operation of the DualModeReadAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeReadAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_DualModeReadAddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
readAddress	Device address from where PMP read will happen

Function

void PLIB_PMP_DualModeReadAddressSet (PMP_MODULE_ID index,

uint32_t readAddress)

PLIB_PMP_DualModeWriteAddressGet Function

Gets the current write address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_DualModeWriteAddressGet(PMP_MODULE_ID index);
```

Returns

- writeAddress - Device Write address to be set

Description

This function gets the current write address of the PMP module.

Remarks

This function implements an operation of the DualModeWriteAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeWriteAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint32_t address;
address = PLIB_PMP_DualModeWriteAddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_DualModeWriteAddressGet (PMP_MODULE_ID index)

PLIB_PMP_DualModeWriteAddressSet Function

Sets the address to be written in Dual mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeWriteAddressSet(PMP_MODULE_ID index, uint32_t writeAddress);
```

Returns

None.

Description

This function sets the address to be written to the specified value in Dual mode.

Remarks

This function implements an operation of the DualModeWriteAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeWriteAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_DualModeWriteAddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
writeAddress	Device write address to be set

Function

void PLIB_PMP_DualModeWriteAddressSet (PMP_MODULE_ID index,

uint32_t writeAddress)

PLIB_PMP_DualModeMasterReceive Function

Receives the data in the Master Dual mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_DualModeMasterReceive(PMP_MODULE_ID index);
```

Returns

- uint16_t - Data received

Description

This function receives the data in Dual mode. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the DualModeMasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_DualModeMasterReceive( PMP_ID_0 );
```

```
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_DualModeMasterReceive (PMP_MODULE_ID index)

PLIB_PMP_DualBufferEnable Function

Enables PMP dual Read/Write buffer.

File

plib_pmp.h

C

```
void PLIB_PMP_DualBufferEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables dual Read/Write buffers for PMP module. Once enabled, PMP uses separate registers for read and write.

- Registers used for Reads: PMRADDR and PMRDIN
- Registers used for Writes: PMRWADDR and PMDOUT

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
PLIB_PMP_DualBufferEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_DualBufferEnable (PMP_MODULE_ID index)

PLIB_PMP_DualBufferDisable Function

Disables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_DualBufferDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific PMP module.

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
PLIB_PMP_DualBufferDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_DualBufferDisable (PMP_MODULE_ID index)

PLIB_PMP_DualModeMasterSend Function

Sends the specified data in Dual Master mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeMasterSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data in dual mode. The data flow is from master to slave.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the DualModeMasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_DualModeMasterSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Function

void PLIB_PMP_DualModeMasterSend (PMP_MODULE_ID index,

uint16_t data)

b) Enhanced General Initialization Functions

PLIB_PMP_ReadChipSelectXDisable Function

Configures the Read Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Read Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the ReadChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ReadChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ReadChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ReadChipSelectXEnable Function

Configures the Read Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Read Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the ReadChipSelectEnable feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use PLIB_PMP_ExistsReadChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ReadChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ReadChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_WriteChipSelectXDisable Function

Configures the Write Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Write Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the WriteChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_WriteChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_WriteChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_WriteChipSelectXEnable Function

Configures the Write Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Write Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the WriteChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_WriteChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_WriteChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

c) General Status Functions

PLIB_PMP_IsEnabled Function

Checks whether or not the PMP module is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsEnabled(PMP_MODULE_ID index);
```

Returns

- true - If the PMP module is enabled
- false - if the PMP module is disabled

Description

This function checks whether or not the PMP module is enabled.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpStatus;
pmpStatus = PLIB_PMP_IsEnabled( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_IsEnabled (PMP_MODULE_ID index)

PLIB_PMP_DualBufferIsEnabled Function

Checks whether or not the PMP module is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_DualBufferIsEnabled(PMP_MODULE_ID index);
```

Returns

- true - If the PMP module is enabled
- false - if the PMP module is disabled

Description

This function checks whether or not the PMP module is enabled.

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpStatus;
pmpStatus = PLIB_PMP_DualBufferIsEnabled( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_DualBufferIsEnabled (PMP_MODULE_ID index)

PLIB_PMP_PortIsBusy Function

Identifies if the (Master mode) PMP port is busy.

File

plib_pmp.h

C

```
bool PLIB_PMP_PortIsBusy(PMP_MODULE_ID index);
```

Returns

- true - If the port is busy
- false - If the port is not busy

Description

This function identifies if the PMP port is busy (in Master mode).

Remarks

Works only in Master mode. This function implements an operation of the BusyStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBusyStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_PMP_PortIsBusy( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_PortIsBusy (PMP_MODULE_ID index)

PLIB_PMP_InputOverflowHasOccurred Function

Identifies if there was a receiver overflow error.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputOverflowHasOccurred(PMP_MODULE_ID index);
```

Returns

- true - If the input buffer has overflowed
- false - If the input buffer has not overflowed

Description

This function identifies if there was a receiver overflow error.

Remarks

This function implements an operation of the BufferOverFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferOverFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_InputOverflowHasOccurred (PMP_MODULE_ID index)

PLIB_PMP_OutputUnderflowHasOccurred Function

Identifies if there was an output buffer sent out with no data.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputUnderflowHasOccurred(PMP_MODULE_ID index);
```

Returns

- true - If the input buffer was empty when data was sent
- false - If the output buffer was not empty when data was sent

Description

This function identifies if there was a output buffer was sent out with no data.

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferUnderFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_OutputUnderflowHasOccurred (PMP_MODULE_ID index)

d) Error Status/Control Functions

PLIB_PMP_InputOverflowClear Function

Clears a PMP Overflow error.

File

plib_pmp.h

C

```
void PLIB_PMP_InputOverflowClear(PMP_MODULE_ID index);
```

Returns

None.

Description

This function clears an overflow error. Clearing the error resets the receive buffer.

Remarks

This function implements an operation of the BufferOverFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferOverFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_InputOverflowClear (PMP_MODULE_ID index)

PLIB_PMP_OutputUnderflowClear Function

Clears a PMP output underflow error.

File

plib_pmp.h

C

```
void PLIB_PMP_OutputUnderflowClear(PMP_MODULE_ID index);
```

Returns

None.

Description

This function clears a PMP output underflow error.

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferUnderFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_OutputUnderflowClear (PMP_MODULE_ID index)

e) Data Read and Write Functions

PLIB_PMP_InputBuffersAreFull Function

Gets the state of the input buffers.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputBuffersAreFull(PMP_MODULE_ID index);
```

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Description

This function gets the state of the input buffers.

Remarks

This function implements an operation of the InputBufferFull feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferFull in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
if(PLIB_PMP_InputBuffersAreFull( PMP_ID_0 ))
{
 value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_InputBuffersAreFull (PMP_MODULE_ID index)

PLIB_PMP_InputBufferXByteReceive Function

Data to be received in Byte mode.

File

plib_pmp.h

C

```
uint8_t PLIB_PMP_InputBufferXByteReceive(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- data - Data to be received

Description

This function specifies the data to be received in Byte mode from the desired PMP module.

Remarks

This function implements an operation of the BufferRead feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferRead in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t mydata;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    // get data from buffer-1
    mydata = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible buffers (valid values are 0 to 3)

Function

uint8_t PLIB_PMP_InputBufferXByteReceive (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_InputBufferXIsFull Function

Gets the state of the identified input buffer.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputBufferXIsFull(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Description

This function gets the state of the identified input buffer.

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
// Check the status of buffer-2
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 2 ))
{
   // get data from buffer 2
   value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	The input buffer number (valid values are 0 to 3)

Function

bool PLIB_PMP_InputBufferXIsFull (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_IsDataReceived Function

Checks and returns if the data has been received in the specified buffer in Slave mode.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsDataReceived(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - Data has been received in the specified buffer
- false - Data has not been received in the specified buffer

Description

This function checks and returns if the data has been received in the specified buffer in Slave mode.

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module should be configured for Slave mode operation.

Example

```
int8_t data;
// Check if data is received on buffer-2
if(PLIB_PMP_IsDataReceived( PMP_ID_0, 2 ))
{
    // get data from buffer-2
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible input buffers (valid values are 0 to 3)

Function

bool PLIB_PMP_IsDataReceived (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_IsDataTransmitted Function

Checks and returns if the data has been transmitted from the specified buffer in Slave mode.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsDataTransmitted(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If data has been transmitted from the specified buffer

- false - If data has not been transmitted from the specified buffer

Description

This function checks and returns if data has been transmitted from the specified buffer.

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module should be configured for Slave mode operation.

Example

```
uint8_t data;
if(PLIB_PMP_IsDataTransmitted( PMP_ID_0, 2 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 2, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible output buffers (valid range is 0 to 3)

Function

bool PLIB_PMP_IsDataTransmitted (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_MasterReceive Function

Receives the data in Master mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_MasterReceive(PMP_MODULE_ID index);
```

Returns

uint16_t - Data received

Description

This function receives the data. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_MasterReceive (PMP_MODULE_ID index)

PLIB_PMP_MasterSend Function

Sends the specified data in Master mode.

File

plib_pmp.h

C

```
void PLIB_PMP_MasterSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data. The data flow is from master to slave.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured for Master mode.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_MasterSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Function

void PLIB_PMP_MasterSend (PMP_MODULE_ID index,

uint16_t data)

PLIB_PMP_OutputBuffersAreEmpty Function

Gets the state of the output buffers.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputBuffersAreEmpty(PMP_MODULE_ID index);
```

Returns

- true - If all output buffers are empty
- false - If all output buffers are not empty

Description

This function returns the state of the output buffers.

Remarks

This function implements an operation of the OutPutBufferEmpty feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutPutBufferEmpty in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value=0xEF;

if(PLIB_PMP_OutputBuffersAreEmpty( PMP_ID_0 ))
{
 PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_OutputBuffersAreEmpty (PMP_MODULE_ID index)

PLIB_PMP_OutputBufferXByteSend Function

Data to be transmitted in Byte mode.

File

plib_pmp.h

C

```
void PLIB_PMP_OutputBufferXByteSend(PMP_MODULE_ID index, uint8_t buffer, uint8_t data);
```

Returns

None.

Description

This function specifies the data to be transmitted in Byte mode for the desired PMP module.

Remarks

This function implements an operation of the BufferWrite feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferWrite in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

buffer	One of the possible output buffers (valid range is 0 to 3)
data	Data to be transmitted

Function

void PLIB_PMP_OutputBufferXByteSend (PMP_MODULE_ID index,
uint8_t buffer,
uint8_t data)

PLIB_PMP_OutputBufferXIsEmpty Function

Gets the state of the input buffer.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputBufferXIsEmpty(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If the identified output buffer is empty
- false - If the identified output buffer is not empty

Description

This function returns the state of the input buffer.

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value = 0xEF;

if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1 ) )
{
 PLIB_PMP_OutputBufferXByteSend( PMP_ID_0,1, value);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	Output buffer number (valid range is 0 to 3)

Function

bool PLIB_PMP_OutputBufferXIsEmpty (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_SlaveReceive Function

Receives the data in Slave mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_SlaveReceive(PMP_MODULE_ID index);
```

Returns

- uint16_t - Data received

Description

This function receives the data. The flow of data is from the master to the slave.

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveRX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsSlaveRX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a slave.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_SlaveReceive( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_SlaveReceive (PMP_MODULE_ID index)

PLIB_PMP_SlaveSend Function

Sends the specified data in Slave mode.

File

plib_pmp.h

C

```
void PLIB_PMP_SlaveSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsSlaveTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured for Slave mode.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_SlaveSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

data	Data to be transmitted

Function

void PLIB_PMP_SlaveSend (PMP_MODULE_ID index,
uint16_t data)

PLIB_PMP_ReadCycleIsStarted Function

Checks whether or not the read cycle on PMP bus is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_ReadCycleIsStarted(PMP_MODULE_ID index);
```

Returns

- true - If the PMP Read cycle is enabled
- false - if the PMP Read cycle is not enabled

Description

This function checks whether or not the read cycle on PMP bus is enabled.

Remarks

This function implements an operation of the StartReadControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStartReadControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpReadStartStatus;
pmpReadStartStatus = PLIB_PMP_ReadCycleIsStarted( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_ReadCycleIsStarted (PMP_MODULE_ID index)

PLIB_PMP_ReadCycleStart Function

Starts a read cycle on the PMP bus.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadCycleStart(PMP_MODULE_ID index);
```

Returns

None.

Description

This function stars a read cycle on the bus for the selected PMP module. This bit is cleared by hardware at the end of the read cycle

Remarks

This function implements an operation of the StartReadControl feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use PLIB_PMP_ExistsStartReadControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadCycleStart( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadCycleStart(PMP_MODULE_ID index)

f) Wait States Initialization Functions

PLIB_PMP_WaitStatesDataHoldSelect Function

Configures the data hold states (after data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesDataHoldSelect(PMP_MODULE_ID index, PMP_DATA_HOLD_STATES dataHoldState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_DATA_HOLD_STATES for the possible settings.

Remarks

This function implements an operation of the DataHoldWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataHoldWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataHoldState	One of the possible values from PMP_DATA_HOLD_STATES

Function

void PLIB_PMP_WaitStatesDataHoldSelect(PMP_MODULE_ID index,
PMP_DATA_HOLD_STATES dataHoldState)

PLIB_PMP_WaitStatesDataSetUpSelect Function

Configures the data wait states (before the data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesDataSetUpSelect(PMP_MODULE_ID index, PMP_DATA_WAIT_STATES dataWaitState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_DATA_WAIT_STATES for the possible settings.

Remarks

This function implements an operation of the DataSetUpWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataSetUpWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_TWO );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataWaitState	One of the possible values from PMP_DATA_WAIT_STATES

Function

void PLIB_PMP_WaitStatesDataSetUpSelect(PMP_MODULE_ID index,
PMP_DATA_WAIT_STATES dataWaitState)

PLIB_PMP_WaitStatesStrobeSelect Function

Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesStrobeSelect(PMP_MODULE_ID index, PMP_STROBE_WAIT_STATES strobeWaitState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_STROBE_WAIT_STATES for the possible settings.

Remarks

This function implements an operation of the DataStrobeWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataStrobeWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
strobeWaitState	One of the possible values from PMP_STROBE_WAIT_STATES

Function

void PLIB_PMP_WaitStatesStrobeSelect(PMP_MODULE_ID index,
PMP_STROBE_WAIT_STATES strobeWaitState)

g) Address Handling Functions

PLIB_PMP_AddressGet Function

Gets the current address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_AddressGet(PMP_MODULE_ID index);
```

Returns

- address - Device address to be set

Description

This function gets the current address of the PMP module.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t address;
address = PLIB_PMP_AddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_AddressGet (PMP_MODULE_ID index)

PLIB_PMP_AddressLinesA0A1Get Function

Gets the value of the address lines PMA0 and PMA1.

File

plib_pmp.h

C

```
uint8_t PLIB_PMP_AddressLinesA0A1Get(PMP_MODULE_ID index);
```

Returns

uint8_t - The two-bit address

Description

This function gets the value of the address lines PMA0 and PMA1. This function is used in the addressable parallel slave port mode.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t bufferAddress;
bufferAddress = PLIB_PMP_AddressLinesA0A1Get( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint8_t PLIB_PMP_AddressLinesA0A1Get (PMP_MODULE_ID index)

PLIB_PMP_AddressLinesA0A1Set Function

Sets the address lines PMA0 and PMA1 with the value specified.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLinesA0A1Set(PMP_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the address lines PMA0 and PMA1 with the value specified. This function is used in the addressable parallel slave port mode.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t bufferAddress = 0x2;
PLIB_PMP_AddressLinesA0A1Set( PMP_ID_0, bufferAddress );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	The two-bit address

Function

void PLIB_PMP_AddressLinesA0A1Set (PMP_MODULE_ID index,

uint8_t address)

PLIB_PMP_AddressSet Function

Sets the current address of the PMP module to the specified address.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressSet(PMP_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function sets the current address of the PMP module to the specified value.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_AddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	Device address to be set

Function

void PLIB_PMP_AddressSet (PMP_MODULE_ID index,

uint32_t address)

h) Port Configuration Functions

PLIB_PMP_AddressPortDisable Function

Disables the port lines specified as PMP address lines.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressPortDisable(PMP_MODULE_ID index, PMP_ADDRESS_PORT portfunctions);
```

Returns

None.

Description

This function disables the port lines specified as PMP address lines. They function as normal I/O lines.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressPortPinControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Example-1
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;
PLIB_PMP_AddressPortDisable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );

//Example-2
PLIB_PMP_AddressPortDisable( PMP_ID_0,
                             ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Function

void PLIB_PMP_AddressPortDisable (PMP_MODULE_ID index,
PMP_ADDRESS_PORT portfunctions)

PLIB_PMP_AddressPortEnable Function

Enables the port lines specified as PMP address lines.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressPortEnable(PMP_MODULE_ID index, PMP_ADDRESS_PORT portfunctions);
```

Returns

None.

Description

This function enables the port lines specified as PMP address lines.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressPortPinControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Example-1
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;
PLIB_PMP_AddressPortEnable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );

//Example-2
PLIB_PMP_AddressPortEnable( PMP_ID_0,
                            ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Function

void PLIB_PMP_AddressPortEnable (PMP_MODULE_ID index,

PMP_ADDRESS_PORT portfunctions)

PLIB_PMP_ReadWriteStrobePortDisable Function

Disables the PMP module read strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePortDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the read strobe port of the PMP module.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWriteStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePortDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadWriteStrobePortDisable (PMP_MODULE_ID index)

PLIB_PMP_ReadWriteStrobePortEnable Function

Enables the PMP module read strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePortEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the read strobe port of the PMP module.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWriteStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePortEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadWriteStrobePortEnable (PMP_MODULE_ID index)

PLIB_PMP_WriteEnableStrobePortDisable Function

Disables the PMP module write strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePortDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the write strobe port of the PMP module.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePortDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_WriteEnableStrobePortDisable (PMP_MODULE_ID index)

PLIB_PMP_WriteEnableStrobePortEnable Function

Enables the PMP module write strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePortEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the write strobe port of the PMP module.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePortEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_WriteEnableStrobePortEnable (PMP_MODULE_ID index)

i) Polarity Configuration Functions

PLIB_PMP_AddressLatchPolaritySelect Function

Sets the address latch strobe polarity.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchPolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets the address latch polarity to the level specified.

Remarks

This function implements an operation of the AddressLatchPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchPolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchPolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_AddressLatchPolaritySelect(PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_ChipSelectXPolaritySelect Function

Sets the specified Chip Select polarity.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXPolaritySelect(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets the specified Chip Select polarity to the level specified.

Remarks

This function implements an operation of the ChipXPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipXPolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXPolaritySelect( PMP_ID_0,
                                    chipSelect,
                                    PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for Chip Select
polarity	Possible polarity levels

Function

void PLIB_PMP_ChipSelectXPolaritySelect (PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_ReadWriteStrobePolaritySelect Function

Sets the polarity of the read strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets polarity of the read strobe to the level specified.

Remarks

This function implements an operation of the ReadWritePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWritePolarity in your application to to automatically determine

whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_ReadWriteStrobePolaritySelect (PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_WriteEnableStrobePolaritySelect Function

Sets the polarity of the write enable strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets the polarity of the write enable strobe to the level specified.

Remarks

This function implements an operation of the WriteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_WriteEnableStrobePolaritySelect (PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

j) Feature Existence Functions

PLIB_PMP_ExistsAddressControl Function

Identifies whether the AddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressControl feature is supported on the device
- false - The AddressControl feature is not supported on the device

Description

This function identifies whether the AddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressSet
- PLIB_PMP_AddressGet
- PLIB_PMP_AddressLinesA0A1Set
- PLIB_PMP_AddressLinesA0A1Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressLatchPolarity Function

Identifies whether the AddressLatchPolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressLatchPolarity(PMP_MODULE_ID index);
```

Returns

- true - The AddressLatchPolarity feature is supported on the device
- false - The AddressLatchPolarity feature is not supported on the device

Description

This function identifies whether the AddressLatchPolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_AddressLatchPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressLatchPolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressLatchStrobePortControl Function

Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressLatchStrobePortControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressLatchStrobePortControl feature is supported on the device
- false - The AddressLatchStrobePortControl feature is not supported on the device

Description

This function identifies whether the AddressLatchStrobePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressLatchStrobeEnable
- PLIB_PMP_AddressLatchStrobeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressLatchStrobePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressPortPinControl Function

Identifies whether the AddressPortPinControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressPortPinControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressPortPinControl feature is supported on the device
- false - The AddressPortPinControl feature is not supported on the device

Description

This function identifies whether the AddressPortPinControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressPortEnable
- PLIB_PMP_AddressPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressPortPinControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferOverFlow Function

Identifies whether the BufferOverFlow feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferOverFlow(PMP_MODULE_ID index);
```

Returns

- true - The BufferOverFlow feature is supported on the device
- false - The BufferOverFlow feature is not supported on the device

Description

This function identifies whether the BufferOverFlow feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InputOverflowHasOccurred
- PLIB_PMP_InputOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferOverFlow(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferRead Function

Identifies whether the BufferRead feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferRead(PMP_MODULE_ID index);
```

Returns

- true - The BufferRead feature is supported on the device
- false - The BufferRead feature is not supported on the device

Description

This function identifies whether the BufferRead feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBufferXByteReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferRead(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferType Function

Identifies whether the BufferType feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferType(PMP_MODULE_ID index);
```

Returns

- true - The BufferType feature is supported on the device
- false - The BufferType feature is not supported on the device

Description

This function identifies whether the BufferType feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBufferTypeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferType(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferUnderFlow Function

Identifies whether the BufferUnderFlow feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferUnderFlow(PMP_MODULE_ID index);
```

Returns

- true - The BufferUnderFlow feature is supported on the device
- false - The BufferUnderFlow feature is not supported on the device

Description

This function identifies whether the BufferUnderFlow feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OutputUnderflowHasOccurred
- PLIB_PMP_OutputUnderflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferUnderFlow(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferWrite Function

Identifies whether the BufferWrite feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferWrite(PMP_MODULE_ID index);
```

Returns

- true - The BufferWrite feature is supported on the device
- false - The BufferWrite feature is not supported on the device

Description

This function identifies whether the BufferWrite feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_OutputBufferXByteSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferWrite(PMP_MODULE_ID index)

PLIB_PMP_ExistsBusyStatus Function

Identifies whether the BusyStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBusyStatus(PMP_MODULE_ID index);
```

Returns

- true - The BusyStatus feature is supported on the device
- false - The BusyStatus feature is not supported on the device

Description

This function identifies whether the BusyStatus feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_PortIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBusyStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipSelectEnable Function

Identifies whether the ChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ChipSelectXEnable
- PLIB_PMP_ChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipSelectoperation Function

Identifies whether the ChipSelectoperation feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipSelectoperation(PMP_MODULE_ID index);
```

Returns

- true - The ChipSelectoperation feature is supported on the device
- false - The ChipSelectoperation feature is not supported on the device

Description

This function identifies whether the ChipSelectoperation feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectFunctionSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipSelectoperation(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipXPolarity Function

Identifies whether the ChipXPolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipXPolarity(PMP_MODULE_ID index);
```

Returns

- true - The ChipXPolarity feature is supported on the device
- false - The ChipXPolarity feature is not supported on the device

Description

This function identifies whether the ChipXPolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectXPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipXPolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsCSXActiveStatus Function

Identifies whether the CSXActiveStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsCSXActiveStatus(PMP_MODULE_ID index);
```

Returns

- true - The CSXActiveStatus feature is supported on the device
- false - The CSXActiveStatus feature is not supported on the device

Description

This function identifies whether the CSXActiveStatus feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectXIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsCSXActiveStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataHoldWaitStates Function

Identifies whether the DataHoldWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataHoldWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataHoldWaitStates feature is supported on the device
- false - The DataHoldWaitStates feature is not supported on the device

Description

This function identifies whether the DataHoldWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesDataHoldSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataHoldWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataSetUpWaitStates Function

Identifies whether the DataSetUpWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataSetUpWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataSetUpWaitStates feature is supported on the device
- false - The DataSetUpWaitStates feature is not supported on the device

Description

This function identifies whether the DataSetUpWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesDataSetUpSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataSetUpWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataStrobeWaitStates Function

Identifies whether the DataStrobeWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataStrobeWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataStrobeWaitStates feature is supported on the device
- false - The DataStrobeWaitStates feature is not supported on the device

Description

This function identifies whether the DataStrobeWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesStrobeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataStrobeWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataTransferSize Function

Identifies whether the DataTransferSize feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataTransferSize(PMP_MODULE_ID index);
```

Returns

- true - The DataTransferSize feature is supported on the device
- false - The DataTransferSize feature is not supported on the device

Description

This function identifies whether the DataTransferSize feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_DataSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataTransferSize(PMP_MODULE_ID index)

PLIB_PMP_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsEnableControl(PMP_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_Disable
- PLIB_PMP_Enable
- PLIB_PMP_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsEnableControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsIncrementMode Function

Identifies whether the IncrementMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsIncrementMode(PMP_MODULE_ID index);
```

Returns

- true - The IncrementMode feature is supported on the device
- false - The IncrementMode feature is not supported on the device

Description

This function identifies whether the IncrementMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressIncrementModeSelect
- PLIB_PMP_AddressIncrementModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsIncrementMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsInputBufferFull Function

Identifies whether the InputBufferFull feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInputBufferFull(PMP_MODULE_ID index);
```

Returns

- true - The InputBufferFull feature is supported on the device
- false - The InputBufferFull feature is not supported on the device

Description

This function identifies whether the InputBufferFull feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBuffersAreFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInputBufferFull(PMP_MODULE_ID index)

PLIB_PMP_ExistsInputBufferXStatus Function

Identifies whether the InputBufferXStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInputBufferXStatus(PMP_MODULE_ID index);
```

Returns

- true - The InputBufferXStatus feature is supported on the device
- false - The InputBufferXStatus feature is not supported on the device

Description

This function identifies whether the InputBufferXStatus feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InputBufferXIsFull
- PLIB_PMP_IsDataReceived

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInputBufferXStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsInterruptMode Function

Identifies whether the InterruptMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInterruptMode(PMP_MODULE_ID index);
```

Returns

- true - The InterruptMode feature is supported on the device
- false - The InterruptMode feature is not supported on the device

Description

This function identifies whether the InterruptMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InterruptModeSelect
- PLIB_PMP_InterruptModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInterruptMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsMasterRXTX Function

Identifies whether the MasterRXTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsMasterRXTX(PMP_MODULE_ID index);
```

Returns

- true - The MasterRXTX feature is supported on the device
- false - The MasterRXTX feature is not supported on the device

Description

This function identifies whether the MasterRXTX feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_MasterSend
- PLIB_PMP_MasterReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsMasterRXTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsMUXModeSelect Function

Identifies whether the MUXModeSelect feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsMUXModeSelect(PMP_MODULE_ID index);
```

Returns

- true - The MUXModeSelect feature is supported on the device
- false - The MUXModeSelect feature is not supported on the device

Description

This function identifies whether the MUXModeSelect feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_MultiplexModeSelect
- PLIB_PMP_MultiplexModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsMUXModeSelect(PMP_MODULE_ID index)

PLIB_PMP_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOperationMode(PMP_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OperationModeSelect
- PLIB_PMP_OperationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOperationMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsOutPutBufferEmpty Function

Identifies whether the OutPutBufferEmpty feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOutPutBufferEmpty(PMP_MODULE_ID index);
```

Returns

- true - The OutPutBufferEmpty feature is supported on the device
- false - The OutPutBufferEmpty feature is not supported on the device

Description

This function identifies whether the OutPutBufferEmpty feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_OutputBuffersAreEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOutPutBufferEmpty(PMP_MODULE_ID index)

PLIB_PMP_ExistsOutputBufferXStatus Function

Identifies whether the OutputBufferXStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOutputBufferXStatus(PMP_MODULE_ID index);
```

Returns

- true - The OutputBufferXStatus feature is supported on the device
- false - The OutputBufferXStatus feature is not supported on the device

Description

This function identifies whether the OutputBufferXStatus feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OutputBufferXIsEmpty
- PLIB_PMP_IsDataTransmitted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOutputBufferXStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadWritePolarity Function

Identifies whether the ReadWritePolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadWritePolarity(PMP_MODULE_ID index);
```

Returns

- true - The ReadWritePolarity feature is supported on the device
- false - The ReadWritePolarity feature is not supported on the device

Description

This function identifies whether the ReadWritePolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ReadWriteStrobePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadWritePolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadWriteStrobePortControl Function

Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadWriteStrobePortControl(PMP_MODULE_ID index);
```

Returns

- true - The ReadWriteStrobePortControl feature is supported on the device
- false - The ReadWriteStrobePortControl feature is not supported on the device

Description

This function identifies whether the ReadWriteStrobePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadWriteStrobePortEnable
- PLIB_PMP_ReadWriteStrobePortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadWriteStrobePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsSlaveRX Function

Identifies whether the SlaveRX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsSlaveRX(PMP_MODULE_ID index);
```

Returns

- true - The SlaveRX feature is supported on the device
- false - The SlaveRX feature is not supported on the device

Description

This function identifies whether the SlaveRX feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_SlaveReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsSlaveRX(PMP_MODULE_ID index)

PLIB_PMP_ExistsSlaveTX Function

Identifies whether the SlaveTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsSlaveTX(PMP_MODULE_ID index);
```

Returns

- true - The SlaveTX feature is supported on the device
- false - The SlaveTX feature is not supported on the device

Description

This function identifies whether the SlaveTX feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_SlaveSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsSlaveTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsStopInIdleControl Function

Identifies whether the StopInIdleControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsStopInIdleControl(PMP_MODULE_ID index);
```

Returns

- true - The StopInIdleControl feature is supported on the device
- false - The StopInIdleControl feature is not supported on the device

Description

This function identifies whether the StopInIdleControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_StopInIdleEnable
- PLIB_PMP_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsStopInIdleControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteEnablePolarity Function

Identifies whether the WriteEnablePolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteEnablePolarity(PMP_MODULE_ID index);
```

Returns

- true - The WriteEnablePolarity feature is supported on the device
- false - The WriteEnablePolarity feature is not supported on the device

Description

This function identifies whether the WriteEnablePolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WriteEnableStrobePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteEnablePolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteEnablePortControl Function

Identifies whether the WriteEnablePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteEnablePortControl(PMP_MODULE_ID index);
```

Returns

- true - The WriteEnablePortControl feature is supported on the device
- false - The WriteEnablePortControl feature is not supported on the device

Description

This function identifies whether the WriteEnablePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_WriteEnableStrobePortEnable
- PLIB_PMP_WriteEnableStrobePortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteEnablePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualBufferControl Function

Identifies whether the DualBufferControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualBufferControl(PMP_MODULE_ID index);
```

Returns

- true - The DualBufferControl feature is supported on the device
- false - The DualBufferControl feature is not supported on the device

Description

This function identifies whether the DualBufferControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualBufferDisable
- PLIB_PMP_DualBufferEnable
- PLIB_PMP_DualBufferIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualBufferControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeMasterRXTX Function

Identifies whether the DualModeMasterRXTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeMasterRXTX(PMP_MODULE_ID index);
```

Returns

- true - The DualModeMasterRXTX feature is supported on the device
- false - The DualModeMasterRXTX feature is not supported on the device

Description

This function identifies whether the DualModeMasterRXTX feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeMasterSend
- PLIB_PMP_DualModeMasterReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeMasterRXTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeReadAddressControl Function

Identifies whether the DualModeReadAddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeReadAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The DualModeReadAddressControl feature is supported on the device
- false - The DualModeReadAddressControl feature is not supported on the device

Description

This function identifies whether the DualModeReadAddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeReadAddressSet
- PLIB_PMP_DualModeReadAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeReadAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeWriteAddressControl Function

Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeWriteAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The DualModeWriteAddressControl feature is supported on the device
- false - The DualModeWriteAddressControl feature is not supported on the device

Description

This function identifies whether the DualModeWriteAddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeWriteAddressSet
- PLIB_PMP_DualModeWriteAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeWriteAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadChipSelectEnable Function

Identifies whether the ReadChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The ReadChipSelectEnable feature is supported on the device
- false - The ReadChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ReadChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadChipSelectXEnable
- PLIB_PMP_ReadChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteChipSelectEnable Function

Identifies whether the WriteChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The WriteChipSelectEnable feature is supported on the device
- false - The WriteChipSelectEnable feature is not supported on the device

Description

This function identifies whether the WriteChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_WriteChipSelectXEnable
- PLIB_PMP_WriteChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsStartReadControl Function

Identifies whether the StartReadControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsStartReadControl(PMP_MODULE_ID index);
```

Returns

- true - The StartReadControl feature is supported on the device
- false - The StartReadControl feature is not supported on the device

Description

This function identifies whether the StartReadControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadCycleStart
- PLIB_PMP_ReadCycleIsStarted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsStartReadControl(PMP_MODULE_ID index)

k) Data Types and Constants

PMP_ACK_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_USE_ACK_WITH_TIMEOUT,
  PMP_USE_ACK,
```

```
  PMP_ACK_DISABLED
} PMP_ACK_MODE;
```

Members

Members	Description
PMP_USE_ACK_WITH_TIMEOUT	Acknowledge used with a timeout
PMP_USE_ACK	Acknowledge is used
PMP_ACK_DISABLED	Acknowledge is not used

Description

PMP Acknowledge Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_HOLD_LATCH_WAIT_STATES Enumeration

PMP hold after address latch strobe wait states configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER,
  PMP_ADDRESS_HOLD_ONE_QUARTER
} PMP_ADDRESS_HOLD_LATCH_WAIT_STATES;
```

Members

Members	Description
PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER	Data Wait of 1 1/4 Peripheral Bus Clock Cycles
PMP_ADDRESS_HOLD_ONE_QUARTER	Data Wait of 1/4 Peripheral Bus Clock Cycles

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP holds after address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_LATCH Enumeration

Address Latch Strobe configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_LATCH_UPPER,
  PMP_ADDRESS_LATCH_HIGH,
  PMP_ADDRESS_LATCH_LOW
} PMP_ADDRESS_LATCH;
```

Members

Members	Description
PMP_ADDRESS_LATCH_UPPER	Address latch upper
PMP_ADDRESS_LATCH_HIGH	Address latch high
PMP_ADDRESS_LATCH_LOW	Address latch low

Description

PMP Address Latch

This data type defines the different configurations by which the PMP address latch strobes can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_LATCH_WAIT_STATES Enumeration

PMP address latch strobe wait states configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_WAIT_THREE_AND_HALF,
  PMP_ADDRESS_WAIT_TWO_AND_HALF,
  PMP_ADDRESS_WAIT_ONE_AND_HALF,
  PMP_ADDRESS_WAIT_HALF
} PMP_ADDRESS_LATCH_WAIT_STATES;
```

Members

Members	Description
PMP_ADDRESS_WAIT_THREE_AND_HALF	Data Wait of 3 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_TWO_AND_HALF	Data Wait of 2 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_ONE_AND_HALF	Data Wait of 1 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_HALF	Data Wait of 1/2 Peripheral Bus Clock Cycles

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_PORT Enumeration

Defines the different address lines that are available for configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_PMA0_PORT,
  PMP_PMA1_PORT,
  PMP_PMA2_TO_PMA13_PORTS,
  PMP_PMA2_PORT,
  PMP_PMA3_PORT,
  PMP_PMA4_PORT,
  PMP_PMA5_PORT,
  PMP_PMA6_PORT,
  PMP_PMA7_PORT,
  PMP_PMA8_PORT,
  PMP_PMA9_PORT,
  PMP_PMA10_PORT,
  PMP_PMA11_PORT,
  PMP_PMA12_PORT,
  PMP_PMA13_PORT,
  PMP_PMA14_PORT,
  PMP_PMA15_PORT,
  PMP_PMA16_TO_PMA22_PORTS,
```

```
  PMP_PMA2_TO_PMA10_PORTS
} PMP_ADDRESS_PORT;
```

Members

Members	Description
PMP_PMA0_PORT	Address line 0 port (make as general purpose I/O or PMP address line)
PMP_PMA1_PORT	Address line 1 port (make as general purpose I/O or PMP address line)
PMP_PMA2_TO_PMA13_PORTS	Address line 2 to 13 ports (make as general purpose I/O or PMP address lines)
PMP_PMA2_PORT	Address line 2 port (make as general purpose I/O or PMP address line)
PMP_PMA3_PORT	Address line 3 port (make as general purpose I/O or PMP address line)
PMP_PMA4_PORT	Address line 4 port (make as general purpose I/O or PMP address line)
PMP_PMA5_PORT	Address line 5 port (make as general purpose I/O or PMP address line)
PMP_PMA6_PORT	Address line 6 port (make as general purpose I/O or PMP address line)
PMP_PMA7_PORT	Address line 7 port (make as general purpose I/O or PMP address line)
PMP_PMA8_PORT	Address line 8 port (make as general purpose I/O or PMP address line)
PMP_PMA9_PORT	Address line 9 port (make as general purpose I/O or PMP address line)
PMP_PMA10_PORT	Address line 10 port (make as general purpose I/O or PMP address line)
PMP_PMA11_PORT	Address line 11 port (make as general purpose I/O or PMP address line)
PMP_PMA12_PORT	Address line 12 port (make as general purpose I/O or PMP address line)
PMP_PMA13_PORT	Address line 13 port (make as general purpose I/O or PMP address line)
PMP_PMA14_PORT	Address line 14 port (make as general purpose I/O or PMP address line)
PMP_PMA15_PORT	Address line 15 port (make as general purpose I/O or PMP address line)
PMP_PMA16_TO_PMA22_PORTS	Address line 16 to 22 ports (make as general purpose I/O or PMP address lines)
PMP_PMA2_TO_PMA10_PORTS	Address line 2 to 10 ports (make as general purpose I/O or PMP address lines)

Description

PMP Address Port Pins

This data type defines the different address that can be configured by the PMP module. The user application can make each pin as a general purpose I/O or part of PMP functionality.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h). These enumerator values can be passed in a bitwise ORed fashion to select multiple ports at a time.

PMP_ALTERNATE_MASTER_WAIT_STATES Enumeration

PMP alternate master wait states.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ALTERNATE_MASTER_WAIT_10,
  PMP_ALTERNATE_MASTER_WAIT_9,
  PMP_ALTERNATE_MASTER_WAIT_8,
  PMP_ALTERNATE_MASTER_WAIT_7,
  PMP_ALTERNATE_MASTER_WAIT_6,
  PMP_ALTERNATE_MASTER_WAIT_5,
  PMP_ALTERNATE_MASTER_WAIT_4,
  PMP_ALTERNATE_MASTER_WAIT_3
} PMP_ALTERNATE_MASTER_WAIT_STATES;
```

Members

Members	Description
PMP_ALTERNATE_MASTER_WAIT_10	Wait of 10 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_9	Wait of 9 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_8	Wait of 8 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_7	Wait of 7 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_6	Wait of 6 Alternate Master Cycles

PMP_ALTERNATE_MASTER_WAIT_5	Wait of 5 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_4	Wait of 4 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_3	Wait of 3 Alternate Master Cycles

Description

PMP Alternate Master Wait States

This data type defines the different configurations by which the PMP alternate master wait states can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_CHIP_SELECT Enumeration

PMP Chip Select data type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_CHIP_SELECT_ONE,
  PMP_CHIP_SELECT_TWO
} PMP_CHIP_SELECT;
```

Members

Members	Description
PMP_CHIP_SELECT_ONE	Chip Select One
PMP_CHIP_SELECT_TWO	Chip Select Two

Description

PMP Chip Select

This data type defines the different Chip Select lines of the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_CHIPSELECT_FUNCTION Enumeration

Defines different functions available for the Chip Select lines multiplexed with address lines.

File

plib_pmp_help.h

C

```
typedef enum {
  PMCS1_AS_CHIP_SELECT,
  PMCS1_AS_ADDRESS_LINE,
  PMCS1_PMCS2_AS_ADDRESS_LINES,
  PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT,
  PMCS1_AND_PMCS2_AS_CHIP_SELECT
} PMP_CHIPSELECT_FUNCTION;
```

Members

Members	Description
PMCS1_AS_CHIP_SELECT	CS1 used as Chip Select
PMCS1_AS_ADDRESS_LINE	CS1 used as address line
PMCS1_PMCS2_AS_ADDRESS_LINES	CS1 and CS2 used as address lines
PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT	CS1 used as address line and CS2 as Chip Select
PMCS1_AND_PMCS2_AS_CHIP_SELECT	Both CS1 and CS2 used as Chip Selects

Description

PMP Chip Select pin functions

This data type defines different functions of Chip Select pins.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_HOLD_STATES Enumeration

PMP Data Hold after strobe wait state.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_HOLD_FOUR,
  PMP_DATA_HOLD_THREE,
  PMP_DATA_HOLD_TWO,
  PMP_DATA_HOLD_ONE
} PMP_DATA_HOLD_STATES;
```

Members

Members	Description
PMP_DATA_HOLD_FOUR	Data Hold of 4 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_THREE	Data Hold of 3 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_TWO	Data Hold of 2 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_ONE	Data Hold of 1 Peripheral Bus Clock Cycles

Description

PMP Data Hold after strobe wait state

This data type defines the different data Hold after strobe wait states.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_LENGTH Enumeration

Possible data lengths handled by the PMP module.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_8_BITS,
  PMP_DATA_16_BITS,
  PMP_DATA_32_BITS
} PMP_DATA_LENGTH;
```

Members

Members	Description
PMP_DATA_8_BITS	8-bit data length
PMP_DATA_16_BITS	16-bit data length
PMP_DATA_32_BITS	32-bit data length

Description

PMP data length

This data type defines the possible data lengths handled by the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_SIZE Enumeration

PMP data size.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_SIZE_8_BITS = 0x0,
  PMP_DATA_SIZE_16_BITS = 0x1
} PMP_DATA_SIZE;
```

Members

Members	Description
PMP_DATA_SIZE_8_BITS = 0x0	Data length of 8-bits
PMP_DATA_SIZE_16_BITS = 0x1	Data length of 16-bits

Description

PMP DATA Size

This data type defines the different configurations for the data lengths handled by the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_WAIT_STATES Enumeration

PMP data setup time configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_WAIT_FOUR,
  PMP_DATA_WAIT_THREE,
  PMP_DATA_WAIT_TWO,
  PMP_DATA_WAIT_ONE
} PMP_DATA_WAIT_STATES;
```

Members

Members	Description
PMP_DATA_WAIT_FOUR	Data Wait of 4 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_THREE	Data Wait of 3 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_TWO	Data Wait of 2 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_ONE	Data Wait of 1 Peripheral Bus Clock Cycles

Description

PMP Data setup to read/write/address phase wait state configuration

This data type defines the different wait state configurations for setup to read/write/address phase.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INCREMENT_MODE Enumeration

PMP address incrementing configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_BUFFERS_AUTO_INCREMENT,
  PMP_ADDRESS_AUTO_DECREMENT,
  PMP_ADDRESS_AUTO_INCREMENT,
  PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE
} PMP_INCREMENT_MODE;
```

Members

Members	Description
PMP_BUFFERS_AUTO_INCREMENT	Read and write buffers auto-increment
PMP_ADDRESS_AUTO_DECREMENT	Decrement PMP Address by one every read/write cycle
PMP_ADDRESS_AUTO_INCREMENT	Increment PMP Address by one every read/write cycle
PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE	PMP Address does not change

Description

PMP Increment Modes

This data type defines the different configurations by which the PMP address incrementing can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INPUT_BUFFER_TYPE Enumeration

PMP Input Buffers type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_INPUT_BUFFER_TTL,
  PMP_INPUT_BUFFER_SCHMITT
} PMP_INPUT_BUFFER_TYPE;
```

Members

Members	Description
PMP_INPUT_BUFFER_TTL	TTL input buffer
PMP_INPUT_BUFFER_SCHMITT	Schmitt trigger input buffer

Description

PMP Input Buffers type

This data type defines the input buffer types.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INTERRUPT_MODE Enumeration

PMP interrupt request mode data type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_INTERRUPT_BUFFER_3_IS_ACCESSED,
  PMP_INTERRUPT_EVERY_RW_CYCLE,
```

```
  PMP_INTERRUPT_NONE
} PMP_INTERRUPT_MODE;
```

Members

Members	Description
PMP_INTERRUPT_BUFFER_3_IS_ACCESSED	Interrupt generated when Read/Write Buffer 3 is read/written
PMP_INTERRUPT_EVERY_RW_CYCLE	Interrupt Occurs Every Read/Write Cycle
PMP_INTERRUPT_NONE	No interrupt generated

Description

PMP Interrupt Mode

This data type defines the different configurations by which the PMP can be configured for interrupt requests.

Remarks

None.

PMP_MASTER_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ALTERNATE_MASTER_DIRECT,
  PMP_ALTERNATE_MASTER,
  PMP_CPU_MASTER
} PMP_MASTER_MODE;
```

Members

Members	Description
PMP_ALTERNATE_MASTER_DIRECT	Alternate Master has PMP control with Direct I/O access
PMP_ALTERNATE_MASTER	Alternate Master has PMP control
PMP_CPU_MASTER	CPU has PMP control

Description

PMP Master Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_MODULE_ID Enumeration

Possible instances of the PMP module.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ID_0
} PMP_MODULE_ID;
```

Members

Members	Description
PMP_ID_0	first instance of the PMP

Description

PMP module ID

This data type defines the possible instances of the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_MUX_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_MUX_ADDRESS_PHASES_3,
  PMP_MUX_ADDRESS_PHASES_2,
  PMP_MUX_ADDRESS_PHASES_1,
  PMP_MUX_LINES_16_ADDRESS_16_DATA,
  PMP_MUX_LINES_16_ADDRESS_8_DATA,
  PMP_MUX_LINES_8_ADDRESS_8_DATA,
  PMP_MUX_NONE
} PMP_MUX_MODE;
```

Members

Members	Description
PMP_MUX_ADDRESS_PHASES_3	Lower address bits are multiplexed with data bits using 3 address phase
PMP_MUX_ADDRESS_PHASES_2	Lower address bits are multiplexed with data bits using 2 address phases
PMP_MUX_ADDRESS_PHASES_1	Lower address bits are multiplexed with data bits using 1 address phase
PMP_MUX_LINES_16_ADDRESS_16_DATA	16 address lines are multiplexed on 16 data lines
PMP_MUX_LINES_16_ADDRESS_8_DATA	16 address lines multiplexed on 8 data lines
PMP_MUX_LINES_8_ADDRESS_8_DATA	8 bits of address is multiplexed on 8 data lines
PMP_MUX_NONE	No multiplexing of address and data lines

Description

PMP Multiplex Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_OPERATION_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED,
  PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT,
  PMP_BUFFERED_SLAVE,
  PMP_ENHANCED_SLAVE,
  PMP_LEGACY_SLAVE
} PMP_OPERATION_MODE;
```

Members

Members	Description
PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED	Master mode 1, the read and the write strobes share a single line. The enable strobe is used to decode the info sent on read/write strobe line
PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT	Master mode 2, the read and write strobes are on independent lines
PMP_BUFFERED_SLAVE	Buffered Slave mode
PMP_ENHANCED_SLAVE	Enhanced Slave mode

PMP_LEGACY_SLAVE	Legacy Parallel Slave mode

Description

PMP Operation Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_PMBE_PORT Enumeration

Defines the available Byte Enable ports.

File

plib_pmp_help.h

C

```
typedef enum {
  PMBE_0,
  PMBE_1
} PMP_PMBE_PORT;
```

Members

Members	Description
PMBE_0	Byte Enable Port-0
PMBE_1	Byte Enable Port-1

Description

PMP Byte Enable port.

This data type defines the available Byte Enable ports.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_POLARITY_LEVEL Enumeration

Possible polarity levels for the PMP pins.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_POLARITY_ACTIVE_HIGH,
  PMP_POLARITY_ACTIVE_LOW
} PMP_POLARITY_LEVEL;
```

Members

Members	Description
PMP_POLARITY_ACTIVE_HIGH	Active high polarity
PMP_POLARITY_ACTIVE_LOW	Active low polarity

Description

PMP Pins Polarity

This data type defines the possible polarity levels for the PMP pins.

Remarks

None.

PMP_STROBE_WAIT_STATES Enumeration

PMP strobe signal wait time configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_STROBE_WAIT_FIFTEEN,
  PMP_STROBE_WAIT_FOURTEEN,
  PMP_STROBE_WAIT_THIRTEEN,
  PMP_STROBE_WAIT_TWELVE,
  PMP_STROBE_WAIT_ELEVEN,
  PMP_STROBE_WAIT_TEN,
  PMP_STROBE_WAIT_NINE,
  PMP_STROBE_WAIT_EIGHT,
  PMP_STROBE_WAIT_SEVEN,
  PMP_STROBE_WAIT_SIX,
  PMP_STROBE_WAIT_FIVE,
  PMP_STROBE_WAIT_FOUR,
  PMP_STROBE_WAIT_THREE,
  PMP_STROBE_WAIT_TWO,
  PMP_STROBE_WAIT_ONE,
  PMP_STROBE_WAIT_NONE
} PMP_STROBE_WAIT_STATES;
```

Members

Members	Description
PMP_STROBE_WAIT_FIFTEEN	Strobe Wait of 15 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOURTEEN	Strobe Wait of 14 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THIRTEEN	Strobe Wait of 13 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWELVE	Strobe Wait of 12 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ELEVEN	Strobe Wait of 11 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TEN	Strobe Wait of 10 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_NINE	Strobe Wait of 9 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_EIGHT	Strobe Wait of 8 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SEVEN	Strobe Wait of 7 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SIX	Strobe Wait of 6 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FIVE	Strobe Wait of 5 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOUR	Strobe Wait of 4 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THREE	Strobe Wait of 3 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWO	Strobe Wait of 2 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ONE	Strobe Wait of 1 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_NONE	No wait states

Description

PMP Read to Byte enable strobe configuration

This data type defines the different configurations by which the PMP strobe wait signal time can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_pmp.h	Parallel Master Port (PMP) Peripheral Library Interface Header.
plib_pmp_help.h	Parallel Master Port (PMP) Peripheral Library Help Header.

Description

This section lists the source and header files used by the library.

plib_pmp.h

Parallel Master Port (PMP) Peripheral Library Interface Header.

Functions

	Name	Description
	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.
	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_DualBufferDisable	Disables the specific PMP module.
	PLIB_PMP_DualBufferEnable	Enables PMP dual Read/Write buffer.
	PLIB_PMP_DualBufferIsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_DualModeMasterReceive	Receives the data in the Master Dual mode.
	PLIB_PMP_DualModeMasterSend	Sends the specified data in Dual Master mode.
	PLIB_PMP_DualModeReadAddressGet	Gets the current read address of the PMP module.
	PLIB_PMP_DualModeReadAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeWriteAddressGet	Gets the current write address of the PMP module.
	PLIB_PMP_DualModeWriteAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_ExistsAddressControl	Identifies whether the AddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchPolarity	Identifies whether the AddressLatchPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchStrobePortControl	Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressPortPinControl	Identifies whether the AddressPortPinControl feature exists on the PMP module.
	PLIB_PMP_ExistsBufferOverFlow	Identifies whether the BufferOverFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferRead	Identifies whether the BufferRead feature exists on the PMP module.
	PLIB_PMP_ExistsBufferType	Identifies whether the BufferType feature exists on the PMP module.
	PLIB_PMP_ExistsBufferUnderFlow	Identifies whether the BufferUnderFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferWrite	Identifies whether the BufferWrite feature exists on the PMP module.
	PLIB_PMP_ExistsBusyStatus	Identifies whether the BusyStatus feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectoperation	Identifies whether the ChipSelectoperation feature exists on the PMP module.
	PLIB_PMP_ExistsChipXPolarity	Identifies whether the ChipXPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsCSXActiveStatus	Identifies whether the CSXActiveStatus feature exists on the PMP module.
	PLIB_PMP_ExistsDataHoldWaitStates	Identifies whether the DataHoldWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataSetUpWaitStates	Identifies whether the DataSetUpWaitStates feature exists on the PMP module.

	PLIB_PMP_ExistsDataStrobeWaitStates	Identifies whether the DataStrobeWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataTransferSize	Identifies whether the DataTransferSize feature exists on the PMP module.
	PLIB_PMP_ExistsDualBufferControl	Identifies whether the DualBufferControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeMasterRXTX	Identifies whether the DualModeMasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeReadAddressControl	Identifies whether the DualModeReadAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeWriteAddressControl	Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the PMP module.
	PLIB_PMP_ExistsIncrementMode	Identifies whether the IncrementMode feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferFull	Identifies whether the InputBufferFull feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferXStatus	Identifies whether the InputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsInterruptMode	Identifies whether the InterruptMode feature exists on the PMP module.
	PLIB_PMP_ExistsMasterRXTX	Identifies whether the MasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsMUXModeSelect	Identifies whether the MUXModeSelect feature exists on the PMP module.
	PLIB_PMP_ExistsOperationMode	Identifies whether the OperationMode feature exists on the PMP module.
	PLIB_PMP_ExistsOutPutBufferEmpty	Identifies whether the OutPutBufferEmpty feature exists on the PMP module.
	PLIB_PMP_ExistsOutputBufferXStatus	Identifies whether the OutputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsReadChipSelectEnable	Identifies whether the ReadChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsReadWritePolarity	Identifies whether the ReadWritePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsReadWriteStrobePortControl	Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveRX	Identifies whether the SlaveRX feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveTX	Identifies whether the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsStartReadControl	Identifies whether the StartReadControl feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies whether the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsWriteChipSelectEnable	Identifies whether the WriteChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies whether the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies whether the WriteEnablePortControl feature exists on the PMP module.
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.
	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_MasterReceive	Receives the data in Master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.

	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.
	PLIB_PMP_PortIsBusy	Identifies if the (Master mode) PMP port is busy.
	PLIB_PMP_ReadChipSelectXDisable	Configures the Read Chip Select.
	PLIB_PMP_ReadChipSelectXEnable	Configures the Read Chip Select.
	PLIB_PMP_ReadCycleIsStarted	Checks whether or not the read cycle on PMP bus is enabled.
	PLIB_PMP_ReadCycleStart	Starts a read cycle on the PMP bus.
	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.
	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WriteChipSelectXDisable	Configures the Write Chip Select.
	PLIB_PMP_WriteChipSelectXEnable	Configures the Write Chip Select.
	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.
	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

Description

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the PMP Peripheral Library.

File Name

plib_pmp.h

Company

Microchip Technology Inc.

plib_pmp_help.h

Parallel Master Port (PMP) Peripheral Library Help Header.

Enumerations

	Name	Description
	PMP_ACK_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES	PMP hold after address latch strobe wait states configuration.
	PMP_ADDRESS_LATCH	Address Latch Strobe configuration.
	PMP_ADDRESS_LATCH_WAIT_STATES	PMP address latch strobe wait states configuration.
	PMP_ADDRESS_PORT	Defines the different address lines that are available for configuration.
	PMP_ALTERNATE_MASTER_WAIT_STATES	PMP alternate master wait states.
	PMP_CHIP_SELECT	PMP Chip Select data type.
	PMP_CHIPSELECT_FUNCTION	Defines different functions available for the Chip Select lines multiplexed with address lines.
	PMP_DATA_HOLD_STATES	PMP Data Hold after strobe wait state.
	PMP_DATA_LENGTH	Possible data lengths handled by the PMP module.
	PMP_DATA_SIZE	PMP data size.
	PMP_DATA_WAIT_STATES	PMP data setup time configuration.
	PMP_INCREMENT_MODE	PMP address incrementing configuration.
	PMP_INPUT_BUFFER_TYPE	PMP Input Buffers type.
	PMP_INTERRUPT_MODE	PMP interrupt request mode data type.

	PMP_MASTER_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_MODULE_ID	Possible instances of the PMP module.
	PMP_MUX_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_OPERATION_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_PMBE_PORT	Defines the available Byte Enable ports.
	PMP_POLARITY_LEVEL	Possible polarity levels for the PMP pins.
	PMP_STROBE_WAIT_STATES	PMP strobe signal wait time configuration.

Description

PMP/EPMP Peripheral Library Help Header

This header file contains the data types and constants that make up the interface to the PMP Peripheral Library.

File Name

plib_pmp_help.h

Company

Microchip Technology Inc.

Prefetch Cache Peripheral Library

This section describes the Prefetch Cache Peripheral Library.

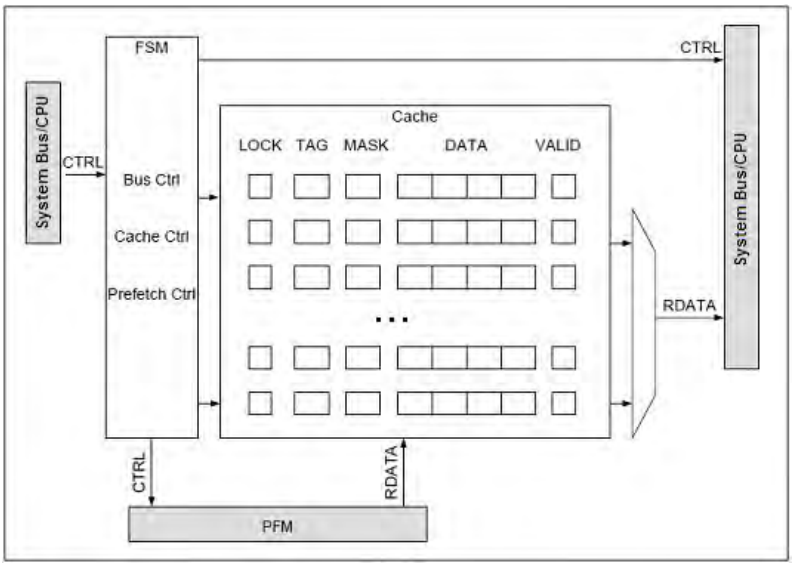
Introduction

This library provides a low-level abstraction of the Prefetch Cache module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Prefetch Cache module increases system performance for most applications by:

- Predictive prefetching of instructions ahead of the current program counter, hiding the access time of the Flash memory
- Instruction and data caching



The program Flash memory (PFM) contains a Prefetch module and a number of cache lines, each containing four words. Some of the cache lines may be allocated to data or peripherals. Each cache line has an associated tag. The tag contains the address of the data in the cache line, and a number of status bits. The number of cache lines and number and type of status bits may vary between devices. Refer to your specific device data sheet for details.

Using the Library

This topic describes the basic architecture of the Prefetch Cache Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_pcache.h`

The interface to the Prefetch Cache Peripheral Library is defined in the `plib_pcache.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Prefetch Cache Peripheral Library must include `peripheral.h`.

Library File:

The Prefetch Cache Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Prefetch Cache module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

The Prefetch Cache module is a performance enhancing module included in some microcontrollers. When running at high clock rates, wait states must be inserted into program Flash memory (PFM) read transactions to meet the access time of the PFM. Wait states can be hidden to the core

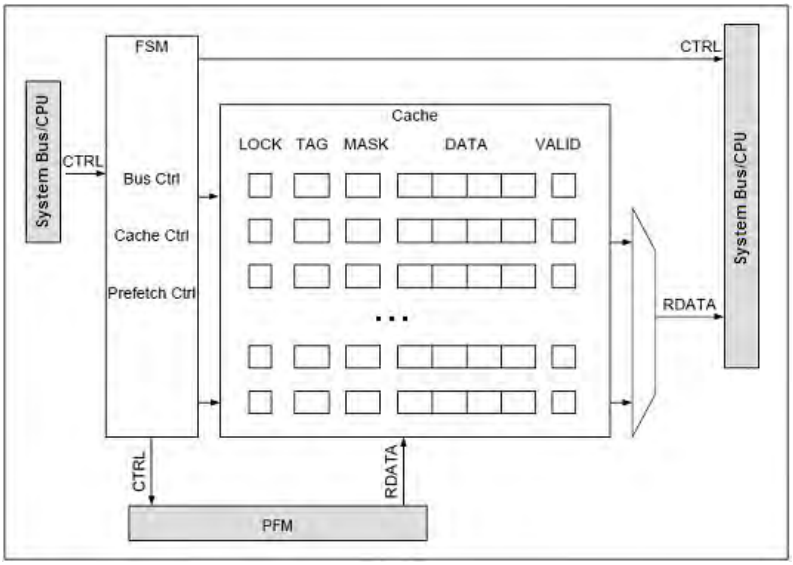
by prefetching and storing instructions in a temporary holding area that the CPU can access quickly.

There are two main functions that the Prefetch Cache module performs: Caching instructions when they are accessed, and prefetching instructions from the PFM before they are needed.

The cache holds a subset of the cacheable memory in temporary holding spaces known as cache lines. Each cache line has a tag describing what it is currently holding, and the address where it is mapped. Normally, the cache lines just hold a copy of what is currently in memory to make data available to the CPU without wait states.

CPU-requested data may or may not be in the cache. A cache-miss occurs if the CPU requests cacheable data that is not in the cache. In this case, a read is performed to the PFM at the correct address, the data is supplied to the cache and to the CPU. A cache-hit occurs if the cache contains the data that the CPU requests. In the case of a cache-hit, data is supplied to the CPU without wait states.

The second main function of the Prefetch Cache module is to prefetch CPU instructions. The module calculates the address of the next cache line and performs a read of the PFM to get the next cache line. This line is placed into a prefetch cache buffer in anticipation of executing straight-line code.



Cache Control

This library provides a set of functions to control the behavior of the overall Cache Module. Those functions include:

- Setting or reading the number of clock wait cycles
- Setting or reading the number of cache lines allocated to data
- Controlling which cache lines are invalidated on a PFM write cycle

Cache Line Operations

This library provides a set of functions to read, write and control the behavior of individual Cache Lines. Operations on individual Cache Lines are performed using a two-step process:

1. Select the individual cache line with PLIB_PCACHE_CacheLineSelect.
2. Perform the required cache line operation using any of the routines in the Cache Line Operations section. The operations available are:
   - Set the cache line to data or instruction type
   - Mark the cache line as valid or invalid
   - Lock or unlock the cache line
   - Read or write the Flash type (boot or program) associated with the cache line
   - Read or write the address associated with the cache line
   - Read or write the mask field to force a match between the cache line and multiple physical addresses
   - Read or write any individual word in the cache line data array

Once the operation is performed, the cache line remains selected until PLIB_PCACHE_CacheLineDeselect is performed. Only one cache line can be selected at a time.

Cache Status

The cache maintains a count of hits and misses for profiling purposes. The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. Accesses to non-cacheable regions do not affect the counters. These values can be read or written using the Cache Status routines.

The cache uses a Least Recently Used (LRU) algorithm to replace cache lines on a miss. The specific algorithm may vary between devices. In some devices, the state of the LRU algorithm encoding bits can be read with PLIB_PCACHE_LeastRecentlyUsedStateRead.

Prefetch Control

Predictive prefetch is controlled by the Prefetch Control routines. Devices with no L1 CPU cache can prefetch from cacheable or non-cacheable regions or both. Devices with L1 CPU cache can prefetch from any address, CPU instructions or CPU instructions and data. Predictive prefetch

can be disabled for either type of device.

Prefetch Status

The Prefetch Cache module maintains a count of prefetch aborts for profiling purposes. The prefetch abort counter is incremented each time a prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can be read or written using the Prefetch Status routines.

Flash ECC

On some devices, the Flash ECC module has the ability to detect single and double bit errors on PFM reads. When a single bit error is detected, the error is corrected and the Single Error Correction (SEC) counter is decremented. When the SEC counter reaches zero, an interrupt is generated to the CPU. The user has the ability to enable or disable the SEC interrupt. It is disabled by default. The user may set the SEC counter value at any time. The maximum counter value may be device-specific and is defined for each processor by the PLIB_PCACHE_MAX_SEC_COUNT constant.

When a double bit error is detected, the device will set the DED status to true and generate a bus exception to the initiator.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Prefetch Cache module.

Library Interface Section	Description
Cache Control Functions	Provides Cache Control interface routines.
Cache Line Functions	Provides Cache Line interface routines.
Cache Status Functions	Provides Cache Status interface routines.
Prefetch Control Functions	Provides Prefetch Control interface routines.
Prefetch Status Functions	Provides Prefetch Status interface routines.
Flash ECC Functions	Provides Flash ECC interface routines.
Feature Existence Functions	Provides interface routines that determine whether or not certain features are available.

How the Library Works

Provides information on how the library works.

Description

This library can be used to optimize the performance of the processor while executing out of cacheable program Flash memory. This is done by implementing the following:

- Instruction caching
- Data caching
- Cache line control
- Predictive prefetching
- ECC detection

Arrays

The cache consists of two arrays: tag and data. The data array contains a number of lines of program instructions or data words. The word size is the same as the processor data width.

The tag array contains a number of corresponding fields, each containing:

- Mask – address mask value
- Tag – tag address to match against
- Valid – indicates whether the data in the cache line is valid or not
- Lock – indicates the line is locked to hold its contents
- Type – instruction, data or peripheral

Note: Not all Tag fields are available on all devices. The number of cache lines may vary between devices. Refer to the specific device data sheet for details.

Example Cache Line

Operations

This library allows the programmer to perform various operations on the cache tag and data arrays:

- Enable/Disable
- Set the Wait state
- Read, write, lock, invalidate

Note: Not all operations are available on all devices. Refer to the specific device data sheet for details.

Cache Control Operations

Cache Wait States

When operating at high clock rates, the Prefetch Cache module must insert wait states into Program Flash Memory (PFM) read operations to meet PFM access time requirements. One Wait state is equivalent to one clock cycle delay. At reset, the Prefetch Cache Module is set to the maximum number of wait states for the device. The PFM read time can be optimized for the device by selecting the minimum number of wait states for the clock rate used. The number of wait states for a given clock frequency may vary between devices.

Note: Refer to the specific device data sheet for details on how to calculate the optimum number of wait states for your device.

Example: Setting the Number of PFM wait states based on the System Clock

This example assumes one wait state for every 20 MHz of system clock.

```
if (sysclk < 20000000)
{
 PLIB_PCACHE_WaitStateSet(0);
}
else if (sysclk < 40000000)
{
 PLIB_PCACHE_WaitStateSet(1);
}
...
```

Data Caching

Some Prefetch Cache modules allow the user to allocate cache lines to data. This is to provide improved CPU access to constant data stored in PFM. By default, data caching is disabled. The Prefetch Cache Peripheral Library allows the user to set the number of cache lines allocated to data. The number of lines available may vary among devices. The available settings are provided by the device-dependent PLIB_PCACHE_DATA_ENABLE enumeration.

Example: Allocating two Cache Lines to Data

```
PLIB_PCACHE_DataCacheEnableSet(PLIB_PCACHE_DATA_2LINE);
```

Cache Line Invalidating on PFM Write

It is not possible to execute out of cache while programming the PFM. The PFM controller stalls the cache during the programming sequence. Therefore, user code that initiates a programming sequence should not be located in a cacheable address region.

During program operation, the Prefetch Cache is flushed by invalidating some or all of the cache lines. To invalidate all of the cache lines on a PFM write operation, the function PLIB_PCACHE_InvalidateOnPFMProgramAll should be called prior to the PFM write. This will invalidate and unlock every cache line during the write operation. The cache tags and masks are also cleared for all lines.

If PLIB_PCACHE_InvalidateOnPFMProgramUnlocked is called before the PFM write, only lines that are not locked will be forced invalid.

By default, only unlocked cache lines are invalidated during a PFM write.

Cache Line Operations

Performing a cache line operation is a two-step process. Before calling a Cache Line operation function, the cache line must be selected and write-enabled. Once selected and write-enabled, any number of cache line operations may be performed on the selected line until the line is cleared or a different line is selected and write-enabled.

Cache lines are selected and write-enabled using PLIB_PCACHE_CacheLineSelect. Cache lines are write-disabled using PLIB_PCACHE_CacheLineDeselect.

The following example shows operations on two different cache lines. The code comments describe each operation.

Example: Cache Line Operations

```
uint32_t cacheLine[4] = {0x01234567, 0x12345678, 0x23456789, 0x34567890};

//Select Cache Line 4
PLIB_PCACHE_CacheLineSelect(4);

//Invalidate Cache Line 4
```

```
PLIB_PCACHE_CacheLineInvalid();

//Select Cache Line 6
PLIB_PCACHE_CacheLineSelect(6);

//Set Cache Line 6 to Data Type
PLIB_PCACHE_CacheLineData();

//Write Data to Cache Line 6
for (i=0; i<PLIB_PCACHE_NUM_WORDS_PER_LINE; i++)
{
    PLIB_PCACHE_WordWrite(i, cacheLine[i]);
}

//Lock Cache Line 6
PLIB_PCACHE_CacheLineLock();

//Disable Writes to Cache Line 6
PLIB_PCACHE_CacheLineDeselect();
```

Cache Status Operations

Cache Statistics Counters

It is often useful to gather statistics related to cache utilization for performance or profiling reasons. Some devices with a Prefetch Cache module maintain cache hit and miss counters for this purpose. This library provides routines to read and write the cache hit and miss counters.

The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Non-cacheable accesses do not modify this value. This value can be read with PLIB_PCACHE_CacheHitRead.

Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. This value can be read with PLIB_PCACHE_CacheMissRead.

Cache Replacement State

The Prefetch Cache module uses a pseudo-Least Recently Used (LRU) algorithm to replace cache lines. The specific algorithm may vary among devices.

Note: Refer to the specific device data sheet for details on a specific device.

Some devices maintain the state of the LRU algorithm for use by software. The state of the LRU algorithm can be read with PLIB_PCACHE_LeastRecentlyUsedStateRead.

Prefetch Control Operations

The Prefetch Cache module is configured differently for different CPU configurations. For devices with a CPU L1 cache, the Prefetch Cache module holds one line of CPU instructions, one line of CPU data, and two lines of peripheral data. For devices without a CPU L1 cache, the Prefetch Cache module holds up to 16 lines of instructions and up to four lines of data. Control of either configuration is done using the PLIB_PCACHE_PrefetchEnableSet function.

Note: Refer to the specific device data sheet for the Prefetch Cache module configuration for a specific device.

For the CPU L1 configuration, the Prefetch Cache module can be configured to enable any address, CPU instructions and data, or CPU instructions only. For devices without a CPU L1 cache, predictive prefetch can be enabled for cacheable regions, uncacheable regions, or both. In either configuration, predictive prefetch may be disabled entirely.

The PLIB_PCACHE_PrefetchEnableSet and PLIB_PCACHE_PrefetchEnableGet functions take or return the enumerated value PLIB_PCACHE_PREFETCH_ENABLE. This enumeration will be defined differently for different CPU configurations.

Example: Enable Prefetch for Non-Cacheable Regions Only (no L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS);
```

Example: Enable Prefetch for CPU Instructions and CPU Data (L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA);
```

Prefetch Status Operations

Prefetch Abort Statistics Counter

Much like the cache hit/miss counters, the Prefetch Abort module maintains a prefetch abort counter. This counter is incremented each time an automatic prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can be read using the PLIB_PCACHE_PrefetchAbortRead function.

Flash ECC Operations

The Prefetch Cache module may handle and report information about two error types: ECC Double-bit Error Detected (DED) and ECC Single-bit Error Corrected (SEC).

A read from the Flash memory that results in a program Flash memory ECC DED causes the Prefetch Cache module to return a bus exception to the initiator. An ECC DED will also set the DED status bit. The state of the DED status bit may be read using the PLIB_PCACHE_FlashDEDStatusGet function.

A single-bit error read from the Flash memory is not a critical error, and as such is reported as an interrupt. The user has the option to enable or disable the SEC interrupt, and to set the number of SEC events that must occur before an SEC interrupt is generated. An SEC interrupt is generated when all of the following conditions are met:

- The Flash SEC counter is zero
- The Flash SEC interrupt is enabled
- An SEC event occurs

The Flash SEC counter is decremented once each time an SEC event occurs, and holds at zero. When a Flash SEC interrupt is triggered, the Flash SEC status bit is set. Once an SEC interrupt occurs, the Flash SEC status must be cleared by software and the Flash SEC counter must be reloaded by software.

Example: Set Flash SEC to Interrupt after three SEC events

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(3);
PLIB_PCACHE_FlashSECIntEnable();
```

It is possible to generate an SEC interrupt for testing purposes by enabling the Flash SEC interrupt and setting the Flash SEC status while the Flash SEC counter is zero.

Example: Generate an SEC Interrupt

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(0);
PLIB_PCACHE_FlashSECIntEnable();
PLIB_PCACHE_FlashSECStatusSet();
```

Configuring the Library

The library is configured for the supported Prefetch Cache module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Cache Control Functions

	Name	Description
	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_WaitStateGet	Gets the Prefetch Cache access time.
	PLIB_PCACHE_WaitStateSet	Sets the Prefetch Cache access time.

b) Cache Line Functions

	Name	Description
	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.
	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.

	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.
	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.
	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

c) Cache Status Functions

	Name	Description
	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache Least Recently Used (LRU) encoding bits.

d) Prefetch Control Functions

	Name	Description
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive Prefetch state for the Prefetch Cache module.

e) Prefetch Status Functions

	Name	Description
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.

f) Flash ECC Functions

	Name	Description
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.

g) Feature Existence Functions

	Name	Description
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.

	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMProgram	Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.

h) Data Types and Constants

	Name	Description
	PLIB_PCACHE_MAX_SEC_COUNT	Defines the maximum value for single bit error counter.
	PLIB_PCACHE_NUM_LINES	Defines the number of available Prefetch Cache Lines.
	PLIB_PCACHE_NUM_WORDS_PER_LINE	Defines the number of Words per Prefetch Cache Line.
	PCACHE_MODULE_ID	Lists the possible Module IDs for the Prefetch Cache.
	PLIB_PCACHE_DATA_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.
	PLIB_PCACHE_PREFETCH_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.

Description

This section describes the Application Programming Interface (API) functions of the Prefetch Cache Peripheral Library.

Refer to each section for a detailed description.

a) Cache Control Functions

PLIB_PCACHE_DataCacheEnableGet Function

Gets the data cache enable bits.

File

plib_pcache.h

C

```
PLIB_PCACHE_DATA_ENABLE PLIB_PCACHE_DataCacheEnableGet(PCACHE_MODULE_ID index);
```

Returns

PLIB_PCACHE_DATA_ENABLE

Description

This function gets the data cache enable bits.

Remarks

At reset, data caching is disabled, and this function will return PLIB_PCACHE_DATA_DISABLE.

This function implements an operation of the DataCacheEnable feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use PLIB_RTCC_ExistsDataCacheEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_DATA_ENABLE dcache_en;
dcache_en = PLIB_PCACHE_DataCacheEnableGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_PCACHE_DATA_ENABLE PLIB_PCACHE_DataCacheEnableGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_DataCacheEnableSet Function

Sets the data cache enable bits.

File

plib_pcache.h

C

```
void PLIB_PCACHE_DataCacheEnableSet(PCACHE_MODULE_ID index, PLIB_PCACHE_DATA_ENABLE dcache_en);
```

Returns

None.

Description

This function sets the data cache enable bits.

Remarks

Data caching is disabled at reset.

This function implements an operation of the DataCacheEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsDataCacheEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_DataCacheEnableSet(PCACHE_ID_0, PLIB_PCACHE_DATA_4LINE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dcache_en	PLIB_PCACHE_DATA_ENABLE

Function

void PLIB_PCACHE_DataCacheEnableSet(PCACHE_MODULE_ID index,
PLIB_PCACHE_DATA_ENABLE dcache_en)

PLIB_PCACHE_InvalidateOnPFMProgramAll Function

Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.

File

plib_pcache.h

C

```
void PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.

Remarks

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsInvalidateOnPFMProgram in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_MODULE_ID index)

PLIB_PCACHE_InvalidateOnPFMProgramUnlocked Function

Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.

File

plib_pcache.h

C

```
void PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.

Remarks

This is the default state at reset.

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsInvalidateOnPFMProgram in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_MODULE_ID index)

PLIB_PCACHE_WaitStateGet Function

Gets the Prefetch Cache access time.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_WaitStateGet(PCACHE_MODULE_ID index);
```

Returns

Number of clock cycles.

Description

This function gets the Prefetch Cache access time in terms of SYSCLK wait. states.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t wait;
wait = PLIB_PCACHE_WaitStateGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_WaitStateGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_WaitStateSet Function

Sets the Prefetch Cache access time.

File

plib_pcache.h

C

```
void PLIB_PCACHE_WaitStateSet(PCACHE_MODULE_ID index, uint32_t clocks);
```

Returns

None.

Description

This function sets the Prefetch Cache access time in terms of SYSCLK wait. states.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_WaitStateSet(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clocks	Identifies the wait state in terms of number of SYSCLK cycles

Function

void PLIB_PCACHE_WaitStateSet(PCACHE_MODULE_ID index, uint32_t clocks)

b) Cache Line Functions

PLIB_PCACHE_CacheLineAddrGet Function

Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheLineAddrGet(PCACHE_MODULE_ID index);
```

Returns

The address in the cache line TAG register.

Description

This function gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineAddr in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t addr;
addr = PLIB_PCACHE_CacheLineAddrGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheLineAddrGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineAddrSet Function

Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineAddrSet(PCACHE_MODULE_ID index, uint32_t addr);
```

Returns

None.

Description

This function Sets the TAG address in the cache line previously write- enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineAddr in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t addr = 0x1FC01234;
PLIB_PCACHE_CacheLineAddrSet(PCACHE_ID_0, addr);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
addr	Address to program into the cache line TAG register

Function

void PLIB_PCACHE_CacheLineAddrSet(PCACHE_MODULE_ID index, uint32_t addr)

PLIB_PCACHE_CacheLineData Function

Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineData(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line previously enabled by PLIB_PCACHE_CacheLineSelect to data type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineData(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineData(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineDeselect Function

Disables write access to the cache line specified by cache_line.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineDeselect(PCACHE_MODULE_ID index, uint32_t cache_line);
```

Returns

None.

Description

This function disables write access to the cache line specified by cache_line.

Remarks

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineDeselect(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cache_line	Specifies the cache line to disable write access

Function

void PLIB_PCACHE_CacheLineDeselect(PCACHE_MODULE_ID index,
uint32_t cache_line)

PLIB_PCACHE_CacheLineFlashTypeBoot Function

Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineFlashTypeInst Function

Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineFlashTypeIsInst Function

Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line resides in Instruction Flash
- false - The cache line resides in Boot Flash

Description

This function returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing

in Instruction Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool LineFlashType;
LineFlashType = PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineInst Function

Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineInst(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineInvalid Function

Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineInvalid(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function invalidates the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineInvalid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineInvalid(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsInst Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsInst(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is instruction
- false - The cache line is data

Description

This function returns true if the cache line previously enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineType;
CacheLineType = PLIB_PCACHE_CacheLineIsInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_CacheLineIsInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsLocked Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsLocked(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is locked
- false - The cache line is unlocked

Description

This function returns true if the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineLocked;
CacheLineLocked = PLIB_PCACHE_CacheLineIsLocked(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineIsLocked(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsValid Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsValid(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is valid
- false - The cache line is invalid

Description

This function returns true if the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineValid;
CacheLineValid = PLIB_PCACHE_CacheLineIsValid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineIsValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineLock Function

Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineLock(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function locks the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineLock(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineLock(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineMaskGet Function

Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheLineMaskGet(PCACHE_MODULE_ID index);
```

Returns

- Address mask - Bits set to '1' will force a match (valid bits: <15:5>).

Description

This function returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineMask in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t mask;
mask = PLIB_PCACHE_CacheLineMaskGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheLineMaskGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineMaskSet Function

Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineMaskSet(PCACHE_MODULE_ID index, uint32_t mask);
```

Returns

None.

Description

This function sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineMask in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t mask = 0x0C0;
PLIB_PCACHE_CacheLineMaskSet(PCACHE_ID_0, mask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mask	Address mask (bits set to '1' will force a match; valid bits: <15:5>)

Function

void PLIB_PCACHE_CacheLineMaskSet(PCACHE_MODULE_ID index, uint32_t mask)

PLIB_PCACHE_CacheLineSelect Function

Enables write access to the cache line specified by cache_line.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineSelect(PCACHE_MODULE_ID index, uint32_t cache_line);
```

Returns

None.

Description

This function enables write access to the cache line specified by cache_line.

Remarks

At reset, all cache lines are read-only.

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineSelect(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cache_line	Specifies the cache line to enable write access

Function

void PLIB_PCACHE_CacheLineSelect(PCACHE_MODULE_ID index,

uint32_t cache_line)

PLIB_PCACHE_CacheLineUnlock Function

Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineUnlock(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineUnlock(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineUnlock(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineValid Function

Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineValid(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function validates the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineValid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_WordRead Function

Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_WordRead(PCACHE_MODULE_ID index, uint32_t word);
```

Returns

A 32-bit unsigned word from the cache data array.

Description

This function reads the word specified by word from the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t word;
word = PLIB_PCACHE_WordRead(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read
word	Location of the word in the data array to be read

Function

uint32_t PLIB_PCACHE_WordRead(PCACHE_MODULE_ID index, uint32_t word)

PLIB_PCACHE_WordWrite Function

Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_WordWrite(PCACHE_MODULE_ID index, uint32_t word, uint32_t data);
```

Returns

None.

Description

This function writes the word (specified by word parameter) with data (specified by data parameter) to the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t word = 0;
uint32_t data = 0xDEADBEEF;
PLIB_PCACHE_WordWrite(PCACHE_ID_0, word, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written.
word	Location of the word in the data array to be written
data	32-bit unsigned word to write to cache data array

Function

void PLIB_PCACHE_WordWrite(PCACHE_MODULE_ID index, uint32_t word,

uint32_t data)

c) Cache Status Functions

PLIB_PCACHE_CacheHitRead Function

Reads the number of cache hits.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheHitRead(PCACHE_MODULE_ID index);
```

Returns

The number of cache hits.

Description

This function reads and returns the number of cache hits.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheHit in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t cache_hits;
cache_hits = PLIB_PCACHE_CacheHitRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheHitRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheHitWrite Function

Sets the number of cache hits.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheHitWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function sets the number of cache hits.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheHit in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheHitWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache hits to write

Function

void PLIB_PCACHE_CacheHitWrite(PCACHE_MODULE_ID index, uint32_t data)

PLIB_PCACHE_CacheMissRead Function

Reads the number of cache misses.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheMissRead(PCACHE_MODULE_ID index);
```

Returns

The number of cache misses.

Description

This function reads and returns the number of cache misses.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheMiss in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t cache_misses;
cache_misses = PLIB_PCACHE_CacheMissRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Function

uint32_t PLIB_PCACHE_CacheMissRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheMissWrite Function

Sets the number of cache misses.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheMissWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function sets the number of cache misses.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific

device data sheet to determine availability or use PLIB_RTCC_ExistsCacheMiss in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheMissWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache misses to write

Function

void PLIB_PCACHE_CacheMissWrite(PCACHE_MODULE_ID index, uint32_t data)

PLIB_PCACHE_LeastRecentlyUsedStateRead Function

Reads the state of the cache Least Recently Used (LRU) encoding bits.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_MODULE_ID index);
```

Returns

The state of the LRU encoding bits represented as a 25-bit unsigned integer.

Description

This function reads the state of the cache LRU encoding bits.

Remarks

This function implements an operation of the LeastRecentlyUsedState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsLeastRecentlyUsedState in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t lru_state;
lru_state = PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Function

uint32_t PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_MODULE_ID index)

d) Prefetch Control Functions

PLIB_PCACHE_PrefetchEnableGet Function

Gets the predictive Prefetch state for the Prefetch Cache module.

File

plib_pcache.h

C

```
PLIB_PCACHE_PREFETCH_ENABLE PLIB_PCACHE_PrefetchEnableGet(PCACHE_MODULE_ID index);
```

Returns

PLIB_PCACHE_PREFETCH_ENABLE

Description

This function gets the predictive Prefetch state for the Prefetch Cache module.

Remarks

At reset, the Prefetch Cache module is disabled and this function will return PLIB_PCACHE_DISABLE.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PREFETCH_ENABLE region;
region = PLIB_PCACHE_PrefetchEnableGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_PCACHE_PREFETCH_ENABLE PLIB_PCACHE_PrefetchEnableGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_PrefetchEnableSet Function

Sets the predictive Prefetch state for the Prefetch Cache module.

File

plib_pcache.h

C

```
void PLIB_PCACHE_PrefetchEnableSet(PCACHE_MODULE_ID index, PLIB_PCACHE_PREFETCH_ENABLE region);
```

Returns

None.

Description

This function sets the predictive Prefetch state for the Prefetch Cache module.

Remarks

At reset, the Prefetch Cache module is disabled.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PrefetchEnableSet(PCACHE_ID_0,
                              PLIB_PCACHE_PREFETCH_ENABLE_ALL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

region	PLIB_PCACHE_PREFETCH_ENABLE

Function

void PLIB_PCACHE_PrefetchEnableSet(PCACHE_MODULE_ID index,
PLIB_PCACHE_PREFETCH_ENABLE region)

e) Prefetch Status Functions

PLIB_PCACHE_PrefetchAbortRead Function

Reads the number of prefetch aborts.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_PrefetchAbortRead(PCACHE_MODULE_ID index);
```

Returns

The number of prefetch aborts.

Description

This function reads and returns the number of prefetch aborts.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchAbort in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t pfabt;
pfabt = PLIB_PCACHE_PrefetchAbortRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_PrefetchAbortRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_PrefetchAbortWrite Function

Sets the number of prefetch aborts.

File

plib_pcache.h

C

```
void PLIB_PCACHE_PrefetchAbortWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function Sets the number of prefetch aborts.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchAbort in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PrefetchAbortWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of prefetch aborts

Function

void PLIB_PCACHE_PrefetchAbortWrite(PCACHE_MODULE_ID index, uint32_t data)

f) Flash ECC Functions

PLIB_PCACHE_FlashDEDStatusClear Function

Clears a bus exception caused by a double-bit error.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashDEDStatusClear(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function clears a bus exception caused by a double-bit error.

Remarks

The DED status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashDEDStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashDEDStatusClear(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Function

void PLIB_PCACHE_FlashDEDStatusClear(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashDEDStatusGet Function

Determines if a bus exception was caused by a double-bit error.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_FlashDEDStatusGet(PCACHE_MODULE_ID index);
```

Returns

- true - A double-bit error was detected
- false - A double-bit error was not detected

Description

This function determines if a bus exception was caused by a double-bit error.

Remarks

The Double-bit Error Detected (DED) status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashDEDStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dedStatus;
dedStatus = PLIB_PCACHE_FlashDEDStatusGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_FlashDEDStatusGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECCountGet Function

Returns the current value of the Flash SEC counter.

File

plib_pcache.h

C

```
uint8_t PLIB_PCACHE_FlashSECCountGet(PCACHE_MODULE_ID index);
```

Returns

- count - Number of SEC events to occur before the SEC status is set to true

Description

This function returns the current value of the Flash SEC counter.

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECCount in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t count;
count = PLIB_PCACHE_FlashSECCountGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint8_t PLIB_PCACHE_FlashSECCountGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECCountSet Function

Sets the number of single-bit error corrections that must occur before the SEC status is set to true.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECCountSet(PCACHE_MODULE_ID index, uint8_t count);
```

Returns

None.

Description

This function sets the number of single-bit error corrections that must occur before the SEC status is set to true.

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECCount in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECCountSet(PCACHE_ID_0, 255);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Number of SEC events to occur before SEC status is set to true

Function

void PLIB_PCACHE_FlashSECCountSet(PCACHE_MODULE_ID index, uint8_t count)

PLIB_PCACHE_FlashSECIntDisable Function

Disables an interrupt on SEC.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECIntDisable(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function disables an interrupt on SEC.

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECInt in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECIntDisable(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_PCACHE_FlashSECIntDisable(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECIntEnable Function

Enables an interrupt on SEC.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECIntEnable(PCACHE_MODULE_ID index);
```

Returns

None

Description

This function enables an interrupt on SEC.

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECInt in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECIntEnable(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_PCACHE_FlashSECIntEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusClear Function

Sets the single-bit error corrected status to false.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECStatusClear(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the single-bit error corrected status to false.

Remarks

The SEC status must be cleared in response to a Prefetch Cache SEC interrupt.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECStatusClear(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Function

void PLIB_PCACHE_FlashSECStatusClear(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusGet Function

Determines if a SEC event triggered a Prefetch Cache interrupt.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_FlashSECStatusGet(PCACHE_MODULE_ID index);
```

Returns

- true - A single-bit error was corrected
- false - A double-bit error was not corrected

Description

This function determines if a SEC event triggered a Prefetch Cache interrupt.

Remarks

The SEC bit is set when a single-bit error is corrected and the SEC counter is zero. If the Flash SEC interrupt is enabled, an error event is reported to the CPU by a Prefetch Cache interrupt event.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool secStatus;
secStatus = PLIB_PCACHE_FlashSECStatusGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_FlashSECStatusGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusSet Function

Sets the single-bit error corrected status to true.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECStatusSet(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the single-bit error corrected status to true.

Remarks

This function is provided for code testing and debug purposes. Setting the SEC status while the SEC count is zero and the SEC interrupt is enabled will trigger a Prefetch Cache interrupt to the CPU.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECStatusSet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_FlashSECStatusSet(PCACHE_MODULE_ID index)

g) Feature Existence Functions

PLIB_PCACHE_ExistsCacheHit Function

Identifies that the CacheHit feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheHit(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheHit feature is supported on the device
- false - The CacheHit feature is not supported on the device

Description

This interface identifies that the CacheHit feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheHitRead

- PLIB_PCACHE_CacheHitWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheHit(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLine Function

Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLine(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineSelect feature is supported on the device
- false - The CacheLineSelect feature is not supported on the device

Description

This interface identifies that the CacheLineSelect feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineSelect
- PLIB_PCACHE_CacheLineDeselect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLine(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineAddr Function

Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineAddr(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineAddr feature is supported on the device
- false - The CacheLineAddr feature is not supported on the device

Description

This interface identifies that the CacheLineAddr feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineAddrSet
- PLIB_PCACHE_CacheLineAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineAddr(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineFlashType Function

Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineFlashType(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineFlashType feature is supported on the device
- false - The CacheLineFlashType feature is not supported on the device

Description

This interface identifies that the CacheLineFlashType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineFlashTypeBoot
- PLIB_PCACHE_CacheLineFlashTypeInst
- PLIB_PCACHE_CacheLineFlashTypeIsInst

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineFlashType(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineLock Function

Identifies that the CacheLineLock feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineLock(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineLock feature is supported on the device
- false - The CacheLineLock feature is not supported on the device

Description

This interface identifies that the CacheLineLock feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineLock
- PLIB_PCACHE_CacheLineUnlock
- PLIB_PCACHE_CacheLineIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineLock(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineMask Function

Identifies that the CacheLineMask feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineMask(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineMask feature is supported on the device
- false - The CacheLineMask feature is not supported on the device

Description

This interface identifies that the CacheLineMask feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineMaskSet
- PLIB_PCACHE_CacheLineMaskGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineMask(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineType Function

Identifies that the CacheLineType feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineType(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineType feature is supported on the device
- false - The CacheLineType feature is not supported on the device

Description

This interface identifies that the CacheLineType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineData
- PLIB_PCACHE_CacheLineInst
- PLIB_PCACHE_CacheLineIsInst

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineType(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineValid Function

Identifies that the CacheLineValid feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineValid(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineValid feature is supported on the device
- false - The CacheLineValid feature is not supported on the device

Description

This interface identifies that the CacheLineValid feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineValid
- PLIB_PCACHE_CacheLineInvalid
- PLIB_PCACHE_CacheLineIsValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheMiss Function

Identifies that the CacheMiss feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheMiss(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheMiss feature is supported on the device
- false - The CacheMiss feature is not supported on the device

Description

This interface identifies that the CacheMiss feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheMissRead
- PLIB_PCACHE_CacheMissWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheMiss(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsDataCacheEnable Function

Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsDataCacheEnable(PCACHE_MODULE_ID index);
```

Returns

- true - The DataCacheEnable feature is supported on the device
- false - The DataCacheEnable feature is not supported on the device

Description

This interface identifies that the DataCacheEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_DataCacheEnableSet
- PLIB_PCACHE_DataCacheEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsDataCacheEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashDEDStatus Function

Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashDEDStatus(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashDEDStatus feature is supported on the device
- false - The FlashDEDStatus feature is not supported on the device

Description

This interface identifies that the FlashDEDStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashDEDStatusGet
- PLIB_PCACHE_FlashDEDStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashDEDStatus(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECCount Function

Identifies that the FlashSECCount feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECCount(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECCount feature is supported on the device
- false - The FlashSECCount feature is not supported on the device

Description

This interface identifies that the FlashSECCount feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECCountSet
- PLIB_PCACHE_FlashSECCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECCount(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECInt Function

Identifies that the FlashSECInt feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECInt(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECInt feature is supported on the device
- false - The FlashSECInt feature is not supported on the device

Description

This interface identifies that the FlashSECInt feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECIntEnable
- PLIB_PCACHE_FlashSECIntDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECInt(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECStatus Function

Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECStatus(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECStatus feature is supported on the device
- false - The FlashSECStatus feature is not supported on the device

Description

This interface identifies that the FlashSECStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECStatusGet
- PLIB_PCACHE_FlashSECStatusSet
- PLIB_PCACHE_FlashSECStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECStatus(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsInvalidateOnPFMProgram Function

Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsInvalidateOnPFMProgram(PCACHE_MODULE_ID index);
```

Returns

- true - The InvalidateOnPFMProgram feature is supported on the device
- false - The InvalidateOnPFMProgram feature is not supported on the device

Description

This interface identifies that the InvalidateOnPFMProgram feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_InvalidateOnPFMProgramAll
- PLIB_PCACHE_InvalidateOnPFMProgramUnlocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsInvalidateOnPFMProgram(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsLeastRecentlyUsedState Function

Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsLeastRecentlyUsedState(PCACHE_MODULE_ID index);
```

Returns

- true - The LeastRecentlyUsedState feature is supported on the device
- false - The LeastRecentlyUsedState feature is not supported on the device

Description

This interface identifies that the LeastRecentlyUsedState feature is available on the Prefetch Cache module. When this interface returns true, this function is supported on the device:

- PLIB_PCACHE_LeastRecentlyUsedStateRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsLeastRecentlyUsedState(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsPrefetchAbort Function

Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsPrefetchAbort(PCACHE_MODULE_ID index);
```

Returns

- true - The PrefetchAbort feature is supported on the device
- false - The PrefetchAbort feature is not supported on the device

Description

This interface identifies that the PrefetchAbort feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_PrefetchAbortRead
- PLIB_PCACHE_PrefetchAbortWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsPrefetchAbort(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsPrefetchEnable Function

Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsPrefetchEnable(PCACHE_MODULE_ID index);
```

Returns

- true - The PrefetchEnable feature is supported on the device
- false - The PrefetchEnable feature is not supported on the device

Description

This interface identifies that the PrefetchEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_PrefetchEnableSet
- PLIB_PCACHE_PrefetchEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsPrefetchEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsWaitState Function

Identifies that the WaitState feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsWaitState(PCACHE_MODULE_ID index);
```

Returns

- true - The WaitState feature is supported on the device
- false - The WaitState feature is not supported on the device

Description

This interface identifies that the WaitState feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_WaitStateSet
- PLIB_PCACHE_WaitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsWaitState(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsWord Function

Identifies that the Word feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsWord(PCACHE_MODULE_ID index);
```

Returns

- true - The Word feature is supported on the device
- false - The Word feature is not supported on the device

Description

This interface identifies that the Word feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_WordRead
- PLIB_PCACHE_WordWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsWord(PCACHE_MODULE_ID index)

h) Data Types and Constants

PLIB_PCACHE_MAX_SEC_COUNT Macro

Defines the maximum value for single bit error counter.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_MAX_SEC_COUNT 255
```

Description

Maximum Value for Single Bit Error Counter

This definition specifies the maximum value for single bit error counter.

Remarks

The actual maximum value of the SEC counter is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_NUM_LINES Macro

Defines the number of available Prefetch Cache Lines.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_NUM_LINES
```

Description

Number of Cache Lines

This definition specifies the number of Prefetch Cache Lines available.

Remarks

The actual number of cache lines is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_NUM_WORDS_PER_LINE Macro

Defines the number of Words per Prefetch Cache Line.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_NUM_WORDS_PER_LINE
```

Description

Number of Words per Cache Line

This definition specifies the number of Words per Prefetch Cache Line.

Remarks

The actual number of words per cache line is processor specific and is defined in the processor-specific header files (see processor.h).

PCACHE_MODULE_ID Enumeration

Lists the possible Module IDs for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PCACHE_ID_1,
  PCACHE_NUMBER_OF_MODULES
} PCACHE_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the Prefetch Cache.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

PLIB_PCACHE_DATA_ENABLE Enumeration

Lists the possible predictive prefetch states for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PLIB_PCACHE_DATA_DISABLE,
  PLIB_PCACHE_DATA_1LINE,
  PLIB_PCACHE_DATA_2LINE,
  PLIB_PCACHE_DATA_4LINE
} PLIB_PCACHE_DATA_ENABLE;
```

Members

Members	Description
PLIB_PCACHE_DATA_DISABLE	Disable data caching
PLIB_PCACHE_DATA_1LINE	Enable data caching with a size of 1 Line
PLIB_PCACHE_DATA_2LINE	Enable data caching with a size of 2 Lines
PLIB_PCACHE_DATA_4LINE	Enable data caching with a size of 4 Lines

Description

Data Cache Enable

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_PREFETCH_ENABLE Enumeration

Lists the possible predictive prefetch states for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PLIB_PCACHE_PREFETCH_DISABLE,
  PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS,
  PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST,
  PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS,
  PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA,
  PLIB_PCACHE_PREFETCH_ENABLE_ALL
} PLIB_PCACHE_PREFETCH_ENABLE;
```

Members

Members	Description
PLIB_PCACHE_PREFETCH_DISABLE	Disable predictive prefetch enable
PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_ALL	Enable predictive prefetch cache for cacheable and non-cacheable regions

Description

Predictive Prefetch Cache Enable Bits

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_pcache.h	Defines the Prefetch Cache Peripheral Library Interface
help_plib_pcache.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_pcache.h

Defines the Prefetch Cache Peripheral Library Interface

Functions

	Name	Description
	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.
	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.
	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.
	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.
	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.

	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMProgram	Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.
	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache Least Recently Used (LRU) encoding bits.
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_WaitStateGet	Gets the Prefetch Cache access time.
	PLIB_PCACHE_WaitStateSet	Sets the Prefetch Cache access time.
	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Description

Prefetch Cache Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Prefetch Cache Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Prefetch Cache module.

File Name

plib_pcache.h

Company

Microchip Technology Inc.

help_plib_pcache.h

Enumerations

	Name	Description
	PCACHE_MODULE_ID	Lists the possible Module IDs for the Prefetch Cache.
	PLIB_PCACHE_DATA_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.
	PLIB_PCACHE_PREFETCH_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.

Macros

	Name	Description
	PLIB_PCACHE_MAX_SEC_COUNT	Defines the maximum value for single bit error counter.
	PLIB_PCACHE_NUM_LINES	Defines the number of available Prefetch Cache Lines.
	PLIB_PCACHE_NUM_WORDS_PER_LINE	Defines the number of Words per Prefetch Cache Line.

Description

This file is used for documentation purposes

Ports Peripheral Library

This section describes the Ports Peripheral Library.

Introduction

This library provides a low-level abstraction of the I/O Ports module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The general purpose I/O pins can be considered the simplest of peripherals. These I/O pins allow the microcontroller to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

A major challenge in general purpose devices is providing the largest possible set of peripheral features while minimizing the conflict of features on I/O pins. The Peripheral Pin Select (PPS) feature simplifies this challenge by enabling the user's peripheral set selection and their placement on a wide range of I/O pins. By increasing the pin out options available on a particular device, users can better tailor the microcontroller to their entire application. The PPS feature operates over a fixed subset of digital I/O pins. Users may independently map the input and/or output of any one of many digital peripherals to any one of these I/O pins. PPS is performed in software and generally does not require the device to be reprogrammed. Hardware safeguards are included that prevent accidental or spurious changes to the peripheral mapping once it has been established.

Note: Peripheral Pin Select is not available on all devices. Please refer to the specific device data sheet to determine availability of this feature.

Following are some of the key features of the I/O Ports module:

- Individual output pin/port open-drain control
- Individual input pin/port pull-up/down control
- Monitor select inputs and generate interrupt on mismatch condition (Change Notification)
- Operation during Sleep and Idle modes
- Port line analog/digital selection
- Port slew rate control
- PPS peripheral function remapping

Using the Library

This topic describes the basic architecture of the Ports Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ports.h`

The interface to the Ports Peripheral Library is defined in the `plib_ports.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Ports Peripheral Library must include `peripheral.h`.

Library File:

The Ports Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

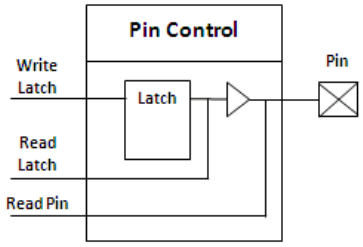
Hardware Abstraction Model

This topic provides detailed information for general purpose I/O and pin mapping.

Description

General Purpose I/O

All port pins have three controls directly associated with their operation as digital I/O. The data direction control determines whether the pin is an input or an output. All port pins are defined as inputs after a Reset. Writes to a port are latched and, if programmed as an output, driven by a buffer to the pin. The value of pin can be read directly from an input buffer, (after it has been synchronized to the clock), regardless of whether the pin direction has been selected as input or output. Additionally, the value being driven by the latch before the output buffer, can also be read.



Internal pull-up resistors are available on selected port pins to eliminate the need to use external pull-up resistors. These pull-ups can be controlled by this library. The open-drain feature allows the generation of outputs higher than VDD on any desired digital-only pins by using external pull-up resistors.

Note: For detailed specifications on the maximum allowed open-drain voltage, refer to the **"Electrical Characteristics"** chapter in the specific device data sheet.

The input change notification feature of the I/O ports allows the microcontrollers to generate interrupt requests to the processor in response to a change of state on selected input pins. This feature is capable of detecting input change of states even in Sleep mode, when the clocks are disabled.

The alternate pin function selections are used to steer specific peripheral input and output functions between different pins.

The output slew rate of some port pins are programmable to select either the standard transition rate or a reduced transition rate of 'x' times the standard to minimize EMI.

Peripheral Pin Select

In addition to general purpose IO control, this library also provides a low-level abstraction of the PPS module on Microchip microcontrollers with a convenient C language interface.

Available Pins

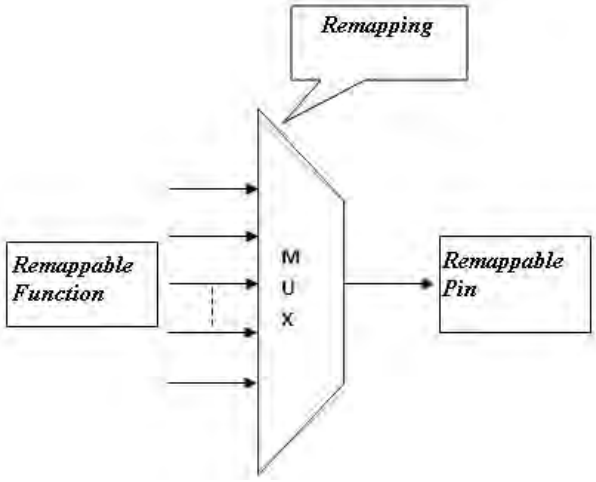
The number of available pins is dependent on the particular device and its pin count. Pins that support the PPS feature include the designation 'RPn' or 'RPIn' in their full pin designation, where 'RP' designates a remappable peripheral and 'n' is the remappable pin number. RP is used to designate pins that support both remappable input and output functions, while RPI indicates pins that only support remappable input functions.

Available Peripherals

The peripherals managed by PPS are all digital-only peripherals. These include general serial communications (UART and SPI), general purpose timer clock inputs, timer-related peripherals (Input Capture and Output Compare), comparator digital output, interrupt-on-change inputs, etc.

In comparison, some digital-only peripheral modules are never included in the peripheral pin select feature. This is because the peripheral's function requires special I/O circuitry on a specific port and cannot be easily connected to multiple pins. These modules include I2C, among others. A similar requirement excludes all modules with analog inputs, such as the Analog-to-Digital Converter (ADC).

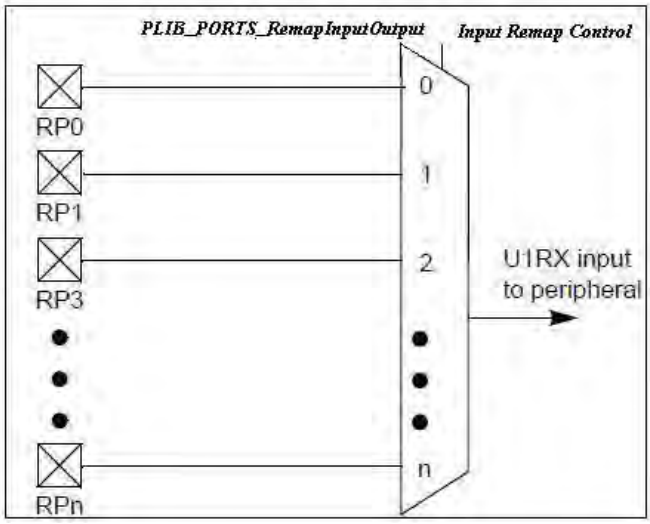
If multiple peripherals are enabled on the same pin(s), there is an internal priority that decides which function is mapped to the pin. When a remappable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin. Priority is given regardless of the type of peripheral that is mapped. Remappable peripherals never take priority over any analog functions associated with the pin. In other words, If an analog function is enabled on the pin, the PPS input will be disabled.



Input Mapping

The inputs of the peripheral pin select options are mapped on the basis of the peripheral.

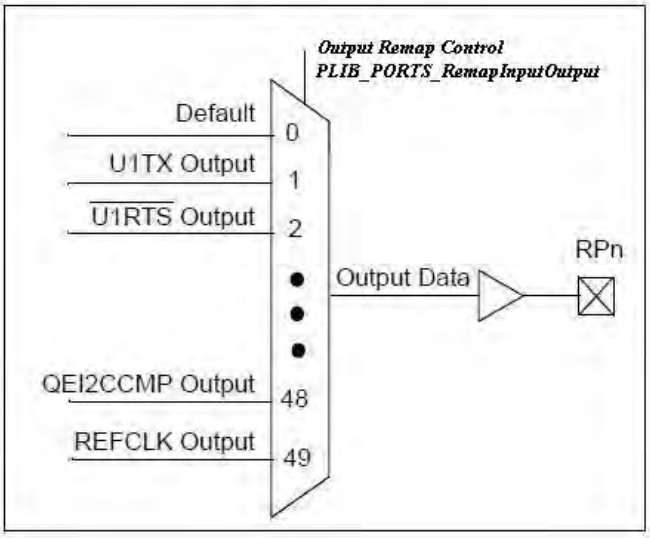
Remappable Input Example for USART 1 Receive



Output Mapping

In contrast to inputs, the outputs of the peripheral pin select options are mapped on the basis of the pin. In this case, a control associated with a particular pin dictates the peripheral output to be mapped.

Multiplexing of Remappable Output for RPn



Mapping Limitations

The control schema of the peripheral select pins is not limited to a small range of fixed peripheral configurations. There are no mutual-exclusion or hardware-enforced lockouts between any of the peripheral mapping SFRs. Literally any combination of peripheral mappings across any or all of the RPn pins is possible. This includes both many-to-one and one-to-many mappings of peripheral inputs and outputs to pins. While such mappings may be technically possible from a configuration point of view, they may not be supportable from an electrical point of view and may cause damage to the part.

Notes:

1. For detailed specifications on the mapping limitations, refer to the **"I/O Ports"** chapter in the specific device data sheet.
2. Regrading pin priorities, the priority of the pins are assigned from left to right, left being the highest priority and the right most being the least priority.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ports module.

Library Interface Section	Description
Port Functions	Port access read/write/toggle/clear interfaces with 8-/16-/32-bit widths based on the selected device.
Port Pin Functions	Port bit/pin read/write/toggle/clear interfaces.
Change Notification Functions	Port change notification.
Peripheral Pin Select Functions	Interface routines for mapping the digital input/output to a specific the PPS remappable input/output pin.
Feature Existence Functions	Determine whether certain features exist.

How the Library Works

How the Library Works

The following topics discuss the processes involved while using a Ports module:

- Ports Control
- Ports Function Remap
- Ports Change Notification
- Special Considerations

Ports Control

Port Functions Usage:

- Port Read - Ports can be read at byte/word level using PLIB_PORTS_Read, and at the bit/pin level using PLIB_PORTS_PinGet
- Port Write - Ports can be written to at byte/word level using PLIB_PORTS_Write and at the bit/pin level using PLIB_PORTS_PinWrite. The function PLIB_PORTS_PinSet can be used to set a specific bit/pin of a port. A mask-based PORTx write can be accomplished using the function PLIB_PORTS_Set with the appropriate mask as a parameter along with the value.
- Port Clear - Ports can be cleared at byte/word level using PLIB_PORTS_Clear with appropriate mask as a parameter and at the bit/pin level using PLIB_PORTS_PinClear
- Port Direction - The ports read direction can be set at byte/word level using PLIB_PORTS_DirectionInputSet and at the bit/pin level using PLIB_PORTS_PinDirectionInputSet. Similarly, the ports write direction can be set at the byte/word level using PLIB_PORTS_DirectionOutputSet and at the bit/pin level using PLIB_PORTS_PinDirectionOutputSet. The direction information can be read using PLIB_PORTS_DirectionGet.
- Port Toggle - Ports can be toggled at the byte/word level using PLIB_PORTS_Toggle with the appropriate mask as a parameter and at the bit/pin level using PLIB_PORTS_PinToggle
- Port Open Drain - Ports can be enabled for open-drain functionality at the byte/word level using the interface PLIB_PORTS_OpenDrainEnable and at the bit/pin level using PLIB_PORTS_PinOpenDrainEnable. Similarly, the ports can be disabled for open-drain functionality at the byte/word level using PLIB_PORTS_OpenDrainDisable and at the bit/pin level using PLIB_PORTS_PinOpenDrainDisable.

Example:

```
// PORT Direction setting for output
   PLIB_PORTS_DirectionOutputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);

// Writing a value into a PORT
   PLIB_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_TYPE)0x1234);

// PORT Direction setting for input
   PLIB_PORTS_DirectionInputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);

// Reading back the previously written value
   PORTS_DATA_TYPE readData = PLIB_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);

// Clearing the PORT
   PLIB_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);

// Writing the value based on the mask, only 0x34 gets written to the PORT
   PLIB_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, 0x1234, (PORTS_DATA_MASK)0x00FF);

// Toggling a PORT
   PLIB_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

Ports Function Remap

Using the PPS Feature:

1. Ensure that any fixed digital peripherals on the pins to be used are disabled.
2. Ensure that pins with analog functionality are set to Digital mode.
3. Enable writing to the Input/Output control register using PLIB_DEVCON_SystemUnlock and PLIB_DEVCON_DeviceRegistersUnlock.
4. Remap the input and/or output pins using the internal PPS module. This can be achieved with the functions PLIB_PORTS_RemapInput and PLIB_PORTS_RemapOutput with the respective parameters defined in the processor specific headers.
5. Lock the Input/Output control register using PLIB_DEVCON_SystemUnlock and PLIB_DEVCON_DeviceRegistersUnlock.
6. Configure and enable newly mapped PPS peripherals.

Example:

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);

// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'
PLIB_PORTS_RemapInput(PORTS_ID_0, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );

// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'
PLIB_PORTS_RemapOutput(PORTS_ID_0, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);

// Lock PPS registers
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
```

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Ports Change Notification

Change Notification Feature Usage

1. Ensure that the change notify pin is configured as a digital input by setting the associated bit in the direction register using the PLIB_PORTS_PinDirectionInputSet or PLIB_PORTS_DirectionInputSet functions.

2. Select the desired change notification pin using PLIB_PORTS_PinChangeNoticeEnable.

3. Turn on the weak pull-up devices (if desired) for the selected change notification pins using PLIB_PORTS_ChangeNoticePullUpEnable (weak pull-ups can be disabled using PLIB_PORTS_ChangeNoticePullUpDisable).

4. Clear the selected change notification interrupt flag.

5. Select the desired interrupt priority for change notification pin.

6. Enable change notification interrupts.

7. Change notification can be disabled after the successful usage using PLIB_PORTS_PinChangeNoticeDisable. Certain microcontrollers support the global control over the change notification feature using the following functions: PLIB_PORTS_ChangeNoticeEnable and PLIB_PORTS_ChangeNoticeDisable. If there are any requirements to control the pull-downs, the following two functions could be used: PLIB_PORTS_ChangeNoticePullDownPerPortEnable and PLIB_PORTS_ChangeNoticePullDownPerPortDisable.

Change Notification Operation in Sleep and Idle Modes

The change notification feature continues to operate during Sleep or Idle mode. Its operation can be enabled or disabled using PLIB_PORTS_ChangeNoticeInIdleEnable or PLIB_PORTS_ChangeNoticeInIdleDisable. If one of the enabled change notification pins changes states, the respective status bit will be set, which can be monitored using PLIB_PORTS_ChangeNoticePerPortHasOccurred.

Example:

```
// Enabling the global change notification
PLIB_PORTS_ChangeNoticeEnable(MY_PORTS_INSTANCE);
// Enabling weak pull-ups for the change notification PIN 10
PLIB_PORTS_ChangeNoticePullUpEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
// Enabling change notification on PIN 10
PLIB_PORTS_PinChangeNoticeEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
```

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the related family reference manual section specified in that chapter.

Special Considerations

Considerations on Ports Usage

When reading a port register, all pins configured as analog input channels will read as cleared.

Pins configured as digital inputs will not operate correctly with analog peripherals. Analog levels on any pin that is defined as a digital input may cause the input buffer of analog peripherals to consume current that exceeds the device specifications.

Pull-ups and pull-downs on change notification pins should always be disabled when the port pin is configured as a digital output.

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically, this instruction would be a NOP.

Considerations for PPS

The ability to control PPS options introduces several considerations into application design that could be overlooked. This is particularly true for several common peripherals that are available only as remappable peripherals.

The main consideration is that the peripheral pin selects are not available on default pins in the device's default (i.e., Reset) state, all PPS inputs are tied to VSS and all PPS outputs are disconnected. This condition requires the user to initialize the device with the proper peripheral configuration before any other application code is executed. For application safety, however, it is best to lock the configuration after writing to the PPS control.

The assignment of a peripheral signal to a particular pin does not automatically perform any other configuration of the pin's I/O circuitry. In theory, this means adding a pin-selectable output to a pin may mean inadvertently driving an existing peripheral input when the output is driven. Users must be familiar with the behavior of other fixed peripherals that share a remappable pin and know when to enable or disable them. To be safe, fixed digital peripherals that share the same pin should be disabled when not in use. In addition, configuring a remappable pin for a specific peripheral does not automatically turn on that feature. The peripheral must be specifically configured for operation and enabled, as if it were tied to a fixed pin. Where this occurs in the application code (immediately following a device Reset and peripheral configuration or inside the main application routine) depends on the peripheral and its use in the application.

A final consideration is that PPS functions neither override analog inputs, nor reconfigure pins with analog functions for digital I/O. If a pin is configured as an analog input on a device Reset, it must be explicitly reconfigured as a digital I/O when used with PPS.

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Configuring the Library

The library is configured for the supported Ports module when the supported processor is chosen in the MPLAB X IDE.

Library Interface

a) Port Pin Functions

	Name	Description
	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
	PLIB_PORTS_PinDirectionInputSet	Makes the selected pin direction input
	PLIB_PORTS_PinDirectionOutputSet	Makes the selected pin direction output
	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
	PLIB_PORTS_PinModePerPortSelect	Enables the selected port pin as analog or digital.
	PLIB_PORTS_PinGetLatched	Reads/Gets data from the selected latch.
	PLIB_PORTS_PinChangeNoticeEdgeHasOccurred	Check Change Notice edge status.
	PLIB_PORTS_PinChangeNoticeEdgeIsEnabled	Check if Change Notice edge is enabled or not.
	PLIB_PORTS_PinSlewRateGet	Gets the slew rate for selected port pin.
	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.

b) Port Functions

	Name	Description
	PLIB_PORTS_Clear	Clears the selected digital port/latch bits.

	PLIB_PORTS_DirectionGet	Reads the direction of the selected digital port.
	PLIB_PORTS_DirectionInputSet	Makes the selected pins direction input.
	PLIB_PORTS_DirectionOutputSet	Makes the selected pins direction output.
	PLIB_PORTS_Read	Reads the selected digital port.
	PLIB_PORTS_Set	Sets the selected bits of the port.
	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
	PLIB_PORTS_Write	Writes the selected digital port/latch.
	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port pins.
	PLIB_PORTS_ReadLatched	Reads and returns data from the selected Latch.
	PLIB_PORTS_ChannelModeSelect	Enables the selected channel pins as analog or digital.
	PLIB_PORTS_ChannelSlewRateSelect	Selects the slew rate for selected channel pins.

c) Peripheral Pin Select Functions

	Name	Description
	PLIB_PORTS_RemapInput	Input function remapping.
	PLIB_PORTS_RemapOutput	Output function remapping.

d) Change Notification Functions

	Name	Description
	PLIB_PORTS_ChangeNoticeDisable	Global Change Notice disable.
	PLIB_PORTS_ChangeNoticeEnable	Global Change Notice enable.
	PLIB_PORTS_ChangeNoticeInIdleDisable	CPU Idle halts the Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdleEnable	CPU Idle mode does not affect Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdlePerPortDisable	Change Notification halts in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticeInIdlePerPortEnable	Allows CN to be working in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticePerPortHasOccured	This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.
	PLIB_PORTS_ChangeNoticePerPortTurnOff	Disables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePerPortTurnOn	Enables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePullDownPerPortDisable	Disables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullDownPerPortEnable	Enables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullUpDisable	Disable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpEnable	Enable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpPerPortDisable	Disables weak pull-up for the selected pin.
	PLIB_PORTS_ChangeNoticePullUpPerPortEnable	Enables the pull-up for selected Change Notice pins.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice interrupt enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Disables CN interrupt for the selected pin.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables CN interrupt for the selected pin.
	PLIB_PORTS_ChangeNoticePerPortHasOccurred	checks the status of change on the pin
	PLIB_PORTS_ChannelChangeNoticeDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticePullDownDisable	Disables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullDownEnable	Enables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeDisable	Disables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeEnable	Enables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeMethodGet	Gets the Change Notice style for the selected port channel.
	PLIB_PORTS_ChannelChangeNoticeMethodSelect	Selects the Change Notice style for selected port channel.
	PLIB_PORTS_AnPinsModeSelect	Enables the selected AN pins as analog or digital.
	PLIB_PORTS_CnPinsDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsPullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_CnPinsPullUpEnable	Enables Change Notice pull-up for the selected channel pins.

e) Feature Existence Functions

	Name	Description
	PLIB_PORTS_ExistsChangeNotice	Identifies whether the ChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeInIdle	Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortInIdle	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortStatus	Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortTurnOn	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullDownPerPort	Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUp	Identifies whether the ChangeNoticePullup feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUpPerPort	Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNotice	Identifies whether the PinChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNoticePerPort	Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinMode	Identifies whether the PinMode feature exists on the Ports module.
	PLIB_PORTS_ExistsPinModePerPort	Identifies whether the PinModePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the Ports module.
	PLIB_PORTS_ExistsLatchRead	Identifies whether the LatchRead feature exists on the Ports module.
	PLIB_PORTS_ExistsAnPinsMode	Identifies whether the AnPinsMode feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeControl	Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeStatus	Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChannelChangeNoticeMethod	Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.
	PLIB_PORTS_ExistsSlewRateControl	Identifies whether the SlewRateControl feature exists on the Ports module.

f) Data Types and Constants

	Name	Description
	PORTS_ANALOG_PIN	Data type defining the different analog input pins.
	PORTS_BIT_POS	Lists the constants that hold different PORTS bit positions.
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANNEL	Identifies the available Ports channels.
	PORTS_DATA_MASK	Data type defining the Ports data mask
	PORTS_DATA_TYPE	Data type defining the Ports data type.
	PORTS_MODULE_ID	Identifies the available Ports modules.
	PORTS_PIN_MODE	Identifies the available pin modes.
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumerations.
	PORTS_REMAP_INPUT_FUNCTION	Data type defining the different remap input function enumerations.
	PORTS_REMAP_INPUT_PIN	Data type defining the different Ports I/O input pins enumerations.
	PORTS_REMAP_OUTPUT_FUNCTION	Data type defining the different remap output function enumerations.
	PORTS_REMAP_OUTPUT_PIN	Data type defining the different Ports I/O output pins enumerations.
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumerations.
	PORTS_AN_PIN	Data type defining the different analog input pins.
	PORTS_CN_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANGE_NOTICE_EDGE	Data type defining the different edge type for change notification.
	PORTS_CHANGE_NOTICE_METHOD	Data type defining the different method of ports pin change notification.

	PORTS_PIN_SLEW_RATE	Data type defining the different slew rates for port pins.

Description

This section describes the Application Programming Interface (API) functions of the Ports Peripheral Library.

Refer to each section for a detailed description.

a) Port Pin Functions

PLIB_PORTS_PinClear Function

Clears the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function clears the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clears port pin RC4
PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinDirectionInputSet Function

Makes the selected pin direction input

File

plib_ports.h

C

```
void PLIB_PORTS_PinDirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function makes the selected pin direction as input

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// make pin RC4 as input
PLIB_PORTS_PinDirectionInputSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS direction that has to be made input

Function

void PLIB_PORTS_PinDirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinDirectionOutputSet Function

Makes the selected pin direction output

File

plib_ports.h

C

```
void PLIB_PORTS_PinDirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function makes the selected pin direction as output

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// make pin RC4 as output
PLIB_PORTS_PinDirectionOutputSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS direction that has to be made output

Function

void PLIB_PORTS_PinDirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinGet Function

Reads/Gets data from the selected digital pin.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

Port pin read data.

Description

This function reads/gets data from the selected digital PORT i/o pin. This function should be used to read the live data at the pin.

Remarks

For reading the Latched data, PLIB_PORTS_PinGetLatched function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// read port pin RC4
bool bitStatus = PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_C,
                                                PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinModeSelect Function

Enables the selected pin as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_PinModeSelect(PORTS_MODULE_ID index, PORTS_ANALOG_PIN pin, PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected pin as analog or digital.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make AN0 pin as Analog
PLIB_PORTS_PinModeSelect(PORTS_ID_0, PORTS_ANALOG_PIN_0, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	Possible values of PORTS_ANALOG_PIN
mode	Possible values of PORTS_PIN_MODE

Function

void PLIB_PORTS_PinModeSelect(PORTS_MODULE_ID index,
PORTS_ANALOG_PIN pin,
PORTS_PIN_MODE mode);

PLIB_PORTS_PinSet Function

Sets the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function sets the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Sets port pin RC4
PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinToggle Function

Toggles the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function toggles the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Toggles port pin RC4
PLIB_PORTS_PinToggle(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinWrite Function

Writes the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos, bool value);
```

Returns

None.

Description

This function writes to the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// write 'one' in port RC4
PLIB_PORTS_PinWrite(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS
value	Value to be written to the specific pin/latch
true	sets the bit, false - clears the bit

Function

void PLIB_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos,
bool value)

PLIB_PORTS_PinModePerPortSelect Function

Enables the selected port pin as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_PinModePerPortSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos,
PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected port pin as analog or digital.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinModePerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC5 pin Analog
PLIB_PORTS_PinModePerPortSelect(PORTS_ID_0, PORT_CHANNEL_C,
                                PORTS_BIT_POS_5, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Function

void PLIB_PORTS_PinModePerPortSelect(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos,
PORTS_PIN_MODE mode);

PLIB_PORTS_PinGetLatched Function

Reads/Gets data from the selected latch.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinGetLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

Latch read data.

Description

This function reads/gets data from the selected PORTx Data Latch, not from the port I/O pins.

Remarks

For reading the Live data from the i/o pin, PLIB_PORTS_PinGet function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsLatchRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// read latch RC4
bool bitStatus = PLIB_PORTS_PinGetLatched(PORTS_ID_0, PORT_CHANNEL_C,
                                                      PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinGetLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticeEdgeHasOccurred Function

Check Change Notice edge status.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinChangeNoticeEdgeHasOccurred(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

- true - Change Notice edge transition has occurred
- false - Change Notice edge transition has not occurred

Description

This function checks whether or no a Change Notice edge transition has occurred on the selected port pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Change Notice edge transition has occurred for pin RC1.
if (PLIB_PORTS_PinChangeNoticeEdgeHasOccurred(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1))
{
```

```
    // do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinChangeNoticeEdgeHasOccurred

```
(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

PLIB_PORTS_PinChangeNoticeEdgeIsEnabled Function

Check if Change Notice edge is enabled or not.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinChangeNoticeEdgeIsEnabled(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos, PORTS_CHANGE_NOTICE_EDGE cnEdgeType);
```

Returns

- true - Selected type of Change Notice Edge is enabled.
- false - Selected type of Change Notice Edge is not enabled.

Description

This function checks if selected type of Change Notice edge is enabled on a particular port pin or not.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Change Notice at rising edge is enabled or not for pin RC1.
if (PLIB_PORTS_PinChangeNoticeEdgeIsEnabled(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1,
PORTS_CHANGE_NOTICE_EDGE_RISING))
{
    // do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS.
cnEdgeType	Type of the edge which has to be checked.

Function

bool PLIB_PORTS_PinChangeNoticeEdgeIsEnabled

```
(
    PORTS_MODULE_ID index,
```

```
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos,
    PORTS_CHANGE_NOTICE_EDGE  cnEdgeType
);
```

PLIB_PORTS_PinSlewRateGet Function

Gets the slew rate for selected port pin.

File

plib_ports.h

C

```
PORTS_PIN_SLEW_RATE PLIB_PORTS_PinSlewRateGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

One of the possible values of PORTS_PIN_SLEW_RATE.

Description

This function gets the slew rate of selected port pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsSlewRateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PORTS_PIN_SLEW_RATE slewRate;

// Get the slew rate of pin RC1
slewRate = PLIB_PORTS_PinSlewRateGet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS.

Function

```
PORTS_PIN_SLEW_RATE PLIB_PORTS_PinSlewRateGet
(
    PORTS_MODULE_ID  index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

PLIB_PORTS_PinOpenDrainDisable Function

Disables the open drain functionality for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables the open drain functionality for the selected pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable open drain for pin RC4
PLIB_PORTS_PinOpenDrainDisable(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	One of the possible values of PORTS_BIT_POS.

Function

void PLIB_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinOpenDrainEnable Function

Enables the open drain functionality for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the open drain functionality for the selected pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable open drain for pin RC4
PLIB_PORTS_PinOpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

b) Port Functions

PLIB_PORTS_Clear Function

Clears the selected digital port/latch bits.

File

plib_ports.h

C

```
void PLIB_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK clearMask);
```

Returns

None.

Description

This function clears the selected digital port/latch bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clears the three least significant Port C bits
PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
clearMask	Identifies the bits to be cleared

Function

void PLIB_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK clearMask)

PLIB_PORTS_DirectionGet Function

Reads the direction of the selected digital port.

File

plib_ports.h

C

```
PORTS_DATA_MASK PLIB_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Direction of the selected port of type PORTS_DATA_MASK

Description

This function reads the direction of the selected digital port.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reads the direction of Port C pins
PORTS_DATA_MASK readDir = PLIB_PORTS_DirectionGet(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

PORTS_DATA_MASK PLIB_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

PLIB_PORTS_DirectionInputSet Function

Makes the selected pins direction input.

File

plib_ports.h

C

```
void PLIB_PORTS_DirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function makes the selected pins direction input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC0, RC1 and RC2 pins as Input
PLIB_PORTS_DirectionInputSet(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins direction that has to be made input

Function

void PLIB_PORTS_DirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_DirectionOutputSet Function

Makes the selected pins direction output.

File

plib_ports.h

C

```
void PLIB_PORTS_DirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function makes the selected pins direction output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC0, RC1 and RC2 pins as Output
PLIB_PORTS_DirectionOutputSet(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins direction that has to be made output

Function

void PLIB_PORTS_DirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_Read Function

Reads the selected digital port.

File

plib_ports.h

C

```
PORTS_DATA_TYPE PLIB_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

data on a port with width PORTS_DATA_TYPE

Description

This function reads from the selected digital port.

Remarks

For reading the Latched data, PLIB_PORTS_ReadLatched function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read PORT C
PORTS_DATA_TYPE readData = PLIB_PORTS_Read(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

PORTS_DATA_TYPE PLIB_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

PLIB_PORTS_Set Function

Sets the selected bits of the port.

File

plib_ports.h

C

```
void PLIB_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function performs an 'AND' operation on the value and mask parameters, and then sets the bits in the port channel that were set by the result of the 'AND' operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// MY_VALUE - 0x1234
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;

// Set the PORT C bit positions 2,4 and 5 (0x0034 = b0000 0000 0011 0100)
PLIB_PORTS_Set(MY_PORTS_INSTANCE, PORT_CHANNEL_C, MY_VALUE, myMask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
value	Consists of information about which port bit has to be set and which not
mask	Identifies the bits which could be intended for setting

Function

void PLIB_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_TYPE value,
PORTS_DATA_MASK mask)

PLIB_PORTS_Toggle Function

Toggles the selected digital port/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK toggleMask);
```

Returns

None.

Description

This function toggles the selected digital port/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Toggles the three least significant Port C bits
PLIB_PORTS_Toggle(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
toggleMask	Identifies the bits to be toggled

Function

void PLIB_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK toggleMask)

PLIB_PORTS_Write Function

Writes the selected digital port/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value);
```

Returns

None.

Description

This function writes to the selected digital port/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Write 0x12 into PORT C
PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_C, 0x12);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the Ports channel A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE

Function

void PLIB_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_TYPE value)

PLIB_PORTS_OpenDrainDisable Function

Disables the open drain functionality for the selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function disables the open drain functionality for the selected port.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Open Drain for RC0, RC1 and RC2 pins
PLIB_PORTS_OpenDrainDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins for the open drain to be disabled

Function

void PLIB_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_OpenDrainEnable Function

Enables the open drain functionality for the selected port pins.

File

plib_ports.h

C

```
void PLIB_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the open drain functionality for the selected port pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Open Drain for RC0, RC1 and RC2 pins
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins for the open drain to be enabled

Function

void PLIB_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_ReadLatched Function

Reads and returns data from the selected Latch.

File

plib_ports.h

C

```
PORTS_DATA_TYPE PLIB_PORTS_ReadLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Latch read data.

Description

This function reads and returns the data from the selected Latch.

Remarks

For reading the Live data, PLIB_PORTS_Read function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsLatchRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read latch C
PORTS_DATA_TYPE bitStatus = PLIB_PORTS_ReadLatched(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

PORTS_DATA_TYPE PLIB_PORTS_ReadLatched
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel
)

PLIB_PORTS_ChannelModeSelect Function

Enables the selected channel pins as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelModeSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK modeMask,
PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected channel pins as analog or digital.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_AnPinsModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinModePerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make pins RC5, RC8 and RC13 Analog
PLIB_PORTS_ChannelModeSelect(PORTS_ID_0, PORT_CHANNEL_C, 0x2120, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
modeMask	Identifies the pins whose mode has to be modified. Modes of the pins whose corresponding bit is '1' get modified, mode of the other pins remains the same.
mode	Possible values of PORTS_PIN_MODE (Analog or Digital)

Function

void PLIB_PORTS_ChannelModeSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK modeMask,

PORTS_PIN_MODE mode

);

PLIB_PORTS_ChannelSlewRateSelect Function

Selects the slew rate for selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelSlewRateSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
channelMask, PORTS_PIN_SLEW_RATE slewRate);
```

Returns

None.

Description

This function selects the slew rate for selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsSlewRateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make slew rate of pins RC5, RC8 and RC13 slowest
PLIB_PORTS_ChannelSlewRateSelect(PORTS_ID_0,
                                 PORT_CHANNEL_C,
                                 0x2120,
                                 PORTS_PIN_SLEW_RATE_SLOWEST);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
channelMask	Identifies the pins for which slew rate has to be modified. Slew rate of the pins which corresponding bit is "1" get modified, slew rate of the other pins remains the same.
slewRate	One of the possible values of PORTS_PIN_SLEW_RATE.

Function

void PLIB_PORTS_ChannelSlewRateSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK channelMask,

PORTS_PIN_SLEW_RATE slewRate

);

c) Peripheral Pin Select Functions

PLIB_PORTS_RemapInput Function

Input function remapping.

File

plib_ports.h

C

```
void PLIB_PORTS_RemapInput(PORTS_MODULE_ID index, PORTS_REMAP_INPUT_FUNCTION inputFunction,
PORTS_REMAP_INPUT_PIN remapInputPin);
```

Returns

None.

Description

This function controls the Input function remapping. It allows user to map any of the input functionality on any of the remappable input pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

IOLOCK bit of configuration register should be clear to allow any remapping. PLIB_DEVCON_DeviceRegistersUnlock function can be used for that purpose. Refer DEVCON PLIB (or System Service) and the specific device data sheet to find more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'
PLIB_PORTS_RemapInput(PORTS_ID_0, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
inputFunction	One of the possible values of PORTS_REMAP_INPUT_FUNCTION
remapInputPin	One of the possible values of PORTS_REMAP_INPUT_PIN

Function

void PLIB_PORTS_RemapInput(PORTS_MODULE_ID index,
PORTS_REMAP_INPUT_FUNCTION inputFunction,
PORTS_REMAP_INPUT_PIN remapInputPin);

PLIB_PORTS_RemapOutput Function

Output function remapping.

File

plib_ports.h

C

```
void PLIB_PORTS_RemapOutput(PORTS_MODULE_ID index, PORTS_REMAP_OUTPUT_FUNCTION outputFunction,
PORTS_REMAP_OUTPUT_PIN remapOutputPin);
```

Returns

None.

Description

This function controls the Output function remapping. it allows user to map any of the output functionality on any of the remappable output pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

IOLOCK bit of configuration register should be clear to allow any remapping. PLIB_DEVCON_DeviceRegistersUnlock function can be used for that purpose. Refer DEVCON PLIB (or System Service) and the specific device data sheet to find more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'
PLIB_PORTS_RemapOutput(PORTS_ID_0, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
outputFunction	One of the possible values of PORTS_REMAP_OUTPUT_FUNCTION
remapOutputPin	One of the possible values of PORTS_REMAP_OUTPUT_PIN

Function

void PLIB_PORTS_RemapOutput(PORTS_MODULE_ID index,
PORTS_REMAP_OUTPUT_FUNCTION outputFunction,
PORTS_REMAP_OUTPUT_PIN remapOutputPin);

d) Change Notification Functions

PLIB_PORTS_ChangeNoticeDisable Function

Global Change Notice disable.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function disables the global Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePerPortTurnOff function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change Notification
PLIB_PORTS_ChangeNoticeDisable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeDisable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeEnable Function

Global Change Notice enable.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function enables the global Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePerPortTurnOn function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change Notification
PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeEnable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdleDisable Function

CPU Idle halts the Change Notice operation.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function halts the Change Notice operation when the CPU enters Idle mode.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticeInIdlePerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Halts the Change notification operation when CPU enters Idle mode
PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdleEnable Function

CPU Idle mode does not affect Change Notice operation.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function makes sure that Change Notice feature continues working in Idle mode.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticeInIdlePerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification feature will be working even when CPU goes to
// Idle mode
PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdlePerPortDisable Function

Change Notification halts in Idle mode for selected channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function makes sure that change notification feature halts in Idle mode for the selected channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeInIdleDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification halts in Idle mode for Port C
PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticeInIdlePerPortEnable Function

Allows CN to be working in Idle mode for selected channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function makes sure that change notification feature keeps working in Idle mode for the selected channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeInIdleEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification continues working in Idle mode for Port C
PLIB_PORTS_ChangeNoticeInIdlePerPortEnable(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticeInIdlePerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePerPortHasOccured Function

File

plib_ports.h

C

```
bool PLIB_PORTS_ChangeNoticePerPortHasOccured(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Description

This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.

PLIB_PORTS_ChangeNoticePerPortTurnOff Function

Disables the change notification for selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function disables the change notification for selected port.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortTurnOn in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change notification for Port C
PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePerPortTurnOn Function

Enables the change notification for selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOn(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function enables the change notification for selected port.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortTurnOn in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change notification for Port C
PLIB_PORTS_ChangeNoticePerPortTurnOn(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticePerPortTurnOn (PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePullDownPerPortDisable Function

Disables the pull-down for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables the pull-down for selected Change Notice pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-down for RC5 pin
PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePullDownPerPortEnable Function

Enables the pull-down for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the pull-down for selected Change Notice pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-down for RC5 pin
PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePullUpDisable Function

Disable pull-up on input change.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function disables pull-up on selected input change notification pin.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePullUpPerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up on pin CN13
PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_ChangeNoticePullUpEnable Function

Enable pull-up on input change.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function enables pull-up on selected input change notification pin

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePullUpPerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up on pin CN13
PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_ChangeNoticePullUpPerPortDisable Function

Disables weak pull-up for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables weak pull-up for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticePullUpDisable function.

Pull-ups on change notification pins should always be disabled when the port pin is configured as a digital output.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for RC5 pin
PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);

PLIB_PORTS_ChangeNoticePullUpPerPortEnable Function

Enables the pull-up for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpPerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function enables the pull-up for selected Change Notice pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticePullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for RC5 pin
PLIB_PORTS_ChangeNoticePullUpPerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullUpPerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticeDisable Function

Port pin Change Notice disable.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticeDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function disables the port pin Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_PinChangeNoticePerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change Notice interrupt for pin CN13
PLIB_PORTS_PinChangeNoticeDisable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_PinChangeNoticeDisable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_PinChangeNoticeEnable Function

Port pin Change Notice interrupt enable.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticeEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function enables the port pin Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_PinChangeNoticePerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change Notice interrupt for pin CN13
PLIB_PORTS_PinChangeNoticeEnable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_PinChangeNoticeEnable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_PinChangeNoticePerPortDisable Function

Disables CN interrupt for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function disables Change Notice interrupt for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for RC5 pin
PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                    PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticePerPortEnable Function

Enables CN interrupt for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticePerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for RC5 pin
PLIB_PORTS_PinChangeNoticePerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                    PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_PinChangeNoticePerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePerPortHasOccurred Function

checks the status of change on the pin

File

plib_ports.h

C

```
bool PLIB_PORTS_ChangeNoticePerPortHasOccurred(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function checks if the change has occurred on the given pin or not.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PORTS_ChangeNoticePerPortHasOccurred( PORTS_ID_0,
                                PORT_CHANNEL_C, PORTS_BIT_POS_4 ) == True)
{
 //do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

bool PLIB_PORTS_ChangeNoticePerPortHasOccurred (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);

PLIB_PORTS_ChannelChangeNoticeDisable Function

Disables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function Disables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticeDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for which change notification is to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticeDisable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticeEnable Function

Enables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticeEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for which change notification is to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticeEnable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticePullDownDisable Function

Disables Change Notice pull-down for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullDownDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function Disables the Change Notice pull-down for the selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-down for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullDownDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for the pull-down to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticePullDownDisable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticePullDownEnable Function

Enables Change Notice pull-down for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullDownEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the Change Notice pull-down for the selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-down for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullDownEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for the pull-down to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticePullDownEnable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticePullUpDisable Function

Disables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullUpDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function Disables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsPullUpDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullUpDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins of the pull-up to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticePullUpDisable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticePullUpEnable Function

Enables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullUpEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsPullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullUpEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                       0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins of the pull-up to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticePullUpEnable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticeEdgeDisable Function

Disables selected type of edge for selected CN pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEdgeDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK edgeRisingMask, PORTS_DATA_MASK edgeFallingMask);
```

Returns

None.

Description

This function Disables selected type of edge (falling or rising) for selected CN pins of a port channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

Change Notice method should be selected as "PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT" using PLIB_PORTS_ChannelChangeNoticeMethodSelect before using this function.

Example

```
// Disable Change Notice at rising edge for RC1 pin and at falling edge for
// RC1 & RC5 pins.
PLIB_PORTS_ChannelChangeNoticeEdgeDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x0002, 0x0022);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
edgeRisingMask	Identifies the pins for which Change Notice has to be enabled for rising edge. Change notice interrupt at rising edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.
edgeFallingMask	Identifies the pins for which Change Notice has to be enabled for falling edge. Change notice interrupt at falling edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.

Function

void PLIB_PORTS_ChannelChangeNoticeEdgeDisable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK edgeRisingMask,

PORTS_DATA_MASK edgeFallingMask

);

PLIB_PORTS_ChannelChangeNoticeEdgeEnable Function

Enables selected type of edge for selected CN pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEdgeEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
edgeRisingMask, PORTS_DATA_MASK edgeFallingMask);
```

Returns

None.

Description

This function Enables selected type of edge (falling or rising) for selected CN pins of a port channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

Change Notice method should be selected as "PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT" using PLIB_PORTS_ChannelChangeNoticeMethodSelect before using this function.

Example

```
// Enable Change Notice at rising edge for RC1 pin and at falling edge for
// RC1 & RC5 pins.
PLIB_PORTS_ChannelChangeNoticeEdgeEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x0002, 0x0022);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

edgeRisingMask	Identifies the pins for which Change Notice has to be enabled for rising edge. Change Notice interrupt at rising edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.
edgeFallingMask	Identifies the pins for which Change Notice has to be enabled for falling edge. Change Notice interrupt at falling edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.

Function

void PLIB_PORTS_ChannelChangeNoticeEdgeEnable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK edgeRisingMask,
PORTS_DATA_MASK edgeFallingMask
);

PLIB_PORTS_ChannelChangeNoticeMethodGet Function

Gets the Change Notice style for the selected port channel.

File

plib_ports.h

C

```
PORTS_CHANGE_NOTICE_METHOD PLIB_PORTS_ChannelChangeNoticeMethodGet(PORTS_MODULE_ID index, PORTS_CHANNEL
channel);
```

Returns

One of the possible values of PORTS_CHANGE_NOTICE_METHOD.

Description

This function gets the Change Notice style (or method) for the selected port channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChannelChangeNoticeMethod in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod;
changeNoticeMethod = PLIB_PORTS_ChannelChangeNoticeMethodGet(PORTS_ID_0,
PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

PORTS_CHANGE_NOTICE_METHOD PLIB_PORTS_ChannelChangeNoticeMethodGet
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel
);

PLIB_PORTS_ChannelChangeNoticeMethodSelect Function

Selects the Change Notice style for selected port channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeMethodSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod);
```

Returns

None.

Description

This function selects the Change Notice style (or method) for selected port channel. It allows user to select whether the Change Notice detection will happen based on edge transition or level transition on all the CN pins of a particular channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChannelChangeNoticeMethod in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PORTS_ChannelChangeNoticeMethodSelect(PORTS_ID_0, PORT_CHANNEL_C,
                                           PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
changeNoticeMethod	One of the possible values of PORTS_CHANGE_NOTICE_METHOD.

Function

void PLIB_PORTS_ChannelChangeNoticeMethodSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod

);

PLIB_PORTS_AnPinsModeSelect Function

Enables the selected AN pins as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_AnPinsModeSelect(PORTS_MODULE_ID index, PORTS_AN_PIN anPins, PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected AN pins as analog or digital.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsAnPinsMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make pins AN5, AN8 and AN13 Analog
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, PORTS_AN_PIN_5 |
                            PORTS_AN_PIN_8 |
                            PORTS_AN_PIN_13,
                            PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
anPins	AN pins whose mode is to be changed. Multiple AN pins can be ORed.
mode	Possible values of PORTS_PIN_MODE (Analog or Digital)

Function

void PLIB_PORTS_AnPinsModeSelect

(

PORTS_MODULE_ID index,

PORTS_AN_PIN anPins,

PORTS_PIN_MODE mode

);

PLIB_PORTS_CnPinsDisable Function

Disables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsDisable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function Disables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsDisable(PORTS_ID_0,
                         CHANGE_NOTICE_PIN_5 |
                         CHANGE_NOTICE_PIN_8 |
                         CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins to be disabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsDisable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

PLIB_PORTS_CnPinsEnable Function

Enables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsEnable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsEnable(PORTS_ID_0,
                        CHANGE_NOTICE_PIN_5 |
                        CHANGE_NOTICE_PIN_8 |
                        CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins to be enabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsEnable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

PLIB_PORTS_CnPinsPullUpDisable Function

Disables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsPullUpDisable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function Disables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticePullUpDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsPullUpDisable(PORTS_ID_0,
                               CHANGE_NOTICE_PIN_5 |
                               CHANGE_NOTICE_PIN_8 |
                               CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins whose pull-up is to be disabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsPullUpDisable

(

PORTS_MODULE_ID index,

PORTS_CN_PIN cnPins

);

PLIB_PORTS_CnPinsPullUpEnable Function

Enables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsPullUpEnable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function enables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticePullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for CN5, CN8 and CN13 pins
```

```
PLIB_PORTS_CnPinsPullUpEnable(PORTS_ID_0, CHANGE_NOTICE_PIN_5 |
                                          CHANGE_NOTICE_PIN_8 |
                                          CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins whose pull-up is to be enabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsPullUpEnable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

e) Feature Existence Functions

PLIB_PORTS_ExistsChangeNotice Function

Identifies whether the ChangeNotice feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNotice(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNotice feature is supported on the device
- false - The ChangeNotice feature is not supported on the device

Description

This function identifies whether the ChangeNotice feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeEnable
- PLIB_PORTS_ChangeNoticeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNotice(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeInIdle Function

Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeInIdle(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeInIdle feature is supported on the device
- false - The ChangeNoticeInIdle feature is not supported on the device

Description

This function identifies whether the ChangeNoticeInIdle feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeInIdleEnable
- PLIB_PORTS_ChangeNoticeInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeInIdle(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortInIdle Function

Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortInIdle(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeInIdlePerPort feature is supported on the device
- false - The ChangeNoticeInIdlePerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticeInIdlePerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeInIdlePerPortEnable
- PLIB_PORTS_ChangeNoticeInIdlePerPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortInIdle(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortStatus Function

Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortStatus(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePerPortStatus feature is supported on the device
- false - The ChangeNoticePerPortStatus feature is not supported on the device

Description

This function identifies whether the ChangeNoticePerPortStatus feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePerPortHasOccured
- PLIB_PORTS_ChangeNoticePerPortHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortStatus(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortTurnOn Function

Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortTurnOn(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePerPortTurnOn feature is supported on the device
- false - The ChangeNoticePerPortTurnOn feature is not supported on the device

Description

This function identifies whether the ChangeNoticePerPortTurnOn feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePerPortTurnOn
- PLIB_PORTS_ChangeNoticePerPortTurnOff

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortTurnOn(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullDownPerPort Function

Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullDownPerPort(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullDownPerPort feature is supported on the device
- false - The ChangeNoticePullDownPerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullDownPerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullDownPerPortEnable
- PLIB_PORTS_ChangeNoticePullDownPerPortDisable
- PLIB_PORTS_ChannelChangeNoticePullDownEnable
- PLIB_PORTS_ChannelChangeNoticePullDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullDownPerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullUp Function

Identifies whether the ChangeNoticePullup feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullUp(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullup feature is supported on the device
- false - The ChangeNoticePullup feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullup feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullUpEnable

- PLIB_PORTS_ChangeNoticePullUpDisable
- PLIB_PORTS_CnPinsPullUpEnable
- PLIB_PORTS_CnPinsPullUpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullUp(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullUpPerPort Function

Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullUpPerPort(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullUpPerPort feature is supported on the device
- false - The ChangeNoticePullUpPerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullUpPerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullUpPerPortEnable
- PLIB_PORTS_ChangeNoticePullUpPerPortDisable
- PLIB_PORTS_ChannelChangeNoticePullUpEnable
- PLIB_PORTS_ChannelChangeNoticePullUpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullUpPerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinChangeNotice Function

Identifies whether the PinChangeNotice feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinChangeNotice(PORTS_MODULE_ID index);
```

Returns

- true - The PinChangeNotice feature is supported on the device
- false - The PinChangeNotice feature is not supported on the device

Description

This function identifies whether the PinChangeNotice feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinChangeNoticeEnable
- PLIB_PORTS_PinChangeNoticeDisable
- PLIB_PORTS_CnPinsEnable
- PLIB_PORTS_CnPinsDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinChangeNotice(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinChangeNoticePerPort Function

Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinChangeNoticePerPort(PORTS_MODULE_ID index);
```

Returns

- true - The PinChangeNoticePerPort feature is supported on the device
- false - The PinChangeNoticePerPort feature is not supported on the device

Description

This function identifies whether the PinChangeNoticePerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinChangeNoticePerPortEnable
- PLIB_PORTS_PinChangeNoticePerPortDisable
- PLIB_PORTS_ChannelChangeNoticeEnable
- PLIB_PORTS_ChannelChangeNoticeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinChangeNoticePerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinMode Function

Identifies whether the PinMode feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinMode(PORTS_MODULE_ID index);
```

Returns

- true - The PinMode feature is supported on the device
- false - The PinMode feature is not supported on the device

Description

This function identifies whether the PinMode (Analog Pin or Digital Pin) feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_PinModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinMode(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinModePerPort Function

Identifies whether the PinModePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinModePerPort(PORTS_MODULE_ID index);
```

Returns

- true - The PinModePerPort feature is supported on the device
- false - The PinModePerPort feature is not supported on the device

Description

This function identifies whether the PinModePerPort (Analog Pin or Digital Pin) feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinModePerPortSelect
- PLIB_PORTS_ChannelModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinModePerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsDirection Function

Identifies whether the PortsDirection feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsDirection(PORTS_MODULE_ID index);
```

Returns

- true - The PortsDirection feature is supported on the device
- false - The PortsDirection feature is not supported on the device

Description

This function identifies whether the PortsDirection feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinDirectionInputSet
- PLIB_PORTS_PinDirectionOutputSet
- PLIB_PORTS_DirectionInputSet
- PLIB_PORTS_DirectionOutputSet
- PLIB_PORTS_DirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsDirection(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsOpenDrain Function

Identifies whether the PortsOpenDrain feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsOpenDrain(PORTS_MODULE_ID index);
```

Returns

- true - The PortsOpenDrain feature is supported on the device
- false - The PortsOpenDrain feature is not supported on the device

Description

This function identifies whether the PortsOpenDrain feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinOpenDrainEnable
- PLIB_PORTS_PinOpenDrainDisable
- PLIB_PORTS_OpenDrainEnable
- PLIB_PORTS_OpenDrainDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsOpenDrain(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsRead Function

Identifies whether the PortsRead feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsRead(PORTS_MODULE_ID index);
```

Returns

- true - The PortsRead feature is supported on the device
- false - The PortsRead feature is not supported on the device

Description

This function identifies whether the PortsRead feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinGet
- PLIB_PORTS_Read

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsRead(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsWrite Function

Identifies whether the PortsWrite feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsWrite(PORTS_MODULE_ID index);
```

Returns

- true - The PortsWrite feature is supported on the device
- false - The PortsWrite feature is not supported on the device

Description

This function identifies whether the PortsWrite feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinWrite
- PLIB_PORTS_PinSet
- PLIB_PORTS_PinClear
- PLIB_PORTS_PinToggle
- PLIB_PORTS_Write
- PLIB_PORTS_Set
- PLIB_PORTS_Toggle
- PLIB_PORTS_Clear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsWrite(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsRemapInput Function

Identifies whether the RemapInput feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsRemapInput(PORTS_MODULE_ID index);
```

Returns

- true - The RemapInput feature is supported on the device
- false - The RemapInput feature is not supported on the device

Description

This function identifies whether the RemapInput feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_RemapInput

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsRemapInput(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsRemapOutput Function

Identifies whether the RemapOutput feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsRemapOutput(PORTS_MODULE_ID index);
```

Returns

- true - The RemapOutput feature is supported on the device
- false - The RemapOutput feature is not supported on the device

Description

This function identifies whether the RemapOutput feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_RemapOutput

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsRemapOutput(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsLatchRead Function

Identifies whether the LatchRead feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsLatchRead(PORTS_MODULE_ID index);
```

Returns

- true - The LatchRead feature is supported on the device
- false - The LatchRead feature is not supported on the device

Description

This function identifies whether the LatchRead feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinGetLatched
- PLIB_PORTS_ReadLatched

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsLatchRead(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsAnPinsMode Function

Identifies whether the AnPinsMode feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsAnPinsMode(PORTS_MODULE_ID index);
```

Returns

- true - The AnPinsMode feature is supported on the device
- false - The AnPinsMode feature is not supported on the device

Description

This function identifies whether the AnPinsMode feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_AnPinsModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsAnPinsMode(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeEdgeControl Function

Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeEdgeControl(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeEdgeControl feature is supported on the device
- false - The ChangeNoticeEdgeControl feature is not supported on the device

Description

This function identifies whether the ChangeNoticeEdgeControl feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelChangeNoticeEdgeEnable
- PLIB_PORTS_ChannelChangeNoticeEdgeDisable
- PLIB_PORTS_PinChangeNoticeEdgeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeEdgeControl(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeEdgeStatus Function

Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeEdgeStatus(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeEdgeStatus feature is supported on the device
- false - The ChangeNoticeEdgeStatus feature is not supported on the device

Description

This function identifies whether the ChangeNoticeEdgeStatus feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_PinChangeNoticeEdgeHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeEdgeStatus(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChannelChangeNoticeMethod Function

Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChannelChangeNoticeMethod(PORTS_MODULE_ID index);
```

Returns

- true - The ChannelChangeNoticeMethod feature is supported on the device
- false - The ChannelChangeNoticeMethod feature is not supported on the device

Description

This function identifies whether the ChannelChangeNoticeMethod feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelChangeNoticeMethodSelect
- PLIB_PORTS_ChannelChangeNoticeMethodGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChannelChangeNoticeMethod(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsSlewRateControl Function

Identifies whether the SlewRateControl feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsSlewRateControl(PORTS_MODULE_ID index);
```

Returns

- true - The SlewRateControl feature is supported on the device
- false - The SlewRateControl feature is not supported on the device

Description

This function identifies whether the SlewRateControl feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelSlewRateSelect
- PLIB_PORTS_PinSlewRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsSlewRateControl(PORTS_MODULE_ID index)

f) Data Types and Constants

PORTS_ANALOG_PIN Enumeration

Data type defining the different analog input pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_ANALOG_PIN_0,
  PORTS_ANALOG_PIN_1,
  PORTS_ANALOG_PIN_2,
  PORTS_ANALOG_PIN_3,
  PORTS_ANALOG_PIN_4,
  PORTS_ANALOG_PIN_5,
  PORTS_ANALOG_PIN_6,
  PORTS_ANALOG_PIN_7,
  PORTS_ANALOG_PIN_8,
  PORTS_ANALOG_PIN_9,
  PORTS_ANALOG_PIN_10,
  PORTS_ANALOG_PIN_11,
  PORTS_ANALOG_PIN_12,
  PORTS_ANALOG_PIN_13,
  PORTS_ANALOG_PIN_14,
  PORTS_ANALOG_PIN_15,
  PORTS_ANALOG_PIN_16,
  PORTS_ANALOG_PIN_17,
  PORTS_ANALOG_PIN_18,
  PORTS_ANALOG_PIN_19,
  PORTS_ANALOG_PIN_20,
  PORTS_ANALOG_PIN_21,
  PORTS_ANALOG_PIN_22,
  PORTS_ANALOG_PIN_23,
  PORTS_ANALOG_PIN_24,
  PORTS_ANALOG_PIN_25,
  PORTS_ANALOG_PIN_26,
  PORTS_ANALOG_PIN_27,
  PORTS_ANALOG_PIN_28,
  PORTS_ANALOG_PIN_29,
  PORTS_ANALOG_PIN_30,
  PORTS_ANALOG_PIN_31,
  PORTS_ANALOG_PIN_32,
  PORTS_ANALOG_PIN_33,
  PORTS_ANALOG_PIN_34,
  PORTS_ANALOG_PIN_35,
  PORTS_ANALOG_PIN_36,
  PORTS_ANALOG_PIN_37,
  PORTS_ANALOG_PIN_38,
  PORTS_ANALOG_PIN_39,
  PORTS_ANALOG_PIN_40,
  PORTS_ANALOG_PIN_41,
  PORTS_ANALOG_PIN_42,
  PORTS_ANALOG_PIN_43,
  PORTS_ANALOG_PIN_44,
  PORTS_ANALOG_PIN_45,
  PORTS_ANALOG_PIN_46,
  PORTS_ANALOG_PIN_47,
  PORTS_ANALOG_PIN_48,
  PORTS_ANALOG_PIN_49
} PORTS_ANALOG_PIN;
```

Members

Members	Description
PORTS_ANALOG_PIN_0	Analog Input Pin 0
PORTS_ANALOG_PIN_1	Analog Input Pin 1
PORTS_ANALOG_PIN_2	Analog Input Pin 2
PORTS_ANALOG_PIN_3	Analog Input Pin 3
PORTS_ANALOG_PIN_4	Analog Input Pin 4
PORTS_ANALOG_PIN_5	Analog Input Pin 5
PORTS_ANALOG_PIN_6	Analog Input Pin 6
PORTS_ANALOG_PIN_7	Analog Input Pin 7
PORTS_ANALOG_PIN_8	Analog Input Pin 8

PORTS_ANALOG_PIN_9	Analog Input Pin 9
PORTS_ANALOG_PIN_10	Analog Input Pin 10
PORTS_ANALOG_PIN_11	Analog Input Pin 11
PORTS_ANALOG_PIN_12	Analog Input Pin 12
PORTS_ANALOG_PIN_13	Analog Input Pin 13
PORTS_ANALOG_PIN_14	Analog Input Pin 14
PORTS_ANALOG_PIN_15	Analog Input Pin 15
PORTS_ANALOG_PIN_16	Analog Input Pin 16
PORTS_ANALOG_PIN_17	Analog Input Pin 17
PORTS_ANALOG_PIN_18	Analog Input Pin 18
PORTS_ANALOG_PIN_19	Analog Input Pin 19
PORTS_ANALOG_PIN_20	Analog Input Pin 20
PORTS_ANALOG_PIN_21	Analog Input Pin 21
PORTS_ANALOG_PIN_22	Analog Input Pin 22
PORTS_ANALOG_PIN_23	Analog Input Pin 23
PORTS_ANALOG_PIN_24	Analog Input Pin 24
PORTS_ANALOG_PIN_25	Analog Input Pin 25
PORTS_ANALOG_PIN_26	Analog Input Pin 26
PORTS_ANALOG_PIN_27	Analog Input Pin 27
PORTS_ANALOG_PIN_28	Analog Input Pin 28
PORTS_ANALOG_PIN_29	Analog Input Pin 29
PORTS_ANALOG_PIN_30	Analog Input Pin 30
PORTS_ANALOG_PIN_31	Analog Input Pin 31
PORTS_ANALOG_PIN_32	Analog Input Pin 32
PORTS_ANALOG_PIN_33	Analog Input Pin 33
PORTS_ANALOG_PIN_34	Analog Input Pin 34
PORTS_ANALOG_PIN_35	Analog Input Pin 35
PORTS_ANALOG_PIN_36	Analog Input Pin 36
PORTS_ANALOG_PIN_37	Analog Input Pin 37
PORTS_ANALOG_PIN_38	Analog Input Pin 38
PORTS_ANALOG_PIN_39	Analog Input Pin 39
PORTS_ANALOG_PIN_40	Analog Input Pin 40
PORTS_ANALOG_PIN_41	Analog Input Pin 41
PORTS_ANALOG_PIN_42	Analog Input Pin 42
PORTS_ANALOG_PIN_43	Analog Input Pin 43
PORTS_ANALOG_PIN_44	Analog Input Pin 44
PORTS_ANALOG_PIN_45	Analog Input Pin 45
PORTS_ANALOG_PIN_46	Analog Input Pin 46
PORTS_ANALOG_PIN_47	Analog Input Pin 47
PORTS_ANALOG_PIN_48	Analog Input Pin 48
PORTS_ANALOG_PIN_49	Analog Input Pin 49

Description

PORTS Analog input pins

This data type defines the different analog input pins.

Remarks

Values of these AN Pin enums are different from the other similar enumerators by name PORTS_AN_PIN.

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_BIT_POS Enumeration

Lists the constants that hold different PORTS bit positions.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_BIT_POS_0,
  PORTS_BIT_POS_1,
  PORTS_BIT_POS_2,
  PORTS_BIT_POS_3,
  PORTS_BIT_POS_4,
  PORTS_BIT_POS_5,
  PORTS_BIT_POS_6,
  PORTS_BIT_POS_7,
  PORTS_BIT_POS_8,
  PORTS_BIT_POS_9,
  PORTS_BIT_POS_10,
  PORTS_BIT_POS_11,
  PORTS_BIT_POS_12,
  PORTS_BIT_POS_13,
  PORTS_BIT_POS_14,
  PORTS_BIT_POS_15
} PORTS_BIT_POS;
```

Members

Members	Description
PORTS_BIT_POS_0	PORT bit position 0
PORTS_BIT_POS_1	PORT bit position 1
PORTS_BIT_POS_2	PORT bit position 2
PORTS_BIT_POS_3	PORT bit position 3
PORTS_BIT_POS_4	PORT bit position 4
PORTS_BIT_POS_5	PORT bit position 5
PORTS_BIT_POS_6	PORT bit position 6
PORTS_BIT_POS_7	PORT bit position 7
PORTS_BIT_POS_8	PORT bit position 8
PORTS_BIT_POS_9	PORT bit position 9
PORTS_BIT_POS_10	PORT bit position 10
PORTS_BIT_POS_11	PORT bit position 11
PORTS_BIT_POS_12	PORT bit position 12
PORTS_BIT_POS_13	PORT bit position 13
PORTS_BIT_POS_14	PORT bit position 14
PORTS_BIT_POS_15	PORT bit position 15

Description

PORTS bit positions

Lists the constants that hold different PORTS bit positions.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_PIN Enumeration

Data type defining the different Change Notification (CN) pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  CN0,
  CN1,
  CN2,
  CN3,
  CN4,
  CN5,
  CN6,
  CN7,
```

```
  CN8,
  CN9,
  CN10,
  CN11,
  CN12,
  CN13,
  CN14,
  CN15,
  CN16,
  CN17,
  CN18,
  CN19,
  CN20,
  CN21
} PORTS_CHANGE_NOTICE_PIN;
```

Members

Members	Description
CN0	Change Notification Pin 0
CN1	Change Notification Pin 1
CN2	Change Notification Pin 2
CN3	Change Notification Pin 3
CN4	Change Notification Pin 4
CN5	Change Notification Pin 5
CN6	Change Notification Pin 6
CN7	Change Notification Pin 7
CN8	Change Notification Pin 8
CN9	Change Notification Pin 9
CN10	Change Notification Pin 10
CN11	Change Notification Pin 11
CN12	Change Notification Pin 12
CN13	Change Notification Pin 13
CN14	Change Notification Pin 14
CN15	Change Notification Pin 15
CN16	Change Notification Pin 16
CN17	Change Notification Pin 17
CN18	Change Notification Pin 18
CN19	Change Notification Pin 19
CN20	Change Notification Pin 20
CN21	Change Notification Pin 21

Description

Change Notification Pins enumeration

This data type defines the different CN pins enumerations.

Remarks

Values of these CN Pin enums are different from the other similar enumerators by name PORTS_CN_PIN. These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANNEL Enumeration

Identifies the available Ports channels.

File

help_plib_ports.h

C

```
typedef enum {
  PORT_CHANNEL_A,
  PORT_CHANNEL_B,
  PORT_CHANNEL_C,
  PORT_CHANNEL_D,
```

```
    PORT_CHANNEL_E,
    PORT_CHANNEL_F,
    PORT_CHANNEL_G,
    PORT_CHANNEL_H,
    PORT_CHANNEL_J,
    PORT_CHANNEL_K
} PORTS_CHANNEL;
```

Members

Members	Description
PORT_CHANNEL_A	Port A
PORT_CHANNEL_B	Port B
PORT_CHANNEL_C	Port C
PORT_CHANNEL_D	Port D
PORT_CHANNEL_E	Port E
PORT_CHANNEL_F	Port F
PORT_CHANNEL_G	Port G
PORT_CHANNEL_H	Port H
PORT_CHANNEL_J	Port J
PORT_CHANNEL_K	Port K

Description

PORT Channel ID

This enumeration identifies the available Ports channels.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

PORTS_DATA_MASK Type

Data type defining the Ports data mask

File

plib_ports.h

C

```
typedef uint16_t PORTS_DATA_MASK;
```

Description

Ports data mask definition

This data type defines the Ports data mask

Remarks

None.

PORTS_DATA_TYPE Type

Data type defining the Ports data type.

File

plib_ports.h

C

```
typedef uint32_t PORTS_DATA_TYPE;
```

Description

Ports data type definition

This data type defines the Ports data type.

Remarks

None.

PORTS_MODULE_ID Enumeration

Identifies the available Ports modules.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_ID_0,
  PORT_NUMBER_OF_MODULES
} PORTS_MODULE_ID;
```

Members

Members	Description
PORT_NUMBER_OF_MODULES	Max number of Instances

Description

PORT Module ID

This enumeration identifies the available Ports modules.

Remarks

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

PORTS_PIN_MODE Enumeration

Identifies the available pin modes.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_PIN_MODE_ANALOG,
  PORTS_PIN_MODE_DIGITAL
} PORTS_PIN_MODE;
```

Members

Members	Description
PORTS_PIN_MODE_ANALOG	Port Pin is in Analog Mode
PORTS_PIN_MODE_DIGITAL	Port pin is in Digital Mode

Description

PORTs Pin Mode

This enumeration identifies the available pin modes.

Remarks

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported.

PORTS_REMAP_FUNCTION Enumeration

Data type defining the different remap function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_REMAP_FUNCTION_EXT_INT1,
  PORTS_REMAP_FUNCTION_EXT_INT2,
  PORTS_REMAP_FUNCTION_EXT_INT3,
  PORTS_REMAP_FUNCTION_EXT_INT4,
  PORTS_REMAP_FUNCTION_IC1,
  PORTS_REMAP_FUNCTION_IC2,
  PORTS_REMAP_FUNCTION_IC3,
  PORTS_REMAP_FUNCTION_IC4,
  PORTS_REMAP_FUNCTION_IC5,
  PORTS_REMAP_FUNCTION_IC6,
  PORTS_REMAP_FUNCTION_IC7,
  PORTS_REMAP_FUNCTION_IC8,
  PORTS_REMAP_FUNCTION_IC9,
  PORTS_REMAP_FUNCTION_OC_FAULTA,
  PORTS_REMAP_FUNCTION_OC_FAULTB,
  PORTS_REMAP_FUNCTION_SPI1_CLOCK,
  PORTS_REMAP_FUNCTION_SPI1_DATA,
  PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_SPI2_CLOCK,
  PORTS_REMAP_FUNCTION_SPI2_DATA,
  PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_SPI3_CLOCK,
  PORTS_REMAP_FUNCTION_SPI3_DATA,
  PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_USART1_CTS,
  PORTS_REMAP_FUNCTION_USART1_RX,
  PORTS_REMAP_FUNCTION_USART2_CTS,
  PORTS_REMAP_FUNCTION_USART2_RX,
  PORTS_REMAP_FUNCTION_USART3_CTS,
  PORTS_REMAP_FUNCTION_USART3_RX,
  PORTS_REMAP_FUNCTION_USART4_CTS,
  PORTS_REMAP_FUNCTION_USART4_RX,
  PORTS_REMAP_FUNCTION_REFCLKI,
  PORTS_REMAP_FUNCTION_NULL,
  PORTS_REMAP_FUNCTION_C1OUT,
  PORTS_REMAP_FUNCTION_C2OUT,
  PORTS_REMAP_FUNCTION_U1TX,
  PORTS_REMAP_FUNCTION_U1RTS,
  PORTS_REMAP_FUNCTION_U2TX,
  PORTS_REMAP_FUNCTION_U2RTS,
  PORTS_REMAP_FUNCTION_SDO1,
  PORTS_REMAP_FUNCTION_SCK1OUT,
  PORTS_REMAP_FUNCTION_SS1OUT,
  PORTS_REMAP_FUNCTION_SDO2,
  PORTS_REMAP_FUNCTION_SCK2OUT,
  PORTS_REMAP_FUNCTION_SS2OUT,
  PORTS_REMAP_FUNCTION_OC1,
  PORTS_REMAP_FUNCTION_OC2,
  PORTS_REMAP_FUNCTION_OC3,
  PORTS_REMAP_FUNCTION_OC4,
  PORTS_REMAP_FUNCTION_OC5,
  PORTS_REMAP_FUNCTION_OC6,
  PORTS_REMAP_FUNCTION_OC7,
  PORTS_REMAP_FUNCTION_OC8,
  PORTS_REMAP_FUNCTION_U3TX,
  PORTS_REMAP_FUNCTION_U3RTS,
  PORTS_REMAP_FUNCTION_U4TX,
  PORTS_REMAP_FUNCTION_U4RTS,
  PORTS_REMAP_FUNCTION_SDO3,
  PORTS_REMAP_FUNCTION_SCK3OUT,
  PORTS_REMAP_FUNCTION_SS3OUT,
  PORTS_REMAP_FUNCTION_OC9,
  PORTS_REMAP_FUNCTION_C3OUT,
  PORTS_REMAP_FUNCTION_REFCLKO
} PORTS_REMAP_FUNCTION;
```

Members

Members	Description
PORTS_REMAP_FUNCTION_EXT_INT1	Input Function Name - External Interrupt 1
PORTS_REMAP_FUNCTION_EXT_INT2	Input Function Name - External Interrupt 2
PORTS_REMAP_FUNCTION_EXT_INT3	Input Function Name - External Interrupt 3
PORTS_REMAP_FUNCTION_EXT_INT4	Input Function Name - External Interrupt 4
PORTS_REMAP_FUNCTION_IC1	Input Function Name - Input Capture 1
PORTS_REMAP_FUNCTION_IC2	Input Function Name - Input Capture 2
PORTS_REMAP_FUNCTION_IC3	Input Function Name - Input Capture 3
PORTS_REMAP_FUNCTION_IC4	Input Function Name - Input Capture 4
PORTS_REMAP_FUNCTION_IC5	Input Function Name - Input Capture 5
PORTS_REMAP_FUNCTION_IC6	Input Function Name - Input Capture 6
PORTS_REMAP_FUNCTION_IC7	Input Function Name - Input Capture 7
PORTS_REMAP_FUNCTION_IC8	Input Function Name - Input Capture 8
PORTS_REMAP_FUNCTION_IC9	Input Function Name - Input Capture 9
PORTS_REMAP_FUNCTION_OC_FAULTA	Input Function Name - Output Compare Fault A
PORTS_REMAP_FUNCTION_OC_FAULTB	Input Function Name - Output Compare Fault B
PORTS_REMAP_FUNCTION_SPI1_CLOCK	Input Function Name - SPI1 Clock
PORTS_REMAP_FUNCTION_SPI1_DATA	Input Function Name - SPI1 Data
PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL	Input Function Name - SPI1 Slave Select
PORTS_REMAP_FUNCTION_SPI2_CLOCK	Input Function Name - SPI2 Clock
PORTS_REMAP_FUNCTION_SPI2_DATA	Input Function Name - SPI2 Data
PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL	Input Function Name - SPI2 Slave Select
PORTS_REMAP_FUNCTION_SPI3_CLOCK	Input Function Name - SPI3 Clock
PORTS_REMAP_FUNCTION_SPI3_DATA	Input Function Name - SPI3 Data
PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL	Input Function Name - SPI3 Slave Select
PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK	Input Function Name - Timer2 External CLock
PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK	Input Function Name - Timer3 External Clock
PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK	Input Function Name - Timer4 External Clock
PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK	Input Function Name - Timer5 External Clock
PORTS_REMAP_FUNCTION_USART1_CTS	Input Function Name - USART1 Clear To Send
PORTS_REMAP_FUNCTION_USART1_RX	Input Function Name - USART1 Receive
PORTS_REMAP_FUNCTION_USART2_CTS	Input Function Name - USART2 Clear To Send
PORTS_REMAP_FUNCTION_USART2_RX	Input Function Name - USART2 Receive
PORTS_REMAP_FUNCTION_USART3_CTS	Input Function Name - USART3 Clear To Send
PORTS_REMAP_FUNCTION_USART3_RX	Input Function Name - USART3 Receive
PORTS_REMAP_FUNCTION_USART4_CTS	Input Function Name - USART4 Clear To Send
PORTS_REMAP_FUNCTION_USART4_RX	Input Function Name - USART4 Receive
PORTS_REMAP_FUNCTION_REFCLKI	Input Function Name - Reference Clock Input
PORTS_REMAP_FUNCTION_NULL	Output Function Name - Null
PORTS_REMAP_FUNCTION_C1OUT	Output Function Name - Comparator 1 Output
PORTS_REMAP_FUNCTION_C2OUT	Output Function Name - Comparator 2 Output
PORTS_REMAP_FUNCTION_U1TX	Output Function Name - UART1 Transmit
PORTS_REMAP_FUNCTION_U1RTS	Output Function Name - UART1 Request To Send
PORTS_REMAP_FUNCTION_U2TX	Output Function Name - UART2 Transmit
PORTS_REMAP_FUNCTION_U2RTS	Output Function Name - UART2 Request To Send
PORTS_REMAP_FUNCTION_SDO1	Output Function Name - SPI1 Data Output
PORTS_REMAP_FUNCTION_SCK1OUT	Output Function Name - SPI1 Clock Output
PORTS_REMAP_FUNCTION_SS1OUT	Output Function Name - SPI1 Slave Select Output
PORTS_REMAP_FUNCTION_SDO2	Output Function Name - SPI2 Data Output
PORTS_REMAP_FUNCTION_SCK2OUT	Output Function Name - SPI2 Clock Output
PORTS_REMAP_FUNCTION_SS2OUT	Output Function Name - SPI2 Slave Select Output
PORTS_REMAP_FUNCTION_OC1	Output Function Name - Output Compare 1
PORTS_REMAP_FUNCTION_OC2	Output Function Name - Output Compare 2

PORTS_REMAP_FUNCTION_OC3	Output Function Name - Output Compare 3
PORTS_REMAP_FUNCTION_OC4	Output Function Name - Output Compare 4
PORTS_REMAP_FUNCTION_OC5	Output Function Name - Output Compare 5
PORTS_REMAP_FUNCTION_OC6	Output Function Name - Output Compare 6
PORTS_REMAP_FUNCTION_OC7	Output Function Name - Output Compare 7
PORTS_REMAP_FUNCTION_OC8	Output Function Name - Output Compare 8
PORTS_REMAP_FUNCTION_U3TX	Output Function Name - UART3 Transmit
PORTS_REMAP_FUNCTION_U3RTS	Output Function Name - UART3 Request To Send
PORTS_REMAP_FUNCTION_U4TX	Output Function Name - UART4 Transmit
PORTS_REMAP_FUNCTION_U4RTS	Output Function Name - UART4 Request To Send
PORTS_REMAP_FUNCTION_SDO3	Output Function Name - SPI3 Data Output
PORTS_REMAP_FUNCTION_SCK3OUT	Output Function Name - SPI3 Clock Output
PORTS_REMAP_FUNCTION_SS3OUT	Output Function Name - SPI3 Slave Select Output
PORTS_REMAP_FUNCTION_OC9	Output Function Name - Output Compare 9
PORTS_REMAP_FUNCTION_C3OUT	Output Function Name - Comparator 3 Output
PORTS_REMAP_FUNCTION_REFCLKO	Output Function Name - Reference Clock Output

Description

Remap Function Enumeration

This data type defines the different remap function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_INPUT_FUNCTION Enumeration

Data type defining the different remap input function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  INPUT_FUNC_INT1,
  INPUT_FUNC_INT2,
  INPUT_FUNC_INT3,
  INPUT_FUNC_INT4,
  INPUT_FUNC_T2CK,
  INPUT_FUNC_T3CK,
  INPUT_FUNC_T4CK,
  INPUT_FUNC_T5CK,
  INPUT_FUNC_T6CK,
  INPUT_FUNC_T7CK,
  INPUT_FUNC_T8CK,
  INPUT_FUNC_T9CK,
  INPUT_FUNC_IC1,
  INPUT_FUNC_IC2,
  INPUT_FUNC_IC3,
  INPUT_FUNC_IC4,
  INPUT_FUNC_IC5,
  INPUT_FUNC_IC6,
  INPUT_FUNC_IC7,
  INPUT_FUNC_IC8,
  INPUT_FUNC_IC9,
  INPUT_FUNC_OCFA,
  INPUT_FUNC_OCFB,
  INPUT_FUNC_U1RX,
  INPUT_FUNC_U1CTS,
  INPUT_FUNC_U2RX,
  INPUT_FUNC_U2CTS,
  INPUT_FUNC_U3RX,
  INPUT_FUNC_U3CTS,
  INPUT_FUNC_U4RX,
  INPUT_FUNC_U4CTS,
  INPUT_FUNC_U5RX,
```

```
    INPUT_FUNC_U5CTS,
    INPUT_FUNC_U6RX,
    INPUT_FUNC_U6CTS,
    INPUT_FUNC_SDI1,
    INPUT_FUNC_SS1,
    INPUT_FUNC_SDI2,
    INPUT_FUNC_SS2,
    INPUT_FUNC_SDI3,
    INPUT_FUNC_SS3,
    INPUT_FUNC_SDI4,
    INPUT_FUNC_SS4,
    INPUT_FUNC_SDI5,
    INPUT_FUNC_SS5,
    INPUT_FUNC_SDI6,
    INPUT_FUNC_SS6,
    INPUT_FUNC_C1RX,
    INPUT_FUNC_C2RX,
    INPUT_FUNC_REFCLKI1,
    INPUT_FUNC_REFCLKI3,
    INPUT_FUNC_REFCLKI4,
    REMAP_NOT_SUPPORTED
} PORTS_REMAP_INPUT_FUNCTION;
```

Description

Remap Input Function Enumeration

This data type defines the different remap input function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_INPUT_PIN Enumeration

Data type defining the different Ports I/O input pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
    INPUT_PIN_RPD2,
    INPUT_PIN_RPG8,
    INPUT_PIN_RPF4,
    INPUT_PIN_RPD10,
    INPUT_PIN_RPF1,
    INPUT_PIN_RPB9,
    INPUT_PIN_RPB10,
    INPUT_PIN_RPC14,
    INPUT_PIN_RPB5,
    INPUT_PIN_RPC1,
    INPUT_PIN_RPD14,
    INPUT_PIN_RPG1,
    INPUT_PIN_RPA14,
    INPUT_PIN_RPD6,
    INPUT_PIN_RPD3,
    INPUT_PIN_RPG7,
    INPUT_PIN_RPF5,
    INPUT_PIN_RPD11,
    INPUT_PIN_RPF0,
    INPUT_PIN_RPB1,
    INPUT_PIN_RPE5,
    INPUT_PIN_RPC13,
    INPUT_PIN_RPB3,
    INPUT_PIN_RPC4,
    INPUT_PIN_RPD15,
    INPUT_PIN_RPG0,
    INPUT_PIN_RPA15,
    INPUT_PIN_RPD7,
    INPUT_PIN_RPD9,
    INPUT_PIN_RPG6,
    INPUT_PIN_RPB8,
    INPUT_PIN_RPB15,
```

```
    INPUT_PIN_RPD4,
    INPUT_PIN_RPB0,
    INPUT_PIN_RPE3,
    INPUT_PIN_RPB7,
    INPUT_PIN_RPF12,
    INPUT_PIN_RPD12,
    INPUT_PIN_RPF8,
    INPUT_PIN_RPC3,
    INPUT_PIN_RPE9,
    INPUT_PIN_RPD1,
    INPUT_PIN_RPG9,
    INPUT_PIN_RPB14,
    INPUT_PIN_RPD0,
    INPUT_PIN_RPB6,
    INPUT_PIN_RPD5,
    INPUT_PIN_RPB2,
    INPUT_PIN_RPF3,
    INPUT_PIN_RPF13,
    INPUT_PIN_NC,
    INPUT_PIN_RPF2,
    INPUT_PIN_RPC2,
    INPUT_PIN_RPE8,
    INPUT_PIN_RPF7,
    INPUT_PIN_RPD8,
    INPUT_PIN_RPA0,
    INPUT_PIN_RPB4,
    INPUT_PIN_RPC7,
    INPUT_PIN_RPC0,
    INPUT_PIN_RPC5,
    INPUT_PIN_RPA1,
    INPUT_PIN_RPB11,
    INPUT_PIN_RPA8,
    INPUT_PIN_RPC8,
    INPUT_PIN_RPA9,
    INPUT_PIN_RPA2,
    INPUT_PIN_RPA4,
    INPUT_PIN_RPB13,
    INPUT_PIN_RPC6,
    INPUT_PIN_RPA3,
    INPUT_PIN_RPC9,
    PIN_REMAP_NOT_SUPPORTED
} PORTS_REMAP_INPUT_PIN;
```

Description

PORTS IO Input Pins enumeration

This data type defines the different Ports I/O input pins enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_OUTPUT_FUNCTION Enumeration

Data type defining the different remap output function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
    OUTPUT_FUNC_U3TX,
    OUTPUT_FUNC_U4RTS,
    OUTPUT_FUNC_SDO1,
    OUTPUT_FUNC_SDO2,
    OUTPUT_FUNC_SDO3,
    OUTPUT_FUNC_SDO5,
    OUTPUT_FUNC_SS6,
    OUTPUT_FUNC_OC3,
    OUTPUT_FUNC_OC6,
    OUTPUT_FUNC_REFCLKO4,
    OUTPUT_FUNC_C2OUT,
    OUTPUT_FUNC_C1TX,
```

```
  OUTPUT_FUNC_U1TX,
  OUTPUT_FUNC_U2RTS,
  OUTPUT_FUNC_U5TX,
  OUTPUT_FUNC_U6RTS,
  OUTPUT_FUNC_SDO4,
  OUTPUT_FUNC_OC4,
  OUTPUT_FUNC_OC7,
  OUTPUT_FUNC_REFCLKO1,
  OUTPUT_FUNC_U3RTS,
  OUTPUT_FUNC_U4TX,
  OUTPUT_FUNC_U6TX,
  OUTPUT_FUNC_SS1,
  OUTPUT_FUNC_SS3,
  OUTPUT_FUNC_SS4,
  OUTPUT_FUNC_SS5,
  OUTPUT_FUNC_SDO6,
  OUTPUT_FUNC_OC5,
  OUTPUT_FUNC_OC8,
  OUTPUT_FUNC_C1OUT,
  OUTPUT_FUNC_REFCLKO3,
  OUTPUT_FUNC_U1RTS,
  OUTPUT_FUNC_U2TX,
  OUTPUT_FUNC_U5RTS,
  OUTPUT_FUNC_SS2,
  OUTPUT_FUNC_OC2,
  OUTPUT_FUNC_OC1,
  OUTPUT_FUNC_OC9,
  OUTPUT_FUNC_C2TX,
  OUTPUT_FUNC_C3OUT,
  OUTPUT_FUNC_REFCLKO,
  OUTPUT_FUNC_NO_CONNECT
} PORTS_REMAP_OUTPUT_FUNCTION;
```

Description

Remap Output Function Enumeration

This data type defines the different remap output function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_OUTPUT_PIN Enumeration

Data type defining the different Ports I/O output pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  OUTPUT_PIN_RPA0,
  OUTPUT_PIN_RPA1,
  OUTPUT_PIN_RPA2,
  OUTPUT_PIN_RPA3,
  OUTPUT_PIN_RPA4,
  OUTPUT_PIN_RPA8,
  OUTPUT_PIN_RPA9,
  OUTPUT_PIN_RPA14,
  OUTPUT_PIN_RPA15,
  OUTPUT_PIN_RPB0,
  OUTPUT_PIN_RPB1,
  OUTPUT_PIN_RPB2,
  OUTPUT_PIN_RPB3,
  OUTPUT_PIN_RPB4,
  OUTPUT_PIN_RPB5,
  OUTPUT_PIN_RPB6,
  OUTPUT_PIN_RPB7,
  OUTPUT_PIN_RPB8,
  OUTPUT_PIN_RPB9,
  OUTPUT_PIN_RPB10,
  OUTPUT_PIN_RPB11,
  OUTPUT_PIN_RPB13,
```

```
  OUTPUT_PIN_RPB14,
  OUTPUT_PIN_RPB15,
  OUTPUT_PIN_RPC0,
  OUTPUT_PIN_RPC1,
  OUTPUT_PIN_RPC2,
  OUTPUT_PIN_RPC3,
  OUTPUT_PIN_RPC4,
  OUTPUT_PIN_RPC5,
  OUTPUT_PIN_RPC6,
  OUTPUT_PIN_RPC7,
  OUTPUT_PIN_RPC8,
  OUTPUT_PIN_RPC9,
  OUTPUT_PIN_RPC13,
  OUTPUT_PIN_RPC14,
  OUTPUT_PIN_RPD0,
  OUTPUT_PIN_RPD1,
  OUTPUT_PIN_RPD2,
  OUTPUT_PIN_RPD3,
  OUTPUT_PIN_RPD4,
  OUTPUT_PIN_RPD5,
  OUTPUT_PIN_RPD6,
  OUTPUT_PIN_RPD7,
  OUTPUT_PIN_RPD8,
  OUTPUT_PIN_RPD9,
  OUTPUT_PIN_RPD10,
  OUTPUT_PIN_RPD11,
  OUTPUT_PIN_RPD12,
  OUTPUT_PIN_RPD14,
  OUTPUT_PIN_RPD15,
  OUTPUT_PIN_RPE3,
  OUTPUT_PIN_RPE5,
  OUTPUT_PIN_RPE8,
  OUTPUT_PIN_RPE9,
  OUTPUT_PIN_RPF0,
  OUTPUT_PIN_RPF1,
  OUTPUT_PIN_RPF2,
  OUTPUT_PIN_RPF3,
  OUTPUT_PIN_RPF4,
  OUTPUT_PIN_RPF5,
  OUTPUT_PIN_RPF6,
  OUTPUT_PIN_RPF7,
  OUTPUT_PIN_RPF8,
  OUTPUT_PIN_RPF12,
  OUTPUT_PIN_RPF13,
  OUTPUT_PIN_RPG0,
  OUTPUT_PIN_RPG1,
  OUTPUT_PIN_RPG6,
  OUTPUT_PIN_RPG7,
  OUTPUT_PIN_RPG8,
  OUTPUT_PIN_RPG9,
  REMAP_NOT_SUPPORTED
} PORTS_REMAP_OUTPUT_PIN;
```

Description

PORTS IO Output Pins enumeration

This data type defines the different Ports I/O output pins enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_PIN Enumeration

Data type defining the different remappable input/output enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_REMAP_PIN_RPI0,
  PORTS_REMAP_PIN_RPI1,
```

```
    PORTS_REMAP_PIN_RPI2,
    PORTS_REMAP_PIN_RPI3,
    PORTS_REMAP_PIN_RPI4,
    PORTS_REMAP_PIN_RPI5,
    PORTS_REMAP_PIN_RPI6,
    PORTS_REMAP_PIN_RPI7,
    PORTS_REMAP_PIN_RPI8,
    PORTS_REMAP_PIN_RPI9,
    PORTS_REMAP_PIN_RPI10,
    PORTS_REMAP_PIN_RPI11,
    PORTS_REMAP_PIN_RPI12,
    PORTS_REMAP_PIN_RPI13,
    PORTS_REMAP_PIN_RPI14,
    PORTS_REMAP_PIN_RPI15,
    PORTS_REMAP_PIN_RPI16,
    PORTS_REMAP_PIN_RPI17,
    PORTS_REMAP_PIN_RPI18,
    PORTS_REMAP_PIN_RPI19,
    PORTS_REMAP_PIN_RPI20,
    PORTS_REMAP_PIN_RPI21,
    PORTS_REMAP_PIN_RPI22,
    PORTS_REMAP_PIN_RPI23,
    PORTS_REMAP_PIN_RPI24,
    PORTS_REMAP_PIN_RPI25,
    PORTS_REMAP_PIN_RPI26,
    PORTS_REMAP_PIN_RPI27,
    PORTS_REMAP_PIN_RPI28,
    PORTS_REMAP_PIN_RPI29,
    PORTS_REMAP_PIN_RPI30,
    PORTS_REMAP_PIN_RPI31,
    PORTS_REMAP_PIN_RPI32,
    PORTS_REMAP_PIN_RPI33,
    PORTS_REMAP_PIN_RPI34,
    PORTS_REMAP_PIN_RPI35,
    PORTS_REMAP_PIN_RPI36,
    PORTS_REMAP_PIN_RPI37,
    PORTS_REMAP_PIN_RPI38,
    PORTS_REMAP_PIN_RPI39,
    PORTS_REMAP_PIN_RPI40,
    PORTS_REMAP_PIN_RPI41,
    PORTS_REMAP_PIN_RPI42,
    PORTS_REMAP_PIN_RPI43,
    PORTS_REMAP_PIN_RPI44,
    PORTS_REMAP_PIN_RPI45,
    PORTS_REMAP_PIN_RP0,
    PORTS_REMAP_PIN_RP1,
    PORTS_REMAP_PIN_RP2,
    PORTS_REMAP_PIN_RP3,
    PORTS_REMAP_PIN_RP4,
    PORTS_REMAP_PIN_RP5,
    PORTS_REMAP_PIN_RP6,
    PORTS_REMAP_PIN_RP7,
    PORTS_REMAP_PIN_RP8,
    PORTS_REMAP_PIN_RP9,
    PORTS_REMAP_PIN_RP10,
    PORTS_REMAP_PIN_RP11,
    PORTS_REMAP_PIN_RP12,
    PORTS_REMAP_PIN_RP13,
    PORTS_REMAP_PIN_RP14,
    PORTS_REMAP_PIN_RP15,
    PORTS_REMAP_PIN_RP16,
    PORTS_REMAP_PIN_RP17,
    PORTS_REMAP_PIN_RP18,
    PORTS_REMAP_PIN_RP19,
    PORTS_REMAP_PIN_RP20,
    PORTS_REMAP_PIN_RP21,
    PORTS_REMAP_PIN_RP22,
    PORTS_REMAP_PIN_RP23,
    PORTS_REMAP_PIN_RP24,
    PORTS_REMAP_PIN_RP25,
    PORTS_REMAP_PIN_RP26,
    PORTS_REMAP_PIN_RP27,
    PORTS_REMAP_PIN_RP28,
    PORTS_REMAP_PIN_RP29,
    PORTS_REMAP_PIN_RP30,
```

```
    PORTS_REMAP_PIN_RP31,
    PORTS_REMAP_PIN_RPA0,
    PORTS_REMAP_PIN_RPB3,
    PORTS_REMAP_PIN_RPB4,
    PORTS_REMAP_PIN_RPB15,
    PORTS_REMAP_PIN_RPB7,
    PORTS_REMAP_PIN_RPC7,
    PORTS_REMAP_PIN_RPC0,
    PORTS_REMAP_PIN_RPC5,
    PORTS_REMAP_PIN_RPA1,
    PORTS_REMAP_PIN_RPB5,
    PORTS_REMAP_PIN_RPB1,
    PORTS_REMAP_PIN_RPB11,
    PORTS_REMAP_PIN_RPB8,
    PORTS_REMAP_PIN_RPA8,
    PORTS_REMAP_PIN_RPC8,
    PORTS_REMAP_PIN_RPA9,
    PORTS_REMAP_PIN_RPA2,
    PORTS_REMAP_PIN_RPB6,
    PORTS_REMAP_PIN_RPA4,
    PORTS_REMAP_PIN_RPB13,
    PORTS_REMAP_PIN_RPB2,
    PORTS_REMAP_PIN_RPC6,
    PORTS_REMAP_PIN_RPC1,
    PORTS_REMAP_PIN_RPC3,
    PORTS_REMAP_PIN_RPA3,
    PORTS_REMAP_PIN_RPB14,
    PORTS_REMAP_PIN_RPB0,
    PORTS_REMAP_PIN_RPB10,
    PORTS_REMAP_PIN_RPB9,
    PORTS_REMAP_PIN_RPC9,
    PORTS_REMAP_PIN_RPC2,
    PORTS_REMAP_PIN_RPC4
} PORTS_REMAP_PIN;
```

Members

Members	Description
PORTS_REMAP_PIN_RPI0	Remappable input RPI0
PORTS_REMAP_PIN_RPI1	Remappable input RPI1
PORTS_REMAP_PIN_RPI2	Remappable input RPI2
PORTS_REMAP_PIN_RPI3	Remappable input RPI3
PORTS_REMAP_PIN_RPI4	Remappable input RPI4
PORTS_REMAP_PIN_RPI5	Remappable input RPI5
PORTS_REMAP_PIN_RPI6	Remappable input RPI6
PORTS_REMAP_PIN_RPI7	Remappable input RPI7
PORTS_REMAP_PIN_RPI8	Remappable input RPI8
PORTS_REMAP_PIN_RPI9	Remappable input RPI9
PORTS_REMAP_PIN_RPI10	Remappable input RPI10
PORTS_REMAP_PIN_RPI11	Remappable input RPI11
PORTS_REMAP_PIN_RPI12	Remappable input RPI12
PORTS_REMAP_PIN_RPI13	Remappable input RPI13
PORTS_REMAP_PIN_RPI14	Remappable input RPI14
PORTS_REMAP_PIN_RPI15	Remappable input RPI15
PORTS_REMAP_PIN_RPI16	Remappable input RPI16
PORTS_REMAP_PIN_RPI17	Remappable input RPI17
PORTS_REMAP_PIN_RPI18	Remappable input RPI18
PORTS_REMAP_PIN_RPI19	Remappable input RPI19
PORTS_REMAP_PIN_RPI20	Remappable input RPI20
PORTS_REMAP_PIN_RPI21	Remappable input RPI21
PORTS_REMAP_PIN_RPI22	Remappable input RPI22
PORTS_REMAP_PIN_RPI23	Remappable input RPI23
PORTS_REMAP_PIN_RPI24	Remappable input RPI24
PORTS_REMAP_PIN_RPI25	Remappable input RPI25
PORTS_REMAP_PIN_RPI26	Remappable input RPI26

PORTS_REMAP_PIN_RPI27	Remappable input RPI27
PORTS_REMAP_PIN_RPI28	Remappable input RPI28
PORTS_REMAP_PIN_RPI29	Remappable input RPI29
PORTS_REMAP_PIN_RPI30	Remappable input RPI30
PORTS_REMAP_PIN_RPI31	Remappable input RPI31
PORTS_REMAP_PIN_RPI32	Remappable input RPI32
PORTS_REMAP_PIN_RPI33	Remappable input RPI33
PORTS_REMAP_PIN_RPI34	Remappable input RPI34
PORTS_REMAP_PIN_RPI35	Remappable input RPI35
PORTS_REMAP_PIN_RPI36	Remappable input RPI36
PORTS_REMAP_PIN_RPI37	Remappable input RPI37
PORTS_REMAP_PIN_RPI38	Remappable input RPI38
PORTS_REMAP_PIN_RPI39	Remappable input RPI39
PORTS_REMAP_PIN_RPI40	Remappable input RPI40
PORTS_REMAP_PIN_RPI41	Remappable input RPI41
PORTS_REMAP_PIN_RPI42	Remappable input RPI42
PORTS_REMAP_PIN_RPI43	Remappable input RPI43
PORTS_REMAP_PIN_RPI44	Remappable input RPI44
PORTS_REMAP_PIN_RPI45	Remappable input RPI45
PORTS_REMAP_PIN_RP0	Remappable output RP0
PORTS_REMAP_PIN_RP1	Remappable output RP1
PORTS_REMAP_PIN_RP2	Remappable output RP2
PORTS_REMAP_PIN_RP3	Remappable output RP3
PORTS_REMAP_PIN_RP4	Remappable output RP4
PORTS_REMAP_PIN_RP5	Remappable output RP5
PORTS_REMAP_PIN_RP6	Remappable output RP6
PORTS_REMAP_PIN_RP7	Remappable output RP7
PORTS_REMAP_PIN_RP8	Remappable output RP8
PORTS_REMAP_PIN_RP9	Remappable output RP9
PORTS_REMAP_PIN_RP10	Remappable output RP10
PORTS_REMAP_PIN_RP11	Remappable output RP11
PORTS_REMAP_PIN_RP12	Remappable output RP12
PORTS_REMAP_PIN_RP13	Remappable output RP13
PORTS_REMAP_PIN_RP14	Remappable output RP14
PORTS_REMAP_PIN_RP15	Remappable output RP15
PORTS_REMAP_PIN_RP16	Remappable output RP16
PORTS_REMAP_PIN_RP17	Remappable output RP17
PORTS_REMAP_PIN_RP18	Remappable output RP18
PORTS_REMAP_PIN_RP19	Remappable output RP19
PORTS_REMAP_PIN_RP20	Remappable output RP20
PORTS_REMAP_PIN_RP21	Remappable output RP21
PORTS_REMAP_PIN_RP22	Remappable output RP22
PORTS_REMAP_PIN_RP23	Remappable output RP23
PORTS_REMAP_PIN_RP24	Remappable output RP24
PORTS_REMAP_PIN_RP25	Remappable output RP25
PORTS_REMAP_PIN_RP26	Remappable output RP26
PORTS_REMAP_PIN_RP27	Remappable output RP27
PORTS_REMAP_PIN_RP28	Remappable output RP28
PORTS_REMAP_PIN_RP29	Remappable output RP29
PORTS_REMAP_PIN_RP30	Remappable output RP30
PORTS_REMAP_PIN_RP31	Remappable output RP31
PORTS_REMAP_PIN_RPA0	Remappable output RPA0
PORTS_REMAP_PIN_RPB3	Remappable output RPB3
PORTS_REMAP_PIN_RPB4	Remappable output RPB4
PORTS_REMAP_PIN_RPB15	Remappable output RPB15

PORTS_REMAP_PIN_RPB7	Remappable output RPB7
PORTS_REMAP_PIN_RPC7	Remappable output RPC7
PORTS_REMAP_PIN_RPC0	Remappable output RPC0
PORTS_REMAP_PIN_RPC5	Remappable output RPC5
PORTS_REMAP_PIN_RPA1	Remappable output RPA1
PORTS_REMAP_PIN_RPB5	Remappable output RPB5
PORTS_REMAP_PIN_RPB1	Remappable output RPB1
PORTS_REMAP_PIN_RPB11	Remappable output RPB11
PORTS_REMAP_PIN_RPB8	Remappable output RPB8
PORTS_REMAP_PIN_RPA8	Remappable output RPA8
PORTS_REMAP_PIN_RPC8	Remappable output RPC8
PORTS_REMAP_PIN_RPA9	Remappable output RPA9
PORTS_REMAP_PIN_RPA2	Remappable output RPA2
PORTS_REMAP_PIN_RPB6	Remappable output RPB6
PORTS_REMAP_PIN_RPA4	Remappable output RPA4
PORTS_REMAP_PIN_RPB13	Remappable output RPB13
PORTS_REMAP_PIN_RPB2	Remappable output RPB2
PORTS_REMAP_PIN_RPC6	Remappable output RPC6
PORTS_REMAP_PIN_RPC1	Remappable output RPC1
PORTS_REMAP_PIN_RPC3	Remappable output RPC3
PORTS_REMAP_PIN_RPA3	Remappable output RPA3
PORTS_REMAP_PIN_RPB14	Remappable output RPB14
PORTS_REMAP_PIN_RPB0	Remappable output RPB0
PORTS_REMAP_PIN_RPB10	Remappable output RPB10
PORTS_REMAP_PIN_RPB9	Remappable output RPB9
PORTS_REMAP_PIN_RPC9	Remappable output RPC9
PORTS_REMAP_PIN_RPC2	Remappable output RPC2
PORTS_REMAP_PIN_RPC4	Remappable output RPC4

Description

Remappable Input/Output Enumeration

This data type defines the different remappable input/output enumerations.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

PORTS_AN_PIN Enumeration

Data type defining the different analog input pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_AN_PIN_0,
  PORTS_AN_PIN_1,
  PORTS_AN_PIN_2,
  PORTS_AN_PIN_3,
  PORTS_AN_PIN_4,
  PORTS_AN_PIN_5,
  PORTS_AN_PIN_6,
  PORTS_AN_PIN_7,
  PORTS_AN_PIN_8,
  PORTS_AN_PIN_9,
  PORTS_AN_PIN_10,
  PORTS_AN_PIN_11,
  PORTS_AN_PIN_12,
  PORTS_AN_PIN_13,
  PORTS_AN_PIN_14,
  PORTS_AN_PIN_15,
```

```
  PORTS_AN_PIN_ALL
} PORTS_AN_PIN;
```

Members

Members	Description
PORTS_AN_PIN_0	AN Pin 0
PORTS_AN_PIN_1	AN Pin 1
PORTS_AN_PIN_2	AN Pin 2
PORTS_AN_PIN_3	AN Pin 3
PORTS_AN_PIN_4	AN Pin 4
PORTS_AN_PIN_5	AN Pin 5
PORTS_AN_PIN_6	AN Pin 6
PORTS_AN_PIN_7	AN Pin 7
PORTS_AN_PIN_8	AN Pin 8
PORTS_AN_PIN_9	AN Pin 9
PORTS_AN_PIN_10	AN Pin 10
PORTS_AN_PIN_11	AN Pin 11
PORTS_AN_PIN_12	AN Pin 12
PORTS_AN_PIN_13	AN Pin 13
PORTS_AN_PIN_14	AN Pin 14
PORTS_AN_PIN_15	AN Pin 15
PORTS_AN_PIN_ALL	All the AN Pins

Description

Analog input pins

This data type defines the different analog input pins.

Remarks

Values of these AN Pin enums are different from the other similar enumerators by name PORTS_ANALOG_PIN.

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CN_PIN Enumeration

Data type defining the different Change Notification (CN) pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  CHANGE_NOTICE_PIN_0,
  CHANGE_NOTICE_PIN_1,
  CHANGE_NOTICE_PIN_2,
  CHANGE_NOTICE_PIN_3,
  CHANGE_NOTICE_PIN_4,
  CHANGE_NOTICE_PIN_5,
  CHANGE_NOTICE_PIN_6,
  CHANGE_NOTICE_PIN_7,
  CHANGE_NOTICE_PIN_8,
  CHANGE_NOTICE_PIN_9,
  CHANGE_NOTICE_PIN_10,
  CHANGE_NOTICE_PIN_11,
  CHANGE_NOTICE_PIN_12,
  CHANGE_NOTICE_PIN_13,
  CHANGE_NOTICE_PIN_14,
  CHANGE_NOTICE_PIN_15,
  CHANGE_NOTICE_PIN_16,
  CHANGE_NOTICE_PIN_17,
  CHANGE_NOTICE_PIN_18,
  CHANGE_NOTICE_PIN_19,
  CHANGE_NOTICE_PIN_20,
  CHANGE_NOTICE_PIN_21,
  CHANGE_NOTICE_PIN_ALL
} PORTS_CN_PIN;
```

Members

Members	Description
CHANGE_NOTICE_PIN_0	Change Notice Pin 0
CHANGE_NOTICE_PIN_1	Change Notice Pin 1
CHANGE_NOTICE_PIN_2	Change Notice Pin 2
CHANGE_NOTICE_PIN_3	Change Notice Pin 3
CHANGE_NOTICE_PIN_4	Change Notice Pin 4
CHANGE_NOTICE_PIN_5	Change Notice Pin 5
CHANGE_NOTICE_PIN_6	Change Notice Pin 6
CHANGE_NOTICE_PIN_7	Change Notice Pin 7
CHANGE_NOTICE_PIN_8	Change Notice Pin 8
CHANGE_NOTICE_PIN_9	Change Notice Pin 9
CHANGE_NOTICE_PIN_10	Change Notice Pin 10
CHANGE_NOTICE_PIN_11	Change Notice Pin 11
CHANGE_NOTICE_PIN_12	Change Notice Pin 12
CHANGE_NOTICE_PIN_13	Change Notice Pin 13
CHANGE_NOTICE_PIN_14	Change Notice Pin 14
CHANGE_NOTICE_PIN_15	Change Notice Pin 15
CHANGE_NOTICE_PIN_16	Change Notice Pin 16
CHANGE_NOTICE_PIN_17	Change Notice Pin 17
CHANGE_NOTICE_PIN_18	Change Notice Pin 18
CHANGE_NOTICE_PIN_19	Change Notice Pin 19
CHANGE_NOTICE_PIN_20	Change Notice Pin 20
CHANGE_NOTICE_PIN_21	Change Notice Pin 21
CHANGE_NOTICE_PIN_ALL	All the Change Notice Pins

Description

Change Notice Pins enumeration

This data type defines the different CN pins enumerations.

Remarks

Values of these CN Pin enums are different from the other similar enumerators by name PORTS_CHANGE_NOTICE_PIN. These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_EDGE Enumeration

Data type defining the different edge type for change notification.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_CHANGE_NOTICE_EDGE_RISING = 0,
  PORTS_CHANGE_NOTICE_EDGE_FALLING = 1
} PORTS_CHANGE_NOTICE_EDGE;
```

Members

Members	Description
PORTS_CHANGE_NOTICE_EDGE_RISING = 0	Change Notification on rising edge
PORTS_CHANGE_NOTICE_EDGE_FALLING = 1	Change Notification on falling edge

Description

PORTS Change Notice Edge Type

This data type defines the different edge type for change notification.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_METHOD Enumeration

Data type defining the different method of ports pin change notification.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_CHANGE_NOTICE_METHOD_SAMPLED = 0,
  PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT = 1
} PORTS_CHANGE_NOTICE_METHOD;
```

Members

Members	Description
PORTS_CHANGE_NOTICE_METHOD_SAMPLED = 0	Change notification happens when there is level transition
PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT = 1	Change notification happens when there is edge transition

Description

PORTS Pin Change Notice Method

This data type defines the different method of ports pin change notification.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_PIN_SLEW_RATE Enumeration

Data type defining the different slew rates for port pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_PIN_SLEW_RATE_FASTEST = 0x00,
  PORTS_PIN_SLEW_RATE_FAST = 0x01,
  PORTS_PIN_SLEW_RATE_SLOW = 0x02,
  PORTS_PIN_SLEW_RATE_SLOWEST = 0x03
} PORTS_PIN_SLEW_RATE;
```

Members

Members	Description
PORTS_PIN_SLEW_RATE_FASTEST = 0x00	Slew rate fastest
PORTS_PIN_SLEW_RATE_FAST = 0x01	Slew rate fast
PORTS_PIN_SLEW_RATE_SLOW = 0x02	Slew rate slow
PORTS_PIN_SLEW_RATE_SLOWEST = 0x03	Slew rate slowest

Description

PORTS Pin Slew Rate

This data type defines the different slew rates for port pins.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

Files

Files

Name	Description
plib_ports.h	Ports Peripheral Library Interface header for Ports function definitions.
help_plib_ports.h	This is file help_plib_ports.h.

Description

This section lists the source and header files used by the library.

plib_ports.h

Ports Peripheral Library Interface header for Ports function definitions.

Functions

	Name	Description
	PLIB_PORTS_AnPinsModeSelect	Enables the selected AN pins as analog or digital.
	PLIB_PORTS_ChangeNoticeDisable	Global Change Notice disable.
	PLIB_PORTS_ChangeNoticeEnable	Global Change Notice enable.
	PLIB_PORTS_ChangeNoticeInIdleDisable	CPU Idle halts the Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdleEnable	CPU Idle mode does not affect Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdlePerPortDisable	Change Notification halts in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticeInIdlePerPortEnable	Allows CN to be working in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticePerPortHasOccured	This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.
	PLIB_PORTS_ChangeNoticePerPortHasOccurred	checks the status of change on the pin
	PLIB_PORTS_ChangeNoticePerPortTurnOff	Disables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePerPortTurnOn	Enables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePullDownPerPortDisable	Disables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullDownPerPortEnable	Enables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullUpDisable	Disable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpEnable	Enable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpPerPortDisable	Disables weak pull-up for the selected pin.
	PLIB_PORTS_ChangeNoticePullUpPerPortEnable	Enables the pull-up for selected Change Notice pins.
	PLIB_PORTS_ChannelChangeNoticeDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeEdgeDisable	Disables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeEnable	Enables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeMethodGet	Gets the Change Notice style for the selected port channel.
	PLIB_PORTS_ChannelChangeNoticeMethodSelect	Selects the Change Notice style for selected port channel.
	PLIB_PORTS_ChannelChangeNoticePullDownDisable	Disables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullDownEnable	Enables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelModeSelect	Enables the selected channel pins as analog or digital.
	PLIB_PORTS_ChannelSlewRateSelect	Selects the slew rate for selected channel pins.
	PLIB_PORTS_Clear	Clears the selected digital port/latch bits.
	PLIB_PORTS_CnPinsDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsPullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_CnPinsPullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_DirectionGet	Reads the direction of the selected digital port.
	PLIB_PORTS_DirectionInputSet	Makes the selected pins direction input.
	PLIB_PORTS_DirectionOutputSet	Makes the selected pins direction output.
	PLIB_PORTS_ExistsAnPinsMode	Identifies whether the AnPinsMode feature exists on the Ports module.

	PLIB_PORTS_ExistsChangeNotice	Identifies whether the ChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeControl	Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeStatus	Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeInIdle	Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortInIdle	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortStatus	Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortTurnOn	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullDownPerPort	Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUp	Identifies whether the ChangeNoticePullup feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUpPerPort	Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChannelChangeNoticeMethod	Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.
	PLIB_PORTS_ExistsLatchRead	Identifies whether the LatchRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNotice	Identifies whether the PinChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNoticePerPort	Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinMode	Identifies whether the PinMode feature exists on the Ports module.
	PLIB_PORTS_ExistsPinModePerPort	Identifies whether the PinModePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the Ports module.
	PLIB_PORTS_ExistsSlewRateControl	Identifies whether the SlewRateControl feature exists on the Ports module.
	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port pins.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEdgeHasOccurred	Check Change Notice edge status.
	PLIB_PORTS_PinChangeNoticeEdgeIsEnabled	Check if Change Notice edge is enabled or not.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice interrupt enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Disables CN interrupt for the selected pin.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables CN interrupt for the selected pin.
	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
	PLIB_PORTS_PinDirectionInputSet	Makes the selected pin direction input
	PLIB_PORTS_PinDirectionOutputSet	Makes the selected pin direction output
	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
	PLIB_PORTS_PinGetLatched	Reads/Gets data from the selected latch.
	PLIB_PORTS_PinModePerPortSelect	Enables the selected port pin as analog or digital.
	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.
	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
	PLIB_PORTS_PinSlewRateGet	Gets the slew rate for selected port pin.
	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
	PLIB_PORTS_Read	Reads the selected digital port.
	PLIB_PORTS_ReadLatched	Reads and returns data from the selected Latch.
	PLIB_PORTS_RemapInput	Input function remapping.

	PLIB_PORTS_RemapOutput	Output function remapping.
	PLIB_PORTS_Set	Sets the selected bits of the port.
	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
	PLIB_PORTS_Write	Writes the selected digital port/latch.

Types

	Name	Description
	PORTS_DATA_MASK	Data type defining the Ports data mask
	PORTS_DATA_TYPE	Data type defining the Ports data type.

Description

Ports Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ports Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Ports peripheral.

File Name

plib_ports.h

Company

Microchip Technology Inc.

help_plib_ports.h

Enumerations

	Name	Description
	PORTS_AN_PIN	Data type defining the different analog input pins.
	PORTS_ANALOG_PIN	Data type defining the different analog input pins.
	PORTS_BIT_POS	Lists the constants that hold different PORTS bit positions.
	PORTS_CHANGE_NOTICE_EDGE	Data type defining the different edge type for change notification.
	PORTS_CHANGE_NOTICE_METHOD	Data type defining the different method of ports pin change notification.
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANNEL	Identifies the available Ports channels.
	PORTS_CN_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_MODULE_ID	Identifies the available Ports modules.
	PORTS_PIN_MODE	Identifies the available pin modes.
	PORTS_PIN_SLEW_RATE	Data type defining the different slew rates for port pins.
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumerations.
	PORTS_REMAP_INPUT_FUNCTION	Data type defining the different remap input function enumerations.
	PORTS_REMAP_INPUT_PIN	Data type defining the different Ports I/O input pins enumerations.
	PORTS_REMAP_OUTPUT_FUNCTION	Data type defining the different remap output function enumerations.
	PORTS_REMAP_OUTPUT_PIN	Data type defining the different Ports I/O output pins enumerations.
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumerations.

Description

This is file help_plib_ports.h.

Power Peripheral Library

This section describes the Power Peripheral Library.

Introduction

This library provides a low-level abstraction of the Power Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Power Controller is a key component of a microcontroller. Power control features allow the user to obtain the balance of power consumption and performance that is best suited to their application.

The features provided in this library discuss specific software commands that are used dynamically to place the device in low-power (i.e., power-saving) modes, which disable power and clocking of selected features. When implementing a system to achieve the lowest possible power consumption while maintain the needed performance, the oscillator configuration and clocking of all peripherals must also be considered. Oscillator configuration and peripheral clocking are addressed in the Oscillator Peripheral Library. In addition, each peripheral may have its own low-power mode settings that are controlled by the library of that peripheral. The user is advised to refer to the libraries of the peripherals used for their particular application.

Key features present on a power controller include (a microcontroller can support one or more of these power features):

- **Power Source Configuration**: Provides the ability to enable and disable a source that may regulate the power consumption in the device
- **Power Status Flags:** Indicate the status of the particular power feature in the device

Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Using the Library

This topic describes the basic architecture of the Power Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_power.h`

The interface to the Power Peripheral Library is defined in the `plib_power.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Power Peripheral Library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific data sheet to determine availability.

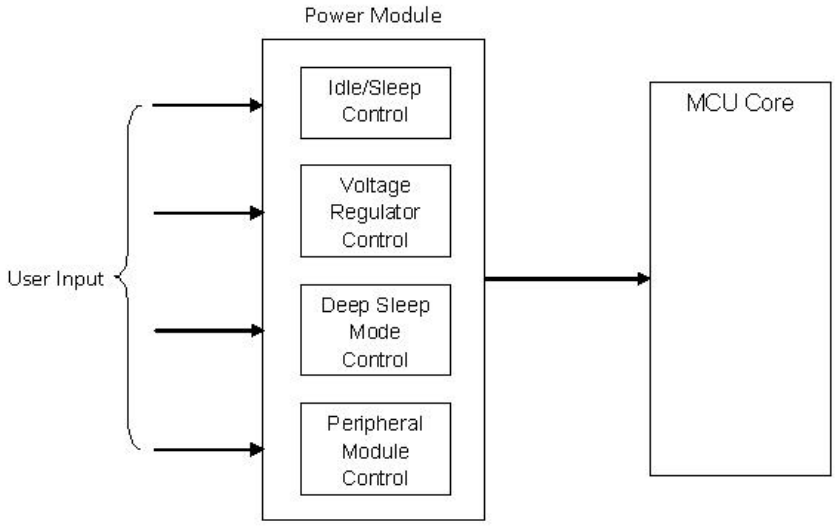
Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Power Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Block Diagram



This section discusses the various power-saving features implemented in hardware. Combinations of these methods can be used to selectively tailor an application's power consumption, while still maintaining critical or timing-sensitive application features.

Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Instruction-based Power-Saving Modes

In addition to full-power operation, otherwise known as Run mode, PIC microcontrollers also provide other instruction-based power-saving modes. By powering down to different modes, different functional areas of the microcontroller allow progressive reductions of operating and idle power consumption. In addition, these modes can be tailored for more power reduction, but at a trade-off of some operating features.

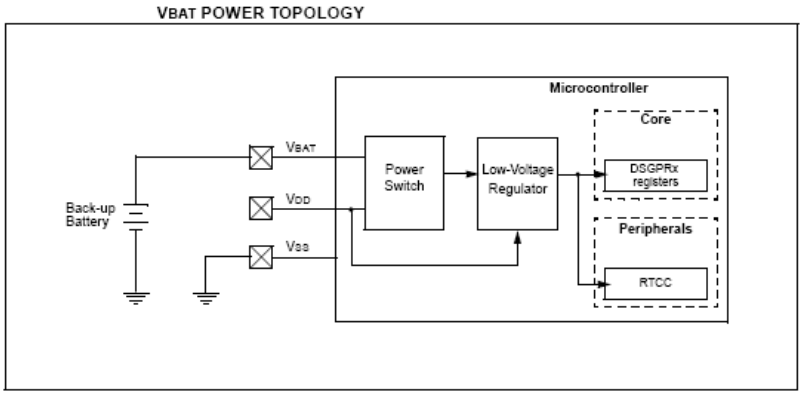
- **Idle Mode:** In Idle mode, the CPU is halted, but the System Clock (SYSCLK) source is still enabled. This allows peripherals to continue to operate when the CPU is halted. Peripherals can be individually configured to halt when entering Idle mode by setting their respective SIDL bit. Latency when exiting Idle mode is very low due to the CPU oscillator source remaining active.
- **Sleep Mode:** The CPU, SYSCLK source and any peripherals that operate on the system clock source are disabled
- **Deep Sleep Mode:** The CPU SYSCLK source, and all the peripherals, with the exception of the Real-Time Clock and Calendar (RTCC) and the Deep Sleep Watchdog Timer (DSWDT), are disabled. This is the lowest power mode for the device. The power to RAM and Flash is also disabled. Deep Sleep mode represents the lowest power mode available without removing power from the application.

The instruction-based power-saving modes are exited as a result of several different hardware triggers. When the device exits one of these three operating modes, it is said to 'wake-up'.

Hardware-based Power-Saving Mode

VBAT mode is a hardware-based power-saving mode that maintains only the most critical operations when a power loss occurs on VDD. The mode does this by powering the systems from a back-up power source connected to the VBAT pin. In this mode, the RTCC can run even when there is no power on VDD.

VBAT mode is entered whenever power is removed from VDD. An on-chip power switch detects the power loss from VDD and connects the VBAT pin to the low-voltage regulator. Entering VBAT mode requires that a power source, distinct from the main VDD power source, be available on the VBAT pin and that VDD be completely removed from the VDD pin(s). Removing VDD can be either unintentional, as in a power failure, or as part of a deliberate power reduction strategy.



Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Selective Peripheral Power Control

Sleep and Idle modes allow users to substantially reduce power consumption by slowing or stopping the CPU clock. Even so, peripheral modules may still remain clocked, and thus, consume some amount of power. There may be cases where the application needs what these modes do not provide: the ability to allocate limited power resources to the CPU while eliminating power consumption from the peripherals. PIC microcontrollers address this requirement by allowing peripheral modules to be selectively enabled or disabled, reducing or eliminating their power consumption.

In addition to the selective peripheral power control implemented as part of this library, devices will also include power saving features that are implemented in their specific libraries. These features include the ability to stop (shutdown) automatically when entering idle mode. On some peripherals the ability to continue operation in Sleep mode is available. Always review the libraries for the peripherals used by your application for specific and unique features that will help minimize power consumption.

Disabling Peripheral Modules

Most of the peripheral modules in the PIC microcontroller family architecture can be selectively disabled, reducing or essentially eliminating their power consumption during all operating modes.

All peripheral modules (except for I/O ports) also have a Control bit that can disable their functionality. These bits, known as the Peripheral Module Disable (PMD) bits, are generically named: 'xxMD'. These bits are located in the PMDx Special Function Registers (SFRs). In contrast to the module enable bits, the xxMD bit must be set to a '1' to disable the module.

While the PMD and module enable bits both disable a peripheral's functionality, the PMD bit completely shuts down the peripheral, effectively powering down all circuits and removing all clock sources. This has the additional effect of making any of the module's control and buffer registers, mapped in the SFR space, unavailable for operations. In other words, when the PMD bit is used to disable a module, the peripheral ceases to exist until the PMD bit is cleared.

The PMD bit is most useful in highly power-sensitive applications, where even tiny savings in power consumption can determine the ability of an application to function. In these cases, the bits can be set before the main body of the application to remove those peripherals that will not be needed.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Power Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring the Power-Saving features and enabling or disabling the particular power feature.
Status Functions	This section provides a function to read the status of a particular power feature.
HLVD Functions	This section provides High/Low-Voltage Detect functions.
Deep Sleep Functions	This section provides Deep Sleep mode functions.
Feature Existence Functions	This section provides functions that can be used to determine available features.

Note: The interface provided is a superset of all power functionality available on the device. Refer to the **"Power-Saving Features"** chapter in the specific device data sheet for availability.

How the Library Works

Provides information on how the library works.

Description

Usage of this library is partitioned into two major sections.

Initialization

The steps that are required to initialize the Power Controller module vary for different microcontrollers. Refer to the specific device data sheet to determine the correct initialization sequence. The following information provides a general overview.

Power Source Setup

- A module can be disabled for the microcontroller to be able to save power. Use the function PLIB_POWER_ModuleDisable to disable the particular module. POWER_MODULE_DISABLE provides a list of possible modules sources.
- On most microcontrollers, the voltage regulator can be controlled during Sleep mode. It can be enabled (turned on) using PLIB_POWER_VoltageRegulatorEnable and disabled (turned off) using PLIB_POWER_VoltageRegulatorDisable.

- On some microcontrollers, Deep Sleep mode is enabled using PLIB_POWER_DeepSleepModeEnable. The Deep Sleep mode is disabled using PLIB_POWER_DeepSleepModeDisable.

Power Status

Status of the particular power feature indicates the previous status of that feature. This feature can be used to determine whether the device was in Sleep or Idle mode before waking up. It also allows software to clear the previous status of the device.

Configuring the Library

The library is configured for the supported Power Controller modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_POWER_DeepSleepModeDisable	Disables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeEnable	Enables Deep Sleep mode.
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Checks to see whether or not the selected peripheral is enabled.
	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.
	PLIB_POWER_DeepSleepGPRsRetentionDisable	Disables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRsRetentionEnable	Enables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepModeIsEnabled	Returns the enable/disable status of Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleDisable	Disables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleEnable	Enables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRelease	Releases I/O pins upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRetain	Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepWakeupEventDisable	Disables wake-up from Deep Sleep mode by the selected event.
	PLIB_POWER_DeepSleepWakeupEventEnable	Enables wake-up from the Deep Sleep mode by the selected event.

b) Status Functions

	Name	Description
	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_HighVoltageOnVDD1V8HasOccurred	Returns the DDR2 High Voltage Detection status of the device.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.
	PLIB_POWER_DeepSleepStatusClear	Clears the Deep Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepEventStatusClear	Clear any events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepEventStatusGet	Returns the events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepModeHasOccurred	Returns the Deep Sleep mode status of the device.

c) HLVD Functions

	Name	Description
	PLIB_POWER_HLVDBandGapVoltageIsStable	Returns the status of Band Gap voltage.
	PLIB_POWER_HLVDDisable	Disables High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDEnable	Enables High/Low-Voltage Detection (HLVD) on VDD.
	PLIB_POWER_HLVDIsEnabled	Returns the enable/disable status of High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDLimitSelect	Selects the HLVD limit.
	PLIB_POWER_HLVDModeSelect	Selects the Voltage Detection mode.
	PLIB_POWER_HLVDStatusGet	Returns the HLVD event status.
	PLIB_POWER_HLVDStopInIdleDisable	Continues HLVD operation when the device enters Idle mode.

	PLIB_POWER_HLVDStopInIdleEnable	Discontinues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleIsEnabled	Returns the Stop in Idle mode status of the HLVD operation.

d) Deep Sleep Functions

	Name	Description
	PLIB_POWER_DeepSleepGPRRead	Reads 32-bit data from the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepGPRWrite	Writes 32-bit data into the Deep Sleep General Purpose Register.

e) Feature Existence Functions

	Name	Description
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the Power module.
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the Power module.
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the Power module.
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepEventStatus	Identifies whether the DeepSleepEventStatus feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPROperation	Identifies whether the DeepSleepGPROperation feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPRsRetentionControl	Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModuleControl	Identifies whether the DeepSleepModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepPortPinsStateControl	Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.
	PLIB_POWER_ExistsHighVoltageOnVDD1V8	Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModeOccurrence	Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepWakeupEventControl	Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDBandGapVoltageStability	Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.
	PLIB_POWER_ExistsHLVDEnableControl	Identifies whether the HLVDEnableControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDLimitSelection	Identifies whether the HLVDLimitSelection feature exists on the Power module.
	PLIB_POWER_ExistsHLVDModeControl	Identifies whether the HLVDModeControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStatus	Identifies whether the HLVDStatus feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStopInIdleControl	Identifies whether the HLVDStopInIdleControl feature exists on the Power module.

f) Data Types and Constants

	Name	Description
	DEEP_SLEEP_EVENT	Lists the events that occurred during Deep Sleep mode.
	DEEP_SLEEP_GPR	Lists the Deep Sleep General Purpose Registers (GPRs).
	DEEP_SLEEP_MODULE	Lists the modules that can be enabled/disabled during Deep Sleep mode.
	DEEP_SLEEP_WAKE_UP_EVENT	Lists the events that can be used to wake the device from Deep Sleep mode.
	POWER_MODULE	List of the modules whose power can be controlled.
	POWER_MODULE_ID	Identifies the supported Power modules.
	HLVD_LIMIT	Lists the voltage limits for the HLVD module.
	HLVD_MODE	Lists the modes for the High/Low Voltage Detection (HLVD) module.

Description

This section describes the Application Programming Interface (API) functions of the Power Peripheral Library.

Refer to each section for a detailed description.

a) Initialization Functions

PLIB_POWER_DeepSleepModeDisable Function

Disables Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModeDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModeDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepModeDisable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeEnable Function

Enables Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModeEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModeEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepModeEnable (POWER_MODULE_ID index)

PLIB_POWER_PeripheralModuleDisable Function

Disable the power supply for the module selected in the source.

File

plib_power.h

C

```
void PLIB_POWER_PeripheralModuleDisable(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

None.

Description

This function completely shuts down the selected peripheral, effectively powering down all circuits and removing all clock sources. This has the additional affect of making any of the module's control and buffer registers, which are mapped in the SFR space, unavailable for operations.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Disabling a peripheral module while it's ON bit is set, may result in undefined behavior. The ON bit for the associated peripheral module must be cleared prior to disable a module via this API.

Preconditions

The PMDLOCK bit of the Configuration register should be cleared using the function PLIB_DEVCON_DeviceRegistersUnlock (this function allows changes in the PMD registers). Refer to the Device Control (DEVCON) Peripheral Library Help (or System Service) and the specific device data sheet for more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PMD registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PMD_REGISTERS);
// Disable ADC module
PLIB_POWER_PeripheralModuleDisable (POWER_ID_0, POWER_MODULE_ADC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be disabled from POWER_MODULE enum

Function

void PLIB_POWER_PeripheralModuleDisable (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_PeripheralModuleEnable Function

Enable the power supply for the module selected in the source.

File

plib_power.h

C

```
void PLIB_POWER_PeripheralModuleEnable(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

None.

Description

This function enables the power supply for the selected module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Preconditions

The PMDLOCK bit of the Configuration register should be cleared using the function PLIB_DEVCON_DeviceRegistersUnlock (this function allows changes in the PMD registers). Refer to the Device Control (DEVCON) Peripheral Library Help (or System Service) and the specific device data sheet for more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PMD registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PMD_REGISTERS);
// Enable ADC module
PLIB_POWER_PeripheralModuleEnable (POWER_ID_0, POWER_MODULE_ADC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from POWER_MODULE enum

Function

void PLIB_POWER_PeripheralModuleEnable (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_PeripheralModuleIsEnabled Function

Checks to see whether or not the selected peripheral is enabled.

File

plib_power.h

C

```
bool PLIB_POWER_PeripheralModuleIsEnabled(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

- true - Power of the selected peripheral is enabled
- false - Power of the selected peripheral is disabled

Description

This function checks to see whether or not the selected peripheral is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_POWER_PeripheralModuleIsEnabled ( POWER_MODULE_0, POWER_MODULE_ADC ))
{
```

```
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from the POWER_MODULE enum

Function

bool PLIB_POWER_PeripheralModuleIsEnabled (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_VoltageRegulatorDisable Function

Disables the voltage regulator during Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_VoltageRegulatorDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_VoltageRegulatorDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_VoltageRegulatorDisable (POWER_MODULE_ID index)

PLIB_POWER_VoltageRegulatorEnable Function

Enable the voltage regulator during Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_VoltageRegulatorEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_VoltageRegulatorEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_VoltageRegulatorEnable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepGPRsRetentionDisable Function

Disables the General Purpose Registers retention.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRsRetentionDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the Deep Sleep General Purpose Registers to retain their values when the device enters Deep Sleep/VBAT mode, which means the power to DEEP_SLEEP_GPR_1 to DEEP_SLEEP_GPR_32 will be dismissed. Power to DEEP_SLEEP_GPR_0 will be retained by default.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPRsRetentionControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepGPRsRetentionDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepGPRsRetentionDisable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepGPRsRetentionEnable Function

Enables the General Purpose Registers retention.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the Deep Sleep General Purpose Registers to retain their values when the device enters Deep Sleep/VBAT mode, which means the power to DEEP_SLEEP_GPR_1 to DEEP_SLEEP_GPR_32 will be maintained. Power to DEEP_SLEEP_GPR_0 will be retained by default.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPRsRetentionControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepGPRsRetentionEnable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeIsEnabled Function

Returns the enable/disable status of Deep Sleep mode.

File

plib_power.h

C

```
bool PLIB_POWER_DeepSleepModeIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - Deep Sleep mode is enabled
- false - Deep Sleep mode is not enabled

Description

This function returns whether or not Deep Sleep mode is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeIsEnabled(POWER_ID_0))
{
    //Enter Deep Sleep by executing Sleep mode.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeepSleepModeIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModuleDisable Function

Disables the module in Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModuleDisable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module);
```

Returns

None.

Description

This function disables the selected module in Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModuleDisable(POWER_ID_0 , DEEP_SLEEP_MODULE_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
module	Possible module from DEEP_SLEEP_MODULE

Function

void PLIB_POWER_DeepSleepModuleDisable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module)

PLIB_POWER_DeepSleepModuleEnable Function

Enables the module in Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModuleEnable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module);
```

Returns

None.

Description

This function enables the selected module in Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModuleEnable(POWER_ID_0 , DEEP_SLEEP_MODULE_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
module	Possible module from DEEP_SLEEP_MODULE

Function

void PLIB_POWER_DeepSleepModuleEnable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module)

PLIB_POWER_DeepSleepPortPinsStateRelease Function

Releases I/O pins upon wake-up from Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepPortPinsStateRelease(POWER_MODULE_ID index);
```

Returns

None.

Description

This function releases the I/O pins upon wake-up from Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepPortPinsStateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepPortPinsStateRelease(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepPortPinsStateRelease (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepPortPinsStateRetain Function

Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepPortPinsStateRetain(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepPortPinsStateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepPortPinsStateRetain(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepPortPinsStateRetain (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepWakeupEventDisable Function

Disables wake-up from Deep Sleep mode by the selected event.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepWakeupEventDisable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event);
```

Returns

None.

Description

This function disables wake-up from Deep Sleep mode by the selected event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepWakeupEventControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepWakeupEventDisable(POWER_ID_0 , DEEP_SLEEP_WAKE_UP_EVENT_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	Possible event from DEEP_SLEEP_WAKE_UP_EVENT

Function

void PLIB_POWER_DeepSleepWakeupEventDisable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event)

PLIB_POWER_DeepSleepWakeupEventEnable Function

Enables wake-up from the Deep Sleep mode by the selected event.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepWakeupEventEnable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event);
```

Returns

None.

Description

This function enables wake-up from the Deep Sleep mode by the selected event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepWakeupEventControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepWakeupEventEnable(POWER_ID_0 , DEEP_SLEEP_WAKE_UP_EVENT_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	Possible event from DEEP_SLEEP_WAKE_UP_EVENT

Function

void PLIB_POWER_DeepSleepWakeupEventEnable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event)

b) Status Functions

PLIB_POWER_ClearIdleStatus Function

Clears the Idle mode status of the device.

File

plib_power.h

C

```
void PLIB_POWER_ClearIdleStatus(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Idle status bit of the device if it was previously in Idle mode. However, it does not mean that it wakes the device from Idle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{
    PLIB_POWER_ClearIdleStatus (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_ClearIdleStatus (POWER_MODULE_ID index)

PLIB_POWER_ClearSleepStatus Function

Clear the Sleep mode status bit of the device.

File

plib_power.h

C

```
void PLIB_POWER_ClearSleepStatus(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Sleep status bit of the device if it was previously in Sleep mode. However, it does not mean that it wakes the device from sleep.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsSleepStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{
    PLIB_POWER_ClearSleepStatus (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_ClearSleepStatus (POWER_MODULE_ID index)

PLIB_POWER_DeviceWasInIdleMode Function

Returns the Idle mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeviceWasInIdleMode(POWER_MODULE_ID index);
```

Returns

- true - The device was in Idle mode before waking up
- false - The device was not in Idle mode

Description

This function returns the Idle mode status of the device. It indicates whether or not the device was in Idle mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeviceWasInIdleMode (POWER_MODULE_ID index)

PLIB_POWER_DeviceWasInSleepMode Function

Returns the Sleep mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeviceWasInSleepMode(POWER_MODULE_ID index);
```

Returns

- true - The device was in Sleep mode before waking up
- false - The device was not in Sleep mode

Description

This function returns the Sleep mode status of the device. It indicates whether or not the device was in Sleep mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsSleepStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeviceWasInSleepMode (POWER_MODULE_ID index)

PLIB_POWER_HighVoltageOnVDD1V8HasOccurred Function

Returns the DDR2 High Voltage Detection status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_HighVoltageOnVDD1V8HasOccurred(POWER_MODULE_ID index);
```

Returns

- true - A high-voltage condition on the VDD1V8 voltage has occurred
- false - A high-voltage condition on the VDD1V8 voltage has not occurred

Description

This function returns the DDR2 High Voltage Detection status of the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHighVoltageOnVDD1V8 in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HighVoltageOnVDD1V8HasOccurred(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HighVoltageOnVDD1V8HasOccurred (POWER_MODULE_ID index)

PLIB_POWER_VoltageRegulatorIsEnabled Function

Provides the status of the voltage regulator during Sleep mode.

File

plib_power.h

C

```
bool PLIB_POWER_VoltageRegulatorIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - The voltage regulator will be enabled in Sleep mode
- false - The voltage regulator will be disabled in Sleep mode

Description

This function provides the status of the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_POWER_VoltageRegulatorIsEnabled(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_VoltageRegulatorIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepStatusClear Function

Clears the Deep Sleep mode status bit of the device.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepStatusClear(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Deep Sleep status bit of the device if it was previously in Deep Sleep mode. However, it does not mean that it wakes the device from Deep Sleep.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModeOccurrence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeHasOccurred(POWER_ID_0))
{
    PLIB_POWER_DeepSleepStatusClear (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepStatusClear (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepEventStatusClear Function

Clear any events that occurred during Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepEventStatusClear(POWER_MODULE_ID index, DEEP_SLEEP_EVENT event);
```

Returns

None.

Description

This function accept the events to clear that occurred during Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepEventStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if((PLIB_POWER_DeepSleepEventStatusGet(POWER_ID_0)& DEEP_SLEEP_EVENT_BOR) != 0)
{
    PLIB_POWER_DeepSleepEventStatusClear(POWER_ID_0, DEEP_SLEEP_EVENT_BOR);
    //BOR event occurred in Deep Sleep Mode
    //Therefore, take appropriate action

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	The event that occurred during the Deep Sleep

Function

void PLIB_POWER_DeepSleepEventStatusClear(POWER_MODULE_ID index , DEEP_SLEEP_EVENT event)

PLIB_POWER_DeepSleepEventStatusGet Function

Returns the events that occurred during Deep Sleep mode.

File

plib_power.h

C

```
DEEP_SLEEP_EVENT PLIB_POWER_DeepSleepEventStatusGet(POWER_MODULE_ID index);
```

Returns

DEEP_SLEEP_EVENT - The Event that has occurred during Deep Sleep

Description

This function returns the events that occurred during Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepEventStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if((PLIB_POWER_DeepSleepEventStatusGet(POWER_ID_0)&
(DEEP_SLEEP_EVENT_RTCC_ALARM|DEEP_SLEEP_EVENT_BOR) != 0)
{
    //BOR or RTCC alarm event occurred in Deep Sleep Mode
    //Therefore, take appropriate action
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

DEEP_SLEEP_EVENT PLIB_POWER_DeepSleepEventStatusGet(POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeHasOccurred Function

Returns the Deep Sleep mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeepSleepModeHasOccurred(POWER_MODULE_ID index);
```

Returns

- true - The device was in Deep Sleep mode before waking up
- false - The device was not in Deep Sleep mode

Description

This function returns whether or not the device was in Deep Sleep mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModeOccurrence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeHasOccurred(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeepSleepModeHasOccurred (POWER_MODULE_ID index)

c) HLVD Functions

PLIB_POWER_HLVDBandGapVoltageIsStable Function

Returns the status of Band Gap voltage.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDBandGapVoltageIsStable(POWER_MODULE_ID index);
```

Returns

- true - Band Gap Voltage is stable
- false - Band Gap Voltage is unstable

Description

This function returns whether the Band Gap Voltage is stable or not.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDBandGapVoltageStability in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool BGStatus;
BGStatus = PLIB_POWER_HLVDBandGapVoltageIsStable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDBandGapVoltageIsStable (POWER_MODULE_ID index)

PLIB_POWER_HLVDDisable Function

Disables High/Low-Voltage Detection on VDD.

File

plib_power.h

C

```
void PLIB_POWER_HLVDDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the HLVD module. The HLVD module is used to detect high/low-voltage conditions on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDDisable (POWER_MODULE_ID index)

PLIB_POWER_HLVDEnable Function

Enables High/Low-Voltage Detection (HLVD) on VDD.

File

plib_power.h

C

```
void PLIB_POWER_HLVDEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the High/Low-Voltage Detection (HLVD) module. The HLVD module is used to detect high/low-voltage conditions on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

Select Low-Voltage or High-Voltage Detection mode by calling PLIB_POWER_HLVDModeSelect.

Example

```
PLIB_POWER_HLVDEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDEnable (POWER_MODULE_ID index)

PLIB_POWER_HLVDIsEnabled Function

Returns the enable/disable status of High/Low-Voltage Detection on VDD.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - HLVD on VDD is enabled
- false - HLVD on VDD is not enabled

Description

This function returns whether or not High/Low-Voltage Detection on VDD is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HLVDIsEnabled(POWER_ID_0))
{
    // HLVD is enabled.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_HLVDLimitSelect Function

Selects the HLVD limit.

File

plib_power.h

C

```
void PLIB_POWER_HLVDLimitSelect(POWER_MODULE_ID index, HLVD_LIMIT limit);
```

Returns

None.

Description

This function selects voltage limit for HLVD on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDLimitSelection in your application to determine whether this feature is available.

Preconditions

The HLVD module should be disabled by calling PLIB_POWER_HLVDDisable.

Example

```
PLIB_POWER_HLVDLimitSelect( POWER_ID_0, HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
limit	Voltage limit for HLVD on VDD

Function

void PLIB_POWER_HLVDLimitSelect (POWER_MODULE_ID index, HLVD_LIMIT limit)

PLIB_POWER_HLVDModeSelect Function

Selects the Voltage Detection mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDModeSelect(POWER_MODULE_ID index, HLVD_MODE mode);
```

Returns

None.

Description

This function selects either the High-Voltage or Low-Voltage Detection mode.

In High-Voltage Detection (HVD) mode, an event occurs when the voltage equals or exceeds the voltage limit.

In Low-Voltage Detection (LVD) mode, an event occurs when the voltage equals or falls below the voltage limit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDModeControl in your application to determine whether this feature is available.

Preconditions

The HLVD module should be disabled by calling PLIB_POWER_HLVDDisable.

Example

```
PLIB_POWER_HLVDModeSelect( POWER_ID_0, HLVD_MODE_HIGH_VOLTAGE_DETECTION );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Voltage Detection mode, either HVD or LVD

Function

void PLIB_POWER_HLVDModeSelect (POWER_MODULE_ID index, HLVD_MODE mode)

PLIB_POWER_HLVDStatusGet Function

Returns the HLVD event status.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDStatusGet(POWER_MODULE_ID index);
```

Returns

- true - HLVD event interrupt is active
- false - HLVD event interrupt is not active

Description

This function returns the HLVD event interrupt status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HLVDStatusGet(POWER_ID_0))
{
    //HLVD event has occurred.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDStatusGet(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleDisable Function

Continues HLVD operation when the device enters Idle mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDStopInIdleDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function continues HLVD operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDStopInIdleDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDStopInIdleDisable(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleEnable Function

Discontinues HLVD operation when the device enters Idle mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDStopInIdleEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function discontinues HLVD operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDStopInIdleEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDStopInIdleEnable(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleIsEnabled Function

Returns the Stop in Idle mode status of the HLVD operation.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - HLVD discontinues operation when the device enters Idle mode
- false - HLVD continues operation when the device enters Idle mode

Description

This function returns whether HLVD continues or discontinues operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool sidl_status;
sidl_status = PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_MODULE_ID index)

d) Deep Sleep Functions

PLIB_POWER_DeepSleepGPRRead Function

Reads 32-bit data from the Deep Sleep General Purpose Register.

File

plib_power.h

C

```
uint32_t PLIB_POWER_DeepSleepGPRRead(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr);
```

Returns

32-bit data from the selected Deep Sleep General Purpose Register.

Description

This function reads 32-bit value from the Deep Sleep General Purpose Register. User can store content in these registers (any application critical content) before entering the Deep Sleep/VBAT modes and read them upon exit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPROperation in your application to determine whether this feature is available.

Preconditions

The content of the DEEP_SLEEP_GPR_0 register is retained even in the Deep Sleep and VBAT modes, but the DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 registers are disabled by default in Deep Sleep and VBAT modes. They can be enabled with PLIB_POWER_DeepSleepGPRsRetentionEnable.

Example

```
uint32_t data;
//Enable power to DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 in Deep Sleep
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);

//Write 32-bit data into the DEEP_SLEEP_GPR_1
PLIB_POWER_DeepSleepGPRWrite(POWER_ID_0, DEEP_SLEEP_GPR_1, 0x1234);

//Enter the Deep Sleep mode and Exit
//Now read the data from DEEP_SLEEP_GPR_1
data = PLIB_POWER_DeepSleepGPRRead(POWER_ID_0, DEEP_SLEEP_GPR_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
gpr	The available Deep Sleep General Purpose Register

Function

uint32_t PLIB_POWER_DeepSleepGPRRead(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr)

PLIB_POWER_DeepSleepGPRWrite Function

Writes 32-bit data into the Deep Sleep General Purpose Register.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRWrite(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr, uint32_t data);
```

Returns

None.

Description

This function accepts a 32-bit value to write into the Deep Sleep General Purpose Register. User can store content in these registers (any application critical content) before entering the Deep Sleep/VBAT modes and read them upon exit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPROperation in your application to determine whether this feature is available.

Preconditions

The content of the DEEP_SLEEP_GPR_0 register is retained even in the Deep Sleep and VBAT modes, but the DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 registers are disabled by default in Deep Sleep and VBAT modes. They can be enabled with PLIB_POWER_DeepSleepGPRsRetentionEnable.

Example

```
uint32_t data;
//Enable power to DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 in Deep Sleep
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);

//Write 32-bit data into the DEEP_SLEEP_GPR_1
PLIB_POWER_DeepSleepGPRWrite(POWER_ID_0, DEEP_SLEEP_GPR_1, 0x1234);

//Enter the Deep Sleep mode and Exit
//Now read the data from DEEP_SLEEP_GPR_1
data = PLIB_POWER_DeepSleepGPRRead(POWER_ID_0, DEEP_SLEEP_GPR_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
gpr	The available Deep Sleep General Purpose Register
data	32-bit data to write into the Deep Sleep General Purpose Register

Function

void PLIB_POWER_DeepSleepGPRWrite(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr, uint32_t data)

e) Feature Existence Functions

PLIB_POWER_ExistsDeepSleepMode Function

Identifies whether the DeepSleepModeControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepMode(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModeControl feature is supported on the device
- false - The DeepSleepModeControl feature is not supported on the device

Description

This function identifies whether the DeepSleepModeControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepModeEnable
- PLIB_POWER_DeepSleepModeDisable
- PLIB_POWER_DeepSleepModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepMode(POWER_MODULE_ID index)

PLIB_POWER_ExistsIdleStatus Function

Identifies whether the IdleStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsIdleStatus(POWER_MODULE_ID index);
```

Returns

- true - The IdleStatus feature is supported on the device
- false - The IdleStatus feature is not supported on the device

Description

This function identifies whether the IdleStatus feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeviceWasInIdleMode
- PLIB_POWER_ClearIdleStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsIdleStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsPeripheralModuleControl Function

Identifies whether the PeripheralModuleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsPeripheralModuleControl(POWER_MODULE_ID index);
```

Returns

- true - The PeripheralModuleControl feature is supported on the device
- false - The PeripheralModuleControl feature is not supported on the device

Description

This function identifies whether the PeripheralModuleControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_PeripheralModuleDisable
- PLIB_POWER_PeripheralModuleEnable
- PLIB_POWER_PeripheralModuleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsPeripheralModuleControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsSleepStatus Function

Identifies whether the SleepStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsSleepStatus(POWER_MODULE_ID index);
```

Returns

- true - The SleepStatus feature is supported on the device
- false - The SleepStatus feature is not supported on the device

Description

This function identifies whether the SleepStatus feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeviceWasInSleepMode
- PLIB_POWER_ClearSleepStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsSleepStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsVoltageRegulatorControl Function

Identifies whether the VoltageRegulatorControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsVoltageRegulatorControl(POWER_MODULE_ID index);
```

Returns

- true - The VoltageRegulatorControl feature is supported on the device
- false - The VoltageRegulatorControl feature is not supported on the device

Description

This function identifies whether the VoltageRegulatorControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_VoltageRegulatorEnable
- PLIB_POWER_VoltageRegulatorDisable
- PLIB_POWER_VoltageRegulatorIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsVoltageRegulatorControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepEventStatus Function

Identifies whether the DeepSleepEventStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepEventStatus(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepEventStatus feature is supported on the device
- false - The DeepSleepEventStatus feature is not supported on the device

Description

This function identifies whether the DeepSleepEventStatus feature is available on the Power module. When this function returns true, these

functions are supported on the device:

- PLIB_POWER_DeepSleepEventStatusGet
- PLIB_POWER_DeepSleepEventStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepEventStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepGPROperation Function

Identifies whether the DeepSleepGPROperation feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepGPROperation(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepGPROperation feature is supported on the device
- false - The DeepSleepGPROperation feature is not supported on the device

Description

This function identifies whether the DeepSleepGPROperation feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepGPRWrite
- PLIB_POWER_DeepSleepGPRRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepGPROperation(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepGPRsRetentionControl Function

Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepGPRsRetentionControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepGPRsRetentionControl feature is supported on the device
- false - The DeepSleepGPRsRetentionControl feature is not supported on the device

Description

This function identifies whether the DeepSleepGPRsRetentionControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepGPRsRetentionEnable
- PLIB_POWER_DeepSleepGPRsRetentionDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepGPRsRetentionControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepModuleControl Function

Identifies whether the DeepSleepModuleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepModuleControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModuleControl feature is supported on the device
- false - The DeepSleepModuleControl feature is not supported on the device

Description

This function identifies whether the DeepSleepModuleControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepModuleEnable
- PLIB_POWER_DeepSleepModuleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepModuleControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepPortPinsStateControl Function

Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepPortPinsStateControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepPortPinsStateControl feature is supported on the device
- false - The DeepSleepPortPinsStateControl feature is not supported on the device

Description

This function identifies whether the DeepSleepPortPinsStateControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepPortPinsStateRetain
- PLIB_POWER_DeepSleepPortPinsStateRelease

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepPortPinsStateControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHighVoltageOnVDD1V8 Function

Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHighVoltageOnVDD1V8(POWER_MODULE_ID index);
```

Returns

- true - The HighVoltageOnVDD1V8 feature is supported on the device
- false - The HighVoltageOnVDD1V8 feature is not supported on the device

Description

This function identifies whether the HighVoltageOnVDD1V8 feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HighVoltageOnVDD1V8HasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHighVoltageOnVDD1V8(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepModeOccurrence Function

Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepModeOccurrence(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModeOccurrence feature is supported on the device
- false - The DeepSleepModeOccurrence feature is not supported on the device

Description

This function identifies whether the DeepSleepModeOccurrence feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_DeepSleepModeHasOccurred
- PLIB_POWER_DeepSleepStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepModeOccurrence(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepWakeupEventControl Function

Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepWakeupEventControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepWakeupEventControl feature is supported on the device
- false - The DeepSleepWakeupEventControl feature is not supported on the device

Description

This function identifies whether the DeepSleepWakeupEventControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepWakeupEventEnable
- PLIB_POWER_DeepSleepWakeupEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepWakeupEventControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDBandGapVoltageStability Function

Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDBandGapVoltageStability(POWER_MODULE_ID index);
```

Returns

- true - The HLVDBandGapVoltageStability feature is supported on the device
- false - The HLVDBandGapVoltageStability feature is not supported on the device

Description

This function identifies whether the HLVDBandGapVoltageStability feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDBandGapVoltageIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDBandGapVoltageStability(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDEnableControl Function

Identifies whether the HLVDEnableControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDEnableControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDEnableControl feature is supported on the device
- false - The HLVDEnableControl feature is not supported on the device

Description

This function identifies whether the HLVDEnableControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDEnable
- PLIB_POWER_HLVDDisable
- PLIB_POWER_HLVDIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDEnableControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDLimitSelection Function

Identifies whether the HLVDLimitSelection feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDLimitSelection(POWER_MODULE_ID index);
```

Returns

- true - The HLVDLimitSelection feature is supported on the device
- false - The HLVDLimitSelection feature is not supported on the device

Description

This function identifies whether the HLVDLimitSelection feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDLimitSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDLimitSelection(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDModeControl Function

Identifies whether the HLVDModeControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDModeControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDModeControl feature is supported on the device
- false - The HLVDModeControl feature is not supported on the device

Description

This function identifies whether the HLVDModeControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDModeControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDStatus Function

Identifies whether the HLVDStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDStatus(POWER_MODULE_ID index);
```

Returns

- true - The HLVDStatus feature is supported on the device
- false - The HLVDStatus feature is not supported on the device

Description

This function identifies whether the HLVDStatus feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDStopInIdleControl Function

Identifies whether the HLVDStopInIdleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDStopInIdleControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDStopInIdleControl feature is supported on the device
- false - The HLVDStopInIdleControl feature is not supported on the device

Description

This function identifies whether the HLVDStopInIdleControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDStopInIdleEnable
- PLIB_POWER_HLVDStopInIdleDisable
- PLIB_POWER_HLVDStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDStopInIdleControl(POWER_MODULE_ID index)

f) Data Types and Constants

DEEP_SLEEP_EVENT Enumeration

Lists the events that occurred during Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_EVENT_BOR,
  DEEP_SLEEP_EVENT_RTCC_ALARM,
  DEEP_SLEEP_EVENT_EXTERNAL_INTERRUPT,
  DEEP_SLEEP_EVENT_FAULT_DETECTION,
  DEEP_SLEEP_EVENT_WDT_TIMEOUT,
  DEEP_SLEEP_EVENT_MCLR
} DEEP_SLEEP_EVENT;
```

Members

Members	Description
DEEP_SLEEP_EVENT_BOR	BOR event has occurred during Deep Sleep
DEEP_SLEEP_EVENT_RTCC_ALARM	RTCC alarm has occurred during Deep Sleep
DEEP_SLEEP_EVENT_EXTERNAL_INTERRUPT	Interrupt-On-Change on pin INT0 has occurred during Deep Sleep
DEEP_SLEEP_EVENT_FAULT_DETECTION	Fault has occurred during Deep Sleep and some Deep Sleep configuration settings may have been corrupted
DEEP_SLEEP_EVENT_WDT_TIMEOUT	DSWDT time-out has occurred during Deep Sleep
DEEP_SLEEP_EVENT_MCLR	Master Clear (MCLR) has occurred during Deep Sleep

Description

Deep Sleep events

This enumeration lists the events that occurred during Deep Sleep mode.

Remarks

Not all events exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_GPR Enumeration

Lists the Deep Sleep General Purpose Registers (GPRs).

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_GPR_0,
  DEEP_SLEEP_GPR_1,
  DEEP_SLEEP_GPR_2,
  DEEP_SLEEP_GPR_3,
  DEEP_SLEEP_GPR_4,
  DEEP_SLEEP_GPR_5,
  DEEP_SLEEP_GPR_6,
  DEEP_SLEEP_GPR_7,
  DEEP_SLEEP_GPR_8,
  DEEP_SLEEP_GPR_9,
  DEEP_SLEEP_GPR_10,
  DEEP_SLEEP_GPR_11,
  DEEP_SLEEP_GPR_12,
  DEEP_SLEEP_GPR_13,
  DEEP_SLEEP_GPR_14,
  DEEP_SLEEP_GPR_15,
  DEEP_SLEEP_GPR_16,
  DEEP_SLEEP_GPR_17,
  DEEP_SLEEP_GPR_18,
  DEEP_SLEEP_GPR_19,
  DEEP_SLEEP_GPR_20,
  DEEP_SLEEP_GPR_21,
  DEEP_SLEEP_GPR_22,
  DEEP_SLEEP_GPR_23,
  DEEP_SLEEP_GPR_24,
  DEEP_SLEEP_GPR_25,
  DEEP_SLEEP_GPR_26,
  DEEP_SLEEP_GPR_27,
  DEEP_SLEEP_GPR_28,
  DEEP_SLEEP_GPR_29,
  DEEP_SLEEP_GPR_30,
  DEEP_SLEEP_GPR_31,
  DEEP_SLEEP_GPR_32
} DEEP_SLEEP_GPR;
```

Members

Members	Description
DEEP_SLEEP_GPR_0	Deep Sleep General Purpose Register 0. This will be active in Deep Sleep by default.
DEEP_SLEEP_GPR_1	Deep Sleep General Purpose Register 1.
DEEP_SLEEP_GPR_2	Deep Sleep General Purpose Register 2.
DEEP_SLEEP_GPR_3	Deep Sleep General Purpose Register 3.
DEEP_SLEEP_GPR_4	Deep Sleep General Purpose Register 4.
DEEP_SLEEP_GPR_5	Deep Sleep General Purpose Register 5.
DEEP_SLEEP_GPR_6	Deep Sleep General Purpose Register 6.
DEEP_SLEEP_GPR_7	Deep Sleep General Purpose Register 7.
DEEP_SLEEP_GPR_8	Deep Sleep General Purpose Register 8.
DEEP_SLEEP_GPR_9	Deep Sleep General Purpose Register 9.
DEEP_SLEEP_GPR_10	Deep Sleep General Purpose Register 10.
DEEP_SLEEP_GPR_11	Deep Sleep General Purpose Register 11.
DEEP_SLEEP_GPR_12	Deep Sleep General Purpose Register 12.
DEEP_SLEEP_GPR_13	Deep Sleep General Purpose Register 13.
DEEP_SLEEP_GPR_14	Deep Sleep General Purpose Register 14.
DEEP_SLEEP_GPR_15	Deep Sleep General Purpose Register 15.
DEEP_SLEEP_GPR_16	Deep Sleep General Purpose Register 16.
DEEP_SLEEP_GPR_17	Deep Sleep General Purpose Register 17.

DEEP_SLEEP_GPR_18	Deep Sleep General Purpose Register 18.
DEEP_SLEEP_GPR_19	Deep Sleep General Purpose Register 19.
DEEP_SLEEP_GPR_20	Deep Sleep General Purpose Register 20.
DEEP_SLEEP_GPR_21	Deep Sleep General Purpose Register 21.
DEEP_SLEEP_GPR_22	Deep Sleep General Purpose Register 22.
DEEP_SLEEP_GPR_23	Deep Sleep General Purpose Register 23.
DEEP_SLEEP_GPR_24	Deep Sleep General Purpose Register 24.
DEEP_SLEEP_GPR_25	Deep Sleep General Purpose Register 25.
DEEP_SLEEP_GPR_26	Deep Sleep General Purpose Register 26.
DEEP_SLEEP_GPR_27	Deep Sleep General Purpose Register 27.
DEEP_SLEEP_GPR_28	Deep Sleep General Purpose Register 28.
DEEP_SLEEP_GPR_29	Deep Sleep General Purpose Register 29.
DEEP_SLEEP_GPR_30	Deep Sleep General Purpose Register 30.
DEEP_SLEEP_GPR_31	Deep Sleep General Purpose Register 31.
DEEP_SLEEP_GPR_32	Deep Sleep General Purpose Register 32.

Description

Deep Sleep General Purpose Register set.

This enumeration lists the Deep Sleep General Purpose Registers. These registers can be used to save some application critical content while entering into Deep Sleep mode and can be read upon exit.

Remarks

Not all GPRs exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_MODULE Enumeration

Lists the modules that can be enabled/disabled during Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_MODULE_RTCC
} DEEP_SLEEP_MODULE;
```

Members

Members	Description
DEEP_SLEEP_MODULE_RTCC	RTCC module

Description

Deep Sleep Module

This enumeration lists the modules that can be enabled/disabled during Deep Sleep mode.

Remarks

Not all modules exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_WAKE_UP_EVENT Enumeration

Lists the events that can be used to wake the device from Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_WAKE_UP_EVENT_RTCC,
  DEEP_SLEEP_WAKE_UP_EVENT_EXTERNAL_INTERRUPT
} DEEP_SLEEP_WAKE_UP_EVENT;
```

Members

Members	Description
DEEP_SLEEP_WAKE_UP_EVENT_RTCC	Deep Sleep wake-up by RTCC alarm event
DEEP_SLEEP_WAKE_UP_EVENT_EXTERNAL_INTERRUPT	Deep Sleep wake-up by Interrupt-On-Change (IOC) on INT0 pin

Description

Deep Sleep Wake-up events

This enumeration lists the events that can be used to wake the device from Deep Sleep mode.

Remarks

Not all events exist on all devices. Please refer to the specific device data sheet for availability.

POWER_MODULE Enumeration

List of the modules whose power can be controlled.

File

help_plib_power.h

C

```
typedef enum {
  POWER_MODULE_ADC1,
  POWER_MODULE_CTMU,
  POWER_MODULE_CVR,
  POWER_MODULE_HLVD,
  POWER_MODULE_HVD1V8,
  POWER_MODULE_CMP1,
  POWER_MODULE_CMP2,
  POWER_MODULE_CMP3,
  POWER_MODULE_IC1,
  POWER_MODULE_IC2,
  POWER_MODULE_IC3,
  POWER_MODULE_IC4,
  POWER_MODULE_IC5,
  POWER_MODULE_IC6,
  POWER_MODULE_IC7,
  POWER_MODULE_IC8,
  POWER_MODULE_IC9,
  POWER_MODULE_OC1,
  POWER_MODULE_OC2,
  POWER_MODULE_OC3,
  POWER_MODULE_OC4,
  POWER_MODULE_OC5,
  POWER_MODULE_OC6,
  POWER_MODULE_OC7,
  POWER_MODULE_OC8,
  POWER_MODULE_OC9,
  POWER_MODULE_TMR1,
  POWER_MODULE_TMR2,
  POWER_MODULE_TMR3,
  POWER_MODULE_TMR4,
  POWER_MODULE_TMR5,
  POWER_MODULE_TMR6,
  POWER_MODULE_TMR7,
  POWER_MODULE_TMR8,
  POWER_MODULE_TMR9,
  POWER_MODULE_UART1,
  POWER_MODULE_UART2,
  POWER_MODULE_UART3,
  POWER_MODULE_UART4,
  POWER_MODULE_UART5,
  POWER_MODULE_UART6,
  POWER_MODULE_SPI1,
  POWER_MODULE_SPI2,
  POWER_MODULE_SPI3,
  POWER_MODULE_SPI4,
  POWER_MODULE_SPI5,
  POWER_MODULE_SPI6,
  POWER_MODULE_I2C1,
```

```
    POWER_MODULE_I2C2,
    POWER_MODULE_I2C3,
    POWER_MODULE_I2C4,
    POWER_MODULE_I2C5,
    POWER_MODULE_USB,
    POWER_MODULE_CAN1,
    POWER_MODULE_CAN2,
    POWER_MODULE_RTCC,
    POWER_MODULE_REF_CLK_OUTPUT,
    POWER_MODULE_REF_CLK_OUTPUT1,
    POWER_MODULE_REF_CLK_OUTPUT2,
    POWER_MODULE_REF_CLK_OUTPUT3,
    POWER_MODULE_REF_CLK_OUTPUT4,
    POWER_MODULE_REF_CLK_OUTPUT5,
    POWER_MODULE_PMP,
    POWER_MODULE_EBI,
    POWER_MODULE_GPU,
    POWER_MODULE_GLCD,
    POWER_MODULE_SDHC,
    POWER_MODULE_SQI,
    POWER_MODULE_ETHERNET,
    POWER_MODULE_DMA,
    POWER_MODULE_RANDOM_NUM_GENERATOR,
    POWER_MODULE_DDR2,
    POWER_MODULE_CRYPTO
} POWER_MODULE;
```

Members

Members	Description
POWER_MODULE_ADC1	ADC module
POWER_MODULE_CTMU	Charge Time Measurement Unit module
POWER_MODULE_CVR	Comparator Voltage Reference module
POWER_MODULE_HLVD	Low-Voltage Detection module
POWER_MODULE_HVD1V8	High-Voltage Detection module
POWER_MODULE_CMP1	Comparator module 1
POWER_MODULE_CMP2	Comparator module 2
POWER_MODULE_CMP3	Comparator module 3
POWER_MODULE_IC1	Input Capture 1 module
POWER_MODULE_IC2	Input Capture 2 module
POWER_MODULE_IC3	Input Capture 3 module
POWER_MODULE_IC4	Input Capture 4 module
POWER_MODULE_IC5	Input Capture 5 module
POWER_MODULE_IC6	Input Capture 6 module
POWER_MODULE_IC7	Input Capture 7 module
POWER_MODULE_IC8	Input Capture 8 module
POWER_MODULE_IC9	Input Capture 9 module
POWER_MODULE_OC1	Output Compare 1 module
POWER_MODULE_OC2	Output Compare 2 module
POWER_MODULE_OC3	Output Compare 3 module
POWER_MODULE_OC4	Output Compare 4 module
POWER_MODULE_OC5	Output Compare 5 module
POWER_MODULE_OC6	Output Compare 6 module
POWER_MODULE_OC7	Output Compare 7 module
POWER_MODULE_OC8	Output Compare 8 module
POWER_MODULE_OC9	Output Compare 9 module
POWER_MODULE_TMR1	Timer1 module
POWER_MODULE_TMR2	Timer2 module
POWER_MODULE_TMR3	Timer3 module
POWER_MODULE_TMR4	Timer4 module
POWER_MODULE_TMR5	Timer5 module
POWER_MODULE_TMR6	Timer6 module
POWER_MODULE_TMR7	Timer7 module

POWER_MODULE_TMR8	Timer8 module
POWER_MODULE_TMR9	Timer9 module
POWER_MODULE_UART1	UART1 module
POWER_MODULE_UART2	UART2 module
POWER_MODULE_UART3	UART3 module
POWER_MODULE_UART4	UART4 module
POWER_MODULE_UART5	UART5 module
POWER_MODULE_UART6	UART6 module
POWER_MODULE_SPI1	SPI1 module
POWER_MODULE_SPI2	SPI2 module
POWER_MODULE_SPI3	SPI3 module
POWER_MODULE_SPI4	SPI4 module
POWER_MODULE_SPI5	SPI5 module
POWER_MODULE_SPI6	SPI6 module
POWER_MODULE_I2C1	I2C1 module
POWER_MODULE_I2C2	I2C2 module
POWER_MODULE_I2C3	I2C3 module
POWER_MODULE_I2C4	I2C4 module
POWER_MODULE_I2C5	I2C5 module
POWER_MODULE_USB	USB module
POWER_MODULE_CAN1	CAN1 module
POWER_MODULE_CAN2	CAN2 module
POWER_MODULE_RTCC	Real-Time Clock and Calender module
POWER_MODULE_REF_CLK_OUTPUT	Reference Clock Output module
POWER_MODULE_REF_CLK_OUTPUT1	Reference Clock Output module 1
POWER_MODULE_REF_CLK_OUTPUT2	Reference Clock Output module 2
POWER_MODULE_REF_CLK_OUTPUT3	Reference Clock Output module 3
POWER_MODULE_REF_CLK_OUTPUT4	Reference Clock Output module 4
POWER_MODULE_REF_CLK_OUTPUT5	Reference Clock Output module 5
POWER_MODULE_PMP	Parallel Master Port module
POWER_MODULE_EBI	External Bus Interface module
POWER_MODULE_GPU	Graphics Processing Unit module
POWER_MODULE_GLCD	Graphics LCD module
POWER_MODULE_SDHC	Secure Digital Host Controller module
POWER_MODULE_SQI	Serial Quad Interface module
POWER_MODULE_ETHERNET	Ethernet module
POWER_MODULE_DMA	Data Memory Access module
POWER_MODULE_RANDOM_NUM_GENERATOR	Random Number Generator module
POWER_MODULE_DDR2	DDR2 module
POWER_MODULE_CRYPTO	Cryptographic module

Description

POWER module enumeration

This enumeration lists the modules whose power can be controlled by the Peripheral Module Disable (PMD) Registers.

Remarks

Not all modules exist on all devices. Please refer to the specific device data sheet for availability.

POWER_MODULE_ID Enumeration

Identifies the supported Power modules.

File

help_plib_power.h

C

```
typedef enum {
```

```
  POWER_ID_0,
  POWER_NUMBER_OF_MODULES
} POWER_MODULE_ID;
```

Members

Members	Description
POWER_ID_0	POWER Module ID
POWER_NUMBER_OF_MODULES	Number of available POWER modules.

Description

Power Module ID

This enumeration identifies the Power modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller.

Remarks

The caller should not rely on the specific numbers assigned to any of these values, as they may change from one processor to the next.

Not all modules are available on all microcontrollers. Refer to the "Power-Saving Features" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

HLVD_LIMIT Enumeration

Lists the voltage limits for the HLVD module.

File

help_plib_power.h

C

```
typedef enum {
  HLVD_LIMIT_TRIP_POINT_4,
  HLVD_LIMIT_TRIP_POINT_5,
  HLVD_LIMIT_TRIP_POINT_6,
  HLVD_LIMIT_TRIP_POINT_7,
  HLVD_LIMIT_TRIP_POINT_8,
  HLVD_LIMIT_TRIP_POINT_9,
  HLVD_LIMIT_TRIP_POINT_10,
  HLVD_LIMIT_TRIP_POINT_11,
  HLVD_LIMIT_TRIP_POINT_12,
  HLVD_LIMIT_TRIP_POINT_13,
  HLVD_LIMIT_TRIP_POINT_14,
  HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN
} HLVD_LIMIT;
```

Members

Members	Description
HLVD_LIMIT_TRIP_POINT_4	Voltage limit is trip point 4
HLVD_LIMIT_TRIP_POINT_5	Voltage limit is trip point 5
HLVD_LIMIT_TRIP_POINT_6	Voltage limit is trip point 6
HLVD_LIMIT_TRIP_POINT_7	Voltage limit is trip point 7
HLVD_LIMIT_TRIP_POINT_8	Voltage limit is trip point 8
HLVD_LIMIT_TRIP_POINT_9	Voltage limit is trip point 9
HLVD_LIMIT_TRIP_POINT_10	Voltage limit is trip point 10
HLVD_LIMIT_TRIP_POINT_11	Voltage limit is trip point 11
HLVD_LIMIT_TRIP_POINT_12	Voltage limit is trip point 12
HLVD_LIMIT_TRIP_POINT_13	Voltage limit is trip point 13
HLVD_LIMIT_TRIP_POINT_14	Voltage limit is trip point 14
HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN	Voltage limit is external analog voltage on the pin HLVDIN

Description

High/Low-Voltage Detection limits

This enumeration lists the voltage limits that can be used as reference to High/Low-Voltage Detection on VDD.

Refer to the â€œElectrical Characteristicsâ€• chapter of the specific device data sheet for the actual trip points (i.e., voltages).

Remarks

Not all voltage limits exist on all devices. Please refer to the specific device data sheet for availability.

HLVD_MODE Enumeration

Lists the modes for the High/Low Voltage Detection (HLVD) module.

File

help_plib_power.h

C

```
typedef enum {
  HLVD_MODE_LOW_VOLTAGE_DETECTION,
  HLVD_MODE_HIGH_VOLTAGE_DETECTION
} HLVD_MODE;
```

Members

Members	Description
HLVD_MODE_LOW_VOLTAGE_DETECTION	In Low-Voltage Detection (LVD) mode, an event occurs when the voltage equals or falls below the voltage limit.
HLVD_MODE_HIGH_VOLTAGE_DETECTION	In High-Voltage Detection (HVD) mode, an event occurs when the voltage equals or exceeds the voltage limit.

Description

High/Low-Voltage Detection modes

This enumeration lists the modes those can be used for High/Low Voltage Detection on VDD.

Remarks

None.

Files

Files

Name	Description
plib_power.h	Defines the Power Peripheral Library interface.
help_plib_power.h	This is file help_plib_power.h.

Description

This section lists the source and header files used by the library.

plib_power.h

Defines the Power Peripheral Library interface.

Functions

	Name	Description
	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepEventStatusClear	Clear any events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepEventStatusGet	Returns the events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepGPRRead	Reads 32-bit data from the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepGPRsRetentionDisable	Disables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRsRetentionEnable	Enables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRWrite	Writes 32-bit data into the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepModeDisable	Disables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeEnable	Enables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeHasOccurred	Returns the Deep Sleep mode status of the device.
	PLIB_POWER_DeepSleepModeIsEnabled	Returns the enable/disable status of Deep Sleep mode.

	PLIB_POWER_DeepSleepModuleDisable	Disables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleEnable	Enables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRelease	Releases I/O pins upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRetain	Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepStatusClear	Clears the Deep Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepWakeupEventDisable	Disables wake-up from Deep Sleep mode by the selected event.
	PLIB_POWER_DeepSleepWakeupEventEnable	Enables wake-up from the Deep Sleep mode by the selected event.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_ExistsDeepSleepEventStatus	Identifies whether the DeepSleepEventStatus feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPROperation	Identifies whether the DeepSleepGPROperation feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPRsRetentionControl	Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModeOccurrence	Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModuleControl	Identifies whether the DeepSleepModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepPortPinsStateControl	Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepWakeupEventControl	Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.
	PLIB_POWER_ExistsHighVoltageOnVDD1V8	Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.
	PLIB_POWER_ExistsHLVDBandGapVoltageStability	Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.
	PLIB_POWER_ExistsHLVDEnableControl	Identifies whether the HLVDEnableControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDLimitSelection	Identifies whether the HLVDLimitSelection feature exists on the Power module.
	PLIB_POWER_ExistsHLVDModeControl	Identifies whether the HLVDModeControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStatus	Identifies whether the HLVDStatus feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStopInIdleControl	Identifies whether the HLVDStopInIdleControl feature exists on the Power module.
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the Power module.
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the Power module.
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the Power module.
	PLIB_POWER_HighVoltageOnVDD1V8HasOccurred	Returns the DDR2 High Voltage Detection status of the device.
	PLIB_POWER_HLVDBandGapVoltageIsStable	Returns the status of Band Gap voltage.
	PLIB_POWER_HLVDDisable	Disables High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDEnable	Enables High/Low-Voltage Detection (HLVD) on VDD.
	PLIB_POWER_HLVDIsEnabled	Returns the enable/disable status of High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDLimitSelect	Selects the HLVD limit.
	PLIB_POWER_HLVDModeSelect	Selects the Voltage Detection mode.
	PLIB_POWER_HLVDStatusGet	Returns the HLVD event status.
	PLIB_POWER_HLVDStopInIdleDisable	Continues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleEnable	Discontinues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleIsEnabled	Returns the Stop in Idle mode status of the HLVD operation.
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Checks to see whether or not the selected peripheral is enabled.

	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.

Description

Power Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Power Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Power Controller module.

File Name

plib_power.h

Company

Microchip Technology Inc.

help_plib_power.h

Enumerations

	Name	Description
	DEEP_SLEEP_EVENT	Lists the events that occurred during Deep Sleep mode.
	DEEP_SLEEP_GPR	Lists the Deep Sleep General Purpose Registers (GPRs).
	DEEP_SLEEP_MODULE	Lists the modules that can be enabled/disabled during Deep Sleep mode.
	DEEP_SLEEP_WAKE_UP_EVENT	Lists the events that can be used to wake the device from Deep Sleep mode.
	HLVD_LIMIT	Lists the voltage limits for the HLVD module.
	HLVD_MODE	Lists the modes for the High/Low Voltage Detection (HLVD) module.
	POWER_MODULE	List of the modules whose power can be controlled.
	POWER_MODULE_ID	Identifies the supported Power modules.

Description

This is file help_plib_power.h.

Reset Peripheral Library

This section describes the Reset Peripheral Library.

Introduction

This library provides a low-level abstraction of the Reset Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Reset Controller is a key component of a microcontroller. The Reset Controller gives the reset status of the device at any point of time.

The following are the key features present on a Reset Controller:

- **Reset Source Configuration**: Provides the ability to enable and disable a source that may reset the device
- **Reset Status:** Identifies the source of a reset and clears it

A microcontroller can support one or more of the previously described reset modes.

Note: Please refer to the **"Resets"** chapter in the specific device data sheet to determine the exact set of supported features.

Using the Library

This topic describes the basic architecture of the Reset Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_reset.h`

The interface to the Reset Peripheral Library is defined in the `plib_reset.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Reset Peripheral Library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the **"Resets"** chapter in the specific device data sheet for availability.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

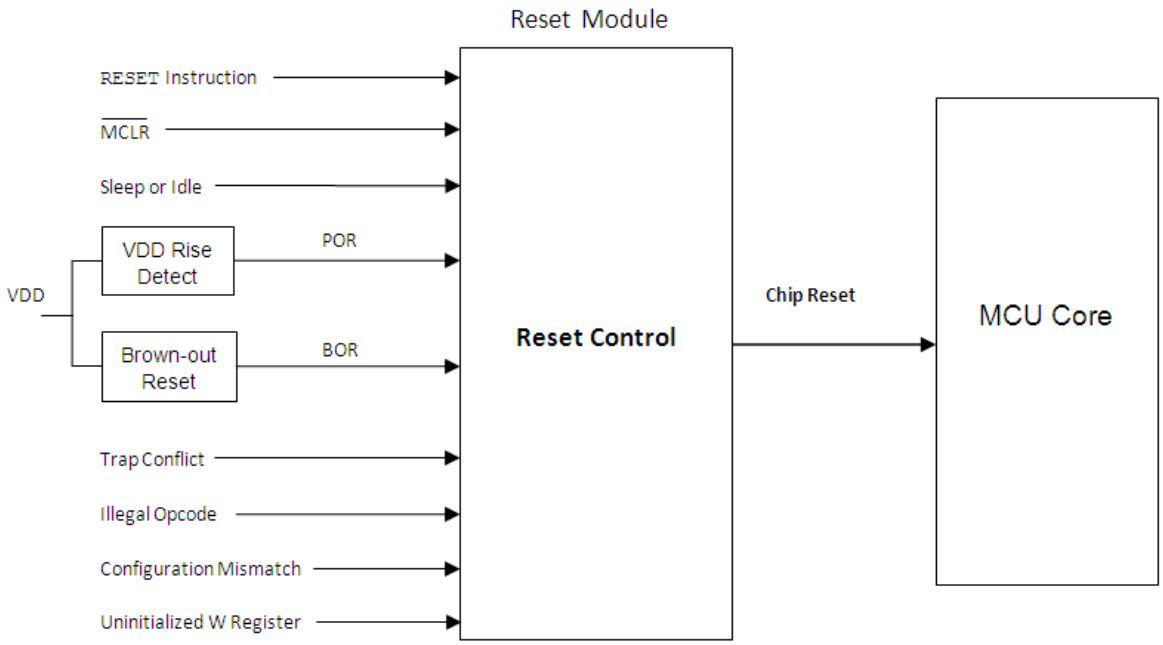
Hardware Abstraction Model

This library provides a low-level abstraction of the Reset Controller module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure illustrates the hardware abstraction model for the Reset Peripheral Library.

Hardware Abstraction Model



The **Reset Controller** receives requests from multiple **Reset Sources**. The **Reset Controller** controls the overall operation of the reset controller.

Source Enable Configuration enables or disables a particular source of reset. Some of the reset sources such as Traps are not controlled by the users, whereas others can be user controlled.

Note: Not all features are available on all devices. Refer to the **"Resets"** chapter in the specific device data sheet for availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Reset Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring and enabling a reset.
Status Functions	This section provides a function to read the status of a particular reset.
NMI Events Functions	This section provides functions for NMI events.
Feature Existence Functions	This section provides functions that determine whether certain features exist.

Note: Please refer to the **"Resets"** chapter in the specific device data sheet to determine the exact set of supported features.

How the Library Works

Initialization

The steps that are required to initialize the Reset Controller module vary for different microcontrollers. Refer to the **"Resets"** chapter in the specific device data sheet for the correct initialization sequence. The following information provides a general overview.

Description

Reset Source Setup

Some reset sources need to be enabled for it to be able to Reset the device. Use the function PLIB_RESET_SoftwareResetEnable to enable the reset source.

Reset Status

The status of the reset indicates which source generated the reset. The status of the reset source can be obtained using the function PLIB_RESET_ReasonGet. The reason for the reset can be cleared using the function PLIB_RESET_ReasonClear.

Configuring the Library

The library is configured for the supported Reset Controller modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_RESET_SoftwareResetEnable	Enables the software reset.

b) Status Functions

	Name	Description
	PLIB_RESET_ReasonClear	Clears the RCON register.
	PLIB_RESET_ReasonGet	Returns the reason for a reset.
	PLIB_RESET_ConfigRegReadErrorGet	Gets the Configuration register read error.
	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

c) NMI Events Functions

	Name	Description
	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value.
	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter.
	PLIB_RESET_NmiEventClear	Clears the NMI event
	PLIB_RESET_NmiEventTrigger	Triggers the NMI event.
	PLIB_RESET_NmiReasonGet	Returns the reason for the Non-Maskable Interrupt (NMI).

d) Feature Existence Functions

	Name	Description
	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the Reset module.
	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the Reset module.
	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the Reset module.
	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the Reset module.
	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the Reset module.
	PLIB_RESET_ExistsWdtoInSleep	Identifies whether the WdtoInSleep feature exists on the Reset module.

e) Data Types and Constants

	Name	Description
	RESET_MODULE_ID	Identifies the supported Reset modules.
	RESET_REASON	Lists the reset sources.
	RESET_CONFIG_REG_READ_ERROR	Lists the Configuration register read errors.
	RESET_NMI_COUNT_TYPE	Defines NMI counter data type
	RESET_NMI_REASON	Lists the NMI reasons.

Description

This section describes the Application Programming Interface (API) functions of the Reset Peripheral Library.

Refer to each section for a detailed description.

a) Initialization Functions

PLIB_RESET_SoftwareResetEnable Function

Enables the software reset.

File

plib_reset.h

C

```
void PLIB_RESET_SoftwareResetEnable(RESET_MODULE_ID index);
```

Returns

None.

Description

This function triggers a software reset.

Remarks

This function implements an operation of the SoftwareResetTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsSoftwareResetTrigger function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_SoftwareResetEnable.

Example

```
//Call system service to unlock the system
PLIB_RESET_SoftwareResetEnable( RESET_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RESET_SoftwareResetEnable (RESET_MODULE_ID index)

b) Status Functions

PLIB_RESET_ReasonClear Function

Clears the RCON register.

File

plib_reset.h

C

```
void PLIB_RESET_ReasonClear(RESET_MODULE_ID index, RESET_REASON reason);
```

Returns

None

Description

This function clears the particular reset bit in the RCON register. Multiple reasons for a reset can be ORed together and given as an input parameter to clear them simultaneously.

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsResetReasonStatus function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RESET_ReasonClear( RESET_ID_0, RESET_REASON_MCLR | RESET_REASON_POWERON );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
reason	Select the reset type from enum RESET_REASON

Function

void PLIB_RESET_ReasonClear(RESET_MODULE_ID index, RESET_REASON reason)

PLIB_RESET_ReasonGet Function

Returns the reason for a reset.

File

plib_reset.h

C

```
RESET_REASON PLIB_RESET_ReasonGet(RESET_MODULE_ID index);
```

Returns

RESET_REASON - Type of reset (when there is more than one reason for a reset, this function will logically OR (bitwise) the reasons and return the value.

Description

This function returns the reason a reset has occurred for the selected device.

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsResetReasonStatus function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
RESET_REASON type;

type = PLIB_RESET_ReasonGet ( RESET_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_REASON PLIB_RESET_ReasonGet (RESET_MODULE_ID index)

PLIB_RESET_ConfigRegReadErrorGet Function

Gets the Configuration register read error.

File

plib_reset.h

C

```
RESET_CONFIG_REG_READ_ERROR PLIB_RESET_ConfigRegReadErrorGet(RESET_MODULE_ID index);
```

Returns

RESET_CONFIG_REG_READ_ERROR - Type of Configuration Register Read Error

Description

This function returns the Configuration register read error, if any, after the reset.

Remarks

This function implements an operation of the ConfigRegReadError feature. This feature may not be available on all devices. Refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsConfigRegReadError function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_ConfigRegReadErrorGet( RESET_ID_0 )== PRIMARY_CONFIG_REG_READ_ERROR )
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_CONFIG_REG_READ_ERROR PLIB_RESET_ConfigRegReadErrorGet(RESET_MODULE_ID index);

PLIB_RESET_WdtTimeOutHasOccurredInSleep Function

Returns the state of the device when WDT time-out occurred

File

plib_reset.h

C

```
bool PLIB_RESET_WdtTimeOutHasOccurredInSleep(RESET_MODULE_ID index);
```

Returns

- true - The device was in Sleep mode when a WDT time-out occurred
- false - The device was not in Sleep mode when a WDT time-out occurred

Description

This function returns whether or not the the device was in Sleep mode when a WDT time-out event occurred.

Remarks

This function implements an operation of the WdtoInSleep feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsWdtoInSleep function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_WdtTimeOutHasOccurredInSleep( RESET_ID_0 ))
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RESET_WdtTimeOutHasOccurredInSleep (RESET_MODULE_ID index)

c) NMI Events Functions

PLIB_RESET_NmiCounterValueGet Function

Gets the NMI counter value.

File

plib_reset.h

C

```
RESET_NMI_COUNT_TYPE PLIB_RESET_NmiCounterValueGet(RESET_MODULE_ID index);
```

Returns

nmi_count - NMI counter value

Description

This function returns the NMI Reset counter value.

Remarks

This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiCounter function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
RESET_NMI_COUNT_TYPE nmi_count;
nmi_count = PLIB_RESET_NmiCounterValueGet( RESET_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_NMI_COUNT_TYPE PLIB_RESET_NmiCounterValueGet (RESET_MODULE_ID index)

PLIB_RESET_NmiCounterValueSet Function

Sets the NMI counter.

File

plib_reset.h

C

```
void PLIB_RESET_NmiCounterValueSet(RESET_MODULE_ID index, RESET_NMI_COUNT_TYPE nmi_count);
```

Returns

None.

Description

This function sets the NMI counter to be expired for a WDT or DMT reset.

Remarks

NMI counter value range may vary between devices. Please refer to the specific device data sheet. This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiCounter function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiCounterValueSet

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiCounterValueSet( RESET_ID_0,  0x54);
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_count	NMI counter value

Function

void PLIB_RESET_NmiCounterValueSet (RESET_MODULE_ID index, RESET_NMI_COUNT_TYPE nmi_count)

PLIB_RESET_NmiEventClear Function

Clears the NMI event

File

plib_reset.h

C

```
void PLIB_RESET_NmiEventClear(RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason);
```

Returns

None

Description

This function clears the NMI event. If a WDTO or DMTO NMI event is cleared before the NMI counter reaches zero, no device Reset is asserted.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiEventTrigger.

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiEventClear( RESET_ID_0, SOFTWARE_NMI );
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	Sets of reasons that can cause a NMI

Function

void PLIB_RESET_NmiEventClear (RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason)

PLIB_RESET_NmiEventTrigger Function

Triggers the NMI event.

File

plib_reset.h

C

```
void PLIB_RESET_NmiEventTrigger(RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason);
```

Returns

None

Description

This function triggers the NMI. In case of a Deadman Timer (DMT) or Watchdog Timer (WDT) NMI event, it will also trigger the NMI counter to start the countdown.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiEventTrigger.

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiEventTrigger( RESET_ID_0, SOFTWARE_NMI );
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	Sets of reasons which can cause NMI

Function

void PLIB_RESET_NmiEventTrigger (RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason)

PLIB_RESET_NmiReasonGet Function

Returns the reason for the Non-Maskable Interrupt (NMI).

File

plib_reset.h

C

```
RESET_NMI_REASON PLIB_RESET_NmiReasonGet(RESET_MODULE_ID index);
```

Returns

RESET_NMI_REASON - Sets of reasons that can cause a NMI.

Description

This function returns the reason that caused the NMI.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_NmiReasonGet( RESET_ID_0 )== WDTO_NMI )
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_NMI_REASON PLIB_RESET_NmiReasonGet (RESET_MODULE_ID index)

d) Feature Existence Functions

PLIB_RESET_ExistsResetReasonStatus Function

Identifies whether the ResetReasonStatus feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsResetReasonStatus(RESET_MODULE_ID index);
```

Returns

- true - The ResetReasonStatus feature is supported on the device
- false - The ResetReasonStatus feature is not supported on the device

Description

This function identifies whether the ResetReasonStatus feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_ReasonGet
- PLIB_RESET_ReasonClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsResetReasonStatus(RESET_MODULE_ID index)

PLIB_RESET_ExistsSoftwareResetTrigger Function

Identifies whether the SoftwareResetTrigger feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsSoftwareResetTrigger(RESET_MODULE_ID index);
```

Returns

- true - The SoftwareResetTrigger feature is supported on the device
- false - The SoftwareResetTrigger feature is not supported on the device

Description

This function identifies whether the SoftwareResetTrigger feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_SoftwareResetEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsSoftwareResetTrigger(RESET_MODULE_ID index)

PLIB_RESET_ExistsConfigRegReadError Function

Identifies whether the ConfigRegReadError feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsConfigRegReadError(RESET_MODULE_ID index);
```

Returns

- true - The ConfigRegReadError feature is supported on the device
- false - The ConfigRegReadError feature is not supported on the device

Description

This function identifies whether the ConfigRegReadError feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_ConfigRegReadErrorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsConfigRegReadError(RESET_MODULE_ID index)

PLIB_RESET_ExistsNmiControl Function

Identifies whether the NmiControl feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsNmiControl(RESET_MODULE_ID index);
```

Returns

- true - The NmiControl feature is supported on the device
- false - The NmiControl feature is not supported on the device

Description

This function identifies whether the NmiControl feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_NmiReasonGet

- PLIB_RESET_NmiEventTrigger
- PLIB_RESET_NmiEventClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsNmiControl(RESET_MODULE_ID index)

PLIB_RESET_ExistsNmiCounter Function

Identifies whether the NmiCounter feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsNmiCounter(RESET_MODULE_ID index);
```

Returns

- true - The NmiCounter feature is supported on the device
- false - The NmiCounter feature is not supported on the device

Description

This function identifies whether the NmiCounter feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_NmiCounterValueSet
- PLIB_RESET_NmiCounterValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsNmiCounter(RESET_MODULE_ID index)

PLIB_RESET_ExistsWdtoInSleep Function

Identifies whether the WdtoInSleep feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsWdtoInSleep(RESET_MODULE_ID index);
```

Returns

- true - The WdtoInSleep feature is supported on the device

- false - The WdtoInSleep feature is not supported on the device

Description

This function identifies whether the WdtoInSleep feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_WdtTimeOutHasOccurredInSleep

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsWdtoInSleep(RESET_MODULE_ID index)

e) Data Types and Constants

RESET_MODULE_ID Enumeration

Identifies the supported Reset modules.

File

help_plib_reset.h

C

```
typedef enum {
  RESET_ID_0
} RESET_MODULE_ID;
```

Members

Members	Description
RESET_ID_0	RESET Module ID 0

Description

Reset Module ID

This enumeration identifies the Reset modules that are available on the microcontroller. This is the superset of all the possible instances that might be available on a Microchip microcontroller.

Remarks

Caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the "Resets" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

RESET_REASON Enumeration

Lists the reset sources.

File

help_plib_reset.h

C

```
typedef enum {
  RESET_REASON_NONE = 0x00000000,
  RESET_REASON_POWERON = 0x00000003,
  RESET_REASON_BROWNOUT = 0x00000002,
  RESET_REASON_WDT_TIMEOUT = 0x00000010,
```

```
  RESET_REASON_SOFTWARE = 0x00000040,
  RESET_REASON_MCLR = 0x00000080,
  RESET_REASON_CONFIG_MISMATCH = 0x00000200,
  RESET_REASON_VBAT = 0x00010000,
  RESET_REASON_VBPOR = 0x00020000,
  RESET_REASON_HIGH_VOLTAGE_DETECT = 0x20000000,
  RESET_REASON_DMT_TIMEOUT = 0x00000020,
  RESET_REASON_ALL = 0x200002D3
} RESET_REASON;
```

Members

Members	Description
RESET_REASON_NONE = 0x00000000	The last reset was of unknown type
RESET_REASON_POWERON = 0x00000003	The last reset was a power on reset
RESET_REASON_BROWNOUT = 0x00000002	The last reset was a brown out reset
RESET_REASON_WDT_TIMEOUT = 0x00000010	The last reset was a watch dog timer time-out reset
RESET_REASON_SOFTWARE = 0x00000040	The last reset was a software reset
RESET_REASON_MCLR = 0x00000080	The last reset was a master clear(MCLR) reset
RESET_REASON_CONFIG_MISMATCH = 0x00000200	The last reset was a configuration mismatch reset
RESET_REASON_VBAT = 0x00010000	The last reset was a VBAT reset
RESET_REASON_VBPOR = 0x00020000	The last reset was a VBPOR reset, because of no supply or Brown-out on VBAT pin
RESET_REASON_HIGH_VOLTAGE_DETECT = 0x20000000	The last reset was high voltage detect reset
RESET_REASON_DMT_TIMEOUT = 0x00000020	The last reset was a Deadman Timer time-out reset
RESET_REASON_ALL = 0x200002D3	All reset flags are high

Description

RESET source select enumeration

This enumeration lists the possible reset sources.

Remarks

Not all reset sources exist on all devices. Please refer to the specific device data sheet for availability.

RESET_CONFIG_REG_READ_ERROR Enumeration

Lists the Configuration register read errors.

File

help_plib_reset.h

C

```
typedef enum {
  PRIMARY_CONFIG_REG_READ_ERROR,
  PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR,
  NO_CONFIG_REG_READ_ERROR
} RESET_CONFIG_REG_READ_ERROR;
```

Members

Members	Description
PRIMARY_CONFIG_REG_READ_ERROR	An error occurred during the read of primary configuration registers
PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR	An error occurred during the read of primary and alternate configuration register
NO_CONFIG_REG_READ_ERROR	No error occurred during the read of configuration registers

Description

RESET Config Register Read Error Enumeration

This enumeration lists the possible errors while reading Configuration registers.

Remarks

Not all errors exist on all devices. Please refer to the specific device data sheet for availability.

RESET_NMI_COUNT_TYPE Type

Defines NMI counter data type

File

plib_reset.h

C

```
typedef unsigned int RESET_NMI_COUNT_TYPE;
```

Description

RESET_NMI_COUNT_TYPE data type

NMI counter data type definition.

RESET_NMI_REASON Enumeration

Lists the NMI reasons.

File

help_plib_reset.h

C

```
typedef enum {
  WDTO_NMI,
  DMTO_NMI,
  SOFTWARE_NMI,
  GLOBAL_NMI,
  LVD_NMI,
  CF_NMI,
  WDTS_NMI
} RESET_NMI_REASON;
```

Members

Members	Description
WDTO_NMI	Watch Dog Timer time-out has caused NMI
DMTO_NMI	Deadman Timer time-out has caused NMI
SOFTWARE_NMI	Software triggered NMI will be generated
GLOBAL_NMI	General NMI or user-initiated NMI has occurred
LVD_NMI	Low Voltage Detection Condition has caused NMI
CF_NMI	Clock Failure has caused NMI
WDTS_NMI	Watch Dog Timer in Sleep has caused NMI

Description

NMI Reason enumeration

This enumeration lists the possible NMI sources.

Remarks

Not all NMI sources exist on all devices. Please refer to the specific device data sheet for availability.

Files

Files

Name	Description
plib_reset.h	Defines the Reset Peripheral Library interface.
help_plib_reset.h	This is file help_plib_reset.h.

Description

This section lists the source and header files used by the library.

plib_reset.h

Defines the Reset Peripheral Library interface.

Functions

	Name	Description
	PLIB_RESET_ConfigRegReadErrorGet	Gets the Configuration register read error.
	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the Reset module.
	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the Reset module.
	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the Reset module.
	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the Reset module.
	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the Reset module.
	PLIB_RESET_ExistsWdtoInSleep	Identifies whether the WdtoInSleep feature exists on the Reset module.
	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value.
	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter.
	PLIB_RESET_NmiEventClear	Clears the NMI event
	PLIB_RESET_NmiEventTrigger	Triggers the NMI event.
	PLIB_RESET_NmiReasonGet	Returns the reason for the Non-Maskable Interrupt (NMI).
	PLIB_RESET_ReasonClear	Clears the RCON register.
	PLIB_RESET_ReasonGet	Returns the reason for a reset.
	PLIB_RESET_SoftwareResetEnable	Enables the software reset.
	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

Types

	Name	Description
	RESET_NMI_COUNT_TYPE	Defines NMI counter data type

Description

Reset Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Reset Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Reset Controller module.

File Name

plib_reset.h

Company

Microchip Technology Inc.

help_plib_reset.h

Enumerations

	Name	Description
	RESET_CONFIG_REG_READ_ERROR	Lists the Configuration register read errors.
	RESET_MODULE_ID	Identifies the supported Reset modules.
	RESET_NMI_REASON	Lists the NMI reasons.
	RESET_REASON	Lists the reset sources.

Description

This is file help_plib_reset.h.

RTCC Peripheral Library

This section describes the Real-Time Clock and Calendar (RTCC) Peripheral Library.

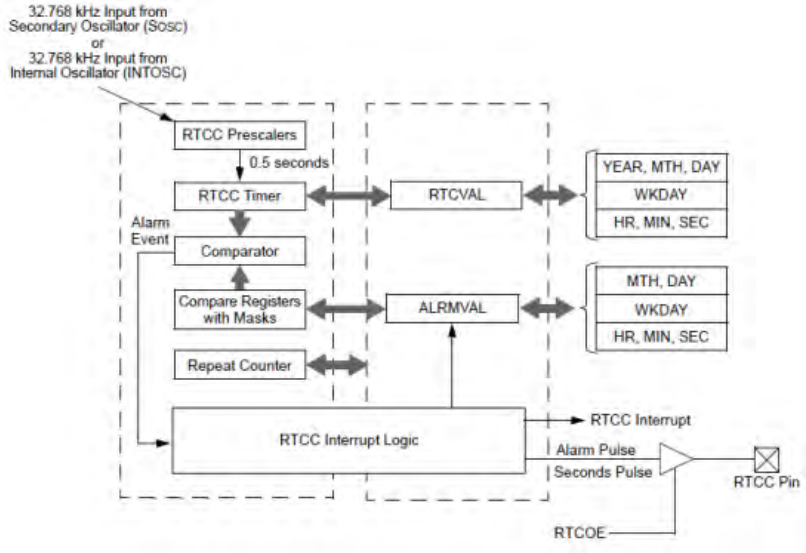
Introduction

This module provides real-time clock and calendar functions. The RTCC is intended for applications where accurate time must be maintained for extended periods with minimum to no intervention from the CPU. The module is optimized for low-power usage to provide extended battery life while keeping track of time.

Description

The RTCC module is a 100-year clock and calendar with automatic leap year detection. The range of the clock is from 00:00:00 (midnight) on January 1, 2000 to 23:59:59 on December 31, 2099. The hours are available in 24-hour (military time) format. The clock provides a granularity of one second with half-second visibility to the user.

RTCC Block Diagram



Operating Modes

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

RTCC Mode

The RTCC module is intended to be clocked by an external real-time clock crystal oscillating at 32.768 kHz. The prescaler divides the crystal oscillator frequency to produce a 1 Hz frequency for the clock and calendar. The current Date and Time are stored in a BCD counter.

Alarm Mode

The RTCC module provides an alarm function that is configurable from a half-second to one year. After the alarm is enabled, the RTCC module can be configured to repeat the alarm at preconfigured intervals. The indefinite repetition of the alarm is provided through the Chime feature.

Using the Library

This topic describes the basic architecture of the RTCC Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_rtcc.h`

The interface to the RTCC Peripheral Library is defined in the `plib_rtcc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the RTCC Peripheral Library should include `peripheral.h`.

Library File: The RTCC Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the peripheral interacts with the framework.

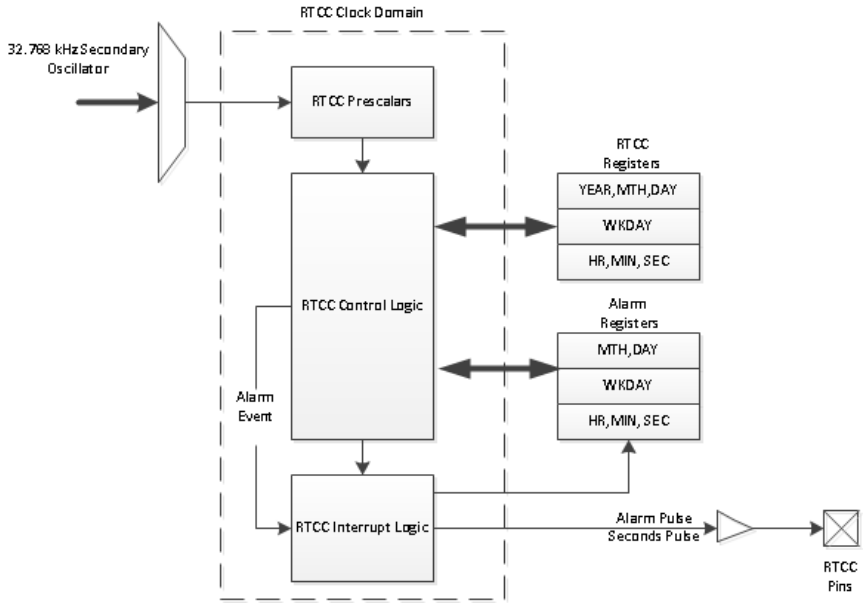
Hardware Abstraction Model

This library provides a low-level abstraction of the RTCC module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The RTCC peripheral library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model



The **RTCC Control Logic** provides the capability to operate the RTCC module in the RTCC and Alarm modes. The control logic also handles the comparison and the counter increments, which in turn control the alarm generation. This module is also responsible for the generation of a square wave at the RTCC output pin. In addition, the RTCC module provides the RTCC drift calibration.

The **RTCC Interrupt Logic** is primarily used in alarm generation. The various configurations of the alarm and their repetition period may be defined.

The **RTCC Registers** store the actual date and time. Separate registers are present for RTCC and alarm functionality.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the RTCC module.

Library Interface Section	Description
General Functions	Provides the setup and configuration of general functionalities in the RTCC module. This section also covers the drift calibration.
RTCC Mode Functions	Provides configuration of the RTCC-related registers. Updating the Date and Time registers along with reading them accurately is performed.
Alarm Mode Functions	Provides configuration of the Alarm-related registers. Updating the Date and Time registers along with reading them accurately is performed. In addition, the rate at which an alarm occurs and the frequency is also configured in this mode.
Other Functions	Provides additional RTCC functions.
Feature Existence Functions	Provides functions that determine whether certain features exist on a device.

How the Library Works

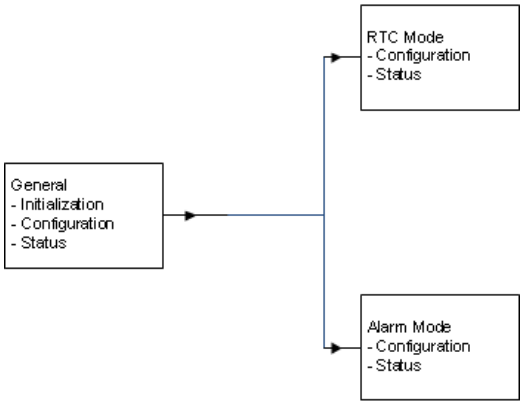
Provides information on how the library works.

Description

The usage model for this library is explained in the following sections.

The following diagram describes the major components of the usage model.

Usage Model



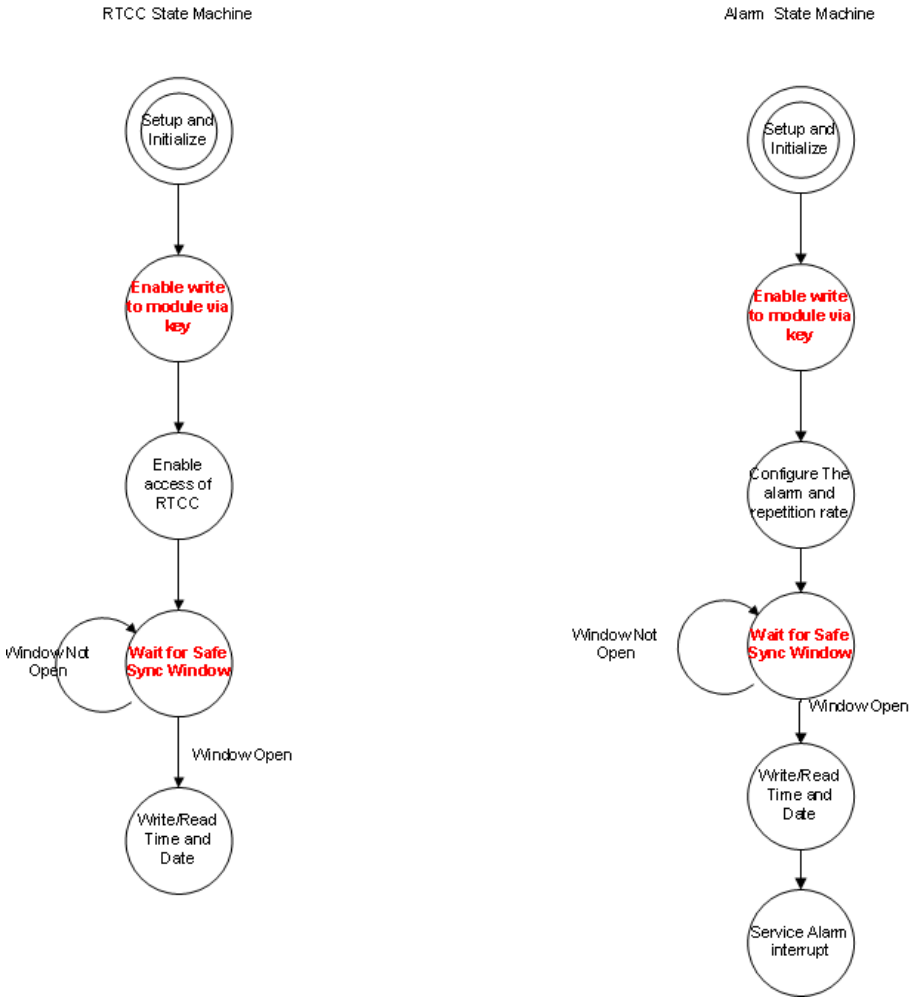
State Machine

This RTCC state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

The following state machine diagram is for RTCC mode and Alarm Mode during normal operation.

RTCC State Machine



State	Associated Function
Setup and Initialization	Refer to mode of RTCC for detailed instructions of setup.

Enable Write via Key Sequence	For RTCC to perform any function, the module has to be enabled via a series of commands in a special sequence. This is abstracted and is achieved using PLIB_RTCC_WriteEnable.
Enable Access of RTC	These include the functions needed to set up the RTCC/Alarm, which will be explained in detail in the following sections.
Wait for Safe Sync Window	To update the RTCC on-the-fly, a safe window period is available, corresponding to 32 clock cycles called RTSYNC. Further sections will explain the functionality in greater detail.
Write/Read Date and Time	The application may read or write the date and time using PLIB_RTCC_RTCDateGet and PLIB_RTCC_RTCTimeGet or PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet, among others. These functions are device-specific and may not be available for all devices.

RTCC Mode Operations

This section describes the RTCC mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

Enabling the RTCC Module for Write Operations

To perform a write to any of the RTCC timer registers the RTCWREN bit must be set. It is recommended that this bit be reset unless a write operation is specifically required.

A sequence of commands, each varying across processor families must be executed to set the RTCWREN.

Example: Enabling a Write

```
// Assume Interrupts are disabled.
// Assume the DMA controller is suspended

PLIB_RTCC_WriteEnable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

// This API performs the writing of the sequence
// 0xaa996655 and 0x556699aa to the SYSKET in MCU32
// and 0x55 and 0xaa in case of MCU 16 and MCU 8
// This API then sets the RTCWREN.
```

Updating RTCC Time and Date

1. Turn off the RTCC using the API PLIB_RTCC_Disable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_RTCSyncStatusGet (this operation may not be required on all devices).
3. Update the Date and Time registers using PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet.
4. Turn on the RTCC module using PLIB_RTCC_Enable.

Example: Updating RTCC Time and Date

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_RTCSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_RTCTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_RTCDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_Enable(MY_RTCC_INSTANCE);
```

Updating RTCC Time and Date using RTSYNC Window

1. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_RTCSyncStatusGet.
2. Update the RTC Date and Time registers using PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet.

Example: Updating RTCC Time and Date Using RTSYNC Window

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

//Disable Critical Interrupts here ....

while(PLIB_RTCC_RTCSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_RTCTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_RTCDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

//Enable Critical Interrupts here ....
```

Updating RTCC Time and Date Using Register Pointers

1. Turn off the RTCC using PLIB_RTCC_Disable.
2. Identify whether the RTCTimeValue and RTCDateValue features exist on the RTCC module using PLIB_RTCC_ExistsRTCTime and PLIB_RTCC_ExistsRTCDate.
3. Update the RTC Date and Time registers using PLIB_RTCC_RTCDateSet
4. and PLIB_RTCC_RTCtimeSet.
5. Turn on the RTCC module using PLIB_RTCC_Enable.

Example: Updating RTCC Time and Date Using Register Pointers

```
// Assume Write Enable has been performed correctly

PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

PLIB_RTCC_RTCYearSet(MY_RTCC_INSTANCE, 0x0007);   // Update the Year

PLIB_RTCC_RTCMonthSet(MY_RTCC_INSTANCE,0x10);    // Set Month to October

PLIB_RTCC_RTCDaySet(MY_RTCC_INSTANCE,0x28);      // Set Day to the 27th

PLIB_RTCC_RTCWeekDaySet(MY_RTCC_INSTANCE,0x01);  // Set the Day of the Week to Sunday

PLIB_RTCC_RTCHourSet(MY_RTCC_INSTANCE,0x10);     // Set the Hour value to 10

PLIB_RTCC_RTCMinuteSet(MY_RTCC_INSTANCE,0x00);   // Set Minute value to 0

PLIB_RTCC_RTCSecondSet(MY_RTCC_INSTANCE,0x00);   // Set Seconds value to 0

PLIB_RTCC_Enable(MY_RTCC_INSTANCE);
```

Alarm Mode Operations

This section describes the Alarm mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

Configuring the One-Time-Per-Day Alarm

1. Turn off the Alarm and also the chime, alarm repeats, and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to zero using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable.

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*Number of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
```

Configuring the Repeat Alarm

1. Turn off the Alarm and also the Chime, alarm repeats, and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (this operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to 10 using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable.

Example: Configuring the Repeat Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 10 /*No of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
```

Configuring the Indefinite One-Per-Day Alarm

1. Turn off the Alarm and also the Chime, alarm repeats and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to '0' using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable and enable chime using PLIB_RTCC_AlarmChimeEnable.

Example: Configuring the Indefinite One-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off
```

```
PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*No of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);

PLIB_RTCC_AlarmChimeEnable(MY_RTCC_INSTANCE);
```

Other Functionality

This section describes the Drift Calibration mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

The real-time crystal input can be calibrated using the periodic auto-adjust feature. When properly calibrated, the RTCC module can provide an error of less than 0.66 seconds per month. Calibration has the ability to eliminate an error of up to 260 ppm.

The calibration is accomplished by finding the number of error clock pulses and writing this value into the calibration bit fields using PLIB_RTCC_DriftCalibrateSet.

Drift Calibration

1. Using another timer resource, the user must find the error of the 32.768 kHz crystal.
2. Once the error is known, it must be converted to the number of error clock pulses per minute, as shown by ***(Ideal Frequency(32,758) - Measured Frequency) * 60 = Error Clocks Per Minute***.
3. If the error is negative, the input to PLIB_RTCC_DriftCalibrateSet is negative. Likewise, if the error is positive, the input is positive. The negative and positive values are the actual number of clock pulses to be added or subtracted from the timer counter per minute.

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
// Assume steps 1 and 2 are performed and the error is determined.

int driftValue = 0x3FD   // 10 bits Adjustment, -3 in value
int T0,T1;

do
{
    T0 = PLIB_RTCC_RTCTimeGet(MY_RTCC_INSTANCE);
    T1 = PLIB_RTCC_RTCTimeGet(MY_RTCC_INSTANCE);
    // where, MY_RTCC_INSTANCE is a specific instance of the hardware peripheral
}while(T0!=T1)     // Read Valid Time Value

if((T0 & 0XFF) == 0) // we are at seconds exactly 00, wait for auto adjust
{
    while PLIB_RTCC_HalfSecondStatusGet(MY_RTCC_INSTANCE); // Wait for the second Half
}

PLIB_RTCC_DriftCalibrateSet(MY_RTCC_INSTANCE, 0x00); //Clear the calibration

PLIB_RTCC_DriftCalibrateSet(MY_RTCC_INSTANCE, 0xdriftValue);
```

Configuring the Library

The library is configured for the supported RTCC modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Functions

	Name	Description
	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.

b) RTCC Mode Functions

	Name	Description
	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.
	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCDayGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSyncStatusGet	The function returns the synchronization status bit.

c) Alarm Mode Functions

	Name	Description
	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.
	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.

	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.
	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmSyncStatusGet	The function returns the synchronization status bit.

d) Other Functions

	Name	Description
	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

e) Feature Existence Functions

	Name	Description
	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.

f) Data Types and Constants

	Name	Description
	RTCC_ALARM_MASK_CONFIGURATION	Data type defining the different configurations for the alarm mask bits.

	RTCC_MODULE_ID	Enumeration: RTCC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	RTCC_VALUE_REGISTER_POINTER	Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

Description

This section describes the Application Programming Interface (API) functions of the RTCC Peripheral Library.

Refer to each section for a detailed description.

a) General Functions

PLIB_RTCC_ClockOutputDisable Function

Disables the specific RTCC module's output pin.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockOutputDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's output pin.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockOutputDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_ClockOutputDisable (RTCC_MODULE_ID index)

PLIB_RTCC_ClockOutputEnable Function

Enables the specific RTCC module's output pin.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockOutputEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's output and generates a square wave using either the alarm or the 1 Hz clock output on the RTCC pin.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockOutputEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_ClockOutputEnable (RTCC_MODULE_ID index)

PLIB_RTCC_ClockRunningStatus Function

Provides the status of the RTCC clock.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ClockRunningStatus(RTCC_MODULE_ID index);
```

Returns

The status of the RTCC clock.

Description

This function provides the status of the RTCC clock.

Remarks

This function implements an operation of the ClockRunning feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsClockRunning in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_RTCC_ClockRunningStatus(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_ClockRunningStatus (RTCC_MODULE_ID index)

b) RTCC Mode Functions

PLIB_RTCC_Disable Function

Disables the specific RTCC module on the device.

File

plib_rtcc.h

C

```
void PLIB_RTCC_Disable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module on the device.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

The RTCC module should be unlocked for writing using the function PLIB_RTCC_WriteEnable before this function is called.

Example

```
PLIB_RTCC_Disable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_Disable (RTCC_MODULE_ID index)

PLIB_RTCC_Enable Function

Enables the specific RTCC module on the device.

File

plib_rtcc.h

C

```
void PLIB_RTCC_Enable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module on the device.

Remarks

By calling this function, the RTCC pins are controlled by the RTCC module. The RTCC module will continue to function when the device is held in reset.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

The RTCC module should be unlocked for writing using the function PLIB_RTCC_WriteEnable before this function is called.

Example

```
PLIB_RTCC_Enable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_Enable (RTCC_MODULE_ID index)

PLIB_RTCC_WriteDisable Function

Disables writing to the specific RTCC module's value registers.

File

plib_rtcc.h

C

```
void PLIB_RTCC_WriteDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables writing to the specific RTCC module's value registers.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWriteEnable in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_WriteDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_WriteDisable (RTCC_MODULE_ID index)

PLIB_RTCC_WriteEnable Function

Enables writing to the specific RTCC module's value registers.

File

plib_rtcc.h

C

```
void PLIB_RTCC_WriteEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables writing to the specific RTCC module's value registers.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWriteEnable in your application to automatically determine whether this feature is available.

Preconditions

The SYSLOCK unlock sequence must be executed prior to calling this function by calling the PLIB_CORE_SysUnlock function.

Example

```
PLIB_RTCC_WriteEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_WriteEnable (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDateGet Function

Returns the contents of the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCDateGet(RTCC_MODULE_ID index);
```

Returns

Date register contents.

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Date;
Date = PLIB_RTCC_RTCDateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCDateGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDateSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCDateSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x06102705;     //Date = 27 Oct 2006 Friday
PLIB_RTCC_RTCDateSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_RTCDateSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCDayGet Function

Returns the contents of the Days bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCDayGet(RTCC_MODULE_ID index);
```

Returns

Days bits in the Date register.

Description

The function returns the contents of the Days bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Day;
Day = PLIB_RTCC_RTCDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDaySet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCDaySet(RTCC_MODULE_ID index, uint32_t day);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Day = 0x27;        //Day = 27th of the month
PLIB_RTCC_RTCDaySet(RTCC_ID_0, Day);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Date register

Function

void PLIB_RTCC_RTCDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCHourGet Function

Returns the contents of the Hours bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCHourGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Hours bits in the Time register.

Description

The function returns the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour;
Hour = PLIB_RTCC_RTCHourGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCHourGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCHourSet Function

Writes the contents of the Hours bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCHourSet(RTCC_MODULE_ID index, uint32_t hour);
```

Returns

None.

Description

The function writes the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour = 0x04;
PLIB_RTCC_RTCHourSet(RTCC_ID_0, Hour);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Time register

Function

uint32_t PLIB_RTCC_RTCHourSet (RTCC_MODULE_ID index, uint32_t hour)

PLIB_RTCC_RTCMinuteGet Function

Returns the contents of the Minutes bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCMinuteGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Minutes bits in the Time register.

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute;
Minute = PLIB_RTCC_RTCMinuteGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCMinuteGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCMinuteSet Function

Writes the contents of Minutes bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCMinuteSet(RTCC_MODULE_ID index, uint32_t minute);
```

Returns

None.

Description

The function writes the contents of these bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute = 0x15;
PLIB_RTCC_RTCMinuteSet(RTCC_ID_0, Minute);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Time register

Function

uint32_t PLIB_RTCC_RTCMinuteSet (RTCC_MODULE_ID index, uint32_t minute)

PLIB_RTCC_RTCMonthGet Function

Returns the contents of the Months bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCMonthGet(RTCC_MODULE_ID index);
```

Returns

Months bits in the Date register.

Description

The function returns the contents of the Months bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Month;
Month = PLIB_RTCC_RTCMonthGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCMonthGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCMonthSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCMonthSet(RTCC_MODULE_ID index, uint32_t month);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Month = 0x10;      //Month = October
PLIB_RTCC_RTCMonthSet(RTCC_ID_0, Month);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
month	The BCD value of the month to set in the Date register

Function

void PLIB_RTCC_RTCMonthSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCSecondGet Function

The function returns the contents of the Seconds bits in the specific RTCC device's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCSecondGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Seconds bits in the Time register.

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second;
Second = PLIB_RTCC_RTCSecondGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCSecondGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCSecondSet Function

Writes the contents of Seconds bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCSecondSet(RTCC_MODULE_ID index, uint32_t second);
```

Returns

None.

Description

The function writes the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second = 0x33;
PLIB_RTCC_RTCSecondSet(RTCC_ID_0, Second);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Time register

Function

uint32_t PLIB_RTCC_RTCSecondSet (RTCC_MODULE_ID index, uint32_t second)

PLIB_RTCC_RTCTimeGet Function

Returns the contents of the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCTimeGet(RTCC_MODULE_ID index);
```

Returns

Time register contents.

Description

The function returns the contents of the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Example

```
uint32_t time;
time = PLIB_RTCC_RTCTimeGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCTimeGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCTimeSet Function

Writes to the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCTimeSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Time register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x04153300;     // Time = 4 hours, 15 minutes, and 33 seconds
PLIB_RTCC_RTCTimeSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_RTCTimeSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCWeekDayGet Function

Returns the contents of the WeekDay bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCWeekDayGet(RTCC_MODULE_ID index);
```

Returns

WeekDay field in the Date register.

Description

The function returns the contents of the WeekDay bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t WeekDay;
WeekDay = PLIB_RTCC_RTCWeekDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCWeekDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCWeekDaySet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCWeekDaySet(RTCC_MODULE_ID index, uint32_t weekday);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t WeekDay = 0x05;        //WeekDay = Friday
PLIB_RTCC_RTCWeekDaySet(RTCC_ID_0, WeekDay);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the Date register

Function

void PLIB_RTCC_RTCWeekDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCYearGet Function

Returns the contents of the Year bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCYearGet(RTCC_MODULE_ID index);
```

Returns

Year bits in the Date register.

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Year;
Year = PLIB_RTCC_RTCYearGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCYearGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCYearSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCYearSet(RTCC_MODULE_ID index, uint32_t year);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Year = 0x06;       //Year = 2006
PLIB_RTCC_RTCYearSet(RTCC_ID_0, Year);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
year	The BCD value of the year to set in the Date register

Function

void PLIB_RTCC_RTCYearSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCSyncStatusGet Function

The function returns the synchronization status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_RTCSyncStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Date and time will change within 32 RTCC clocks
- false - Date and time are safe to read, and will not change soon

Description

The function returns the synchronization status bit, which is used to determine whether it is safe to read the date/time values, or if the values will change within 32 RTCC clocks.

Remarks

This bit is read-only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsSynchronization in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RTCC_RTCSyncStatusGet(RTCC_ID_0))
{
...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_RTCSyncStatusGet (RTCC_MODULE_ID index)

c) Alarm Mode Functions

PLIB_RTCC_AlarmChimeDisable Function

Disables the specific RTCC module's chime.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmChimeDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's chime. The alarm repeat count value bits stop once they reach zero.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmChimeControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_AlarmChimeDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmChimeDisable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmChimeEnable Function

Enables the specific RTCC module's chime.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmChimeEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's chime. The alarm repeat count bits are allowed to rollover.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmChimeControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_AlarmChimeEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmChimeEnable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmDateGet Function

Returns the contents of the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmDateGet(RTCC_MODULE_ID index);
```

Returns

Value register.

Description

The function returns the contents of the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Date;
Date = PLIB_RTCC_AlarmDateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmDateGet (AlarmC_MODULE_ID index)

PLIB_RTCC_AlarmDateSet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDateSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x06102705;     //Date = 27 Oct 2006 Friday
PLIB_RTCC_AlarmDateSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	The value to set the Alarm Date register to, in BCD format

Function

void PLIB_RTCC_AlarmDateSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmDayGet Function

Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmDayGet(RTCC_MODULE_ID index);
```

Returns

Days bits in the Alarm Date register.

Description

The function returns the contents of Day bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Day;
Day = PLIB_RTCC_AlarmDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmDaySet Function

Writes to the specific RTCC module's Alarm Date value register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDaySet(RTCC_MODULE_ID index, uint32_t day);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Day = 0x27;        //Day = 27th of the month
PLIB_RTCC_AlarmDaySet(RTCC_ID_0, Day);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Alarm Date register

Function

void PLIB_RTCC_AlarmDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmDisable Function

Disables the specific RTCC module's alarm.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's alarm. RTCC Alarm bit shouldn't be modified when RTCC on bit is enabled and PLIB_RTCC_AlarmSyncStatusGet (ALRMSYNC bit) returns true. Meaning the check RTCC ON (RTCCON<15>) bit and the ALRMSYNC bit should be equal to '1'.

Remarks

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// check RTCC enable bit and PLIB_RTCC_AlarmSyncStatusGet return value and
// then modify the Alarm bit.
PLIB_RTCC_AlarmDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmDisable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmEnable Function

Enables the specific RTCC module's alarm.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's alarm. RTCC Alarm bit shouldn't be modified when RTCC on bit is enabled and

PLIB_RTCC_AlarmSyncStatusGet (ALRMSYNC bit) returns true. Meaning the check RTCC ON (RTCCON<15>) bit and the ALRMSYNC bit should be equal to '1'.

Remarks

The alarm enable bit is cleared automatically after an alarm event whenever the alarm is not set up to repeat, and chime is disabled.

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// check RTCC enable bit and PLIB_RTCC_AlarmSyncStatusGet return value and
// then modify the Alarm bit.
PLIB_RTCC_AlarmEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmEnable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmHourGet Function

Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmHourGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Hours bits in the Alarm Time register.

Description

The function returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour;
Hour = PLIB_RTCC_AlarmHourGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmHourGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmHourSet Function

The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmHourSet(RTCC_MODULE_ID index, uint32_t hour);
```

Returns

None

Description

Returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour = 0x04;
PLIB_RTCC_AlarmHourSet(RTCC_ID_0, Hour);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmHourSet (RTCC_MODULE_ID index, uint32_t hour)

PLIB_RTCC_AlarmMaskModeSelect Function

Sets the specific RTCC module's alarm mask Configuration bits.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMaskModeSelect(RTCC_MODULE_ID index, RTCC_ALARM_MASK_CONFIGURATION data);
```

Returns

None.

Description

This function sets the specific RTCC module's alarm mask Configuration bits.

Remarks

The actual definition of this enumeration is device-specific.

This function implements an operation of the AlarmMaskControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmMaskControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 0;
PLIB_RTCC_AlarmMaskModeSelect(RTCC_ID_0, RTCC_ALARM_EVERY_HALF_SECOND);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm mask Configuration bits

Function

void PLIB_RTCC_AlarmMaskModeSelect (RTCC_MODULE_ID index, RTCC_ALARM_MASK_CONFIGURATION data)

PLIB_RTCC_AlarmMinuteGet Function

Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmMinuteGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Minutes bits in the Alarm Time register.

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute;
Minute = PLIB_RTCC_AlarmMinuteGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmMinuteGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmMinuteSet Function

Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMinuteSet(RTCC_MODULE_ID index, uint32_t minute);
```

Returns

None

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute = 0x15;
PLIB_RTCC_AlarmMinuteSet(RTCC_ID_0, Minute);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmMinuteSet (RTCC_MODULE_ID index, uint32_t minute)

PLIB_RTCC_AlarmMonthGet Function

Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmMonthGet(RTCC_MODULE_ID index);
```

Returns

Months bits in the Date register.

Description

The function returns the contents of the Months bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Month;
Month = PLIB_RTCC_AlarmMonthGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmMonthGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmMonthSet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMonthSet(RTCC_MODULE_ID index, uint32_t month);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Month = 0x10;      //Month = October
PLIB_RTCC_AlarmMonthSet(RTCC_ID_0, Month);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
month	The BCD value of the month to set in the Alarm Date register

Function

void PLIB_RTCC_AlarmMonthSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmPulseInitialGet Function

Returns the state of the initial alarm pulse.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_AlarmPulseInitialGet(RTCC_MODULE_ID index);
```

Returns

- 1 - Logical High
- 0 - Logical Low

Description

This function returns the state of the initial alarm pulse.

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmPulseInitial in your application to automatically determine whether this feature is available.

Preconditions

The ALRMEN bit should be '1' indicating the PLIB_RTCC_AlarmEnable function was called.

Example

```
bool PulseValue;
PulseValue = PLIB_RTCC_AlarmPulseInitialGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_AlarmPulseInitialGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmPulseInitialSet Function

Enables the determination of the initial alarm pulse.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmPulseInitialSet(RTCC_MODULE_ID index, bool data);
```

Returns

None.

Description

This function enables the determination of initial alarm pulse.

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmPulseInitial in your application to automatically determine whether this feature is available.

Preconditions

The ALRMEN bit should be '0' indicating the PLIB_RTCC_AlarmDisable was called. This function must not be called when the RTCC is ON and the Alarm Sync is 1.

Example

```
bool data = 0;
PLIB_RTCC_AlarmPulseInitialSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmPulseInitialSet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmRepeatCountGet Function

Reads the specific RTCC module's alarm repeat counter.

File

plib_rtcc.h

C

```
uint8_t PLIB_RTCC_AlarmRepeatCountGet(RTCC_MODULE_ID index);
```

Returns

uint8_t - The current value of the alarm repeat counter

Description

This function reads the specific RTCC module's alarm repeat counter.

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmRepeatControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t currentCount;
currentCount = PLIB_RTCC_AlarmRepeatCountGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_RTCC_AlarmRepeatCountGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmRepeatCountSet Function

Sets the specific RTCC module's alarm repeat counter.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmRepeatCountSet(RTCC_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function sets the specific RTCC module's alarm repeat counter.

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmRepeatControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 0xFF;
PLIB_RTCC_AlarmRepeatCountSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm repeat counter bits

Function

void PLIB_RTCC_AlarmRepeatCountSet (RTCC_MODULE_ID index, uint8_t data)

PLIB_RTCC_AlarmSecondGet Function

Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmSecondGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Seconds bits in the Alarm Time register.

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second;
Second = PLIB_RTCC_AlarmSecondGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmSecondGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmSecondSet Function

Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmSecondSet(RTCC_MODULE_ID index, uint32_t second);
```

Returns

None

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second = 0x33;
PLIB_RTCC_AlarmSecondSet(RTCC_ID_0, Second);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmSecondSet (RTCC_MODULE_ID index, uint32_t second)

PLIB_RTCC_AlarmTimeGet Function

Returns the contents of the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmTimeGet(RTCC_MODULE_ID index);
```

Returns

Value register.

Description

The function returns the contents of the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t time;
time = PLIB_RTCC_AlarmTimeGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmTimeGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmTimeSet Function

Writes to the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmTimeSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Time register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x04153300;     // Time = 4 hours, 15 minutes, and 33 seconds
PLIB_RTCC_AlarmTimeSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmTimeSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmValueRegisterPointer Function

Sets the specific RTCC module's Alarm register pointer.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmValueRegisterPointer(RTCC_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function sets the specific RTCC module's Alarm register pointer.

Remarks

None.

Preconditions

None.

Example

```
uint8_t data = 2;
PLIB_RTCC_AlarmValueRegisterPointer(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm register pointer

Function

void PLIB_RTCC_AlarmValueRegisterPointer (RTCC_MODULE_ID index, uint8_t data)

PLIB_RTCC_AlarmWeekDayGet Function

Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmWeekDayGet(RTCC_MODULE_ID index);
```

Returns

WeekDay bits in the Alarm Date register.

Description

The function returns the contents of Weekday bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t WeekDay;
WeekDay = PLIB_RTCC_AlarmWeekDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmWeekDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmWeekDaySet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmWeekDaySet(RTCC_MODULE_ID index, uint32_t weekday);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t WeekDay = 0x05;        //WeekDay = Friday
PLIB_RTCC_AlarmWeekDaySet(RTCC_ID_0, WeekDay);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the field of the Alarm Date register

Function

void PLIB_RTCC_AlarmWeekDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmSyncStatusGet Function

The function returns the synchronization status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_AlarmSyncStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Alarm repeat count may change as a result of a half-second rollover during a read.
- false - Alarm repeat count is safe to read, and will not change in less than 32 RTC clocks.

Description

The function returns the synchronization status bit, which is used to determine whether it is safe to read the date/time values, or if the values will change within 32 RTCC clocks.

Remarks

This bit is read-only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmSynchronization in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RTCC_AlarmSyncStatusGet(RTCC_ID_0))
{
...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_AlarmSyncStatusGet (RTCC_MODULE_ID index)

d) Other Functions

PLIB_RTCC_ClockSourceSelect Function

Selects the clock source for the RTCC module.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockSourceSelect(RTCC_MODULE_ID index, RTCC_CLOCK_SOURCE_SELECT source);
```

Returns

None.

Description

This function determines which clock source the RTCC module will use depending on the features of the device.

Remarks

This function implements an operation of the ClockSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsClockSelect in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockSourceSelect(RTCC_ID_0, RTCC_CLOCK_SOURCE_SELECT_NONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Which clock source will be used

Function

void PLIB_RTCC_ClockSourceSelect (RTCC_MODULE_ID index, RTCC_CLOCK_SOURCE_SELECT source)

PLIB_RTCC_DriftCalibrateGet Function

Reads the specific RTCC module's drift calibration bits.

File

plib_rtcc.h

C

```
uint16_t PLIB_RTCC_DriftCalibrateGet(RTCC_MODULE_ID index);
```

Returns

uint16_t - The current drift calibration value

Description

This function reads the specific RTCC module's drift calibration bits.

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCalibration in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t calibrationbits;
calibrationbits = PLIB_RTCC_DriftCalibrateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_RTCC_DriftCalibrateGet (RTCC_MODULE_ID index)

PLIB_RTCC_DriftCalibrateSet Function

Sets the specific RTCC module's drift calibration bits.

File

plib_rtcc.h

C

```
void PLIB_RTCC_DriftCalibrateSet(RTCC_MODULE_ID index, uint16_t calibrationbits);
```

Returns

None.

Description

This function sets the specific RTCC module's drift calibration bits. The error between the system clock and the external clock has to be computed and calibration input must be provided to this function.

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCalibration in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t calibrationbits = 3; //Positive 3 adjustment derived from the formula
                              // Error = (Ideal Freq(32758) - Measured)*60;
PLIB_RTCC_DriftCalibrateSet(RTCC_ID_0, calibrationbits);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
calibrationbits	Drift calibration bits

Function

void PLIB_RTCC_DriftCalibrateSet (RTCC_MODULE_ID index, uint16_t calibrationbits)

PLIB_RTCC_HalfSecondStatusGet Function

The function returns the half second status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_HalfSecondStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Second half period of a second
- false - First half period of a second

Description

The function returns the half second status bit, which is used in the calibration procedure. When the seconds byte is zero, the calibration value must be updated when the half second bit becomes '1'.

Remarks

This bit is read-only. It is cleared to '0' on a write to the seconds value. This function implements an operation of the HalfSecond feature. This

feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsHalfSecond in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Wait for the half second status bit to be '1'.
while(PLIB_RTCC_HalfSecondStatusGet(RTCC_ID_0));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_HalfSecondStatusGet (RTCC_MODULE_ID index)

PLIB_RTCC_OutputSelect Function

Selects which signal will be presented on the RTCC pin

File

plib_rtcc.h

C

```
void PLIB_RTCC_OutputSelect(RTCC_MODULE_ID index, RTCC_OUTPUT_SELECT data);
```

Returns

None.

Description

This function selects which signal will be presented on the RTCC pin.

Remarks

The RTCC module's output pin should be enabled using the function PLIB_RTCC_OutputEnable.

This function implements an operation of the OutputSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputSelect in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_OutputSelect(RTCC_ID_0, RTCC_OUTPUT_SECONDS_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Enumerated value of which signal to present

Function

void PLIB_RTCC_OutputSelect (RTCC_MODULE_ID index, RTCC_OUTPUT_SELECT data)

PLIB_RTCC_StopInIdleDisable Function

Continues normal RTCC operation when the device enters Idle mode.

File

plib_rtcc.h

C

```
void PLIB_RTCC_StopInIdleDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function continues normal RTCC operation when the device enters Idle mode.

Remarks

None.

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsStopInIdle in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_StopInIdleDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_StopInIdleDisable(RTCC_MODULE_ID index)

PLIB_RTCC_StopInIdleEnable Function

Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

File

plib_rtcc.h

C

```
void PLIB_RTCC_StopInIdleEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables access to the RTCC module by the PBCLK when the CPU is in Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsStopInIdle in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_StopInIdleEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_StopInIdleEnable(RTCC_MODULE_ID index)

e) Feature Existence Functions

PLIB_RTCC_ExistsAlarmChimeControl Function

Identifies whether the AlarmChimeControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmChimeControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmChimeControl feature is supported on the device
- false - The AlarmChimeControl feature is not supported on the device

Description

This function identifies whether the AlarmChimeControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmChimeEnable
- PLIB_RTCC_AlarmChimeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmChimeControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmControl Function

Identifies whether the AlarmControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmControl feature is supported on the device
- false - The AlarmControl feature is not supported on the device

Description

This function identifies whether the AlarmControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmEnable
- PLIB_RTCC_AlarmDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmDate Function

Identifies whether the AlarmDate feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmDate(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmDate feature is supported on the device
- false - The AlarmDate feature is not supported on the device

Description

This function identifies whether the AlarmDate feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmDateGet
- PLIB_RTCC_AlarmDateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmDate(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmMaskControl Function

Identifies whether the AlarmMaskControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmMaskControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmMaskControl feature is supported on the device
- false - The AlarmMaskControl feature is not supported on the device

Description

This function identifies whether the AlarmMaskControl feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_AlarmMaskModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmMaskControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmPulseInitial Function

Identifies whether the AlarmPulseInitial feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmPulseInitial(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmPulseInitial feature is supported on the device
- false - The AlarmPulseInitial feature is not supported on the device

Description

This function identifies whether the AlarmPulseInitialValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmPulseInitialSet
- PLIB_RTCC_AlarmPulseInitialGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmPulseInitial(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmRepeatControl Function

Identifies whether the AlarmRepeatControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmRepeatControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmRepeatControl feature is supported on the device
- false - The AlarmRepeatControl feature is not supported on the device

Description

This function identifies whether the AlarmRepeatControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmRepeatCountSet
- PLIB_RTCC_AlarmRepeatCountRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmRepeatControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmTime Function

Identifies whether the AlarmTime feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmTime(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmTime feature is supported on the device
- false - The AlarmTime feature is not supported on the device

Description

This function identifies whether the AlarmTime feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmTimeGet
- PLIB_RTCC_AlarmTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmTime(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsCalibration Function

Identifies whether the Calibration feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsCalibration(RTCC_MODULE_ID index);
```

Returns

- true - The Calibration feature is supported on the device
- false - The Calibration feature is not supported on the device

Description

This function identifies whether the Calibration feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_DriftCalibrate

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsCalibration(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsClockRunning Function

Identifies whether the ClockRunning feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsClockRunning(RTCC_MODULE_ID index);
```

Returns

- true - The ClockRunning feature is supported on the device
- false - The ClockRunning feature is not supported on the device

Description

This function identifies whether the ClockRunning feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_ClockRunningStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsClockRunning(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsClockSelect Function

Identifies whether the ClockSelect feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsClockSelect(RTCC_MODULE_ID index);
```

Returns

- true - The ClockSelect feature is supported on the device
- false - The ClockSelect feature is not supported on the device

Description

This function identifies whether the ClockSelect feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_SourceClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsClockSelect(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsEnableControl(RTCC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_Enable
- PLIB_RTCC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsEnableControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsHalfSecond Function

Identifies whether the HalfSecond feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsHalfSecond(RTCC_MODULE_ID index);
```

Returns

- true - The HalfSecond feature is supported on the device
- false - The HalfSecond feature is not supported on the device

Description

This function identifies whether the HalfSecond feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_HalfSecondStatusBit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsHalfSecond(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsOutputControl Function

Identifies whether the OutputControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsOutputControl(RTCC_MODULE_ID index);
```

Returns

- true - The OutputControl feature is supported on the device
- false - The OutputControl feature is not supported on the device

Description

This function identifies whether the OutputControl feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_ClockOutputEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsOutputControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsOutputSelect Function

Identifies whether the OutputSelect feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsOutputSelect(RTCC_MODULE_ID index);
```

Returns

- true - The OutputSelect feature is supported on the device
- false - The OutputSelect feature is not supported on the device

Description

This function identifies whether the OutputSelect feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_SecondsClockOutputSelect
- PLIB_RTCC_AlarmPulseOutputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsOutputSelect(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsRTCDate Function

Identifies whether the RTCDateValue feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsRTCDate(RTCC_MODULE_ID index);
```

Returns

- true - The RTCDate feature is supported on the device
- false - The RTCDate feature is not supported on the device

Description

This function identifies whether the RTCDateValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_RTCDateGet
- PLIB_RTCC_RTCDateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsRTCDate(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsRTCTime Function

Identifies whether the RTCTimeValue feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsRTCTime(RTCC_MODULE_ID index);
```

Returns

- true - The RTCTime feature is supported on the device
- false - The RTCTime feature is not supported on the device

Description

This function identifies whether the RTCTimeValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_RTCTimeGet
- PLIB_RTCC_RTCTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsRTCTime(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsStopInIdleControl(RTCC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_StopInIdleEnable
- PLIB_RTCC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsStopInIdleControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsSynchronization Function

Identifies whether the Synchronization feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsSynchronization(RTCC_MODULE_ID index);
```

Returns

- true - The Synchronization feature is supported on the device
- false - The Synchronization feature is not supported on the device

Description

This function identifies whether the Synchronization feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_RTCSyncStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsSynchronization(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsWriteEnable Function

Identifies whether the WriteEnable feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsWriteEnable(RTCC_MODULE_ID index);
```

Returns

- true - The WriteEnable feature is supported on the device
- false - The WriteEnable feature is not supported on the device

Description

This function identifies whether the WriteEnable feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_WriteEnable
- PLIB_RTCC_WriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsWriteEnable(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmSynchronization Function

Identifies whether the AlarmSynchronization feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmSynchronization(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmSynchronization feature is supported on the device
- false - The AlarmSynchronization feature is not supported on the device

Description

This function identifies whether the AlarmSynchronization feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_AlarmSyncGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmSynchronization(RTCC_MODULE_ID index)

f) Data Types and Constants

RTCC_ALARM_MASK_CONFIGURATION Enumeration

Data type defining the different configurations for the alarm mask bits.

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_ALARM_EVERY_HALF_SECOND,
  RTCC_ALARM_EVERY_SECOND,
  RTCC_ALARM_EVERY_10_SECONDS,
  RTCC_ALARM_EVERY_MINUTE,
  RTCC_ALARM_EVERY_10_MINUTES,
  RTCC_ALARM_EVERY_HOUR,
  RTCC_ALARM_ONCE_A_DAY,
  RTCC_ALARM_ONCE_A_WEEK,
  RTCC_ALARM_ONCE_A_MONTH,
  RTCC_ALARM_ONCE_A_YEAR
} RTCC_ALARM_MASK_CONFIGURATION;
```

Members

Members	Description
RTCC_ALARM_EVERY_HALF_SECOND	Alarm every half second
RTCC_ALARM_EVERY_SECOND	Alarm every second
RTCC_ALARM_EVERY_10_SECONDS	Alarm every 10 seconds
RTCC_ALARM_EVERY_MINUTE	Alarm every minute
RTCC_ALARM_EVERY_10_MINUTES	Alarm every 10 minutes
RTCC_ALARM_EVERY_HOUR	Alarm every hour
RTCC_ALARM_ONCE_A_DAY	Alarm every day
RTCC_ALARM_ONCE_A_WEEK	Alarm every week
RTCC_ALARM_ONCE_A_MONTH	Alarm every month
RTCC_ALARM_ONCE_A_YEAR	Alarm every year

Description

Alarm Mask Configuration

This data type defines the different configurations for accessing the alarm mask Configuration bits.

Remarks

The actual definition of this enumeration is device-specific.

RTCC_MODULE_ID Enumeration

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_ID_1,
  RTCC_NUMBER_OF_MODULES
} RTCC_MODULE_ID;
```

Members

Members	Description
RTCC_NUMBER_OF_MODULES	The total number of modules available.

Description

Enumeration: RTCC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

RTCC_VALUE_REGISTER_POINTER Enumeration

Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_REG_YEAR,
  RTCC_REG_DAY_MONTH,
  RTCC_REG_HOURS_WEEKDAY,
  RTCC_REG_MINUTES_SECONDS
} RTCC_VALUE_REGISTER_POINTER;
```

Members

Members	Description
RTCC_REG_YEAR	The year register is being pointed
RTCC_REG_DAY_MONTH	The day and month register is being pointed
RTCC_REG_HOURS_WEEKDAY	The hours and weekday register is being pointed
RTCC_REG_MINUTES_SECONDS	The minutes and seconds register is being pointed

Description

Value Register Pointer

This data type defines the different configurations by which the RTCC Date and Time Registers can be accessed.

Remarks

The actual definition of this enumeration is device-specific.

Files

Files

Name	Description
plib_rtcc.h	RTCC Peripheral Library interface header for RTCC common definitions.
plib_rtcc_help.h	This is file plib_rtcc_help.h.

Description

This section lists the source and header files used by the library.

plib_rtcc.h

RTCC Peripheral Library interface header for RTCC common definitions.

Functions

	Name	Description
	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.
	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.

	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSyncStatusGet	The function returns the synchronization status bit.
	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.
	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.
	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCDayGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCSyncStatusGet	The function returns the synchronization status bit.

	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.
	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.

Description

Real-Time Clock and Calendar (RTCC) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the RTCC Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the RTCC peripheral.

File Name

plib_rtcc.h

Company

Microchip Technology Inc.

plib_rtcc_help.h

Enumerations

	Name	Description
	RTCC_ALARM_MASK_CONFIGURATION	Data type defining the different configurations for the alarm mask bits.
	RTCC_MODULE_ID	Enumeration: RTCC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	RTCC_VALUE_REGISTER_POINTER	Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

Description

This is file plib_rtcc_help.h.

System Bus Peripheral Library

This section describes the System Bus Peripheral Library.

Introduction

This library provides a low-level abstraction of the System Bus on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The System Bus is a multi-layer crossbar switching matrix that connects initiators to targets. An initiator is a device or peripheral that initiates a data transfer to or from a target. Examples of initiators include the CPU, DMA and Ethernet controller. A target is a memory, bus or peripheral that sends or receives data to/from an initiator.

The System Bus allows the system programmer to control which initiators can access each target by defining permission groups for each initiator. Each target has one or more regions (physical address spaces) that can be individually programmed to allow access to different permission groups. By setting initiator permission groups and target region read/write permissions, the system programmer can allow or deny task or program access to system resources.

Using the Library

This topic describes the basic architecture of the System Bus Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_sb.h`

The interface to the System Bus Peripheral library is defined in the `plib_sb.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the System Bus Peripheral library must include `peripheral.h`.

Library File:

The System Bus Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

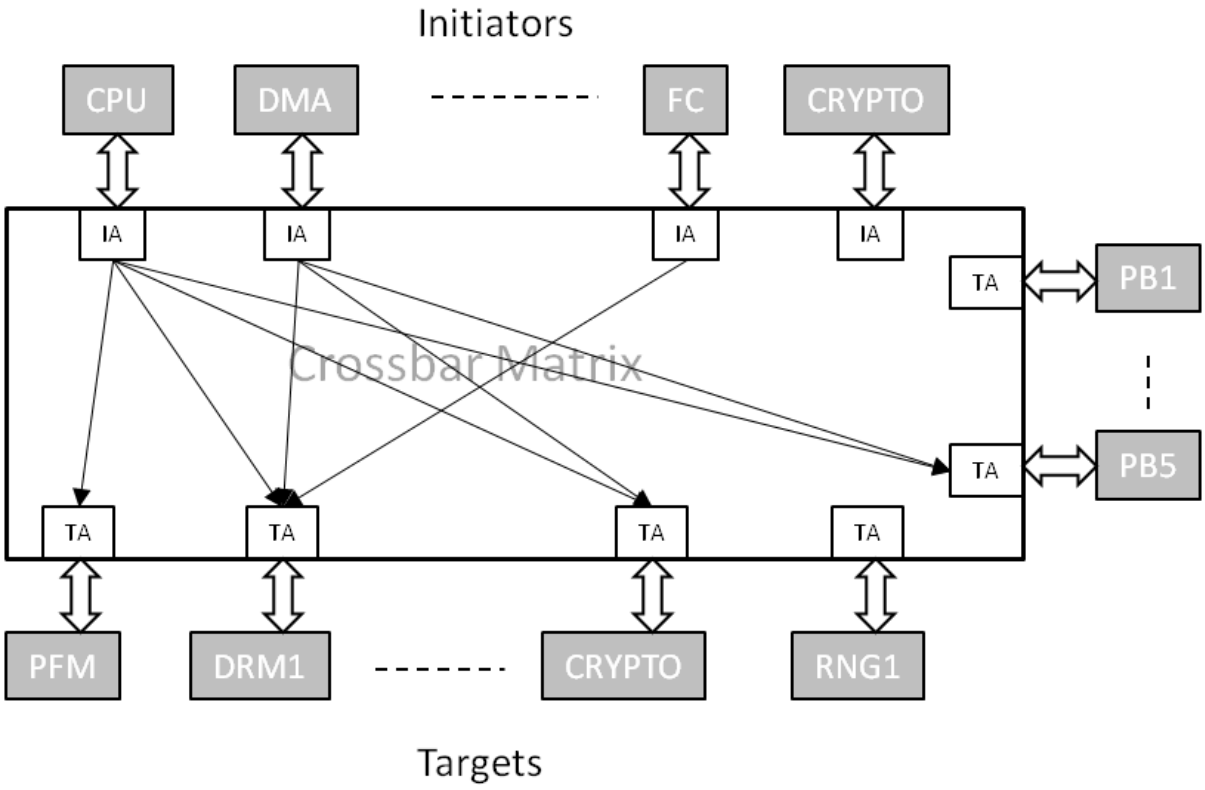
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the System Bus on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

System Bus Software Abstraction Block Diagram



Hardware Abstraction Model

The System Bus Peripheral Library provides interface routines to initialize the bus matrix such that:

- Initiator permission groups are assigned
- Target regions are defined and given read/write access permissions
- Errors are identified, logged and cleared when a permission group violation occurs
- CPU and DMA priorities are set

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the System Bus module.

Library Interface Section	Description
Target Initialization Functions	This section provides target initialization functions.
PGV Error Functions	This section provides PGV error functions.
Other Functions	This section provides additional System Bus functions.
Feature Existence Functions	This section provides functions that determine whether certain features exist.

How the Library Works

The System Bus is a crossbar interconnect matrix that provides for simultaneous data transfer between the CPU, memory, and various peripherals on the PIC32 device. In addition, it provides a protection mechanism that allows trusted code to define and restrict access to memory regions and peripherals. A typical application would be a secure bootloader that prevents a third party or untrusted application from accessing regions of Flash memory that contain privileged code or information. The protection mechanism works by dividing the target physical address space into regions, each of which may be accessed by one or more initiators. Individual targets may have one or more regions. Each region can be individually programmed to allow read or write access to one or more permission groups. Each initiator is assigned to a permission group.

This library provides interface routines to initialize the System Bus and initiator permission groups. On reset or NMI, the CPU is assigned to permission group 0 (full access).

Note: Refer to the **"CPU"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for information on target region mapping and initiator permission groups for a specific device.

Initiator Initialization

Initializing the initiators for the System Bus is a two-step process:

- Set CPU and DMA priorities (if needed)
- Assign permission groups to each initiator

Priority assignments for CPU and DMA

By default, the System Bus uses a least-recently-serviced (LRS) priority arbitration scheme for most initiators, including the CPU and DMA. For applications that require low latency, the user may set the CPU and/or DMA arbitration to fixed high priority. For the CPU, fixed high priority means that the CPU gets arbitration preference while servicing an interrupt over all initiators using LRS arbitration.

Priority is set using the PLIB_SB_CPUPrioritySet and PLIB_SB_DMAPrioritySet library functions.

Initiator permission groups

At reset, all initiators default to permission group 0. Each target region can be individually programmed to allow read or write access to any or all permission groups. By assigning permission groups to initiators and restricting target region read/write access based on permission group, it is possible to restrict initiator access to target regions during normal operations.

Permission groups are assigned using the library function PLIB_SB_InitPermGrpSet.

Target Initialization

Each System Bus target resides in a physical address space that contains one or more regions. Regions are used to provide separate access permissions for different areas within a target address space. Region 0 for any target is defined as the entire address space for that target. At reset, region 0 for all targets is accessible by any initiator with group 0 permissions.

Target regions are defined by base (physical) address and size. Read and write permissions are programmed individually for each region. The number of regions for each target varies, depending on the target. Some regions have fixed addresses, sizes and/or permissions. For example, region 0 for all targets encompasses the entire address space for the target, so the address and size are preprogrammed. Similarly, write permission for PFM is disabled for all initiators except the Flash controller.

Target region initialization is a four step process:

1. Set the base address of a region using PLIB_SB_PGRegionAddrSet. This function takes an enumerated value for the target region. Only valid target regions should be enumerated for any specific device.
2. Set the size of the region using PLIB_SB_PGRegionSizeSet.
3. Set the read permissions using PLIB_SB_PGRegionReadPermSet.
4. Set the write permissions using PLIB_SB_PGRegionWritePermSet.

Note: Please refer to the **"Memory Organization"** chapter in the specific device data sheet or the family reference manual section specified that chapter for information on target region mapping and initiator permission groups.

PGV Error Handling

A permission group violation (PGV) occurs when an initiator attempts to access a target without the proper permissions. Any permission group violation will trigger an interrupt to the CPU. If enabled, the System Bus will report a PGV to the error log registers in the System Bus. The PGV interrupt handler can then use this library to retrieve information about the violation and act accordingly.

By default, error reporting is disabled in the System Bus and must be enabled during the start-up or boot code. Errors are divided into two categories: primary and secondary. Primary errors are errors arising from secure accesses, such as trying to access a target region without the proper permissions. Secondary errors are errors arising from non-secure accesses, such as trying to access a memory location outside of a target region.

PIC32 devices support logging of primary errors only. Primary error logging is enabled using the library function PLIB_SB_PGVErrorReportPrimaryEnable. Primary error reporting can be disabled using the library function PLIB_SB_PGVErrorReportPrimaryDisable. Error logging must be enabled or disabled separately for each target.

Once a permission group violation error is logged, the user can use this library to retrieve information from the System Bus error log registers. Each target has its own set of error log registers. The error information available and the library API used to retrieve it are shown in the following table.

Error Log Data	System Bus Peripheral Library Function
Has the specified target reported an error?	PLIB_SB_PGVErrorStatus
Has the specified target reported more than one error since last cleared?	PLIB_SB_PGVErrorMulti
Type of error.	PLIB_SB_PGVErrorCode
Initiator ID.	PLIB_SB_PGVErrorInitiatorID
OCP command code of the transaction.	PLIB_SB_PGVErrorCommandCode

The region number of the specified target accessed during the violation.	PLIB_SB_PGVErrorRegion
Permission group of initiator that caused the violation.	PLIB_SB_PGVErrorPermissionGroup

Once an error is logged, it needs to be cleared by software before any data for a subsequent violation can be logged (for the same target). This is typically done in the exception handler that reads the data. Which function is used depends on whether a single or multiple error violation occurred. Single violations are cleared with PLIB_SB_PGVErrorClearSingle, and multiple violations are cleared with PLIB_SB_PGVErrorClearMulti.

Configuring the Library

The library is configured for the supported System Bus module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Target Initialization Functions

	Name	Description
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

b) PGV Error Functions

	Name	Description
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.
	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.
	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.
	PLIB_SB_PGVErrorMulti	Indicates if more than one permission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.
	PLIB_SB_PGVErrGroup0Status	Identifies whether a permission group violation has been reported for the Target Group 0.
	PLIB_SB_PGVErrGroup1Status	Identifies whether a permission group violation has been reported for the Target Group 1.

	PLIB_SB_PGVErrGroup2Status	Identifies whether a permission group violation has been reported for the Target Group 2.
	PLIB_SB_PGVErrGroup3Status	Identifies whether a permission group violation has been reported for the Target Group 3.
	PLIB_SB_PGVErrGroupStatus	Identifies whether a permission group violation has been reported for the specified target group.

c) Other Functions

	Name	Description
	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
	PLIB_SB_DMAPrioritySet	Sets the DMA arbitration policy
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

d) Feature Existence Functions

	Name	Description
	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the System Bus module.
	PLIB_SB_ExistsDMAPriority	Identifies whether the DMAPriority feature exists on the System Bus module.
	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrGroup0Status	Identifies whether the PGVErrGroup0Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup1Status	Identifies whether the PGVErrGroup1Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup2Status	Identifies whether the PGVErrGroup2Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup3Status	Identifies whether the PGVErrGroup3Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroupStatus	Identifies whether the PGVErrGroupStatus feature exists on the SB module

e) Data Types and Constants

	Name	Description
	PLIB_SB_ARB_POLICY	This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.
	PLIB_SB_ERROR	Lists the possible System Bus Transaction Error Codes.
	PLIB_SB_INIT_ID	Lists the possible System Bus Initiator IDs.
	PLIB_SB_INIT_PG	Lists the possible Initiator permission groups
	PLIB_SB_OCP_CMD_CODE	Lists the possible OCP Command codes.
	PLIB_SB_PG_INITIATOR	Lists the possible permission group Initiators.
	PLIB_SB_REGION_PG	This enumeration lists the possible permission groups assigned to a region for read and/or write access.
	PLIB_SB_TGT_ID	Lists the possible System Bus Target IDs.
	PLIB_SB_TGT_REGION	Lists the programmable target regions.
	SB_MODULE_ID	Lists the possible Module IDs for the System Bus.

Description

This section describes the Application Programming Interface (API) functions of the System Bus Peripheral Library.

Refer to each section for a detailed description.

a) Target Initialization Functions

PLIB_SB_PGRegionAddrGet Function

Returns the base address for a permission group region within a target's physical address space.

File

plib_sb.h

C

```
uint32_t PLIB_SB_PGRegionAddrGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region);
```

Returns

uint32_t - The base address of the region.

Description

This function returns the base address for a permission group region within a target's physical address space.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
uint32_t T1R6BaseAddress;
T1R6BaseAddress = PLIB_SB_PGRegionAddrGet(SB_ID_1, PLIB_T1_REGION_6);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the address is returned.

Function

uint32_t PLIB_SB_PGRegionAddrGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region)

PLIB_SB_PGRegionAddrSet Function

Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionAddrSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t phys_addr);
```

Returns

None.

Description

This function sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets have a defined start address which may not be changed. Some regions (such as those containing peripheral SFRs) are fixed and may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

```
// Set up two regions in PFM with full read and write permission
#define REGION_5_BASE_ADDR  0x1D000000
#define REGION_6_BASE_ADDR  (REGION5_BASE_ADDR + (32*1024))
#define REGION_5_SIZE       0x06 // 32KB
#define REGION_6_SIZE       0x05 // 16KB
#define FULL_PERM           (REGION_PG_0 | REGION_PG_1 | REGION_PG_2 | REGION_PG_3)

PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_BASE_ADDR);
PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_BASE_ADDR);

PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_SIZE);
PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_SIZE);

PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);

PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the base address is set.
phys_addr	The base address of the region. Must be aligned to the intended size of the region.

Function

void PLIB_SB_PGRegionAddrSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,

uint32_t phys_addr)

PLIB_SB_PGRegionReadPermClear Function

Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionReadPermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG
readPerm);
```

Returns

None.

Description

This function clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.

region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionReadPermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG readPerm)

PLIB_SB_PGRegionReadPermSet Function

Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionReadPermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG readPerm);
```

Returns

None.

Description

This function sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionReadPermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG readPerm)

PLIB_SB_PGRegionSizeGet Function

Returns the size for a permission group region within a target's physical address space.

File

plib_sb.h

C

```
uint32_t PLIB_SB_PGRegionSizeGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region);
```

Returns

The size of the region.

Description

This function returns the size for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

```
uint32_t T1R6Size;
T1R6Size = PLIB_SB_PGRegionSizeGet(SB_ID_1, PLIB_T1_REGION_6);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.

Function

uint32_t PLIB_SB_PGRegionSizeGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region)

PLIB_SB_PGRegionSizeSet Function

Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionSizeSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t size);
```

Returns

None.

Description

This function sets the size for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.
size	The actual size of the region being programmed is calculated as
follows	2**(size-1)*1024 bytes

Function

void PLIB_SB_PGRegionSizeSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t size)

PLIB_SB_PGRegionWritePermClear Function

Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionWritePermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG
writePerm);
```

Returns

None.

Description

This function clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionWritePermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG writePerm)

PLIB_SB_PGRegionWritePermSet Function

Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionWritePermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG
writePerm);
```

Returns

None.

Description

This function sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionWritePermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG writePerm)

b) PGV Error Functions

PLIB_SB_PGVErrorClearMulti Function

Clears multiple permission group errors for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

mult - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Description

This function clears multiple permission group errors for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool mult;
mult = PLIB_SB_PGVErrorClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

bool PLIB_SB_PGVErrorClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorClearSingle Function

Clears a single permission group error for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

sing - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Description

This function clears a single permission group error for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sing;
sing = PLIB_SB_PGVErrorClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

bool PLIB_SB_PGVErrorClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorCode Function

Returns a value corresponding to the type of error logged for the specified target.

File

plib_sb.h

C

```
PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

PLIB_SB_ERROR enumeration representing the type of SB error logged states.

Description

This function returns a value corresponding to the type of error logged for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_ERROR error;
error = PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Function

PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorCommandCode Function

Returns the OCP command code of the transaction that caused the protection violation for the specified target.

File

plib_sb.h

C

```
PLIB_SB_OCP_CMD_CODE PLIB_SB_PGVErrorCommandCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

OCP command code

Description

This function returns the OCP command code of the transaction that caused the protection violation for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_OCP_CMD_CODE commandCode;
commandCode = PLIB_SB_PGVErrorCommandCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

PLIB_SB_OCP_CMD_CODE PLIB_SB_PGVErrorCommandCode(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorInitiatorID Function

Returns the ID of the Initiator that caused the protection violation

File

plib_sb.h

C

```
PLIB_SB_INIT_ID PLIB_SB_PGVErrorInitiatorID(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

An enumerated value representing the ID of the initiator that caused the protection violation.

Description

This function returns the ID of the Initiator that caused the protection violation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_INIT_ID id;
id = PLIB_SB_PGVErrorInitiatorID(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

PLIB_SB_INIT_ID PLIB_SB_PGVErrorInitiatorID(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorLogClearMulti Function

Clears a multiple protection group violations error from the specified target error log register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorLogClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function clears a multiple protection group violations error from the specified target error log register. Multiple errors are cleared by writing a '1' to both the MULTI and CODE fields of the SBTxELOG1 register for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorLogClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Function

void PLIB_SB_PGVErrorLogClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorLogClearSingle Function

Clears a single protection group violation error from the specified target error log register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorLogClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function clears a single protection group violation error from the specified target error log register. Single errors are cleared by writing a '1' to the CODE field of the SBTxELOG1 register for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorLogClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Function

void PLIB_SB_PGVErrorLogClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorMulti Function

Indicates if more than one permission group violation has occurred since last cleared.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

- true - Multiple permission group violations.
- false - Single or no permission group violations.

Description

This function indicates if more than one permission group violation has occurred since last cleared.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool multiPGV;
multiPGV = PLIB_SB_PGVErrorMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Function

bool PLIB_SB_PGVErrorMulti(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorPermissionGroup Function

Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.

File

plib_sb.h

C

```
int PLIB_SB_PGVErrorPermissionGroup(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
int pg;
pg = PLIB_SB_PGVErrorPermissionGroup(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

int PLIB_SB_PGVErrorPermissionGroup(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorRegion Function

Returns the number of the protection region in the specified target address space that caused the protection violation.

File

plib_sb.h

C

```
int PLIB_SB_PGVErrorRegion(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

Region number in target address space or -1 on unrecognized target.

Description

This function returns the number of the protection region in the specified target address space that caused the protection violation. Note that if there are no other region matches, region 0 (the default region that spans the entire target address space) will always match, and this function will return 0.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
int region;
```

```
region = PLIB_SB_PGVErrorRegion(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

int PLIB_SB_PGVErrorRegion(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorReportPrimaryDisable Function

Disables primary permission group error reporting for the specified target to the SB flag register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorReportPrimaryDisable(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function disables primary permission group error reporting for the specified target to the SB flag register.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorReportPrimaryDisable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

void PLIB_SB_PGVErrorReportPrimaryDisable(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorReportPrimaryEnable Function

Enables primary permission group error reporting for the specified target to the SB flag register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorReportPrimaryEnable(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function enables primary permission group error reporting for the specified target to the SB flag register.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorReportPrimaryEnable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

void PLIB_SB_PGVErrorReportPrimaryEnable(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorStatus Function

Identifies whether a permission group violation has been reported for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorStatus(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrorStatus(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	Target to be checked.

Function

bool PLIB_SB_PGVErrorStatus (SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrGroup0Status Function

Identifies whether a permission group violation has been reported for the Target Group 0.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup0Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP0_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 0.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup0Status(SB_ID_0, PLIB_SB_PGV_GROUP0_T1_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup0Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP0_TGT targetId);

PLIB_SB_PGVErrGroup1Status Function

Identifies whether a permission group violation has been reported for the Target Group 1.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup1Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP1_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 1.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup1Status(SB_ID_0, PLIB_SB_PGV_GROUP1_T11_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup1Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP1_TGT targetId);

PLIB_SB_PGVErrGroup2Status Function

Identifies whether a permission group violation has been reported for the Target Group 2.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup2Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP2_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 2.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup2Status(SB_ID_0, PLIB_SB_PGV_GROUP2_T14_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup2Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP2_TGT targetId);

PLIB_SB_PGVErrGroup3Status Function

Identifies whether a permission group violation has been reported for the Target Group 3.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup3Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP3_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 3.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup3Status(SB_ID_0, PLIB_SB_PGV_GROUP3_T16_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup3Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP3_TGT targetId);

PLIB_SB_PGVErrGroupStatus Function

Identifies whether a permission group violation has been reported for the specified target group.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroupStatus(SB_MODULE_ID index, PLIB_SB_PGV_GROUP_ID groupId);
```

Returns

- true - Target group is reporting a permission group violation.
- false - Target group is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for the specified target group.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroupStatus(SB_ID_0, PLIB_SB_PGV_GROUP0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
groupId	Target group to be checked.

Function

bool PLIB_SB_PGVErrGroupStatus(SB_MODULE_ID index, PLIB_SB_PGV_GROUP_ID groupId);

c) Other Functions

PLIB_SB_CPUPrioritySet Function

Sets the CPU arbitration policy to SRAM when servicing an interrupt

File

plib_sb.h

C

```
void PLIB_SB_CPUPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority);
```

Returns

None.

Description

This function sets the CPU arbitration policy to SRAM when servicing an interrupt.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Preconditions

System must be unlocked before the priority can be set.

Example

```
PLIB_SB_PriorityUnlock();
PLIB_SB_CPUPrioritySet(SB_ID_1, PRIORITY_HI);
```

Parameters

Parameters	Description
priority	Use either high priority or least-recently-serviced algorithm.

Function

void PLIB_SB_CPUPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority)

PLIB_SB_DMAPrioritySet Function

Sets the DMA arbitration policy

File

plib_sb.h

C

```
void PLIB_SB_DMAPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority);
```

Returns

None.

Description

This function sets the DMA arbitration.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Preconditions

System must be unlocked before the priority can be set.

Example

```
PLIB_SB_PriorityUnlock();
void PLIB_SB_DMAPrioritySet(SB_ID_1, PRIORITY_HI);
```

Parameters

Parameters	Description
priority	Use either high priority or least-recently-serviced algorithm.

Function

void PLIB_SB_DMAPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority)

PLIB_SB_InitPermGrpSet Function

Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

File

plib_sb.h

C

```
void PLIB_SB_InitPermGrpSet(SB_MODULE_ID index, PLIB_SB_PG_INITIATOR initiator, PLIB_SB_INIT_PG pg);
```

Returns

None.

Description

This function sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGPG, which is not part of the SB. Permission groups should be assigned by the boot code prior to programming the SB.

Default permission group at reset for all initiators is 0.

After an NMI, the CPU permission group reverts to 0. All other initiator permission groups remain unchanged.

The effective CPU permission group value in debug mode is controlled by boot configuration value DBGPER[2:0]. If DBGPER denies access to the group CPU1PG selects, the effective value selects group 3.

Preconditions

None.

Example

```
PLIB_SB_InitPermGrpSet(SB_ID_1, PLIB_SB_PG_INITIATOR_CPU, PLIB_SB_INIT_PG_1);
```

Parameters

Parameters	Description
initiator	The initiator for which permission groups are assigned.
pg	The permission group(s) to which the initiator is assigned.

Function

void PLIB_SB_InitPermGrpSet(SB_MODULE_ID index, PLIB_SB_PG_INITIATOR initiator, PLIB_SB_INIT_PG pg)

d) Feature Existence Functions

PLIB_SB_ExistsCPUPriority Function

Identifies whether the CPUPriority feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsCPUPriority(SB_MODULE_ID index);
```

Returns

- true - The CPUPriority feature is supported on the device
- false - The CPUPriority feature is not supported on the device

Description

This function identifies whether the CPUPriority feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_CPUPrioritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsCPUPriority(SB_MODULE_ID index)

PLIB_SB_ExistsDMAPriority Function

Identifies whether the DMAPriority feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsDMAPriority(SB_MODULE_ID index);
```

Returns

- true - The DMAPriority feature is supported on the device
- false - The DMAPriority feature is not supported on the device

Description

This function identifies whether the DMAPriority feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_DMAPrioritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsDMAPriority(SB_MODULE_ID index)

PLIB_SB_ExistsInitPermGrp Function

Identifies whether the InitPermGrp feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsInitPermGrp(SB_MODULE_ID index);
```

Returns

- true - The InitPermGrp feature is supported on the device
- false - The InitPermGrp feature is not supported on the device

Description

This function identifies whether the InitPermGrp feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_InitPermGrpSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsInitPermGrp(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegAddr Function

Identifies whether the PGRegAddr feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegAddr(SB_MODULE_ID index);
```

Returns

- true - The PGRegAddr feature is supported on the device
- false - The PGRegAddr feature is not supported on the device

Description

This function identifies whether the PGRegAddr feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionAddrSet
- PLIB_SB_PGRegionAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegAddr(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegRdPerm Function

Identifies whether the PGRegRdPerm feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegRdPerm(SB_MODULE_ID index);
```

Returns

- true - The PGRegRdPerm feature is supported on the device
- false - The PGRegRdPerm feature is not supported on the device

Description

This function identifies whether the PGRegRdPerm feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionReadPermSet
- PLIB_SB_PGRegionReadPermClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegRdPerm(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegSize Function

Identifies whether the PGRegSize feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegSize(SB_MODULE_ID index);
```

Returns

- true - The PGRegSize feature is supported on the device
- false - The PGRegSize feature is not supported on the device

Description

This function identifies whether the PGRegSize feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionSizeSet
- PLIB_SB_PGRegionSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegSize(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegWrPerm Function

Identifies whether the PGRegWrPerm feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegWrPerm(SB_MODULE_ID index);
```

Returns

- true - The PGRegWrPerm feature is supported on the device
- false - The PGRegWrPerm feature is not supported on the device

Description

This function identifies whether the PGRegWrPerm feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionWritePermSet
- PLIB_SB_PGRegionWritePermClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegWrPerm(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClear Function

Identifies whether the PGVErrClear feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClear(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClear feature is supported on the device
- false - The PGVErrClear feature is not supported on the device

Description

This function identifies whether the PGVErrClear feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorMulti
- PLIB_SB_PGVErrorCode
- PLIB_SB_PGVErrorLogClearSingle
- PLIB_SB_PGVErrorLogClearMulti

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClear(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClrMulti Function

Identifies whether the PGVErrClrMulti feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClrMulti(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClrMulti feature is supported on the device
- false - The PGVErrClrMulti feature is not supported on the device

Description

This function identifies whether the PGVErrClrMulti feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorClearMulti

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClrMulti(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClrSingle Function

Identifies whether the PGVErrClrSingle feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClrSingle(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClrSingle feature is supported on the device
- false - The PGVErrClrSingle feature is not supported on the device

Description

This function identifies whether the PGVErrClrSingle feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorClearSingle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClrSingle(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrCmdCode Function

Identifies whether the PGVErrCmdCode feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrCmdCode(SB_MODULE_ID index);
```

Returns

- true - The PGVErrCmdCode feature is supported on the device
- false - The PGVErrCmdCode feature is not supported on the device

Description

This function identifies whether the PGVErrCmdCode feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorCommandCode

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrCmdCode(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrInitID Function

Identifies whether the PGVErrInitID feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrInitID(SB_MODULE_ID index);
```

Returns

- true - The PGVErrInitID feature is supported on the device
- false - The PGVErrInitID feature is not supported on the device

Description

This function identifies whether the PGVErrInitID feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorInitiatorID

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrInitID(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrPG Function

Identifies whether the PGVErrPG feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrPG(SB_MODULE_ID index);
```

Returns

- true - The PGVErrPG feature is supported on the device
- false - The PGVErrPG feature is not supported on the device

Description

This function identifies whether the PGVErrPG feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorPermissionGroup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrPG(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrRegion Function

Identifies whether the PGVErrRegion feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrRegion(SB_MODULE_ID index);
```

Returns

- true - The PGVErrRegion feature is supported on the device
- false - The PGVErrRegion feature is not supported on the device

Description

This function identifies whether the PGVErrRegion feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrRegion(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrRptPri Function

Identifies whether the PGVErrRptPri feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrRptPri(SB_MODULE_ID index);
```

Returns

- true - The PGVErrRptPri feature is supported on the device
- false - The PGVErrRptPri feature is not supported on the device

Description

This function identifies whether the PGVErrRptPri feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorReportPrimaryEnable
- PLIB_SB_PGVErrorReportPrimaryDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrRptPri(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrStatus Function

Identifies whether the PGVErrStatus feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrStatus(SB_MODULE_ID index);
```

Returns

- true - The PGVErrStatus feature is supported on the device
- false - The PGVErrStatus feature is not supported on the device

Description

This function identifies whether the PGVErrStatus feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrStatus(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup0Status Function

Identifies whether the PGVErrGroup0Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup0Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup0Status feature is supported on the device
- false - The PGVErrGroup0Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup0Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup0Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup0Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup1Status Function

Identifies whether the PGVErrGroup1Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup1Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup1Status feature is supported on the device
- false - The PGVErrGroup1Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup1Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup1Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup1Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup2Status Function

Identifies whether the PGVErrGroup2Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup2Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup2Status feature is supported on the device
- false - The PGVErrGroup2Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup2Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup2Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup2Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup3Status Function

Identifies whether the PGVErrGroup3Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup3Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup3Status feature is supported on the device
- false - The PGVErrGroup3Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup3Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup3Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup3Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroupStatus Function

Identifies whether the PGVErrGroupStatus feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroupStatus(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroupStatus feature is supported on the device
- false - The PGVErrGroupStatus feature is not supported on the device

Description

This function identifies whether the PGVErrGroupStatus feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroupStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroupStatus(SB_MODULE_ID index)

e) Data Types and Constants

PLIB_SB_ARB_POLICY Enumeration

This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.

File

help_plib_sb.h

C

```
typedef enum {
  PRIORITY_LRS,
  PRIORITY_HI
} PLIB_SB_ARB_POLICY;
```

Members

Members	Description
PRIORITY_LRS	Least Recently Serviced Arbitration
PRIORITY_HI	High Priority

Description

System Bus Arbitration Policy

This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.

PLIB_SB_ERROR Enumeration

Lists the possible System Bus Transaction Error Codes.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_ERROR_NONE,
  PLIB_SB_ERROR_PGV
} PLIB_SB_ERROR;
```

Members

Members	Description
PLIB_SB_ERROR_NONE	No Error
PLIB_SB_ERROR_PGV	Permission Group Violation

Description

PG Error Code

This enumeration lists the possible transaction error codes for the System Bus.

PLIB_SB_INIT_ID Enumeration

Lists the possible System Bus Initiator IDs.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_INIT_ID_NONE,
  PLIB_SB_INIT_ID_CPU_LRS,
  PLIB_SB_INIT_ID_CPU_HI,
  PLIB_SB_INIT_ID_DMA1_RD_LRS,
  PLIB_SB_INIT_ID_DMA1_RD_HI,
```

```
  PLIB_SB_INIT_ID_DMA1_WR_LRS,
  PLIB_SB_INIT_ID_DMA1_WR_HI,
  PLIB_SB_INIT_ID_USB1,
  PLIB_SB_INIT_ID_ETH1_RD,
  PLIB_SB_INIT_ID_ETH1_WR,
  PLIB_SB_INIT_ID_CAN1,
  PLIB_SB_INIT_ID_CAN2,
  PLIB_SB_INIT_ID_SQI1,
  PLIB_SB_INIT_ID_FLASH_CTL,
  PLIB_SB_INIT_ID_CRYPTO
} PLIB_SB_INIT_ID;
```

Description

System Bus Initiator ID

This enumeration lists the possible System Bus Initiator IDs. This ID is used for self-reporting and error logging. LRS and HI are System Bus arbitration schemes set by the soft configuration register CFGCON.

PLIB_SB_INIT_PG Enumeration

Lists the possible Initiator permission groups

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_INIT_PG_0,
  PLIB_SB_INIT_PG_1,
  PLIB_SB_INIT_PG_2,
  PLIB_SB_INIT_PG_3
} PLIB_SB_INIT_PG;
```

Members

Members	Description
PLIB_SB_INIT_PG_0	Privilege Group 0
PLIB_SB_INIT_PG_1	Privilege Group 1
PLIB_SB_INIT_PG_2	Privilege Group 2
PLIB_SB_INIT_PG_3	Privilege Group 3

Description

System Bus Initiator Permission Groups

This enumeration lists the possible initiator permission groups for the System Bus.

Remarks

These values are used to program the CFGPG soft configuration register, which is not part of the System Bus. This should be done by the boot code to set up the desired initiator permissions prior to programming the System Bus.

PLIB_SB_OCP_CMD_CODE Enumeration

Lists the possible OCP Command codes.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_OCP_CMD_IDLE,
  PLIB_SB_OCP_CMD_WRITE,
  PLIB_SB_OCP_CMD_READ,
  PLIB_SB_OCP_CMD_READEX,
  PLIB_SB_OCP_CMD_WRITE_NON_POST,
  PLIB_SB_OCP_CMD_BROADCAST
} PLIB_SB_OCP_CMD_CODE;
```

Description

OCP Command Codes

This enumeration lists the possible OCP command codes. An OCP command code is logged when a transaction violation occurs. The command code of the offending command can then be read.

PLIB_SB_PG_INITIATOR Enumeration

Lists the possible permission group Initiators.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_PG_INITIATOR_CPU,
  PLIB_SB_PG_INITIATOR_DMA1,
  PLIB_SB_PG_INITIATOR_USB1,
  PLIB_SB_PG_INITIATOR_CAN1,
  PLIB_SB_PG_INITIATOR_CAN2,
  PLIB_SB_PG_INITIATOR_ETH1,
  PLIB_SB_PG_INITIATOR_SQI1,
  PLIB_SB_PG_INITIATOR_FLASH_CTL,
  PLIB_SB_PG_INITIATOR_CRYPTO
} PLIB_SB_PG_INITIATOR;
```

Description

System Bus Initiators

This enumeration lists the possible permission group Initiators.

PLIB_SB_REGION_PG Enumeration

This enumeration lists the possible permission groups assigned to a region for read and/or write access.

File

help_plib_sb.h

C

```
typedef enum {
  REGION_PG_0,
  REGION_PG_1,
  REGION_PG_2,
  REGION_PG_3
} PLIB_SB_REGION_PG;
```

Members

Members	Description
REGION_PG_0	Privilege Group 0 has read/write permission
REGION_PG_1	Privilege Group 1 has read/write permission
REGION_PG_2	Privilege Group 2 has read/write permission
REGION_PG_3	Privilege Group 3 has read/write permission

Description

System Bus Region Permission Groups

Lists the possible permission groups assigned to a region for read and/or write access.

PLIB_SB_TGT_ID Enumeration

Lists the possible System Bus Target IDs.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_TGT_ID_T0,
  PLIB_SB_TGT_ID_T1,
  PLIB_SB_TGT_ID_T2,
  PLIB_SB_TGT_ID_T3,
  PLIB_SB_TGT_ID_T4,
  PLIB_SB_TGT_ID_T5,
  PLIB_SB_TGT_ID_T6,
  PLIB_SB_TGT_ID_T7,
  PLIB_SB_TGT_ID_T8,
  PLIB_SB_TGT_ID_T9,
  PLIB_SB_TGT_ID_T10,
  PLIB_SB_TGT_ID_T11,
  PLIB_SB_TGT_ID_T12,
  PLIB_SB_TGT_ID_T13
} PLIB_SB_TGT_ID;
```

Members

Members	Description
PLIB_SB_TGT_ID_T0	System Bus
PLIB_SB_TGT_ID_T1	Prefetch Module
PLIB_SB_TGT_ID_T2	Data RAM 1
PLIB_SB_TGT_ID_T3	Data RAM 2
PLIB_SB_TGT_ID_T4	External Bus Interface
PLIB_SB_TGT_ID_T5	Peripheral Bus 1
PLIB_SB_TGT_ID_T6	Peripheral Bus 2
PLIB_SB_TGT_ID_T7	Peripheral Bus 3
PLIB_SB_TGT_ID_T8	Peripheral Bus 4
PLIB_SB_TGT_ID_T9	Peripheral Bus 5
PLIB_SB_TGT_ID_T10	USB
PLIB_SB_TGT_ID_T11	Serial Quad Interface
PLIB_SB_TGT_ID_T12	Crypto
PLIB_SB_TGT_ID_T13	Random Number Generator

Description

System Bus Target ID

This enumeration lists the possible System Bus Target IDs.

PLIB_SB_TGT_REGION Enumeration

Lists the programmable target regions.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_T0_REGION_0,
  PLIB_SB_T0_REGION_1,
  PLIB_SB_T1_REGION_0,
  PLIB_SB_T1_REGION_1,
  PLIB_SB_T1_REGION_2,
  PLIB_SB_T1_REGION_3,
  PLIB_SB_T1_REGION_4,
  PLIB_SB_T1_REGION_5,
  PLIB_SB_T1_REGION_6,
  PLIB_SB_T1_REGION_7,
  PLIB_SB_T1_REGION_8,
  PLIB_SB_T2_REGION_0,
  PLIB_SB_T2_REGION_1,
  PLIB_SB_T2_REGION_2,
  PLIB_SB_T3_REGION_0,
  PLIB_SB_T3_REGION_1,
  PLIB_SB_T3_REGION_2,
```

```
    PLIB_SB_T4_REGION_0,
    PLIB_SB_T4_REGION_1,
    PLIB_SB_T4_REGION_2,
    PLIB_SB_T5_REGION_0,
    PLIB_SB_T5_REGION_1,
    PLIB_SB_T5_REGION_2,
    PLIB_SB_T6_REGION_0,
    PLIB_SB_T6_REGION_1,
    PLIB_SB_T7_REGION_0,
    PLIB_SB_T7_REGION_1,
    PLIB_SB_T8_REGION_0,
    PLIB_SB_T8_REGION_1,
    PLIB_SB_T9_REGION_0,
    PLIB_SB_T9_REGION_1,
    PLIB_SB_T10_REGION_0,
    PLIB_SB_T11_REGION_0,
    PLIB_SB_T11_REGION_1,
    PLIB_SB_T12_REGION_0,
    PLIB_SB_T13_REGION_0
} PLIB_SB_TGT_REGION;
```

Members

Members	Description
PLIB_SB_T0_REGION_0	System Bus Region 0
PLIB_SB_T0_REGION_1	System Bus Region 1
PLIB_SB_T1_REGION_0	Prefetch Module Region 0
PLIB_SB_T1_REGION_1	Prefetch Module Region 1
PLIB_SB_T1_REGION_2	Prefetch Module Region 2
PLIB_SB_T1_REGION_3	Prefetch Module Region 3
PLIB_SB_T1_REGION_4	Prefetch Module Region 4
PLIB_SB_T1_REGION_5	Prefetch Module Region 5
PLIB_SB_T1_REGION_6	Prefetch Module Region 6
PLIB_SB_T1_REGION_7	Prefetch Module Region 7
PLIB_SB_T1_REGION_8	Prefetch Module Region 8
PLIB_SB_T2_REGION_0	Data RAM 1 Region 0
PLIB_SB_T2_REGION_1	Data RAM 1 Region 1
PLIB_SB_T2_REGION_2	Data RAM 1 Region 2
PLIB_SB_T3_REGION_0	Data RAM 2 Region 0
PLIB_SB_T3_REGION_1	Data RAM 2 Region 1
PLIB_SB_T3_REGION_2	Data RAM 2 Region 2
PLIB_SB_T4_REGION_0	External Bus Interface Region 0
PLIB_SB_T4_REGION_1	External Bus Interface Region 1
PLIB_SB_T4_REGION_2	External Bus Interface Region 2
PLIB_SB_T5_REGION_0	Peripheral Bus 1 Region 0
PLIB_SB_T5_REGION_1	Peripheral Bus 1 Region 1
PLIB_SB_T5_REGION_2	Peripheral Bus 1 Region 2
PLIB_SB_T6_REGION_0	Peripheral Bus 2 Region 0
PLIB_SB_T6_REGION_1	Peripheral Bus 2 Region 1
PLIB_SB_T7_REGION_0	Peripheral Bus 3 Region 0
PLIB_SB_T7_REGION_1	Peripheral Bus 3 Region 1
PLIB_SB_T8_REGION_0	Peripheral Bus 4 Region 0
PLIB_SB_T8_REGION_1	Peripheral Bus 4 Region 1
PLIB_SB_T9_REGION_0	Peripheral Bus 5 Region 0
PLIB_SB_T9_REGION_1	Peripheral Bus 5 Region 1
PLIB_SB_T10_REGION_0	USB Region 0
PLIB_SB_T11_REGION_0	Serial Quad Interface Region 0
PLIB_SB_T11_REGION_1	Serial Quad Interface Region 1
PLIB_SB_T12_REGION_0	Crypto Region 9
PLIB_SB_T13_REGION_0	Random Number Generator 0

Description

Regions

This enumeration lists the programmable System Bus target regions.

SB_MODULE_ID Enumeration

Lists the possible Module IDs for the System Bus.

File

help_plib_sb.h

C

```
typedef enum {
  SB_ID_1,
  SB_NUMBER_OF_MODULES
} SB_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the System Bus.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

Files

Files

Name	Description
plib_sb.h	Defines the System Bus Peripheral Library interface
help_plib_sb.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_sb.h

Defines the System Bus Peripheral Library interface

Functions

	Name	Description
	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
	PLIB_SB_DMAPrioritySet	Sets the DMA arbitration policy
	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the System Bus module.
	PLIB_SB_ExistsDMAPriority	Identifies whether the DMAPriority feature exists on the System Bus module.
	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrGroup0Status	Identifies whether the PGVErrGroup0Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup1Status	Identifies whether the PGVErrGroup1Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup2Status	Identifies whether the PGVErrGroup2Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup3Status	Identifies whether the PGVErrGroup3Status feature exists on the SB module

	PLIB_SB_ExistsPGVErrGroupStatus	Identifies whether the PGVErrGroupStatus feature exists on the SB module
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the System Bus module.
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGVErrGroup0Status	Identifies whether a permission group violation has been reported for the Target Group 0.
	PLIB_SB_PGVErrGroup1Status	Identifies whether a permission group violation has been reported for the Target Group 1.
	PLIB_SB_PGVErrGroup2Status	Identifies whether a permission group violation has been reported for the Target Group 2.
	PLIB_SB_PGVErrGroup3Status	Identifies whether a permission group violation has been reported for the Target Group 3.
	PLIB_SB_PGVErrGroupStatus	Identifies whether a permission group violation has been reported for the specified target group.
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.
	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.
	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.
	PLIB_SB_PGVErrorMulti	Indicates if more than one permission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.

Description

System Bus Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the System Bus

Peripheral Library for Microchip microcontrollers. The definitions in this file are for the System Bus controller module.

File Name

plib_sb.h

Company

Microchip Technology Inc.

help_plib_sb.h

Enumerations

	Name	Description
	PLIB_SB_ARB_POLICY	This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.
	PLIB_SB_ERROR	Lists the possible System Bus Transaction Error Codes.
	PLIB_SB_INIT_ID	Lists the possible System Bus Initiator IDs.
	PLIB_SB_INIT_PG	Lists the possible Initiator permission groups
	PLIB_SB_OCP_CMD_CODE	Lists the possible OCP Command codes.
	PLIB_SB_PG_INITIATOR	Lists the possible permission group Initiators.
	PLIB_SB_REGION_PG	This enumeration lists the possible permission groups assigned to a region for read and/or write access.
	PLIB_SB_TGT_ID	Lists the possible System Bus Target IDs.
	PLIB_SB_TGT_REGION	Lists the programmable target regions.
	SB_MODULE_ID	Lists the possible Module IDs for the System Bus.

Description

This file is used for documentation purposes

SPI Peripheral Library

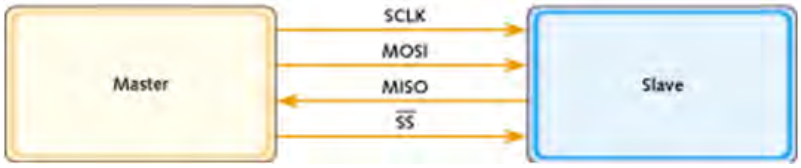
This section describes the Serial Peripheral Interface (SPI) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Serial Peripheral Interface (SPI) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with other peripherals or microcontroller device. These peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc.



SPI is a synchronous serial data link operating at full duplex Master/slave relationship.

Two data lines:

- MOSI – Master Data Output, Slave Data Input
- MISO – Master Data Input, Slave Data Output

Two control lines:

- SCLK – Clock
- /SS – Slave Select (no addressing)

Devices communicate in Master/Slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (Chip Select) lines. The SPI is sometimes referred to as a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses.

Using the Library

This topic describes the basic architecture of the SPI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_spi.h`

The interface to the SPI Peripheral Library is defined in the `plib_spi.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the SPI Peripheral library must include `peripheral.h`.

Library File:

The SPI Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

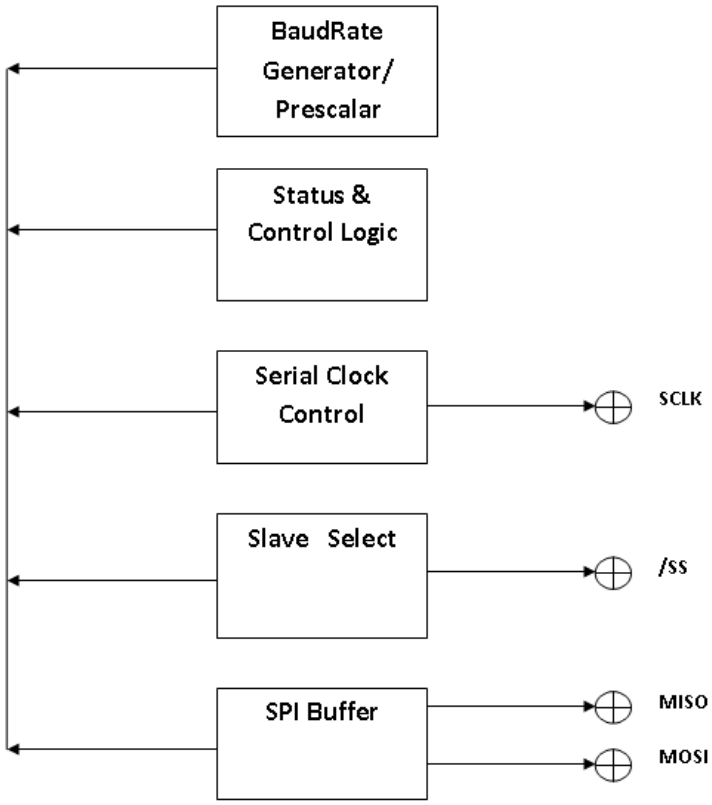
Hardware Abstraction Model

This library provides a low-level abstraction of the SPI module on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SPI Peripheral Library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model



The Serial Peripheral Interface (API) module is a synchronous serial interface useful for communicating with external peripherals and other microcontroller devices.

The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master)
- MOSI: master output, slave input (output from master)
- MISO: master input, slave output (output from slave)
- /SS: slave select (active low, output from master); on certain devices, this pin is implemented using general purpose I/O (GPIO)

The SPI bus can operate with a single master device and with one or more slave devices. To begin a communication, the master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data.

The master then transmits the appropriate chip select bit for the desired chip to a logic '0'. A logic '0' is transmitted because the Chip Select line (/SS) is active low, meaning its off state is a logic '1'; the on state is asserted with a logic 0. The master issues the clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line
- The slave sends a bit on the MISO line; the master reads it from that same line

Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

The **Baud Rate Generator/Prescaler** controls the timing, the desired baud rate can be programmed in the baud rate controller.

In the Master mode, the **clock** becomes the serial clock and is provided to external devices via the SCK pin (the clock can be prescaled by the primary prescaler and the secondary prescaler if device supports).

The **Buffers** are for data transmitted or received by the SPI module over the MISO and MOSI line synchronized with the SCK line by the clock control logic.

The **Status and Control logic**, provide the capability to control different ways of enabling or disabling the master and slave. It also can provides status about the transmitter and receiver.

Active-Low **Slave Select** or Frame Synchronization I/O Pulse allows for a Synchronous Slave mode.

The SPI module supports the following four SPI modes.

- **Standard Mode:** In this mode of operation, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive Buffer. The master provides the serial clock and synchronization signals required to the slave device.
- **Enhanced Buffer Mode:** The operation of this mode is very similar to that of Standard mode. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.
- **Framed Mode:** In this mode of operation, the Frame Master controls the generation of the frame synchronization pulse and provides this pulse to other devices at the Slave Select (/SS) pin. The SPI clock is generated by the SPI Master and is continuously running. The SPI slave module uses a frame synchronization pulse received at the SS pin.

- **Audio Protocol Mode:** The SPI module provides support to the audio protocol modes; with this mode, the SPI module can be interfaced to most codec devices available today to provide PIC microcontroller-based audio solution

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
General Configuration and Status Functions	This section provides a set of functions and data types for configuring the SPI and to read the status of the SPI.
Data Transfer Functions	This section provides a set of functions and data types for Reading and Writing the SPI buffer values.
Transmitter Functions	This section provides a set of functions for transmitter.
Receiver Functions	This section provides a set of functions for receiver.
Framed Mode Functions	Provides control, status, and data transfer routines for Framed SPI mode.
Audio Mode Functions	Provides control, status routines to support audio protocol functionality.
Feature Existence Functions	Provides functions that determine whether certain features exist on a device.

Note: Not all modes are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

How the Library Works

Provides information on how the library works.

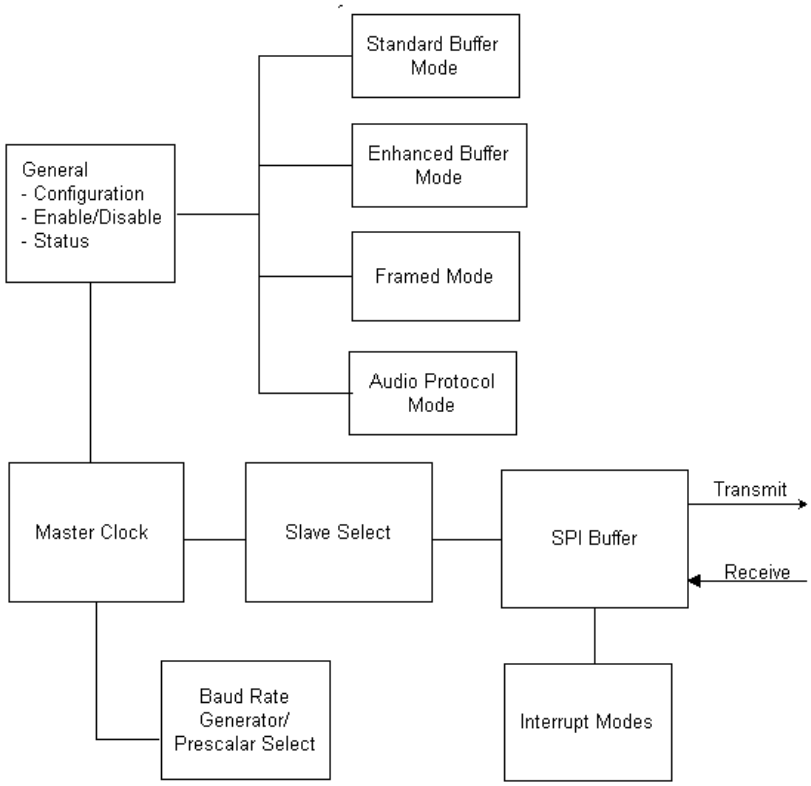
Description

The SPI module has the following operating modes:

- Standard SPI
- Enhanced Buffer SPI
- Framed SPI
- Audio Protocol Interface

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

The following diagram shows the general architecture of the SPI Peripheral Library.



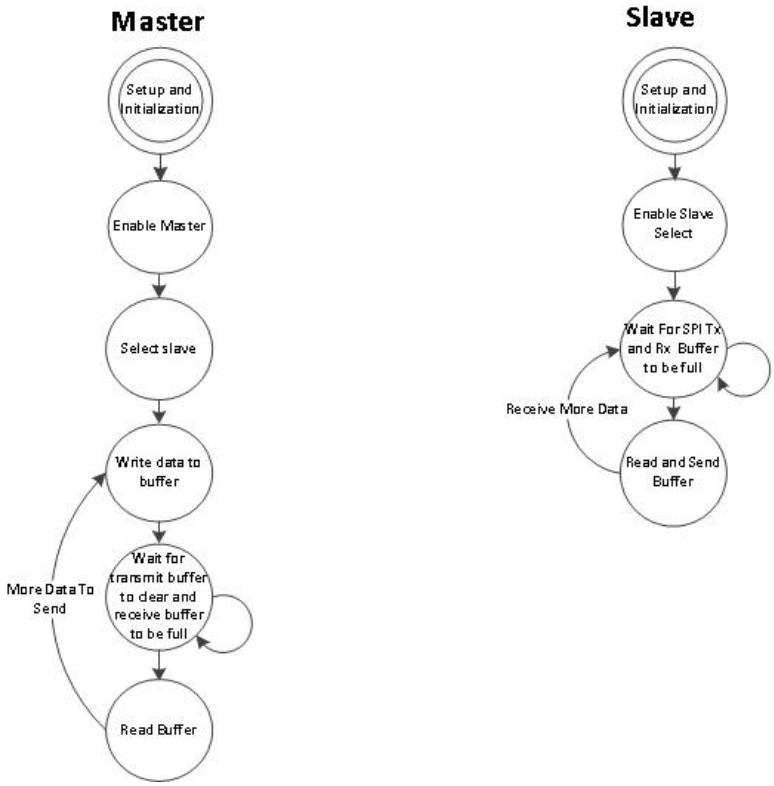
State Machine

This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

SPI State Machine

The following state machine diagrams shows the transmission and reception during normal operation.



SPI Routines

State	Associated Function
Setup and Initialization	Initialize the SPI by setting prescaler/baud rate generator, interrupt modes.
Enable Master	Once the SPI has been appropriately set up and initialized, the state machine enables Master mode (PLIB_SPI_MasterEnable).
Select Slave	Select slave by pulling the /SSx pin low to transmit the data.
Write Data To Transfer	Write data to the buffer to transmit using PLIB_SPI_BufferWrite.
Wait for Transmit Buffer to Clear	Buffer Data will be transmitted to transmit buffer and transmit buffer flag will be full. The state machine waits for transmit buffer to clear. Check for status by calling PLIB_SPI_TransmitBufferIsFull.
Wait for Receive Buffer Full to Set	Slave sends received data back to master. Upon data received receive buffer flag will set. The state machine waits for Receive buffer to clear. Check for status by calling PLIB_SPI_ReceiverBufferIsFull.
Read Buffer	Read the buffer using PLIB_SPI_BufferRead.
Enable Slave Select	Enable slave select by calling PLIB_SPI_PinEnable.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Standard SPI Mode

In Standard (legacy) Master and Slave modes, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive buffer. The module configured as the master module provides the serial clock and synchronization signals (as required) to the slave device.

Standard Master Mode

In Standard Master mode, the input clock is used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Description

Setup

1. Select on which edge of the clock data transmission occurs, using PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect.
2. Set SPI baud rate using PLIB_SPI_BaudRateSet.
3. If the module needs to operate as a Master, use PLIB_SPI_MasterEnable.

Example: Standard Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer (PLIB_SPI_BufferWrite).
2. When contents of buffer are moved to the shift register, the SPI transmit buffer full flag is cleared (this can be verified using PLIB_SPI_TransmitBufferIsFull).
3. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
4. When the transfer is complete, the following events occur:
   - The SPI interrupt flag is set. Interrupts will occur if SPI interrupts are enabled. The SPI interrupt flag is not cleared automatically by the hardware.
   - Also, when the ongoing transmit and receive operation is completed, the contents of the shift register are moved to the SPI receive register.
   - The SPI receive buffer full flag (PLIB_SPI_ReceiverBufferIsFull) is set by the module, indicating that the receive buffer is full. Once the SPI buffer is read by the user application using PLIB_SPI_BufferRead, the hardware clears the SPI receive buffer full flag.
5. If the SPI receive buffer full flag is set when the SPI module needs to transfer data from SPI shift register to SPI receive buffer, the module will set the SPI receive overflow flag (PLIB_SPI_ReceiverHasOverflowed), indicating an overflow condition.
6. Data to be transmitted can be written to SPI buffer (PLIB_SPI_BufferWrite) by the user software at any time as long as the SPI Transmit buffer full flag is clear (PLIB_SPI_TransmitBufferIsFull). The write can occur while SPI shift register is shifting out the previously written data, allowing continuous transmission.

Example: Standard Master Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt
```

```
// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Standard Slave Mode

Describes Standard Slave mode.

Description

Slave Select Synchronization

The Slave Select pin (/SS) allows for Synchronous Slave mode. If the slave select is enabled (PLIB_SPI_PinEnable), transmission and reception are enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the data output (SDO) pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low, using the data held in the SPI Transmit buffer. If slave select is not enabled, the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Status Operation

The function of the SPI Transmit buffer full (PLIB_SPI_TransmitBufferIsFull) is different in the Slave mode of operation. If slave select is disabled (PLIB_SPI_PinDisable), the PLIB_SPI_TransmitBufferIsFull returns a '1' when the SPI buffer is loaded by the user application. It is cleared when the module transfers data from SPI transmit buffer to SPI shift register. This is similar to the SPI Transmit buffer full function in Master mode.

If slave select is enabled (PLIB_SPI_PinEnable), PLIB_SPI_TransmitBufferIsFull returns a '1' when the SPI buffer is loaded by the user application. However,PLIB_SPI_TransmitBufferIsFull returns zero only when the SPI module completes data transmission. A transmission will be aborted when the Slave Select pin goes high and may be retried at a later time. Each data word is held in SPI transmit buffer until all bits are transmitted to the receiver.

Setup

In Standard Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. The clock polarity and clock edge (PLIB_SPI_ClockPolaritySelect) determine upon which edge of the clock data transmission occurs.

Example: Standard Slave Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
```

```
// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/ PLIB_SPI_BufferRead). The remainder of the operation of the module is identical to that of Standard Master mode.

Example: Standard Slave Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
     // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);                // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Enhanced Buffer SPI Mode

The operation of Enhanced Buffer Master and Slave modes is very similar to Standard Master and Slave modes. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.

Enhanced Buffer Master Mode

In Enhanced Buffer Master mode (referred to as a FIFO), the input clock used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Description

Setup

1. Select on which edge of the clock data transmission occurs, using PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect.
2. Set SPI baud rate using PLIB_SPI_BaudRateSet.
3. If the module needs to operate as a Master, use PLIB_SPI_MasterEnable.

Example: Enhanced Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);
```

```
// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

The CPU loads data to be transmitted into the transmit buffer by writing the SPI buffer (PLIB_SPI_BufferWrite). An SPI transmission begins after the first buffer write. Up to all pending transmissions can be loaded. The number of pending transfers is indicated by the Buffer Element Count bits through PLIB_SPI_FIFOCountGet.

In Master mode, this count reflects the number of transfers pending in the transmit buffer. In Slave mode, it reflects the number of unread receptions in the receive buffer. If the Shift register is empty, the first write will immediately load the Shift register, leaving all transmit buffer locations available.

After an SPI transfer completes, the receive buffer location is updated with the received data. The CPU accesses the received data by reading the SPI buffer. After each CPU read, the SPI buffer points to the next buffer location. SPI transfers continue until all pending data transfers have completed.

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer (PLIB_SPI_BufferWrite) and is loaded into the next available transmit buffer location.
2. The SPI transmit buffer full flag (PLIB_SPI_TransmitBufferIsFull) and SPI interrupt flag are set after pending transfers are loaded.
3. The current buffer location's contents are moved to the Shift register. The SPI transmit buffer is cleared by the module if a buffer location is available for a CPU write.
4. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
5. When the transfer is complete, the following events occur:
   - The contents of the SPI shift register are moved into the next available location in the receive buffer.
   - If the last unread location is written by the SPI module, the SPI receive buffer full flag (PLIB_SPI_ReceiveBufferIsFull) is set, indicating that all buffer locations are full. Enable the SPI interrupts. The SPI interrupt flag is not cleared automatically by the hardware.
   - Once the SPI buffer is read (PLIB_SPI_BufferRead) by the user application, the hardware clears the SPI receive buffer full flag (PLIB_SPI_ReceiverBufferIsFull) and the SPI Buffer increments to the next unread receive buffer location. SPI buffer reads beyond the last unread location will not increment the buffer location.
6. When PLIB_SPI_ReceiverBufferIsFull is set, if the SPI module needs to transfer one more data from SPI shift register to the buffer, the module will enable the receive overflow flag (PLIB_SPI_ReceiverHasOverflowed) indicating an overflow condition.
7. Data to be transmitted can be written to the SPI buffer (PLIB_SPI_BufferWrite) by the user application at any time as long as the SPI transmit buffer full flag is clear (PLIB_SPI_TransmitBufferIsFull). Up to all pending transfers can be loaded into the buffer allowing continuous transmission.

Example: Enhanced Master Mode communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Enhanced Buffer Slave Mode

Describes Enhanced Buffer Slave mode.

Description

Slave Select Synchronization

The Slave Select pin allows a Synchronous Slave mode. If the slave select enabled (PLIB_SPI_PinEnable), transmission and reception is enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the SDO pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low using the data held in the SPI transmit buffer. If the Slave select is disabled (PLIB_SPI_PinDisable), the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Operation

The function of the SPI transmit buffer full flag (PLIB_SPI_TransmitBufferIsFull) is different in the Slave mode of operation. If Slave select is disabled (PLIB_SPI_PinDisable), the SPI Transmit Buffer Full flag (PLIB_SPI_TransmitBufferIsFull) is set when the last available buffer location is loaded by the user application. It is cleared when the module transfers data from the buffer to SPI status register and a buffer location is available for a CPU write. This is similar to the SPI transmit buffer in Master mode.

If Slave select is enabled (PLIB_SPI_PinEnable), the SPI transmit buffer is set when the last available buffer location is loaded by the user application. However, it is cleared only when the SPI module completes data transmission, leaving a buffer location available for a CPU write. A transmission will be aborted when the /SS pin goes high and may be retried at a later time. Each data word is held in the buffer until all bits are transmitted to the receiver.

Setup

In Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect determine upon which edge of the clock data transmission occurs. The rest of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_AT_END);//clock polarity and edge is
subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional:  Enable interrupts.
```

```
// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive:

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/PLIB_SPI_BufferRead). The remainder of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);                // send back
}
```

Notes:

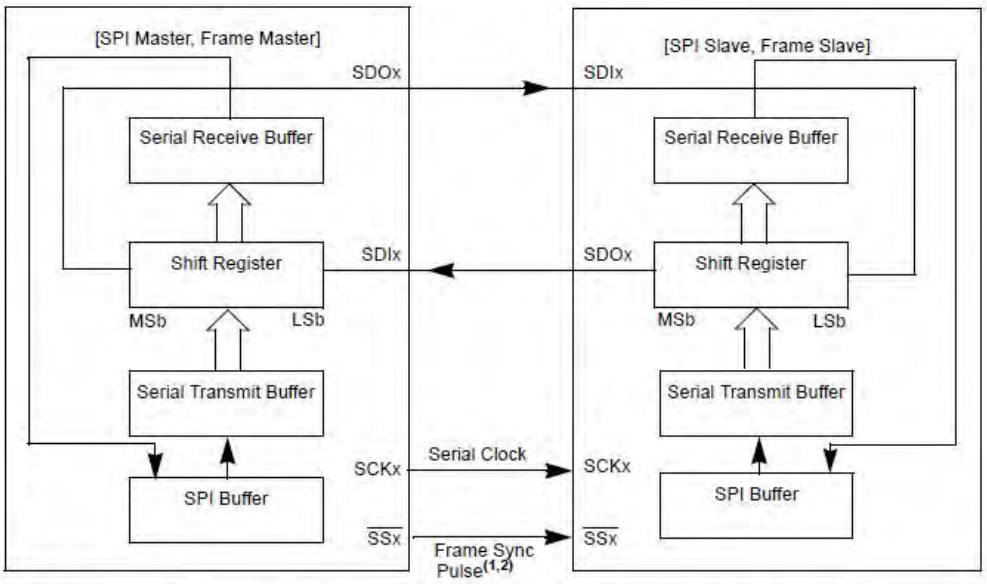
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Framed SPI Modes

The SPI module supports a basic framed SPI protocol while operating in either Master or Slave modes.

Description

The SPI module supports two framed modes of operation. In Frame Master mode, the SPI module generates the frame synchronization pulse and provides this pulse to other devices at the /SS pin. In Frame Slave mode, the SPI module uses a frame synchronization pulse received at the /SS pin.



Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

The framed SPI modes are supported in conjunction with the unframed Master and Slave modes. This makes four framed SPI configurations:

- SPI Master mode and Frame Master mode
- SPI Master mode and Frame Slave mode
- SPI Slave mode and Frame Master mode
- SPI Slave mode and Frame Slave mode

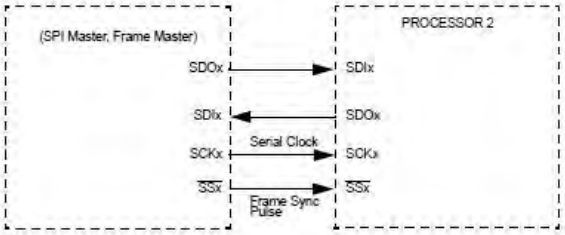
These modes determine whether or not the SPI module generates the serial clock and the frame synchronization pulse.

SPI Master Mode and Frame Master Mode

Describes Master/Frame Master mode.

Description

In Master/Frame Master mode, the SPI module generates both the clock and frame synchronization signals, as shown in the following figure.



In this mode, the serial clock is output continuously at the SCK pin, regardless of whether the module is transmitting. When the SPI buffer is written (PLIB_SPI_BufferWrite), the /SS pin will be driven to its active state on the appropriate transmit edge of the SCK clock, and remain active for one data frame. If the PLIB_SPI_FrameSyncPulseEdgeSelect function decides whether sync pulse precedes the data transmission or coincides with the beginning of the data transmission. The module starts transmitting data on the next transmit edge of the SCK.

Setup

The mode is enabled by calling PLIB_SPI_MasterEnable, PLIB_SPI_FramedCommunicationEnable and PLIB_SPI_FrameSyncPulseDirectionSelect (as output).

Example: SPI Master Mode and Frame Master Mode Communication Setup

```
#define MY_SPI_ID             SPI_ID_1
#define PERIPHERAL_BUS_CLK    80000000
#define SPI_BAUD_RATE         10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
```

```
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Master Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

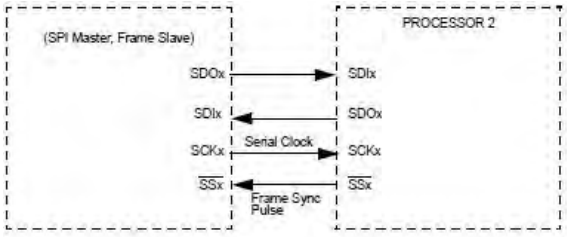
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Master Mode and Frame Slave Mode

Describes Master/Frame Slave mode.

Description

In Master/Frame Slave mode, the module generates the clock signal but uses the slave module's frame synchronization signal for data transmission, as shown in the following figure.



In this mode, the /SS pin is an input and it is sampled on the sample edge of the SPI clock. When it is sampled in its active state, data will be transmitted on the subsequent transmit edge of the SPI clock. The SPI interrupt flag is set when the transmission is complete. The user application must make sure that the correct data is loaded into the SPI buffer for transmission before the signal is received at the /SS pin.

Setup

The mode is enabled by calling PLIB_SPI_MasterEnable, PLIB_SPI_FramedCommunicationEnable, and PLIB_SPI_FrameSyncPulseDirectionSelect.

Example: SPI Master Mode and Frame Slave Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Slave Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

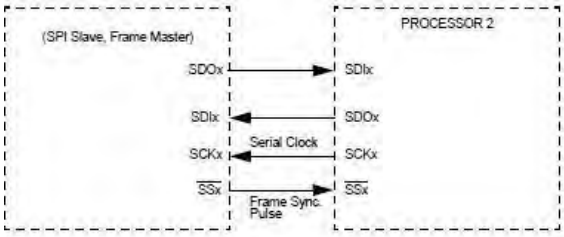
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Slave Mode and Frame Master Mode

Describes Slave/Frame Master mode.

Description

In Slave/Frame Master mode, the module acts as the SPI slave and takes its clock from the other SPI module; however, it produces frame synchronization signals to control data transmission, as shown in the following figure.



The input SPI clock will be continuous in Slave mode. The /SS pin will be an output when the Frame sync pulse is output. Therefore, when the SPI buffer is written, the module drives the /SS pin to the active state on the appropriate transmit edge of the SPI clock for one SPI clock cycle. Data will start transmitting on the appropriate SPI clock transmit edge.

Setup

The mode is enabled by calling PLIB_SPI_SlaveEnable, PLIB_SPI_FramedCommunicationEnable, and PLIB_SPI_FrameSyncPulseDirectionSelect.

Example: SPI Slave Mode and Frame Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Master Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
```

```
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data);                // send back
}
```

Notes:

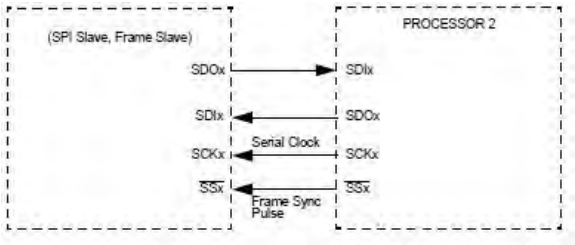
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Slave Mode and Frame Master Mode

Describes Slave/Frame Slave mode.

Description

In Slave/Frame Slave mode, the module obtains both its clock and frame synchronization signal from the master module, as shown in the following figure.



In this mode, both the clock and Slave Select pins will be inputs. The /SS pin is sampled on the sample edge of the SPI clock. When /SS is sampled at its active state, data will be transmitted on the appropriate transmit edge of SCK.

Setup

The mode is enabled by calling PLIB_SPI_SlaveEnable,PLIB_SPI_FramedCommunicationEnable and PLIB_SPI_FrameSyncPulseDirectionSelect (as input).

Example: SPI Slave Mode and Frame Slave Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
```

```
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Slave Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);               // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Audio Protocol Interface Mode

The SPI module can be interfaced to most codec devices available today to provide PIC microcontroller-based audio solutions.

Description

The SPI module provides support to the audio protocol functionality via four standard I/O pins. The four pins that make up the audio protocol interface modes are:

- SDIx: Serial Data Input for receiving sample digital audio data
- SDOx: Serial Data Output for transmitting digital audio data
- SCKx: Serial Clock (also known as bit clock)
- /SSx: Left/Right Channel Clock

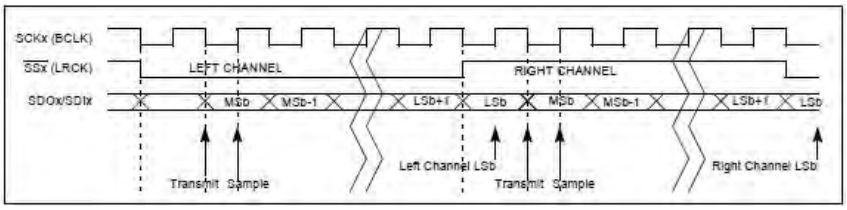
The SPI module supports four audio protocol modes and can be operated in any one of these modes:

Note: Each of the modes can additionally support some or all of the following features. Please refer to the"**Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for more information.

I2S Mode

The Inter-IC Sound (I2S) protocol enables transmission of two channels of digital audio data over a single serial interface. The I2S specification defines a half-duplex interface that supports transmit or receive, but not both at the same time. With both SDO and SDI available, full-duplex

operation is supported by this peripheral, as shown in the following figure.



Data Transmit and Clocking:

- The transmitter shifts the audio sample data's Most Significant bit (MSb) on the first falling edge of SCK after an LRCK transition
- The receiver samples the MSB on the second rising edge of SCK
- The left channel data shifts out while LRCK is low and right channel data is shifted out while LRCK is high
- The data in the left and right channel consists of a single frame

To set the module to I2S mode, the following bits must be set:

- Set SPI to I2S Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active low)
- Select PLIB_SPI_ClockPolaritySelect (as active low)

Setting these bits enables the SDO and LRCK (/SSx) transitions to occur on the falling edge of SCK (BCLK) and sampling of SDI to occur on the rising edge of SCK.

Left-Justified Mode

The Left-Justified mode is similar to I2S mode; however, in this mode, the SPI shifts the audio data's MSb on the first SCK edge that is coincident with an LRCK transition. On the receiver side, the SPI module samples the MSb on the next SCK edge.

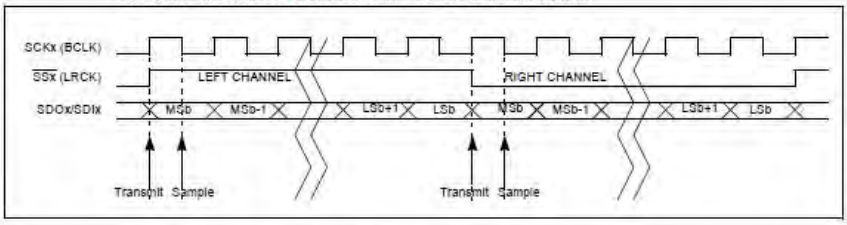
In general, a codec using justified protocols defaults to transmitting data on the rising edge of SCK and receiving data on the falling edge of SCK.

To set the module to Left-Justified mode, the following bits must to be set

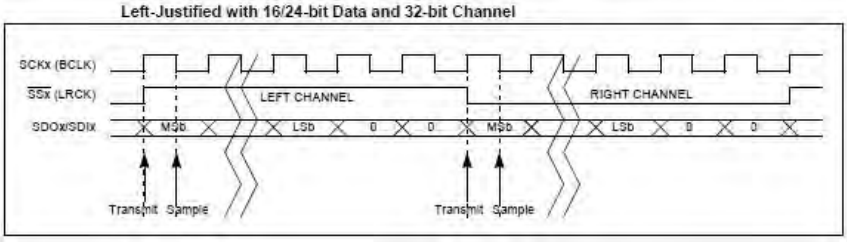
- Set SPI to Left Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active high)
- Select PLIB_SPI_ClockPolaritySelect (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.

Left-Justified with 16-bit Data/Channel or 32-bit Data/Channel



Left-Justified with 16/24-bit Data and 32-bit Channel



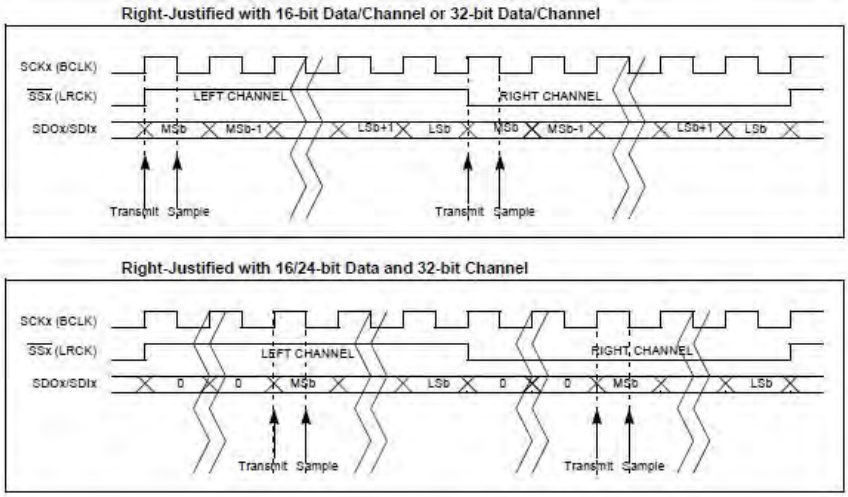
Right-Justified Mode

In Right-Justified mode, the SPI module shifts the audio sample data's MSb after aligning the data to the last clock cycle. The bits preceding the audio sample data can be driven to logic level 0 .

To set the module to Right-Justified mode, the following bits must to be set:

- Set SPI to Right Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active high)
- Select PLIB_SPI_ClockPolaritySelect (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK after the Least Significant bit (LSb) being aligned to the last clock cycle. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.



PCM/DSP Mode

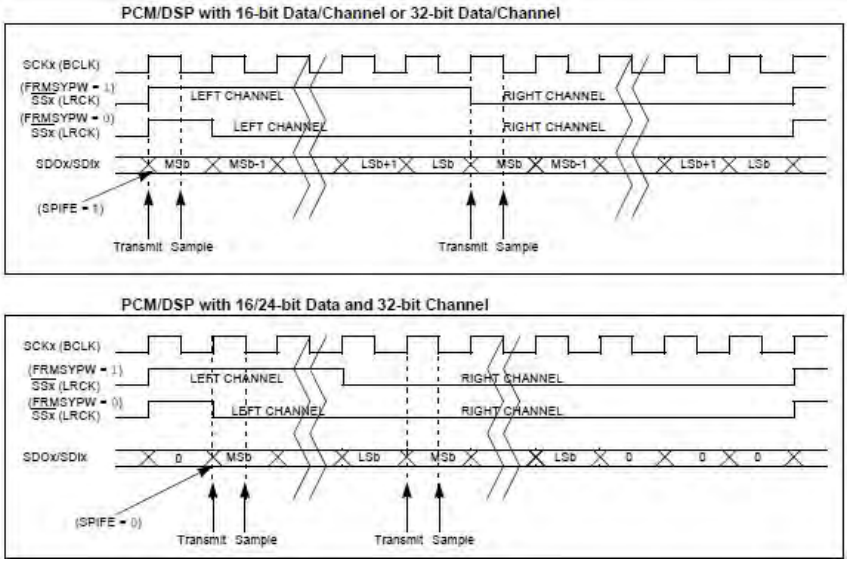
This mode modifies the behavior of LRCK and audio data spacing. In PCM/DSP mode, the LRCK can be a single bit wide (i.e., 1 SCK) or as wide as the audio data (16-, 24-, and 32-bits). The audio data is packed in the frame with the left channel data immediately followed by the right channel data. The frame length is still either 32 or 64 clocks when this device is the master.

In PCM/DSP mode, the transmitter drives the audio data's (left channel) MSb on the first or second transmit edge of SCK (after an LRCK transition). Immediately after the (left channel) LSb, the transmitter drives the (right channel) MSb.

To set the module to Left-Justified mode, the following bit must to be set:

- Set the SPI module to Right-Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect

Refer to the following sample waveform for 16-, 24-, and 32-bit audio data transfers.



Mono Mode Versus Stereo Mode

The SPI module enables the audio data transmission in Mono or Stereo mode by setting PLIB_SPI_AudioTransmitModeSelect. In Stereo mode, the shift register uses each FIFO location once, which gives each channel a unique stream of data for stereo data. In Mono mode, the shift register uses each FIFO location twice, to give each channel the same mono stream of audio data.

Master Mode

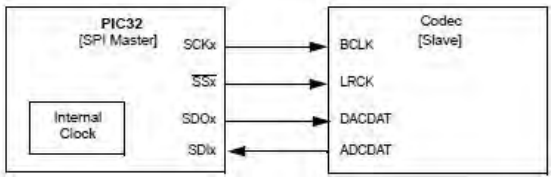
Describes Master mode for the Audio Protocol Interface mode.

Description

A few characteristics of Master mode are:

- This mode enables the device to generate SCK and LRCK pulses as long as the master mode enabled
- The SPI module generates LRCK and SCK continuously in all the cases, regardless of the transmit data while in Master mode
- The SPI module drives the leading edge of LRCK and SCK within 1 SCK period and the serial data shifts in and out continuously even when the Transmit FIFO is empty

The following figure shows a typical interface between the master and slave while in Master mode.



Setup

To configure the device in Audio Protocol Master, enable master mode through PLIB_SPI_MasterEnable and enable audio mode through PLIB_SPI_AudioProtocolEnable.

Example: Audio Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_IN|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: Audio Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;
data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// wait for transfer to complete
while(!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID));

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Slave Mode

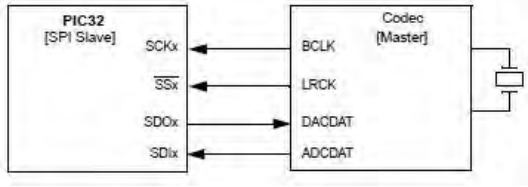
Describes Slave mode for the Audio Protocol Interface mode.

Description

A few characteristics of Slave mode are:

- This mode enables the device to receive SCK and LRCK pulses as long as slave mode is enabled
- The SPI module drives zeros out of SDO, but does not shift data out or in (SDI) until the module receives the LRCK (i.e., the edge that precedes the left channel)
- Once the module receives the leading edge of LRCK, it starts receiving data if PLIB_SPI_PinEnable (enables data in) is selected and the serial data shifts out continuously even when the TX FIFO is empty

The following figure shows the interface between a SPI module in Audio Slave Interface mode to a codec master device.



Setup

The SPI module can be configured in Audio Protocol Slave mode by setting PLIB_SPI_SlaveEnable and enabling the audio protocol through PLIB_SPI_AudioProtocolEnable.

Example: Slave Mode communication Setup

```
#define MY_SPI_ID             SPI_ID_1
#define PERIPHERAL_BUS_CLK    80000000
#define SPI_BAUD_RATE         10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear(MY_SPI_ID);

// Configure General SPI Options
// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/ PLIB_SPI_BufferRead).

Example: Slave Mode communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
  // error
  PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
  data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);               // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Other Features

This topic describes the additional features available in the SPI Peripheral Library.

Communication Mode

The SPI module allows three types of data widths when transmitting and receiving data over a SPI bus. The selection of data width determines the minimum length of the SPI data. The user application should select the appropriate data width to maximize its data throughput. To change the mode of operation on the fly, the SPI module must be idle. If the SPI module is switched off, the new mode will be available when the module is switched on again.

The macro, PLIB_SPI_CommunicationWidthSet, allows the module to communicate in either 8-/16-/32-bit modes. The functionality will be the same for each mode, except for the number of bits that are received and transmitted.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Power-Saving Modes

Note: Each of the following modes can additionally support some or all of the following features. Please refer to the **"Power-Saving Modes"** section of the specific device data sheet or the family reference manual section specified in that chapter for more information.

Sleep Mode

When the device enters Sleep mode, the device clock source and the entire device is shut down. The consequences of entering Sleep depend upon which mode (Master or Slave) the module is configured in at the time that Sleep mode is invoked.

Idle Mode

When the device enters Idle mode, the device clock is operational, but the CPU and selected peripherals are shut down.

The SPI module can continue to operate in Idle mode by calling PLIB_SPI_StopInIdleDisable, or can stop operation in Idle mode by calling PLIB_SPI_StopInIdleEnable.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet to determine availability.

SPI Receive-only Operation

Calling PLIB_SPI_PinDisable (data in pin) disables the transmission at the SDO pin. This allows the SPI module to be configured for a receive-only mode of operation. This function is applicable to all SPI operating modes.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Error Handling

If a new data word has been shifted and the previous buffer contents have not been read, the status can be read by calling PLIB_SPI_ReceiverHasOverflowed. Any received data is not transferred, and further data reception is disabled until the buffer is cleared by calling PLIB_SPI_ReceiverOverflowClear.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Master Mode Clock Frequency

In Master mode, the PBCLK is divided and then used as the serial clock. The division is based on the settings in the SPIxBRG register. The serial clock is output via the SCKx pin to slave devices.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Interrupts

The SPI module has the ability to generate interrupts reflecting the events that occur during the data communication. Interrupts can be enabled or disabled using PLIB_SPI_ErrorInterruptEnable/PLIB_SPI_ErrorInterruptDisable.

The following types of interrupts can be generated:

- Receive data available interrupts are signalled by SPI receive and transmit flag. This event occurs when there is new data assembled in the SPI receive buffer.
- Transmit buffer empty interrupts are signalled by SPI receive and transmit flag. This event occurs when there is space available in the SPI transmit buffer and new data can be written.
- Error interrupts are signalled by SPI receive and transmit flag. This event occurs when there is an overflow condition for the receive buffer when there is an underrun of the transmit buffer, or when a frame error event occurs.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Configuring the Library

The library is configured for the supported SPI modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration and Status Functions

	Name	Description
	PLIB_SPI_BaudRateClockSelect	Selects the type of clock is used by the Baud Rate Generator.
	PLIB_SPI_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_SPI_ClockPolaritySelect	Enables clock polarity.
	PLIB_SPI_CommunicationWidthSelect	Selects the data width for the SPI communication.
	PLIB_SPI_Disable	Disables the SPI module.
	PLIB_SPI_Enable	Enables the SPI module.
	PLIB_SPI_ErrorInterruptDisable	Enables SPI error interrupts.

	PLIB_SPI_ErrorInterruptEnable	Enables SPI error interrupts
	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.
	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

b) Data Transfer Functions

	Name	Description
	PLIB_SPI_BufferClear	Clears the SPI buffer.
	PLIB_SPI_BufferRead	Returns the SPI buffer value.
	PLIB_SPI_BufferAddressGet	Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.
	PLIB_SPI_BufferRead16bit	Returns 16-bit SPI buffer value.
	PLIB_SPI_BufferRead32bit	Returns 32-bit SPI buffer value.
	PLIB_SPI_BufferWrite	Write the data to the SPI buffer.
	PLIB_SPI_BufferWrite16bit	Writes 16-bit data to the SPI buffer.
	PLIB_SPI_BufferWrite32bit	Write 32-bit data to the SPI buffer.

c) Framed Mode Functions

	Name	Description
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameErrorStatusClear	Clears the SPI frame error flag.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.
	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.

d) Audio Mode Functions

	Name	Description
	PLIB_SPI_AudioCommunicationWidthSelect	Selects the data width for the SPI audio communication.
	PLIB_SPI_AudioErrorDisable	Disables the SPI error.
	PLIB_SPI_AudioErrorEnable	Enables the SPI error.
	PLIB_SPI_AudioProtocolDisable	Audio protocol is disabled.
	PLIB_SPI_AudioProtocolEnable	Audio protocol is enabled.
	PLIB_SPI_AudioProtocolModeSelect	Selects the Audio Protocol mode.
	PLIB_SPI_AudioTransmitModeSelect	Selects the transmit audio data format.

e) Transmitter Functions

	Name	Description
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.

	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.
	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.
	PLIB_SPI_TransmitUnderRunStatusClear	Clears the SPI transmit underrun flag.

f) Receiver Functions

	Name	Description
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.

g) Feature Existence Functions

	Name	Description
	PLIB_SPI_ExistsAudioCommunicationWidth	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsAudioErrorControl	Identifies whether the AudioErrorControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolControl	Identifies whether the AudioProtocolControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolMode	Identifies whether the AudioProtocolMode feature exists on the SPI module.
	PLIB_SPI_ExistsAudioTransmitMode	Identifies whether the AudioTransmitMode feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRate	Identifies whether the BaudRate feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRateClock	Identifies whether the BaudRateClock feature exists on the SPI module.
	PLIB_SPI_ExistsBuffer	Identifies whether the Buffer feature exists on the SPI module.
	PLIB_SPI_ExistsBusStatus	Identifies whether the BusStatus feature exists on the SPI module.
	PLIB_SPI_ExistsClockPolarity	Identifies whether the ClockPolarity feature exists on the SPI module.
	PLIB_SPI_ExistsCommunicationWidth	Identifies whether the CommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SPI module.
	PLIB_SPI_ExistsErrorInterruptControl	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOControl	Identifies whether the FIFOControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOCount	Identifies whether the FIFOCount feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOInterruptMode	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFramedCommunication	Identifies whether the FramedCommunication feature exists on the SPI module.
	PLIB_SPI_ExistsFrameErrorStatus	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseCounter	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseDirection	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseEdge	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulsePolarity	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseWidth	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	PLIB_SPI_ExistsInputSamplePhase	Identifies whether the InputSamplePhase feature exists on the SPI module.
	PLIB_SPI_ExistsMasterControl	Identifies whether the MasterControl feature exists on the SPI module.
	PLIB_SPI_ExistsOutputDataPhase	Identifies whether the OutputDataPhase feature exists on the SPI module.
	PLIB_SPI_ExistsPinControl	Identifies whether the PinControl feature exists on the SPI module.
	PLIB_SPI_ExistsReadDataSignStatus	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveBufferStatus	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveFIFOStatus	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	PLIB_SPI_ExistsSlaveSelectControl	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	PLIB_SPI_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferEmptyStatus	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferFullStatus	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.
	PLIB_SPI_Exists16bitBuffer	Identifies whether the Buffer16bit feature exists on the SPI module.
	PLIB_SPI_Exists32bitBuffer	Identifies whether the Buffer32bit feature exists on the SPI module.

h) Data Types and Constants

	Name	Description
	SPI_AUDIO_COMMUNICATION_WIDTH	Defines the list of SPI audio communication width.
	SPI_AUDIO_ERROR	Defines the list of SPI audio error.
	SPI_AUDIO_PROTOCOL	Data type defining the audio protocol mode.
	SPI_AUDIO_TRANSMIT_MODE	Defines the list of SPI transmit audio mode format.
	SPI_BAUD_RATE_CLOCK	Defines the list of SPI Baud Rate Generator (BRG).
	SPI_CLOCK_POLARITY	Defines the list of SPI clock polarity.
	SPI_COMMUNICATION_WIDTH	Defines the list of SPI communication width.
	SPI_DATA_TYPE	Data type defining the SPI data size.
	SPI_ERROR_INTERRUPT	Defines the list of SPI error interrupts.
	SPI_FIFO_INTERRUPT	Defines the list of SPI Buffer Interrupt mode.
	SPI_FIFO_TYPE	Defines the list of SPI buffer mode.
	SPI_FRAME_PULSE_DIRECTION	Defines the list of SPI frame sync pulse direction.
	SPI_FRAME_PULSE_EDGE	Defines the list of SPI frame sync pulse edge.
	SPI_FRAME_PULSE_POLARITY	Defines the list of SPI frame sync pulse polarity.
	SPI_FRAME_PULSE_WIDTH	Defines the list of SPI frame sync pulse width.
	SPI_FRAME_SYNC_PULSE	Data type defining the frame sync pulse counter values.
	SPI_INPUT_SAMPLING_PHASE	Defines the list of SPI data input sample phase.
	SPI_MODULE_ID	Identifies the supported SPI modules.
	SPI_OUTPUT_DATA_PHASE	Defines the list of SPI serial output data changes.
	SPI_PIN	Data type defining the SPI pin.

Description

This section describes the Application Programming Interface (API) functions of the SPI Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration and Status Functions

PLIB_SPI_BaudRateClockSelect Function

Selects the type of clock is used by the Baud Rate Generator.

File

plib_spi.h

C

```
void PLIB_SPI_BaudRateClockSelect(SPI_MODULE_ID index, SPI_BAUD_RATE_CLOCK type);
```

Returns

None.

Description

This function selects the type of clock is used by the Baud Rate Generator.

Remarks

This function implements an operation of the baud rate clock control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBaudRateClock" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BaudRateClockSelect (MY_SPI_INSTANCE, SPI_BAUD_RATE_MCLK_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_BAUD_RATE_CLOCK enumeration values as the SPI baud clock

Function

void PLIB_SPI_BaudRateClockSelect (SPI_MODULE_ID index,
SPI_BAUD_RATE_CLOCK type)

PLIB_SPI_BaudRateSet Function

Sets the baud rate to the desired value.

File

plib_spi.h

C

```
void PLIB_SPI_BaudRateSet(SPI_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

This function implements an operation of the baud rate set feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBaudRate" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BaudRateSet(MY_SPI_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock frequency
baudrate	Baud rate value

Function

void PLIB_SPI_BaudRateSet(SPI_MODULE_ID index, uint32_t clockFrequency,
uint32_t baudRate)

PLIB_SPI_ClockPolaritySelect Function

Enables clock polarity.

File

plib_spi.h

C

```
void PLIB_SPI_ClockPolaritySelect(SPI_MODULE_ID index, SPI_CLOCK_POLARITY polarity);
```

Returns

None.

Description

This function enables clock polarity.

Remarks

This function implements an operation of the clock polarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsClockPolarity" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ClockPolaritySelect(MY_SPI_INSTANCE, SPI_CLOCK_POLARITY_IDLE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_CLOCK_POLARITY enumeration values as the SPI clock polarity

Function

void PLIB_SPI_ClockPolaritySelect(SPI_MODULE_ID index,
SPI_CLOCK_POLARITY polarity)

PLIB_SPI_CommunicationWidthSelect Function

Selects the data width for the SPI communication.

File

plib_spi.h

C

```
void PLIB_SPI_CommunicationWidthSelect(SPI_MODULE_ID index, SPI_COMMUNICATION_WIDTH width);
```

Returns

None.

Description

This function selects the data width for the SPI communication.

Remarks

This function implements an operation of the communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsCommunicationWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_CommunicationWidthSelect(MY_SPI_INSTANCE, SPI_COMMUNICATION_WIDTH_8BITS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
width	One of the SPI_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Function

void PLIB_SPI_CommunicationWidthSelect (SPI_MODULE_ID index,
SPI_COMMUNICATION_WIDTH width)

PLIB_SPI_Disable Function

Disables the SPI module.

File

plib_spi.h

C

```
void PLIB_SPI_Disable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SPI module.

Remarks

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsEnableControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_Disable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_Disable (SPI_MODULE_ID index)

PLIB_SPI_Enable Function

Enables the SPI module.

File

plib_spi.h

C

```
void PLIB_SPI_Enable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI module.

Remarks

The SCKx, SDOx, SDIx and SSx pins must be assigned to available RPn pins before use.

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsEnableControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_Enable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_Enable(SPI_MODULE_ID index)

PLIB_SPI_ErrorInterruptDisable Function

Enables SPI error interrupts.

File

plib_spi.h

C

```
void PLIB_SPI_ErrorInterruptDisable(SPI_MODULE_ID index, SPI_ERROR_INTERRUPT error);
```

Returns

None.

Description

This function enables SPI error interrupts.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsErrorInterruptControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ErrorInterruptDisable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Function

void PLIB_SPI_ErrorInterruptDisable (SPI_MODULE_ID index,
SPI_ERROR_INTERRUPT error)

PLIB_SPI_ErrorInterruptEnable Function

Enables SPI error interrupts

File

plib_spi.h

C

```
void PLIB_SPI_ErrorInterruptEnable(SPI_MODULE_ID index, SPI_ERROR_INTERRUPT error);
```

Returns

None.

Description

This function enables SPI error interrupts.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsErrorInterruptControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ErrorInterruptEnable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Function

void PLIB_SPI_ErrorInterruptEnable (SPI_MODULE_ID index,
SPI_ERROR_INTERRUPT error)

PLIB_SPI_FIFOCountGet Function

Reads the SPI Buffer Element Count bits for either receive or transmit.

File

plib_spi.h

C

```
uint8_t PLIB_SPI_FIFOCountGet(SPI_MODULE_ID index, SPI_FIFO_TYPE type);
```

Returns

CountValue - Buffer element count bits

Description

This function reads the number of SPI transfers pending for Master mode and the number of unread SPI transfers for Slave mode.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOCount" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
uint8_t count = PLIB_SPI_FIFOCountGet(MY_SPI_INSTANCE,SPI_FIFO_TYPE_TRANSMIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_FIFO_TYPE enumeration values

Function

uint8_t PLIB_SPI_FIFOCountGet (SPI_MODULE_ID index, SPI_FIFO_TYPE type)

PLIB_SPI_FIFODisable Function

Disables the SPI enhanced buffer.

File

plib_spi.h

C

```
void PLIB_SPI_FIFODisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SPI enhanced buffer.

Remarks

Enables the legacy standard single buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFODisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FIFODisable (SPI_MODULE_ID index)

PLIB_SPI_FIFOEnable Function

Enables the SPI enhanced buffer.

File

plib_spi.h

C

```
void PLIB_SPI_FIFOEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI enhanced buffer.

Remarks

This enables the enhanced buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFOEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FIFOEnable (SPI_MODULE_ID index)

PLIB_SPI_FIFOInterruptModeSelect Function

Selects the SPI buffer interrupt mode.

File

plib_spi.h

C

```
void PLIB_SPI_FIFOInterruptModeSelect(SPI_MODULE_ID index, SPI_FIFO_INTERRUPT mode);
```

Returns

None.

Description

This function selects the SPI buffer interrupt mode from SPI_FIFO_INTERRUPT.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO interrupt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOInterruptMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFOInterruptModeSelect(MY_SPI_INSTANCE,
                SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_FIFO_INTERRUPT enumeration values as the SPI buffer interrupt mode

Function

void PLIB_SPI_FIFOInterruptModeSelect (SPI_MODULE_ID index,
SPI_FIFO_INTERRUPT mode)

PLIB_SPI_FIFOShiftRegisterIsEmpty Function

Returns the current status of the SPI shift register.

File

plib_spi.h

C

```
bool PLIB_SPI_FIFOShiftRegisterIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - SPI shift register is empty and ready to send or receive
- false - SPI shift register is not empty

Description

This function returns the current status of the SPI shift register.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_FIFOShiftRegisterIsEmpty(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_FIFOShiftRegisterIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_InputSamplePhaseSelect Function

Selects the SPI data input sample phase.

File

plib_spi.h

C

```
void PLIB_SPI_InputSamplePhaseSelect(SPI_MODULE_ID index, SPI_INPUT_SAMPLING_PHASE phase);
```

Returns

None.

Description

This function selects the input sampling phase in Master mode.

Remarks

This function implements an operation of the input sample phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsInputSamplePhase" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_INSTANCE, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
phase	One of the SPI_INPUT_SAMPLING_PHASE as the SPI input sampling phase

Function

void PLIB_SPI_InputSamplePhaseSelect(SPI_MODULE_ID index,
SPI_INPUT_SAMPLING_PHASE phase)

PLIB_SPI_IsBusy Function

Returns the current SPI module activity status.

File

plib_spi.h

C

```
bool PLIB_SPI_IsBusy(SPI_MODULE_ID index);
```

Returns

- true - SPI module is currently busy with some transactions
- false - SPI module is currently idle

Description

This function returns the current SPI module activity status.

Remarks

This function implements an operation of the bus status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBusStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool status = PLIB_SPI_IsBusy(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_IsBusy (SPI_MODULE_ID index)

PLIB_SPI_MasterEnable Function

Enables the SPI in Master mode.

File

plib_spi.h

C

```
void PLIB_SPI_MasterEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI in Master mode.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsMasterControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_MasterEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_MasterEnable(SPI_MODULE_ID index)

PLIB_SPI_OutputDataPhaseSelect Function

Selects serial output data change.

File

plib_spi.h

C

```
void PLIB_SPI_OutputDataPhaseSelect(SPI_MODULE_ID index, SPI_OUTPUT_DATA_PHASE phase);
```

Returns

None.

Description

This function selects serial output data change.

Remarks

This function implements an operation of the output data phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsOutputDataPhase" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_INSTANCE, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	One of the SPI_OUTPUT_DATA_PHASE enumeration values as the SPI serial output data change

Function

void PLIB_SPI_OutputDataPhaseSelect (SPI_MODULE_ID index,
SPI_OUTPUT_DATA_PHASE data)

PLIB_SPI_PinDisable Function

Enables the selected SPI pins.

File

plib_spi.h

C

```
void PLIB_SPI_PinDisable(SPI_MODULE_ID index, SPI_PIN pin);
```

Returns

None.

Description

This function enables the selected SPI pins.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsPinControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PinDisable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Function

void PLIB_SPI_PinDisable (SPI_MODULE_ID index, SPI_PIN pin)

PLIB_SPI_PinEnable Function

Enables the selected SPI pins.

File

plib_spi.h

C

```
void PLIB_SPI_PinEnable(SPI_MODULE_ID index, SPI_PIN pin);
```

Returns

None.

Description

This function enables the selected SPI pins.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsPinControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PinEnable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Function

void PLIB_SPI_PinEnable (SPI_MODULE_ID index,SPI_PIN pin)

PLIB_SPI_ReadDataIsSignExtended Function

Returns the current status of the receive (RX) FIFO sign-extended data.

File

plib_spi.h

C

```
bool PLIB_SPI_ReadDataIsSignExtended(SPI_MODULE_ID index);
```

Returns

- true - Data from RX FIFO is sign-extended
- false - Data from RX FIFO is not sign-extended

Description

This function returns the current status of the receive (RX) FIFO sign-extended data.

Remarks

This function implements an operation of the data sign feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReadDataSignStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_ReadDataIsSignExtended(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReadDataIsSignExtended(SPI_MODULE_ID index)

PLIB_SPI_SlaveEnable Function

Enables the SPI in Slave mode.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI in Slave mode.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsMasterControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveEnable(SPI_MODULE_ID index)

PLIB_SPI_SlaveSelectDisable Function

Disables Master mode slave select.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveSelectDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables Master mode slave select.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsSlaveSelectControl" in your application to automatically determine whether this feature is available.

To disable Slave mode slave select pin (SSEN), PLIB_SPI_PinDisable API can be used.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveSelectDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveSelectDisable(SPI_MODULE_ID index)

PLIB_SPI_SlaveSelectEnable Function

Enables Master mode slave select.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveSelectEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables Master mode slave select.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the Master mode slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsSlaveSelectControl" in your application to automatically determine whether this feature is available.

To enable Slave mode slave select pin (SSEN), PLIB_SPI_PinEnable API can be used.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveSelectEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveSelectEnable(SPI_MODULE_ID index)

PLIB_SPI_StopInIdleDisable Function

Continues module operation when the device enters Idle mode.

File

plib_spi.h

C

```
void PLIB_SPI_StopInIdleDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function sets up the SPI module such that module operation is continued when the device enters Idle mode.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsStopInIdleControl" in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_StopInIdleDisable (SPI_MODULE_ID index)

PLIB_SPI_StopInIdleEnable Function

Discontinues module operation when the device enters Idle mode.

File

plib_spi.h

C

```
void PLIB_SPI_StopInIdleEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function sets up the SPI module such that module operation is disabled when the device enters Idle mode.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsStopInIdleControl" in your application to automatically determine if this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_StopInIdleEnable (SPI_MODULE_ID index)

b) Data Transfer Functions

PLIB_SPI_BufferClear Function

Clears the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BufferClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_BufferClear (SPI_MODULE_ID index)

PLIB_SPI_BufferRead Function

Returns the SPI buffer value.

File

plib_spi.h

C

```
uint8_t PLIB_SPI_BufferRead(SPI_MODULE_ID index);
```

Returns

Reads the SPI buffer.

Description

This function returns the SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is

available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
uint8_t bufferValue = PLIB_SPI_BufferRead(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SPI_BufferRead (SPI_MODULE_ID index)

PLIB_SPI_BufferAddressGet Function

Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.

File

plib_spi.h

C

```
void* PLIB_SPI_BufferAddressGet(SPI_MODULE_ID index);
```

Returns

The address of the SPIxBUF register

Description

This function returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_BufferAddressGet(SPI_MODULE_ID index)

PLIB_SPI_BufferRead16bit Function

Returns 16-bit SPI buffer value.

File

plib_spi.h

C

```
uint16_t PLIB_SPI_BufferRead16bit(SPI_MODULE_ID index);
```

Returns

Returns the SPI 16-bit buffer value.

Description

This function returns 16-bit SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists16bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 16-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

uint16_t bufferValue = PLIB_SPI_BufferRead16bit( MY_SPI_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_SPI_BufferRead16bit (SPI_MODULE_ID index)

PLIB_SPI_BufferRead32bit Function

Returns 32-bit SPI buffer value.

File

plib_spi.h

C

```
uint32_t PLIB_SPI_BufferRead32bit(SPI_MODULE_ID index);
```

Returns

Returns the SPI 32-bit buffer value.

Description

This function returns 32-bit SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists32bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 32-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

uint32_t bufferValue = PLIB_SPI_BufferRead32bit( MY_SPI_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SPI_BufferRead32bit (SPI_MODULE_ID index)

PLIB_SPI_BufferWrite Function

Write the data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite(SPI_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function writes data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BufferWrite ( MY_SPI_INSTANCE, 0xFF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to written to the SPI buffer

Function

void PLIB_SPI_BufferWrite (SPI_MODULE_ID index , uint8_t data)

PLIB_SPI_BufferWrite16bit Function

Writes 16-bit data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite16bit(SPI_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function writes 16-bit data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists16bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 16-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

PLIB_SPI_BufferWrite16bit ( MY_SPI_INSTANCE, 0x55AA );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	16-bit data to be written to the SPI buffer

Function

void PLIB_SPI_BufferWrite16bit (SPI_MODULE_ID index , uint16_t data)

PLIB_SPI_BufferWrite32bit Function

Write 32-bit data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite32bit(SPI_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function writes 32-bit data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists32bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 32-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

PLIB_SPI_BufferWrite ( MY_SPI_INSTANCE, 0x55AA55AA );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	32-bit data to be written to the SPI buffer

Function

void PLIB_SPI_BufferWrite32bit (SPI_MODULE_ID index , uint32_t data)

c) Framed Mode Functions

PLIB_SPI_FramedCommunicationDisable Function

Disables framed SPI support.

File

plib_spi.h

C

```
void PLIB_SPI_FramedCommunicationDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables framed SPI support.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFramedCommunication" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FramedCommunicationDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FramedCommunicationDisable (SPI_MODULE_ID index)

PLIB_SPI_FramedCommunicationEnable Function

Enables framed SPI support.

File

plib_spi.h

C

```
void PLIB_SPI_FramedCommunicationEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables framed SPI support.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFramedCommunication" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FramedCommunicationEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FramedCommunicationEnable (SPI_MODULE_ID index)

PLIB_SPI_FrameErrorStatusGet Function

Returns the current status of the SPI frame error.

File

plib_spi.h

C

```
bool PLIB_SPI_FrameErrorStatusGet(SPI_MODULE_ID index);
```

Returns

- true - Frame error detected
- false - No frame error detected

Description

This function returns the current status of the SPI frame error.

Remarks

Valid only if Frame mode is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameErrorStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool status = PLIB_SPI_FrameErrorStatusGet(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_FrameErrorStatusGet (SPI_MODULE_ID index)

PLIB_SPI_FrameErrorStatusClear Function

Clears the SPI frame error flag.

File

plib_spi.h

C

```
void PLIB_SPI_FrameErrorStatusClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI frame error flag.

Remarks

This function implements an operation of the frame error status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "ExistsFrameErrorStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameErrorStatusClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FrameErrorStatusClear(SPI_MODULE_ID index)

PLIB_SPI_FrameSyncPulseCounterSelect Function

Selects at which character the SPI frame sync pulse is generated.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseCounterSelect(SPI_MODULE_ID index, SPI_FRAME_SYNC_PULSE pulse);
```

Returns

None.

Description

This function selects at which character the SPI frame sync pulse is generated.

Remarks

This is valid only when PLIB_SPI_FramedCommunicationEnable is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseCounter" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_INSTANCE,
                                     SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pulse	One of the SPI_FRAME_SYNC_PULSE enumeration values as the SPI frame sync pulse count

Function

void PLIB_SPI_FrameSyncPulseCounterSelect (SPI_MODULE_ID index,
SPI_FRAME_SYNC_PULSE pulse)

PLIB_SPI_FrameSyncPulseDirectionSelect Function

Selects the frame sync pulse direction.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseDirectionSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_DIRECTION direction);
```

Returns

None.

Description

This function selects the frame sync pulse direction.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseDirection" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_DIRECTION_INPUT );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
direction	One of the SPI_FRAME_PULSE_DIRECTION enumeration values as the SPI frame sync pulse polarity

Function

void PLIB_SPI_FrameSyncPulseDirectionSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_DIRECTION direction)

PLIB_SPI_FrameSyncPulseEdgeSelect Function

Selects the frame sync pulse edge.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseEdgeSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_EDGE edge);
```

Returns

None.

Description

This function selects the frame sync pulse edge.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseEdge" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseEdgeSelect(MY_SPI_INSTANCE,
                SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_EDGE enumeration values as the SPI frame sync pulse edge

Function

void PLIB_SPI_FrameSyncPulseEdgeSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_EDGE edge)

PLIB_SPI_FrameSyncPulsePolaritySelect Function

Selects the frame sync pulse polarity.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulsePolaritySelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_POLARITY polarity);
```

Returns

None.

Description

This function selects the frame sync pulse polarity.

Remarks

Available only for Frame mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulsePolarity" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulsePolaritySelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_FRAME_PULSE_POLARITY enumeration values as the SPI frame sync pulse polarity

Function

void PLIB_SPI_FrameSyncPulsePolaritySelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_POLARITY polarity)

PLIB_SPI_FrameSyncPulseWidthSelect Function

Sets the frame sync pulse width.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseWidthSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_WIDTH width);
```

Returns

None.

Description

This function sets the frame sync pulse width.

Remarks

Length of the word is dependent on the communication mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseWidthSelect (MY_SPI_INSTANCE,SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_WIDTH enumeration values as the SPI frame sync pulse width.

Function

void PLIB_SPI_FrameSyncPulseWidthSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_WIDTH width)

d) Audio Mode Functions

PLIB_SPI_AudioCommunicationWidthSelect Function

Selects the data width for the SPI audio communication.

File

plib_spi.h

C

```
void PLIB_SPI_AudioCommunicationWidthSelect(SPI_MODULE_ID index, SPI_AUDIO_COMMUNICATION_WIDTH mode);
```

Returns

None.

Description

This function selects the data width for the SPI audio communication.

Remarks

This mode is available only when PLIB_SPI_AudioProtocolEnable is enabled.

This function implements an operation of the audio communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioCommunicationWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
```

```
PLIB_SPI_AudioCommunicationWidthSelect(MY_SPI_INSTANCE,
                        SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
width	One of the SPI_AUDIO_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Function

void PLIB_SPI_AudioCommunicationWidthSelect (SPI_MODULE_ID index,
SPI_AUDIO_COMMUNICATION_WIDTH width)

PLIB_SPI_AudioErrorDisable Function

Disables the SPI error.

File

plib_spi.h

C

```
void PLIB_SPI_AudioErrorDisable(SPI_MODULE_ID index, SPI_AUDIO_ERROR error);
```

Returns

None.

Description

This function disables the SPI error.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioErrorControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioErrorDisable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Function

void PLIB_SPI_AudioErrorDisable (SPI_MODULE_ID index, SPI_AUDIO_ERROR error)

PLIB_SPI_AudioErrorEnable Function

Enables the SPI error.

File

plib_spi.h

C

```
void PLIB_SPI_AudioErrorEnable(SPI_MODULE_ID index, SPI_AUDIO_ERROR error);
```

Returns

None.

Description

This function enables the SPI error.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioErrorControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioErrorEnable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Function

void PLIB_SPI_AudioErrorEnable (SPI_MODULE_ID index, SPI_Audio_ERROR error)

PLIB_SPI_AudioProtocolDisable Function

Audio protocol is disabled.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the audio protocol.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_AudioProtocolDisable (SPI_MODULE_ID index)

PLIB_SPI_AudioProtocolEnable Function

Audio protocol is enabled.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the audio protocol.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolControl" in your application to automatically determine whether this feature is available.

Preconditions

Disable the SPI module by calling PLIB_SPI_Disable.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_AudioProtocolEnable (SPI_MODULE_ID index)

PLIB_SPI_AudioProtocolModeSelect Function

Selects the Audio Protocol mode.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolModeSelect(SPI_MODULE_ID index, SPI_AUDIO_PROTOCOL mode);
```

Returns

None.

Description

This function selects the Audio Protocol mode.

Remarks

This function implements an operation of the audio protocol mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_INSTANCE,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_AUDIO_PROTOCOL enumeration values as the audio protocol

Function

void PLIB_SPI_AudioProtocolModeSelect(SPI_MODULE_ID index,
SPI_AUDIO_PROTOCOL mode)

PLIB_SPI_AudioTransmitModeSelect Function

Selects the transmit audio data format.

File

plib_spi.h

C

```
void PLIB_SPI_AudioTransmitModeSelect(SPI_MODULE_ID index, SPI_AUDIO_TRANSMIT_MODE mode);
```

Returns

None.

Description

This function selects the transmit audio data format.

Remarks

This function implements an operation of the audio transmit mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioTransmitMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioTransmitModeSelect (MY_SPI_INSTANCE, SPI_AUDIO_TRANSMIT_MONO);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_TRANSMIT_AUDIO_MODE enumeration values as the transmit audio format

Function

void PLIB_SPI_AudioTransmitModeSelect (SPI_MODULE_ID index,
SPI_AUDIO_TRANSMIT_MODE mode)

e) Transmitter Functions

PLIB_SPI_TransmitBufferIsEmpty Function

Returns the current status of the transmit buffer.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitBufferIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - Transmit buffer is empty
- false - Transmit buffer is not empty

Description

This function returns the current status of the transmit buffer.

Remarks

This function implements an operation of the transmit buffer empty status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitBufferEmptyStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_TransmitBufferIsEmpty(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitBufferIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_TransmitBufferIsFull Function

Returns the current transmit buffer status of the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitBufferIsFull(SPI_MODULE_ID index);
```

Returns

- true - Transmit not yet started, transmit buffer is full
- false - Transmit started, transmit buffer is empty/not full

Description

This function returns the current transmit buffer status of the SPI module.

Remarks

In Standard Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the transmit buffer. Automatically cleared in hardware when the SPI module transfers data from the transmit buffer to the shift register.

In Enhanced Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the last available buffer. Automatically cleared in hardware when the buffer is available for writing.

This function implements an operation of the transmit buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitBufferFullStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool buffullstate = PLIB_SPI_TransmitBufferIsFull (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitBufferIsFull (SPI_MODULE_ID index)

PLIB_SPI_TransmitUnderRunStatusGet Function

Returns the current status of the transmit underrun.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitUnderRunStatusGet(SPI_MODULE_ID index);
```

Returns

- true - Transmit buffer has encountered an underrun condition
- false - Transmit buffer run has not encountered an underrun condition

Description

This function returns the current status of the transmit underrun.

Remarks

Valid in Framed Sync mode.

This function implements an operation of the transmit underrun status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitUnderRunStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_TransmitUnderRunStatusGet(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitUnderRunStatusGet(SPI_MODULE_ID index)

PLIB_SPI_TransmitUnderRunStatusClear Function

Clears the SPI transmit underrun flag.

File

plib_spi.h

C

```
void PLIB_SPI_TransmitUnderRunStatusClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI transmit underrun flag.

Remarks

This function implements an operation of the transmit underrun status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "ExistsTransmitUnderRunStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_TransmitUnderRunStatusClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_TransmitUnderRunStatusClear(SPI_MODULE_ID index)

f) Receiver Functions

PLIB_SPI_ReceiverBufferIsFull Function

Returns the current status of the SPI receive buffer.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverBufferIsFull(SPI_MODULE_ID index);
```

Returns

Receiver Buffer Full Status:

- true - Receive complete, receive buffer is full
- false - Receive is not complete, receive buffer is empty

Description

This function returns the current status of the SPI receive buffer.

Remarks

In Standard Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive buffer. Automatically cleared in hardware when the core reads the SPI buffer, read in the receive buffer.

In Enhanced Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive buffer, filling the last unread buffer. Automatically cleared in hardware when a buffer is available for a transfer from the shift register.

This function implements an operation of the receiver buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiveBufferStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
```

```
// application developer.
bool receivefullstate = PLIB_SPI_ReceiverBufferIsFull (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverBufferIsFull (SPI_MODULE_ID index)

PLIB_SPI_ReceiverFIFOIsEmpty Function

Returns the current status of the SPI receive FIFO.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverFIFOIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - Receive FIFO is empty
- false - Receive FIFO is not empty

Description

This function returns the current status of the SPI receive FIFO.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiveFIFOStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool fifostate = PLIB_SPI_ReceiverFIFOIsEmpty (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverFIFOIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_ReceiverHasOverflowed Function

Returns the current status of the SPI receiver overflow.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverHasOverflowed(SPI_MODULE_ID index);
```

Returns

SPI receiver overflow status:

- true - A new byte/word is completely received and discarded. The user software has not read the previous data in the SPI buffer register.
- false - No Overflow has occurred

Description

This function returns the current status of the SPI receiver overflow.

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiverOverflow" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool overflowstate = PLIB_SPI_ReceiverHasOverflowed(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverHasOverflowed (SPI_MODULE_ID index)

PLIB_SPI_ReceiverOverflowClear Function

Clears the SPI receive overflow flag.

File

plib_spi.h

C

```
void PLIB_SPI_ReceiverOverflowClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI receive overflow flag.

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiverOverflow" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ReceiverOverflowClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_ReceiverOverflowClear(SPI_MODULE_ID index)

g) Feature Existence Functions

PLIB_SPI_ExistsAudioCommunicationWidth Function

Identifies whether the AudioCommunicationWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioCommunicationWidth(SPI_MODULE_ID index);
```

Returns

- true - The AudioCommunicationWidth feature is supported on the device
- false - The AudioCommunicationWidth feature is not supported on the device

Description

This function identifies whether the AudioCommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioCommunicationWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioCommunicationWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioErrorControl Function

Identifies whether the AudioErrorControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioErrorControl(SPI_MODULE_ID index);
```

Returns

- true - The AudioErrorControl feature is supported on the device
- false - The AudioErrorControl feature is not supported on the device

Description

This function identifies whether the AudioErrorControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_AudioErrorEnable
- PLIB_SPI_AudioErrorDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioErrorControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioProtocolControl Function

Identifies whether the AudioProtocolControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioProtocolControl(SPI_MODULE_ID index);
```

Returns

- true - The AudioProtocolControl feature is supported on the device
- false - The AudioProtocolControl feature is not supported on the device

Description

This function identifies whether the AudioProtocolControl feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioProtocolEnable
- PLIB_SPI_AudioProtocolDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioProtocolControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioProtocolMode Function

Identifies whether the AudioProtocolMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioProtocolMode(SPI_MODULE_ID index);
```

Returns

- true - The AudioProtocolMode feature is supported on the device
- false - The AudioProtocolMode feature is not supported on the device

Description

This function identifies whether the AudioProtocolMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioProtocolModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioProtocolMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioTransmitMode Function

Identifies whether the AudioTransmitMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioTransmitMode(SPI_MODULE_ID index);
```

Returns

- true - The AudioTransmitMode feature is supported on the device
- false - The AudioTransmitMode feature is not supported on the device

Description

This function identifies whether the AudioTransmitMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioTransmitModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioTransmitMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBaudRate(SPI_MODULE_ID index);
```

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_BaudRateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBaudRate(SPI_MODULE_ID index)

PLIB_SPI_ExistsBaudRateClock Function

Identifies whether the BaudRateClock feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBaudRateClock(SPI_MODULE_ID index);
```

Returns

- true - The BaudRateClock feature is supported on the device
- false - The BaudRateClock feature is not supported on the device

Description

This function identifies whether the BaudRateClock feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_BaudRateClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBaudRateClock(SPI_MODULE_ID index)

PLIB_SPI_ExistsBuffer Function

Identifies whether the Buffer feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer feature is supported on the device
- false - The Buffer feature is not supported on the device

Description

This function identifies whether the Buffer feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferClear
- PLIB_SPI_BufferRead
- PLIB_SPI_BufferWrite
- PLIB_SPI_BufferAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBuffer(SPI_MODULE_ID index)

PLIB_SPI_ExistsBusStatus Function

Identifies whether the BusStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBusStatus(SPI_MODULE_ID index);
```

Returns

- true - The BusStatus feature is supported on the device
- false - The BusStatus feature is not supported on the device

Description

This function identifies whether the BusStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_IsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBusStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsClockPolarity Function

Identifies whether the ClockPolarity feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsClockPolarity(SPI_MODULE_ID index);
```

Returns

- true - The ClockPolarity feature is supported on the device
- false - The ClockPolarity feature is not supported on the device

Description

This function identifies whether the ClockPolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ClockPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsClockPolarity(SPI_MODULE_ID index)

PLIB_SPI_ExistsCommunicationWidth Function

Identifies whether the CommunicationWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsCommunicationWidth(SPI_MODULE_ID index);
```

Returns

- true - The CommunicationWidth feature is supported on the device
- false - The CommunicationWidth feature is not supported on the device

Description

This function identifies whether the CommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_CommunicationWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsCommunicationWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsEnableControl(SPI_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_Enable
- PLIB_SPI_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsEnableControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsErrorInterruptControl Function

Identifies whether the ErrorInterruptControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsErrorInterruptControl(SPI_MODULE_ID index);
```

Returns

- true - The ErrorInterruptControl feature is supported on the device
- false - The ErrorInterruptControl feature is not supported on the device

Description

This function identifies whether the ErrorInterruptControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_ErrorInterruptEnable
- PLIB_SPI_ErrorInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsErrorInterruptControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOControl Function

Identifies whether the FIFOControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOControl(SPI_MODULE_ID index);
```

Returns

- true - The FIFOControl feature is supported on the device
- false - The FIFOControl feature is not supported on the device

Description

This function identifies whether the FIFOControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FIFOEnable
- PLIB_SPI_FIFODisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOCount Function

Identifies whether the FIFOCount feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOCount(SPI_MODULE_ID index);
```

Returns

- true - The FIFOCount feature is supported on the device
- false - The FIFOCount feature is not supported on the device

Description

This function identifies whether the FIFOCount feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOCount(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOInterruptMode Function

Identifies whether the FIFOInterruptMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOInterruptMode(SPI_MODULE_ID index);
```

Returns

- true - The FIFOInterruptMode feature is supported on the device
- false - The FIFOInterruptMode feature is not supported on the device

Description

This function identifies whether the FIFOInterruptMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOInterruptMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus Function

Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus(SPI_MODULE_ID index);
```

Returns

- true - The FIFOShiftRegisterEmptyStatus feature is supported on the device
- false - The FIFOShiftRegisterEmptyStatus feature is not supported on the device

Description

This function identifies whether the FIFOShiftRegisterEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOShiftRegisterIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsFramedCommunication Function

Identifies whether the FramedCommunication feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFramedCommunication(SPI_MODULE_ID index);
```

Returns

- true - The FramedCommunication feature is supported on the device
- false - The FramedCommunication feature is not supported on the device

Description

This function identifies whether the FramedCommunication feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FramedCommunicationEnable
- PLIB_SPI_FramedCommunicationDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFramedCommunication(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameErrorStatus Function

Identifies whether the FrameErrorStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameErrorStatus(SPI_MODULE_ID index);
```

Returns

- true - The FrameErrorStatus feature is supported on the device
- false - The FrameErrorStatus feature is not supported on the device

Description

This function identifies whether the FrameErrorStatus feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FrameErrorStatusGet
- PLIB_SPI_FrameErrorStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameErrorStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseCounter Function

Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseCounter(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseCounter feature is supported on the device
- false - The FrameSyncPulseCounter feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseCounter feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseCounterSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseCounter(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseDirection Function

Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseDirection(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseDirection feature is supported on the device
- false - The FrameSyncPulseDirection feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseDirection feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseDirectionSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseDirection(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseEdge Function

Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseEdge(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseEdge feature is supported on the device
- false - The FrameSyncPulseEdge feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseEdge feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseEdge(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulsePolarity Function

Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulsePolarity(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulsePolarity feature is supported on the device
- false - The FrameSyncPulsePolarity feature is not supported on the device

Description

This function identifies whether the FrameSyncPulsePolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulsePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulsePolarity(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseWidth Function

Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseWidth(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseWidth feature is supported on the device
- false - The FrameSyncPulseWidth feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsInputSamplePhase Function

Identifies whether the InputSamplePhase feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsInputSamplePhase(SPI_MODULE_ID index);
```

Returns

- true - The InputSamplePhase feature is supported on the device
- false - The InputSamplePhase feature is not supported on the device

Description

This function identifies whether the InputSamplePhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_InputSamplePhaseSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsInputSamplePhase(SPI_MODULE_ID index)

PLIB_SPI_ExistsMasterControl Function

Identifies whether the MasterControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsMasterControl(SPI_MODULE_ID index);
```

Returns

- true - The MasterControl feature is supported on the device
- false - The MasterControl feature is not supported on the device

Description

This function identifies whether the MasterControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_MasterEnable
- PLIB_SPI_SlaveEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsMasterControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsOutputDataPhase Function

Identifies whether the OutputDataPhase feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsOutputDataPhase(SPI_MODULE_ID index);
```

Returns

- true - The OutputDataPhase feature is supported on the device
- false - The OutputDataPhase feature is not supported on the device

Description

This function identifies whether the OutputDataPhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_OutputDataPhaseSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsOutputDataPhase(SPI_MODULE_ID index)

PLIB_SPI_ExistsPinControl Function

Identifies whether the PinControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsPinControl(SPI_MODULE_ID index);
```

Returns

- true - The PinControl feature is supported on the device
- false - The PinControl feature is not supported on the device

Description

This function identifies whether the PinControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_PinEnable
- PLIB_SPI_PinDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsPinControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsReadDataSignStatus Function

Identifies whether the ReadDataSignStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReadDataSignStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReadDataSignStatus feature is supported on the device
- false - The ReadDataSignStatus feature is not supported on the device

Description

This function identifies whether the ReadDataSignStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReadDataIsSignExtended

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReadDataSignStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiveBufferStatus Function

Identifies whether the ReceiveBufferStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiveBufferStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReceiveBufferStatus feature is supported on the device
- false - The ReceiveBufferStatus feature is not supported on the device

Description

This function identifies whether the ReceiveBufferStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReceiverBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiveBufferStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiveFIFOStatus Function

Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiveFIFOStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReceiveFIFOStatus feature is supported on the device
- false - The ReceiveFIFOStatus feature is not supported on the device

Description

This function identifies whether the ReceiveFIFOStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReceiverFIFOIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiveFIFOStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiverOverflow Function

Identifies whether the ReceiverOverflow feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiverOverflow(SPI_MODULE_ID index);
```

Returns

- true - The ReceiverOverflow feature is supported on the device
- false - The ReceiverOverflow feature is not supported on the device

Description

This function identifies whether the ReceiverOverflow feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_ReceiverHasOverflowed
- PLIB_SPI_ReceiverOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiverOverflow(SPI_MODULE_ID index)

PLIB_SPI_ExistsSlaveSelectControl Function

Identifies whether the SlaveSelectControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsSlaveSelectControl(SPI_MODULE_ID index);
```

Returns

- true - The SlaveSelectControl feature is supported on the device
- false - The SlaveSelectControl feature is not supported on the device

Description

This function identifies whether the SlaveSelectControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_SlaveSelectEnable
- PLIB_SPI_SlaveSelectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsSlaveSelectControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsStopInIdleControl(SPI_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_StopInIdleEnable
- PLIB_SPI_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsStopInIdleControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitBufferEmptyStatus Function

Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitBufferEmptyStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitBufferEmptyStatus feature is supported on the device
- false - The TransmitBufferEmptyStatus feature is not supported on the device

Description

This function identifies whether the TransmitBufferEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_TransmitBufferIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitBufferEmptyStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitBufferFullStatus Function

Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitBufferFullStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitBufferFullStatus feature is supported on the device
- false - The TransmitBufferFullStatus feature is not supported on the device

Description

This function identifies whether the TransmitBufferFullStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_TransmitBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitBufferFullStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitUnderRunStatus Function

Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitUnderRunStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitUnderRunStatus feature is supported on the device
- false - The TransmitUnderRunStatus feature is not supported on the device

Description

This function identifies whether the TransmitUnderRunStatus feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_TransmitUnderRunStatusGet
- PLIB_SPI_TransmitUnderRunStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitUnderRunStatus(SPI_MODULE_ID index)

PLIB_SPI_Exists16bitBuffer Function

Identifies whether the Buffer16bit feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_Exists16bitBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer16bit feature is supported on the device
- false - The Buffer16bit feature is not supported on the device

Description

This function identifies whether the Buffer16bit feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferRead16bit
- PLIB_SPI_BufferWrite16bit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_Exists16bitBuffer(SPI_MODULE_ID index)

PLIB_SPI_Exists32bitBuffer Function

Identifies whether the Buffer32bit feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_Exists32bitBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer32bit feature is supported on the device
- false - The Buffer32bit feature is not supported on the device

Description

This function identifies whether the Buffer32bit feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferRead32bit
- PLIB_SPI_BufferWrite32bit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_Exists32bitBuffer(SPI_MODULE_ID index)

h) Data Types and Constants

SPI_AUDIO_COMMUNICATION_WIDTH Enumeration

Defines the list of SPI audio communication width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL
} SPI_AUDIO_COMMUNICATION_WIDTH;
```

Members

Members	Description
SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL	Communication is 24-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL	Communication is 32-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL	Communication is 16-bit Data,16-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL	Communication is 16-bit Data,16-bit FIFO,16-bit Channel/64-bit Frame

Description

SPI Audio Communication Width

This macro defines the list of SPI audio communication width.

SPI_AUDIO_ERROR Enumeration

Defines the list of SPI audio error.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_ERROR_RECEIVE_OVERFLOW,
  SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN
} SPI_AUDIO_ERROR;
```

Members

Members	Description
SPI_AUDIO_ERROR_RECEIVE_OVERFLOW	SPI error for receive overflow
SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN	SPI error for transmit underrun

Description

SPI Audio Error

This macro defines the list of SPI audio error.

SPI_AUDIO_PROTOCOL Enumeration

Data type defining the audio protocol mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_PROTOCOL_PCM_DSP,
  SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED,
  SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED,
  SPI_AUDIO_PROTOCOL_I2S
} SPI_AUDIO_PROTOCOL;
```

Members

Members	Description
SPI_AUDIO_PROTOCOL_PCM_DSP	Audio protocol set to PCM/DSP mode
SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED	Audio protocol set to Right-Justified mode
SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED	Audio protocol set to Left-Justified mode
SPI_AUDIO_PROTOCOL_I2S	Audio protocol set to I2C mode

Description

Audio Protocol Mode enumeration

This data type defining the audio protocol mode.

Remarks

This enumeration is processor specific.

SPI_AUDIO_TRANSMIT_MODE Enumeration

Defines the list of SPI transmit audio mode format.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_TRANSMIT_MONO,
  SPI_AUDIO_TRANSMIT_STEREO
} SPI_AUDIO_TRANSMIT_MODE;
```

Members

Members	Description
SPI_AUDIO_TRANSMIT_MONO	SPI Transmit Audio Data Format is Mono
SPI_AUDIO_TRANSMIT_STEREO	SPI Transmit Audio Data Format is Stereo

Description

Audio Transmit mode format

This macro defines the list of SPI transmit audio mode format.

SPI_BAUD_RATE_CLOCK Enumeration

Defines the list of SPI Baud Rate Generator (BRG).

File

help_plib_spi.h

C

```
typedef enum {
  SPI_BAUD_RATE_MCLK_CLOCK,
  SPI_BAUD_RATE_PBCLK_CLOCK
} SPI_BAUD_RATE_CLOCK;
```

Members

Members	Description
SPI_BAUD_RATE_MCLK_CLOCK	MCLK is used by the Baud Rate Generator
SPI_BAUD_RATE_PBCLK_CLOCK	Peripheral bus clock is used by the Baud Rate Generator

Description

SPI Baud Rate Generator

This macro defines the list of the SPI Baud Rate Generator.

SPI_CLOCK_POLARITY Enumeration

Defines the list of SPI clock polarity.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_CLOCK_POLARITY_IDLE_HIGH,
  SPI_CLOCK_POLARITY_IDLE_LOW
} SPI_CLOCK_POLARITY;
```

Members

Members	Description
SPI_CLOCK_POLARITY_IDLE_HIGH	Idle state for clock is a high level;active state is a low level
SPI_CLOCK_POLARITY_IDLE_LOW	Idle state for clock is a low level;active state is a high level

Description

SPI Clock polarity

This macro defines the list of SPI clock polarity.

SPI_COMMUNICATION_WIDTH Enumeration

Defines the list of SPI communication width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_COMMUNICATION_WIDTH_32BITS,
  SPI_COMMUNICATION_WIDTH_16BITS,
  SPI_COMMUNICATION_WIDTH_8BITS
} SPI_COMMUNICATION_WIDTH;
```

Members

Members	Description
SPI_COMMUNICATION_WIDTH_32BITS	Communication is 32-bit-wide
SPI_COMMUNICATION_WIDTH_16BITS	Communication is word-wide
SPI_COMMUNICATION_WIDTH_8BITS	Communication is byte-wide

Description

SPI Communication Width

This macro defines the list of SPI communication width.

SPI_DATA_TYPE Type

Data type defining the SPI data size.

File

help_plib_spi.h

C

```
typedef uint16_t SPI_DATA_TYPE;
```

Description

SPI Data Type definition

This data type defines the SPI data size.

SPI_ERROR_INTERRUPT Enumeration

Defines the list of SPI error interrupts.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW,
  SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW,
  SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN
} SPI_ERROR_INTERRUPT;
```

Members

Members	Description
SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW	SPI interrupt for frame error
SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW	SPI interrupt for receive overflow error
SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN	SPI interrupt for transmit underrun error

Description

SPI Error Interrupt

This macro defines the list of SPI error interrupts.

SPI_FIFO_INTERRUPT Enumeration

Defines the list of SPI Buffer Interrupt mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY,
  SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY
} SPI_FIFO_INTERRUPT;
```

Members

Members	Description
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL	Interrupt when the transmit buffer is not full
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE	Interrupt when the transmit buffer is half empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY	Interrupt when the transmit buffer is empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE	Interrupt when transmission is complete
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL	Interrupt when receive buffer is full
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE	Interrupt when the receive buffer half full or more
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY	Interrupt when the receive buffer is not empty
SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY	Interrupt when the receive buffer is empty

Description

SPI Buffer Interrupt Mode

This macro defines the list of SPI Buffer Interrupt mode.

SPI_FIFO_TYPE Enumeration

Defines the list of SPI buffer mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FIFO_TYPE_TRNSMIT,
  SPI_FIFO_TYPE_RECEIVE
} SPI_FIFO_TYPE;
```

Members

Members	Description
SPI_FIFO_TYPE_TRNSMIT	Buffer type is transmit
SPI_FIFO_TYPE_RECEIVE	Buffer type is receive

Description

SPI Buffer Mode

This macro defines the list of SPI buffer mode.

SPI_FRAME_PULSE_DIRECTION Enumeration

Defines the list of SPI frame sync pulse direction.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_DIRECTION_OUTPUT,
  SPI_FRAME_PULSE_DIRECTION_INPUT
} SPI_FRAME_PULSE_DIRECTION;
```

Members

Members	Description
SPI_FRAME_PULSE_DIRECTION_OUTPUT	Frame sync pulse direction is output
SPI_FRAME_PULSE_DIRECTION_INPUT	Frame sync pulse direction is input

Description

SPI Frame sync pulse direction

This macro defines the list of SPI frame sync pulse direction.

SPI_FRAME_PULSE_EDGE Enumeration

Defines the list of SPI frame sync pulse edge.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK,
  SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK
} SPI_FRAME_PULSE_EDGE;
```

Members

Members	Description
SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK	Frame sync pulse coincides with first bit clock
SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK	Frame sync pulse precedes first bit clock

Description

SPI Frame sync pulse edge

This macro defines the list of SPI frame sync pulse edge.

SPI_FRAME_PULSE_POLARITY Enumeration

Defines the list of SPI frame sync pulse polarity.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH,
  SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW
} SPI_FRAME_PULSE_POLARITY;
```

Members

Members	Description
SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH	Frame sync pulse is active high
SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW	Frame sync pulse is active low

Description

SPI Frame sync pulse polarity

This macro defines the list of SPI frame sync pulse polarity.

SPI_FRAME_PULSE_WIDTH Enumeration

Defines the list of SPI frame sync pulse width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH,
  SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE
} SPI_FRAME_PULSE_WIDTH;
```

Members

Members	Description
SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH	Frame sync Pulse width as one word length wide
SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE	Frame sync Pulse width as one clock wide

Description

SPI Frame sync pulse width

This macro defines the list of SPI frame sync pulse width.

SPI_FRAME_SYNC_PULSE Enumeration

Data type defining the frame sync pulse counter values.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER
} SPI_FRAME_SYNC_PULSE;
```

Members

Members	Description
SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER	Generate a frame sync pulse on every 32 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER	Generate a frame sync pulse on every 16 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER	Generate a frame sync pulse on every 8 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER	Generate a frame sync pulse on every 4 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER	Generate a frame sync pulse on every 2 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER	Generate a frame sync pulse on every data character

Description

Frame Sync Pulse Counter enumeration

This data type defining the frame sync pulse counter values.

SPI_INPUT_SAMPLING_PHASE Enumeration

Defines the list of SPI data input sample phase.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE,
  SPI_INPUT_SAMPLING_PHASE_AT_END
} SPI_INPUT_SAMPLING_PHASE;
```

Members

Members	Description
SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE	SPI Data Input Sample phase in middle
SPI_INPUT_SAMPLING_PHASE_AT_END	SPI Data Input Sample phase at end

Description

SPI Data Input Sample Phase

This macro defines the list of SPI data input sample phase.

SPI_MODULE_ID Enumeration

Identifies the supported SPI modules.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_ID_1,
  SPI_ID_2,
  SPI_ID_3,
  SPI_ID_4,
  SPI_NUMBER_OF_MODULES
} SPI_MODULE_ID;
```

Members

Members	Description
SPI_ID_1	SPI Module 1 ID
SPI_ID_2	SPI Module 2 ID
SPI_ID_3	SPI Module 3 ID
SPI_ID_4	SPI Module 4 ID
SPI_NUMBER_OF_MODULES	Number of available SPI modules.

Description

SPI Module ID

This enumeration identifies the SPI modules that are available on a microcontroller. This is the superset of all the possible instances that might be available on Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

SPI_OUTPUT_DATA_PHASE Enumeration

Defines the list of SPI serial output data changes.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK,
  SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK
} SPI_OUTPUT_DATA_PHASE;
```

Members

Members	Description
SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK	Serial output data changes on transition from active clock state to idle clock state
SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK	Serial output data changes on transition from idle clock state to active clock state.

Description

SPI Output Data Phase

This macro defines the list of SPI serial output data changes.

SPI_PIN Enumeration

Data type defining the SPI pin.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_PIN_DATA_OUT,
  SPI_PIN_DATA_IN,
  SPI_PIN_SLAVE_SELECT
} SPI_PIN;
```

Members

Members	Description
SPI_PIN_DATA_OUT	SPI data output pin
SPI_PIN_DATA_IN	SPI data input pin
SPI_PIN_SLAVE_SELECT	SPI slave select pin

Description

SPI pin

This data type defining the SPI pin.

Remarks

This enumeration is processor specific.

Files

Files

Name	Description
plib_spi.h	SPI Peripheral Library Interface Header for common definitions.
help_plib_spi.h	Identifies the various enumerations in SPI modules supported

Description

This section lists the source and header files used by the library.

plib_spi.h

SPI Peripheral Library Interface Header for common definitions.

Functions

	Name	Description
	PLIB_SPI_AudioCommunicationWidthSelect	Selects the data width for the SPI audio communication.
	PLIB_SPI_AudioErrorDisable	Disables the SPI error.
	PLIB_SPI_AudioErrorEnable	Enables the SPI error.
	PLIB_SPI_AudioProtocolDisable	Audio protocol is disabled.
	PLIB_SPI_AudioProtocolEnable	Audio protocol is enabled.
	PLIB_SPI_AudioProtocolModeSelect	Selects the Audio Protocol mode.
	PLIB_SPI_AudioTransmitModeSelect	Selects the transmit audio data format.
	PLIB_SPI_BaudRateClockSelect	Selects the type of clock is used by the Baud Rate Generator.
	PLIB_SPI_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_SPI_BufferAddressGet	Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.
	PLIB_SPI_BufferClear	Clears the SPI buffer.
	PLIB_SPI_BufferRead	Returns the SPI buffer value.
	PLIB_SPI_BufferRead16bit	Returns 16-bit SPI buffer value.
	PLIB_SPI_BufferRead32bit	Returns 32-bit SPI buffer value.
	PLIB_SPI_BufferWrite	Write the data to the SPI buffer.
	PLIB_SPI_BufferWrite16bit	Writes 16-bit data to the SPI buffer.
	PLIB_SPI_BufferWrite32bit	Write 32-bit data to the SPI buffer.
	PLIB_SPI_ClockPolaritySelect	Enables clock polarity.
	PLIB_SPI_CommunicationWidthSelect	Selects the data width for the SPI communication.
	PLIB_SPI_Disable	Disables the SPI module.

	PLIB_SPI_Enable	Enables the SPI module.
	PLIB_SPI_ErrorInterruptDisable	Enables SPI error interrupts.
	PLIB_SPI_ErrorInterruptEnable	Enables SPI error interrupts
	PLIB_SPI_Exists16bitBuffer	Identifies whether the Buffer16bit feature exists on the SPI module.
	PLIB_SPI_Exists32bitBuffer	Identifies whether the Buffer32bit feature exists on the SPI module.
	PLIB_SPI_ExistsAudioCommunicationWidth	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsAudioErrorControl	Identifies whether the AudioErrorControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolControl	Identifies whether the AudioProtocolControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolMode	Identifies whether the AudioProtocolMode feature exists on the SPI module.
	PLIB_SPI_ExistsAudioTransmitMode	Identifies whether the AudioTransmitMode feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRate	Identifies whether the BaudRate feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRateClock	Identifies whether the BaudRateClock feature exists on the SPI module.
	PLIB_SPI_ExistsBuffer	Identifies whether the Buffer feature exists on the SPI module.
	PLIB_SPI_ExistsBusStatus	Identifies whether the BusStatus feature exists on the SPI module.
	PLIB_SPI_ExistsClockPolarity	Identifies whether the ClockPolarity feature exists on the SPI module.
	PLIB_SPI_ExistsCommunicationWidth	Identifies whether the CommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SPI module.
	PLIB_SPI_ExistsErrorInterruptControl	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOControl	Identifies whether the FIFOControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOCount	Identifies whether the FIFOCount feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOInterruptMode	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFramedCommunication	Identifies whether the FramedCommunication feature exists on the SPI module.
	PLIB_SPI_ExistsFrameErrorStatus	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseCounter	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseDirection	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseEdge	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulsePolarity	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseWidth	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	PLIB_SPI_ExistsInputSamplePhase	Identifies whether the InputSamplePhase feature exists on the SPI module.
	PLIB_SPI_ExistsMasterControl	Identifies whether the MasterControl feature exists on the SPI module.
	PLIB_SPI_ExistsOutputDataPhase	Identifies whether the OutputDataPhase feature exists on the SPI module.
	PLIB_SPI_ExistsPinControl	Identifies whether the PinControl feature exists on the SPI module.
	PLIB_SPI_ExistsReadDataSignStatus	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveBufferStatus	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveFIFOStatus	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	PLIB_SPI_ExistsSlaveSelectControl	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	PLIB_SPI_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferEmptyStatus	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferFullStatus	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.
	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.
	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusClear	Clears the SPI frame error flag.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.

	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.
	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.
	PLIB_SPI_TransmitUnderRunStatusClear	Clears the SPI transmit underrun flag.
	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.

Description

SPI Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SPI PLIB.

File Name

plib_spi.h

Company

Microchip Technology Inc.

help_plib_spi.h

Identifies the various enumerations in SPI modules supported

Enumerations

	Name	Description
	SPI_AUDIO_COMMUNICATION_WIDTH	Defines the list of SPI audio communication width.
	SPI_AUDIO_ERROR	Defines the list of SPI audio error.
	SPI_AUDIO_PROTOCOL	Data type defining the audio protocol mode.
	SPI_AUDIO_TRANSMIT_MODE	Defines the list of SPI transmit audio mode format.
	SPI_BAUD_RATE_CLOCK	Defines the list of SPI Baud Rate Generator (BRG).
	SPI_CLOCK_POLARITY	Defines the list of SPI clock polarity.
	SPI_COMMUNICATION_WIDTH	Defines the list of SPI communication width.
	SPI_ERROR_INTERRUPT	Defines the list of SPI error interrupts.
	SPI_FIFO_INTERRUPT	Defines the list of SPI Buffer Interrupt mode.
	SPI_FIFO_TYPE	Defines the list of SPI buffer mode.
	SPI_FRAME_PULSE_DIRECTION	Defines the list of SPI frame sync pulse direction.
	SPI_FRAME_PULSE_EDGE	Defines the list of SPI frame sync pulse edge.
	SPI_FRAME_PULSE_POLARITY	Defines the list of SPI frame sync pulse polarity.
	SPI_FRAME_PULSE_WIDTH	Defines the list of SPI frame sync pulse width.
	SPI_FRAME_SYNC_PULSE	Data type defining the frame sync pulse counter values.
	SPI_INPUT_SAMPLING_PHASE	Defines the list of SPI data input sample phase.

	SPI_MODULE_ID	Identifies the supported SPI modules.
	SPI_OUTPUT_DATA_PHASE	Defines the list of SPI serial output data changes.
	SPI_PIN	Data type defining the SPI pin.

Types

	Name	Description
	SPI_DATA_TYPE	Data type defining the SPI data size.

Description

This enumeration identifies the SPI modules that are available on a microcontroller. This is the superset of all the possible instances that might be available on Microchip microcontrollers.

File Name

help_plib_spi.h

Company

Microchip Technology Inc.

SQI Peripheral Library

This section describes the Serial Quad Interface (SQI) Peripheral Library.

Introduction

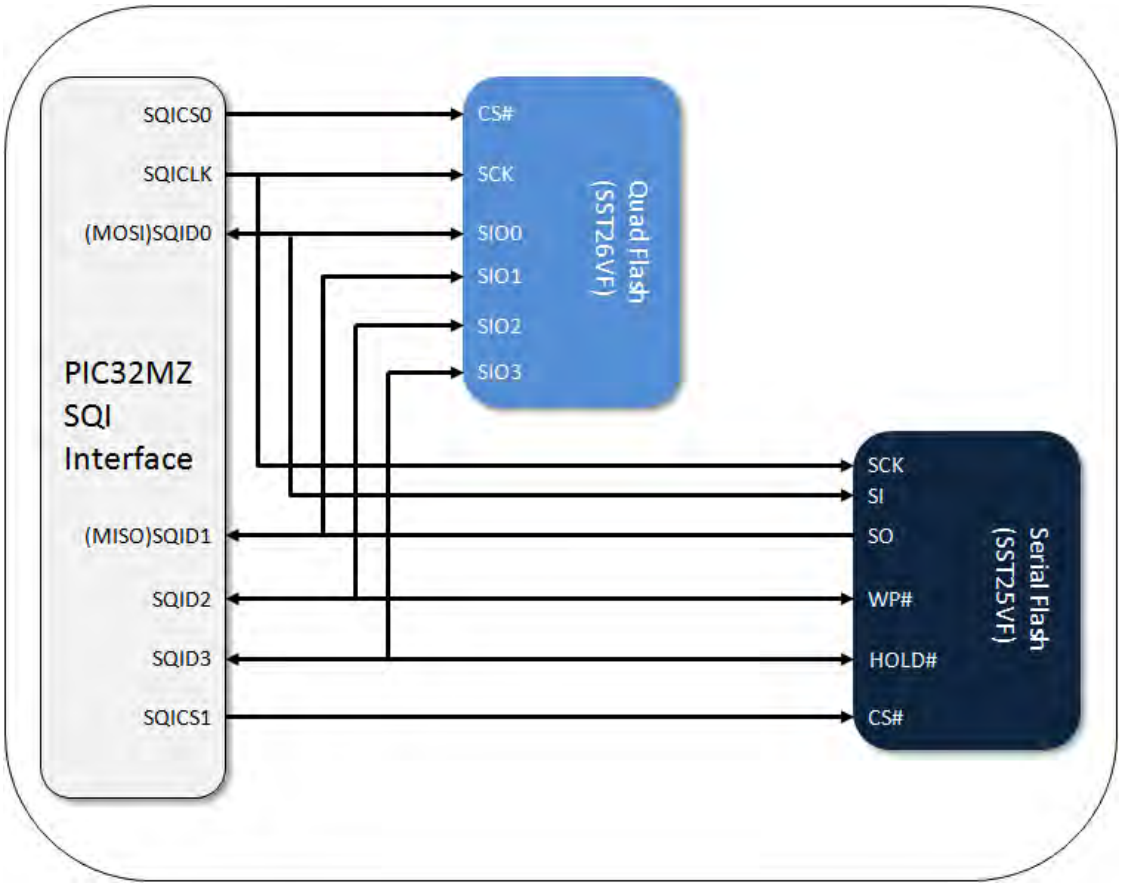
This library provides a low-level abstraction of the Serial Quad Interface (SQI) Peripheral that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the SQI module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The SQI module is a synchronous serial interface that provides access to serial Flash memories and other serial devices. The SQI bus interface consists of four data lines (SQID3-SQID0), a clock line (SQICLK), two select lines (SQICS0 and SQICS1) and the module supports single Lane (identical to SPI), dual Lane, and quad Lane interface modes. SQI operates in only master mode. The SQI module has configurable transmit and receive buffers, programmable baud rates through the internal clock divider, clock phase, and clock polarity control for efficient data operations. Transmit and receive buffers can be accessed through SQITXDATA and SQIRXDATA registers. Similarly, the control buffer can be accessed through the SQI1CON register and is mainly used to pipeline the operations.

The SQI module operates in three transfer modes: DMA, PIO, and XIP using three data flow modes:SPI Mode 0 and Mode 3, and a high-speed Serial Flash mode . All three modes use the control buffer to pipeline the command/data sequences on the SQI bus. Following diagrams describe typical SQI hardware interface and software flow. Figure 1 shows the typical hardware interface of SQI module.

Figure 1: Hardware Interface Diagram



Refer to **Section 46. "Serial Quad Interface (SQI)"** (DS60001244), which is available on the Microchip website for additional details on control register information and operation.

Using the Library

This topic describes the basic architecture of the SQI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_sqi.h`

The interface to the SQI peripheral library is defined in the `plib_sqi.h` header file, which is included by the peripheral library header file,

`peripheral.h`. Any C language source (`.c`) file that uses the SQI peripheral library must include `peripheral.h`.

Library File:

The SQI peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the SQI module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Before jumping into the software abstraction, it is important to understand the SQI hardware a little more in detail.

Figure 2 provides a glimpse of SQI hardware block diagram. As shown the diagram the features of SQI are divided into multiple logic blocks and the SQI peripheral library provides APIs to configure the various logic block thus the features.

The control and status SFRs block contains all the registers that are used to configure the SQI in a specific mode of operation. The PIO mode of operation of SQI requires a set of registers to be configured by the Host processor and writing to/ reading from the TXDATA and RXDATA registers. The XIP logic facilitates the SQI modules eXecute-In-Place mode of operation, where Host processor programs the set of registers that configure the XIP mode and the engine automatically reads the external Flash and the read data gets mapped into a direct memory region. Where as DMA mode requires the Host processor to program a set of buffer descriptors and instructs SQI to point to the address of those buffer descriptors for the reads and writes. As mentioned earlier there are 3 buffers; transmit, receive and control that are accessed through SQIxTXDATA, SQIxRXDATA, and SQIxCON registers, respectively.

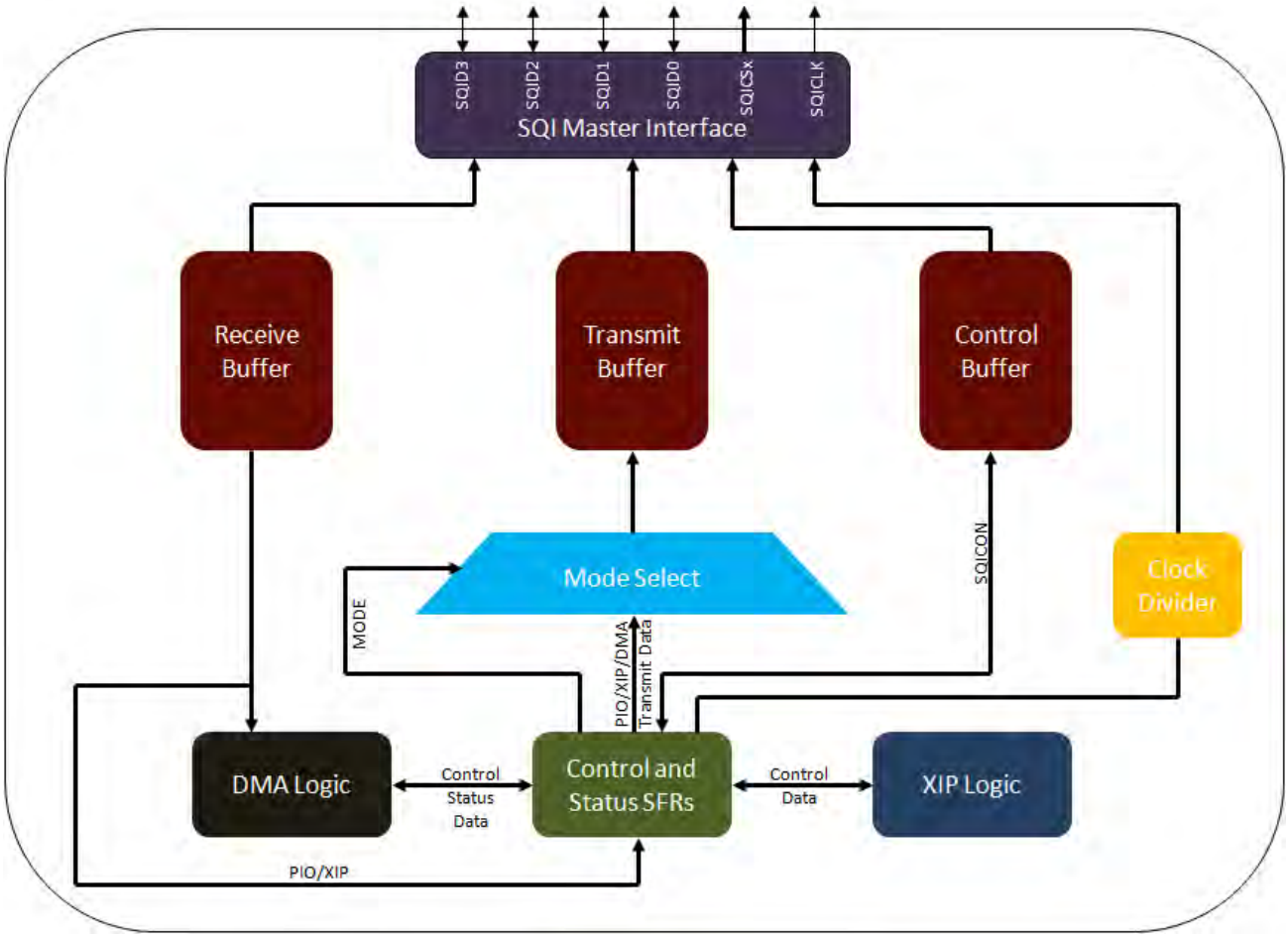


Figure 2: SQI Hardware Block Diagram

The SQI Peripheral Library takes requests from application software or the SQI device driver and controls the SQI hardware as per the inputs passed to the Library Interface. Figure 3 provides the SQI peripheral library software abstraction layer diagram.

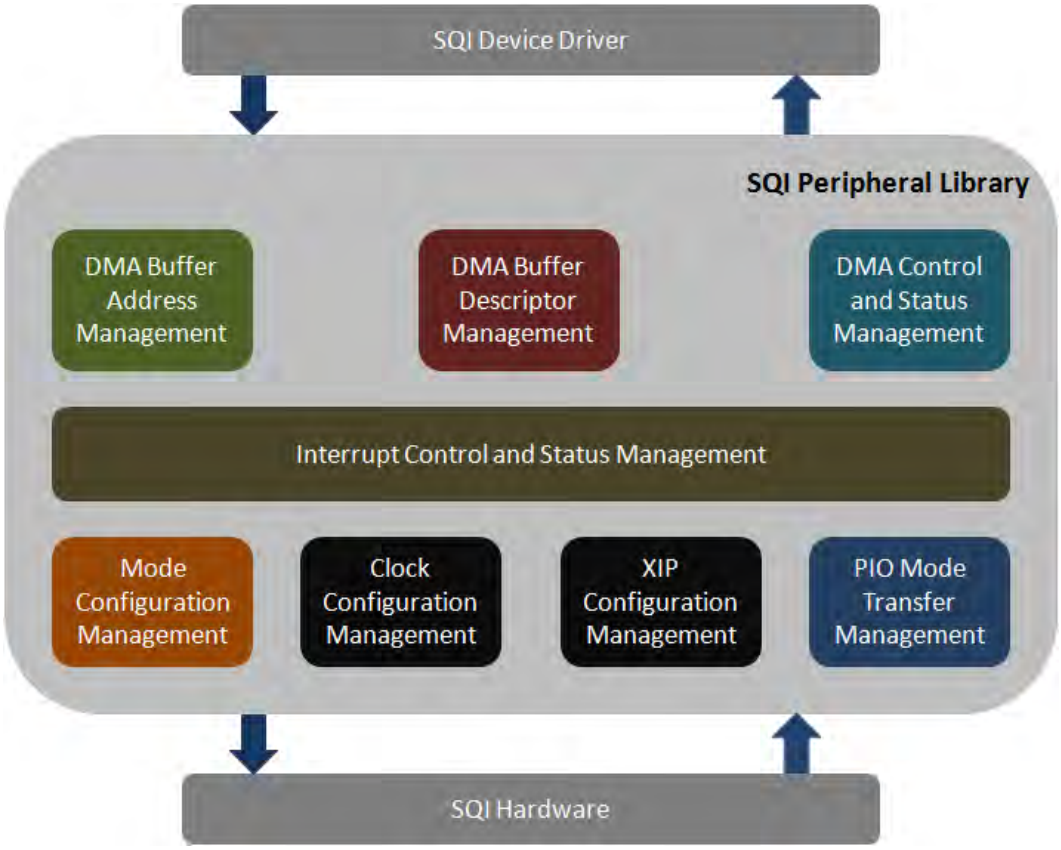


Figure 3: SQI Software Abstraction Model

The major components are SQI Peripheral Library are:

- Configuration Management
- Interrupt Control and Status Management
- DMA Mode Management

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SQI module.

Library Interface Section	Description
Mode Configuration Management Functions	These functions are used to set up transfer mode (DMA/XIP/PIO), data mode (SPI Mode 0/Mode 3/Serial Flash), lane mode (single/dual/quad) and XIP control registers.
Clock Configuration Management Functions	These functions are used to enable the clock, set up a clock divider and check the status of the clock to see if it is stable.
XIP Configuration Management Functions	These functions are used to set up parameters for XIP mode of operation.
PIO Mode Transfer Management Functions	These functions are used to set up TXDATA, RXDATA registers and status checks.
Interrupt Control and Status Management Functions	These functions are used to control the interrupts. SQI supports only status type of interrupts (no error interrupts).
DMA Buffer Address Management Functions	These functions are used to set up the address of the base descriptor, check the address of the current descriptor
DMA Buffer Descriptor Management Functions	These functions are used to set up and control 4 words (BD_CTRL, BD_STATUS (reserved), BD_BUF_ADDR and BD_NEXT_PTR)
DMA Control and Status Management Functions	These functions are used to control the buffer descriptor fetch process and check the status during the buffer descriptor fetches.

How the Library Works

Note: Not all features are available on all devices. Refer to the **"Serial Quad Interface (SQI)"** chapter in the specific device data sheet for availability.

Core Functionality

SQI modules core functionality is to transfer data between the serial device (mostly quad Flash) attached and the PIC microcontroller. To achieve this, SQI module uses 3 transfer modes: DMA, PIO and XIP. Each transfer mode can be configured to use any of the 3 available data flow modes (Mode 0, Mode 3 and Serial Flash Mode) to achieve the transfers. DMA and PIO modes are typically used to transfer data and XIP mode is used for code execution.

Each mode of operation requires setting-up the clock, selecting data flow mode and selecting lane mode and other parameters as required by the device driver or application. SQI peripheral library includes complete set of API necessary to perform these operations.

The following sections describe the SQI core functionality in each mode of operation.

XIP Mode

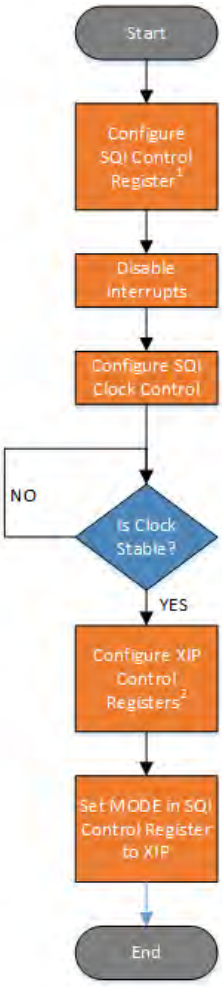
Describes XIP mode.

Description

XIP (eXecute-In-Place) mode is mostly used to execute the code out of the attached serial Flash device. However, this mode can also be used to transfer data as deemed necessary. In order for SQI to operate in XIP mode, host processor would have to set up the two SFRs dedicated to the XIP mode in addition to SQI configuration and clock control SFRs. Please refer to the specific data sheet and family reference manual for the configuration details.

Please note that all the XIP configuration values depend on the parameters of the interfaced serial Flash memory device. In XIP mode, reading values of any other SFRs return the contents of the serial Flash device, the way it is mapped.

The following figure shows the XIP mode software flow.



Notes:
1. This step configures the SQI configuration fields, SQIEN, CSEN, and DATAEN, with the exception of MODE<1:0>.
2. There are two XIP control registers: SQI1XCON1 and SQI1XCON2.

Example:

```
// Example is written to access SST26VF032 Flash device
```

```
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_HS_READ    0x0B
#define SST26VF_MODE_CODE  0x0
#define SST26VF_MODE_BYTES 0x0

// Set up SQI Configuration Bits
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1, SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1, SQI_DATA_OEN_QUAD);

// Disable All Interrupts
PLIB_SQI_InterruptDisable(SQI_ID_1, SQI_ALL_INT);
PLIB_SQI_InterruptSignalDisable(SQI_ID_1, SQI_ALL_INT);

// Set up SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_1);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

// Configure XIP Control 1 SFR
PLIB_SQI_XIPControlWord1Set (SQI_ID_1,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SST26VF_HS_READ,
                             ADDR_BYTES_3,
                             DUMMY_BYTES_2
                            );

// Configure XIP Control 2 SFR
PLIB_SQI_XIPControlWord2Set (SQI_ID_1,
                             SST26VF_MODE_CODE,
                             MODE_BYTES_0,
                             SQI_CS_0
                            );

// Set Transfer Mode to XIP
PLIB_SQI_TransferModeSet(SQI_ID_1, SQI_XFER_MODE_XIP);

// The SQI is now in XIP mode and should start fetching the code
```

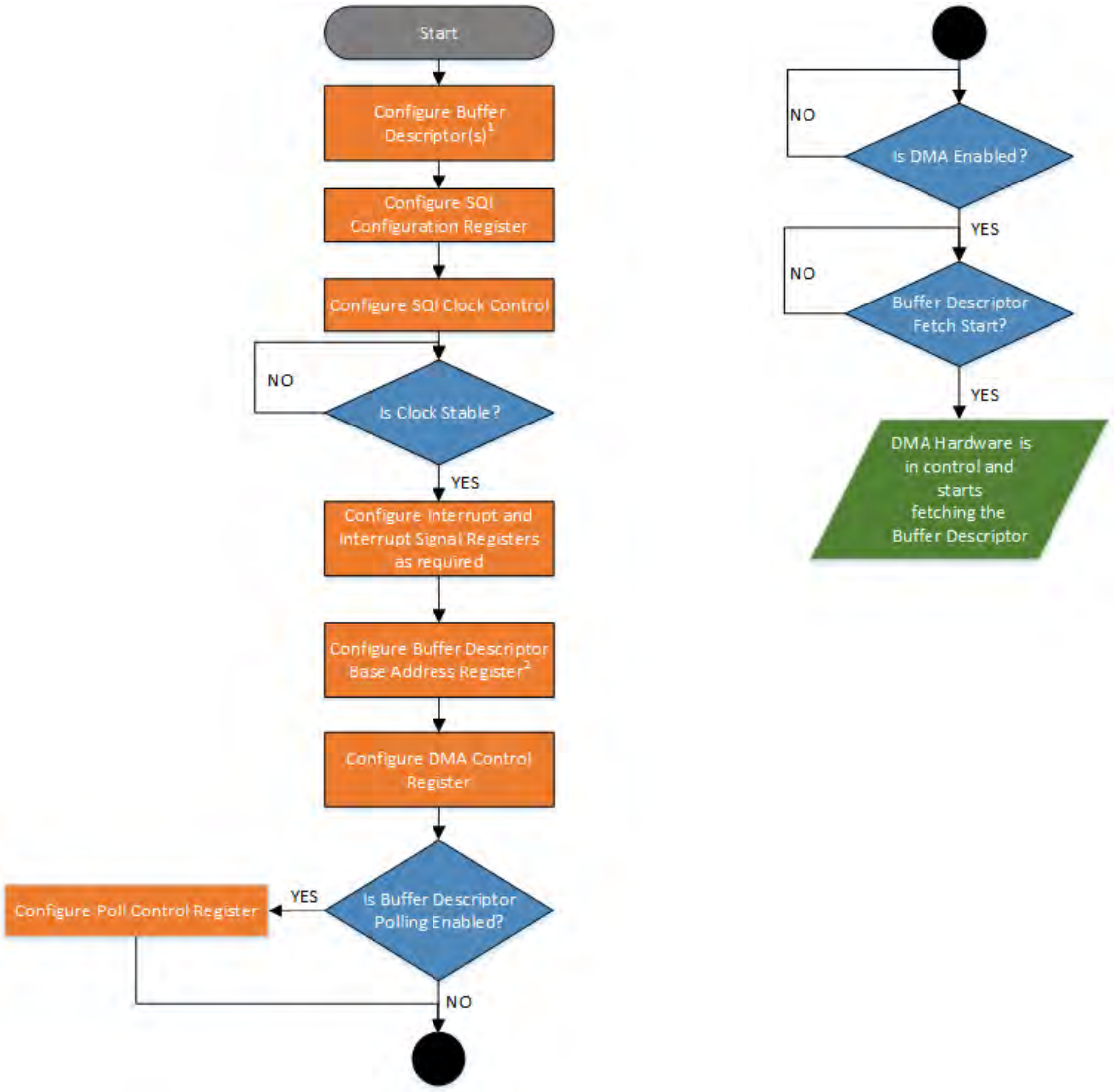
DMA Mode

Describes DMA mode.

Description

DMA mode is used for higher throughput data transfers. In DMA mode, the SQI DMA engine controls the data transfers off-loading the host processor. However, the host processor would have to configure the buffer descriptor and some SQI control SFRs to initiate the DMA process. Once the transactions are in progress, the DMA will be in control and the host processor can get the control through the interrupts or by disabling the DMA.

The following figure shows the DMA mode software flow.



Notes:

1. Buffer descriptor is a data structure in memory containing 4 words (BD_CTRL, BD_STAT, BD_BUFADDR and BD_NXTPTR).
2. Base address register points to first buffer descriptor in the chain.

Example:

```
// Example is written to access SST26VF032 Flash device
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_EQIO       0x38
#define SST26VF_HS_READ    0x0B
#define SST26VF_MODE_CODE  0x0
#define SST26VF_MODE_BYTES 0x0
#define NUMBER_OF_BDs      5
#define POLL_CON_VALUE     10

{

// Set up the Buffer Descriptors
for ( i=0; i <(NUMBER_OF_BDs-1); i++)
{
// Set up Buffer Descriptors. Handle structures as application
// needs.

// Configure SQI Control Fields
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1,SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1,SQI_DATA_OEN_2);
```

```
PLIB_SQI_BurstEnable(SQI_ID_1);
PLIB_SQI_SerialModeSet(SQI_ID_1);
PLIB_SQI_TransferModeSet(SQI_ID_1,SQI_XFER_MODE_PIO);

// Configure the SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_4);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

//Enable Buffer Descriptor Done and Packet Done Interrupts
PLIB_SQI_InterruptEnable(SQI_ID_1,BDDONE);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1,BDDONE);
PLIB_SQI_InterruptEnable(SQI_ID_1,PKTCOMP);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1,PKTCOMP);

// Send EQIO(0x38) Command to SST26VF032 to Enable Quad Lane Mode
// Note: These steps are not in the flow as these are specific to
// one Flash Device (SST26VF032)
PLIB_SQI_ChipSelectSet(SQI_ID_1,SQI_CS_0);
PLIB_SQI_LaneModeSet(SQI_ID_1,SQI_SINGLE_LANE);
PLIB_SQI_TransferCommandSet(SQI_ID_1,SQI_CMD_TRANSMIT);
// Transfers 1 Command Byte, No Address/Dummy Bytes
PLIB_SQI_TransferCountSet(SQI_ID_1, 0x1);
PLIB_SQI_TransmitterDataSend(SQI_ID_1, SST26VF_EQIO);

// Set the SQI in DMA Mode
PLIB_SQI_TransferModeSet(SQI_ID_1,SQI_XFER_MODE_DMA);

// Set up the Base Buffer Descriptor Address
if (!PLIB_SQI_DMAIsActive(SQI_ID_1))
    PLIB_SQI_DMABDBaseAddressSet(SQI_ID_1, (void *) (&MY_BASE_BD_STRUCT));

//Set up DMA Control Register
PLIB_SQI_DMABDEnable(SQI_ID_1);
PLIB_SQI_DMABDPollEnable(SQI_ID_1);

// Set up DMA Polling
If (PLIB_SQI_DMABDPollIsEnabled(SQI_ID_1))
   // Poll DMA every 10 cycles
   PLIB_SQI_PollControlSet(SQI_ID_1, MY_POLL_CONTROL_VALUE);

// Start the DMA Process - DMA Engine is now in Control
If (PLIB_SQI_DMAIsEnabled(SQI_ID_1))
{
   if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
      PLIB_SQI_DMABDFetchStart(SQI_ID_1);

}

}
```

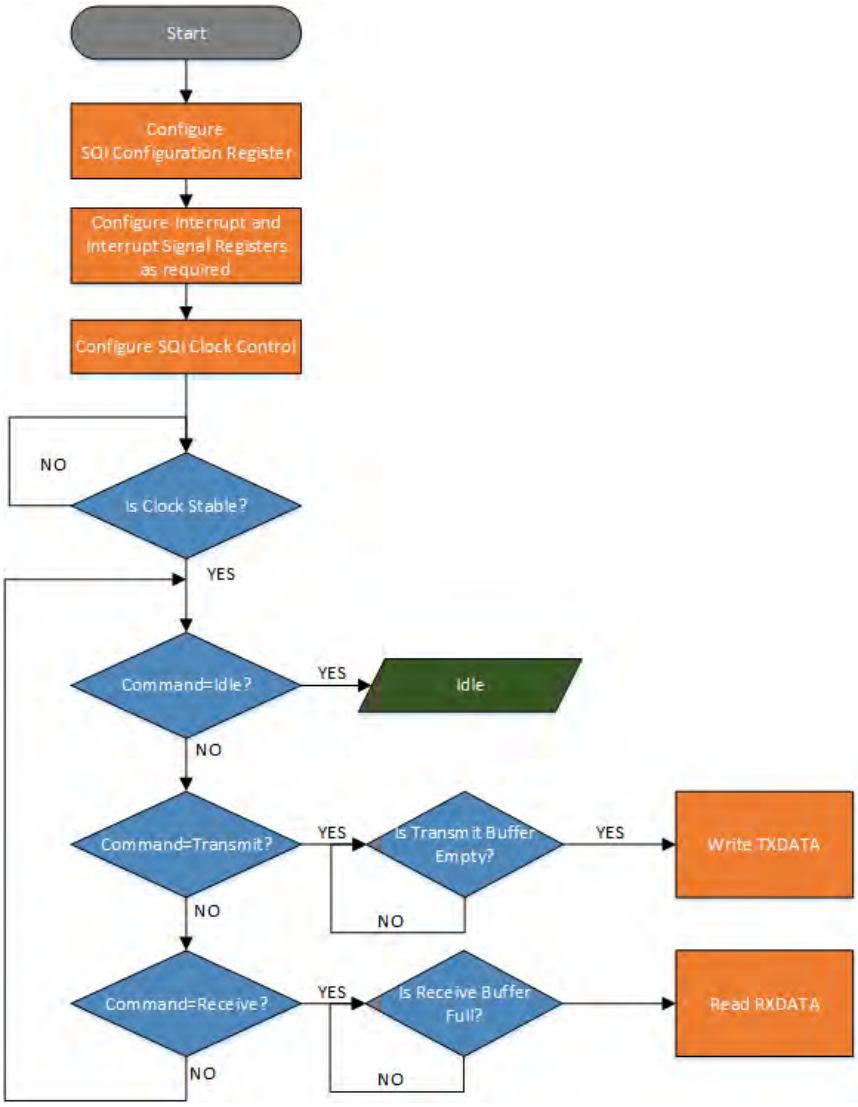
PIO Mode

Describes PIO mode.

Description

PIO mode is mostly used for data transfers. In PIO mode, the SQI module is entirely under the control of host processor. For the SQI module to operate in PIO mode, the host processor would have to set up the configuration, control, and clock control SFRs. Once the transactions are in progress, the host processor can use interrupts and status flags to control the operation. Please refer to the specific device data sheet and family reference manual for the configuration details.

The following figure shows the PIO mode software flow.



Example:

```
// Example is written to access SST26VF032 Flash device
// Example assumes the Flash Device is connected to Chip Select 0
// Example reads the device ID of the SST26VF032

//Global defines
#define FALSE                0
#define TRUE                 1
#define SST26VF_EQIO         0x38
#define SST26VF_HS_READ      0x0B
#define SST26VF_QJEDID_READ  0x9F
#define SST26VF_JEDEC_ID     0xBF2602

{
int32_t jedecID;

// Set up SQI Configuration (SQI1CFG) Register
PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
                       1,
                       SQI_CS_OEN_0,
                       SQI_DATA_OEN_QUAD,
                       0,
                       1,
                       0,
                       0,
                       0,
                       SQI_DATA_FORMAT_MSBF,
                       SQI_DATA_MODE_3,
```

```
                        SQI_XFER_MODE_PIO
                        );

PLIB_SQI_ControlBufferThresholdSet(SQI_ID_1,4);

// Enable Specific Interrupts
PLIB_SQI_InterruptEnable(SQI_ID_1,TXEMPTY|RXFULL/TXTHRIE|RXTHRIE);

// Set up SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_1);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

// Set up SQI Transmit Command Threshold
PLIB_SQI_TxCommandThresholdSet(SQI_ID_1,3);

// Set up SQI Receive Buffer Threshold
PLIB_SQI_TxBufferThresholdIntSet(SQI_ID_1,3);

// Configure SQI Control Fields and Send EQIO(0x38) Command to
// SST26VF032 to Enable Quad Lane Mode
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                        1,
                        SQI_CS_1,
                        SQI_LANE_SINGLE,
                        SQI_CMD_TRANSMIT,
                        1
                        );
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                        0,
                        SQI_CS_1,
                        SQI_LANE_QUAD,
                        SQI_CMD_TRANSMIT,
                        1
                        );
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                        0,
                        SQI_CS_1,
                        SQI_LANE_QUAD,
                        SQI_CMD_RECEIVE,
                        3
                        );
if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1,TXEMPTY) )
   PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_EQIO);

//Configure SQI to read JEDEC-ID in Quad Mode
if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1,TXEMPTY) )
   PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_QJEDID_READ);

if (PLIB_SQI_InterruptFlagGet(SQI_ID_1,RXFULL))
   jedecID = PLIB_SQI_ReceiveData(SQI_ID_1);

if (jedecID == SST26VF_JEDEC_ID)
return TRUE;
else
return FALSE;
}
```

Configuring the Library

The library is configured for the supported SQI module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Mode Configuration Management Functions

	Name	Description
	PLIB_SQI_BurstEnable	Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.
	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
	PLIB_SQI_ControlWordGet	Get the SQI Control Word.
	PLIB_SQI_ControlWordSet	Sets the SQI Control Word.
	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI Chip Select pins.
	PLIB_SQI_DataFormatGet	Returns the data format.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First (LSBF)..
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation.
	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_HoldClear	Clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	Gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	Sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).
	PLIB_SQI_NumberOfReceiveBufferReads	Returns the number of receive buffer reads.
	PLIB_SQI_ReceiveData	Reads the data from the receive buffer.
	PLIB_SQI_ReceiveLatchDisable	Disables the receive latch so receive data is discarded when in transmit mode.
	PLIB_SQI_ReceiveLatchEnable	Enables the receive latch so receive data is latched during transmit mode.
	PLIB_SQI_ReceiveLatchGet	Returns the receive latch status in transmit mode.
	PLIB_SQI_SoftReset	Issues a soft reset to the SQI module.
	PLIB_SQI_TransferModeGet	Returns the SQI transfer mode of operation.
	PLIB_SQI_TransferModeSet	Sets the SQI transfer mode of operation.
	PLIB_SQI_TransmitData	Writes data into the SQI transmit buffer.
	PLIB_SQI_WriteProtectClear	Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	PLIB_SQI_WriteProtectGet	Gets the state of the write-protect feature on SQID2.
	PLIB_SQI_WriteProtectSet	Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.
	PLIB_SQI_ConBufferSoftReset	Issues a soft reset to the SQI control buffer.
	PLIB_SQI_RxBufferSoftReset	Issues a soft reset to the SQI receive buffer.
	PLIB_SQI_TxBufferSoftReset	Issues a soft reset to the SQI transmit buffer.
	PLIB_SQI_TapDelaySet	Sets the tap delays.
	PLIB_SQI_DDRModeGet	Return the SQI data rate mode (single/double) value.
	PLIB_SQI_DDRModeSet	Sets SQI communication to DDR mode.

b) Clock Configuration Management Functions

	Name	Description
	PLIB_SQI_ClockDividerSet	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
	PLIB_SQI_ClockEnable	Enables the SQI transfer clock.
	PLIB_SQI_ClockIsStable	Returns SQI transfer clock state.

c) XIP Configuration Management Functions

	Name	Description
	PLIB_SQI_XIPAddressBytesGet	Returns the number of address bytes.
	PLIB_SQI_XIPAddressBytesSet	Sets the number of address bytes.
	PLIB_SQI_XIPChipSelectGet	Returns the current active Chip Select.

	PLIB_SQI_XIPChipSelectSet	Activates the Chip Select in XIP mode.
	PLIB_SQI_XIPControlWord1Get	Get the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord1Set	Sets the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord2Get	Gets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord2Set	Sets the XIP mode Control Word 2.
	PLIB_SQI_XIPDummyBytesGet	Sets the number of dummy bytes.
	PLIB_SQI_XIPDummyBytesSet	Sets the number of dummy bytes.
	PLIB_SQI_XIPLaneModeSet	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	PLIB_SQI_XIPModeBytesGet	Returns the number of bytes used for mode code command.
	PLIB_SQI_XIPModeBytesSet	Allocates the bytes for mode code command.
	PLIB_SQI_XIPModeCodeGet	Returns the mode code op-code.
	PLIB_SQI_XIPModeCodeSet	Sets the mode code command.
	PLIB_SQI_XIPReadOpcodeGet	Returns the read op code in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read op code in XIP mode.
	PLIB_SQI_XIPControlWord3Get	Gets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord3Set	Sets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord4Get	Gets the XIP mode Control Word 4.
	PLIB_SQI_XIPDDRModeSet	Selects the rate mode (SDR/DDR) for different transactions in XIP mode.
	PLIB_SQI_XIPSDRCommandDDRDataGet	Returns the SQI command in SDR mode and Data in DDR mode bit value.
	PLIB_SQI_XIPSDRCommandDDRDataSet	Sets the SQI command in SDR mode.
	PLIB_SQI_XIPControlWord4Set	Sets the XIP mode Control Word 4.

d) PIO Mode Transfer Management Functions

	Name	Description
	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
	PLIB_SQI_ChipSelectDeassertDisable	Clears the Chip Select deassert.
	PLIB_SQI_ChipSelectDeassertEnable	Sets the Chip Select deassert.
	PLIB_SQI_ChipSelectGet	Returns the Chip Select that is currently active.
	PLIB_SQI_ChipSelectSet	Activates the Chip Select.
	PLIB_SQI_ConfigWordGet	Gets the SQI Configuration Word.
	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
	PLIB_SQI_ReceiveBufferIsUnderrun	Returns the status of receive buffer.
	PLIB_SQI_RxBufferThresholdGet	Returns the receive command threshold.
	PLIB_SQI_RxBufferThresholdIntGet	Sets the receive buffer threshold interrupt.
	PLIB_SQI_RxBufferThresholdIntSet	Sets the receive buffer threshold for interrupt.
	PLIB_SQI_RxBufferThresholdSet	Sets the receive command threshold.
	PLIB_SQI_TransferDirectionGet	Returns the transfer command.
	PLIB_SQI_TransferDirectionSet	Sets the transfer command.
	PLIB_SQI_TransmitBufferFreeSpaceGet	Returns the number of transmit buffer words available.
	PLIB_SQI_TransmitBufferHasOverflowed	Returns the current status of the transmit buffer.
	PLIB_SQI_TxBufferThresholdGet	Returns the transmit command threshold value.
	PLIB_SQI_TxBufferThresholdIntGet	Returns the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdIntSet	Sets the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdSet	Sets the transmit command threshold.

e) Interrupt Control and Status Management Functions

	Name	Description
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the passed interrupt source.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state.
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the passed interrupt signal source.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state.

	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_CommandStatusGet	Returns the SQI command (transmit/receive/idle).
	PLIB_SQI_ConBufWordsAvailable	Returns the number of control buffer words available.

f) DMA Buffer Address Management Functions

	Name	Description
	PLIB_SQI_DMABDBaseAddressGet	Returns the address of the base buffer descriptor.
	PLIB_SQI_DMABDBaseAddressSet	Sets the address of the base buffer descriptor.
	PLIB_SQI_DMABDCurrentAddressGet	Returns the address of the current buffer descriptor in process.

g) DMA Buffer Descriptor Management Functions

	Name	Description
	PLIB_SQI_DMABDControlWordGet	Returns Current Buffer Descriptor Control Word Information.
	PLIB_SQI_DMABDReceiveBufferCountGet	Returns the receive buffer space in bytes.
	PLIB_SQI_DMABDReceiveBufferLengthGet	Returns the receive length in bytes.
	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDFetchStop	Stops the DMA buffer descriptor fetch process.

h) DMA Control and Status Management Functions

	Name	Description
	PLIB_SQI_DMABDFetchStart	Starts the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns whether or not the DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollIsEnabled	Returns whether or not the DMA buffer descriptor poll is enabled.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns whether or no the DMA process has started.
	PLIB_SQI_DMAIsEnabled	Returns whether or not DMA is enabled.

i) Other Functions

	Name	Description
	PLIB_SQI_StatusCheckSet	Sets the Flash status check related control bits.

j) Feature Existence Functions

	Name	Description
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module.
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxBufCount	Identifies whether the BDTxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxLength	Identifies whether the BDTxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxState	Identifies whether the BDTxState feature exists on the SQI module.
	PLIB_SQI_ExistsBurstControl	Identifies whether the BurstControl feature exists on the SQI module.
	PLIB_SQI_ExistsChipSelect	Identifies whether the ChipSelect feature exists on the SQI module.

	PLIB_SQI_ExistsClockControl	Identifies whether the ClockControl feature exists on the SQI module.
	PLIB_SQI_ExistsClockDivider	Identifies whether the ClockDivider feature exists on the SQI module.
	PLIB_SQI_ExistsClockReady	Identifies whether the ClockReady feature exists on the SQI module.
	PLIB_SQI_ExistsConBufThreshold	Identifies whether the ConBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsConfigWord	Identifies whether the ConfigWord feature exists on the SQI module.
	PLIB_SQI_ExistsControlWord	Identifies whether the ControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsCSDeassert	Identifies whether the CSDeassert feature exists on the SQI module.
	PLIB_SQI_ExistsCSOutputEnable	Identifies whether the CSOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataFormat	Identifies whether the DataFormat feature exists on the SQI module.
	PLIB_SQI_ExistsDataModeControl	Identifies whether the DataModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsDataOutputEnable	Identifies whether the DataOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataPinStatus	Identifies whether the DataPinStatus feature exists on the SQI module.
	PLIB_SQI_ExistsDmaEnable	Identifies whether the DMAEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDMAEngineBusy	Identifies whether the DMAEngineBusy feature exists on the SQI module.
	PLIB_SQI_ExistsDMAProcessInProgress	Identifies whether the DMAProcessInProgress feature exists on the SQI module.
	PLIB_SQI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SQI module.
	PLIB_SQI_ExistsHoldPinControl	Identifies whether the HoldPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptControl	Identifies whether the InterruptControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptSignalControl	Identifies whether the InterruptSignalControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptStatus	Identifies whether the InterruptStatus feature exists on the SQI module.
	PLIB_SQI_ExistsLaneMode	Identifies whether the LaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsReceiveLatch	Identifies whether the ReceiveLatch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferCount	Identifies whether the RxBufferCount feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufIntThreshold	Identifies whether the RxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufThreshold	Identifies whether the RxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxData	Identifies whether the RxData feature exists on the SQI module.
	PLIB_SQI_ExistsRxUnderRun	Identifies whether the RxUnderRun feature exists on the SQI module.
	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module.
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module.
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module.
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module.
	PLIB_SQI_ExistsCommandStatus	Identifies whether the CommandStatus feature exists on the SQI module.
	PLIB_SQI_ExistsConBufAvailable	Identifies whether the ConBufWordsAvailable feature exists on the SQI module.
	PLIB_SQI_ExistsConBufferSoftReset	Identifies whether the control buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsDDRMode	Identifies whether the DDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsDMABDFetch	Identifies whether the DMABDFetch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferSoftReset	Identifies whether the receive buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsStatusCheck	Identifies whether the StatusCheckSet feature exists on the SQI module.
	PLIB_SQI_ExistsTapDelay	Identifies whether the TapDelay feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferSoftReset	Identifies whether the transmit buffer soft reset feature exists on the SQI module.

	PLIB_SQI_ExistsXIPControlWord3	Identifies whether the XIPControlWord3 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord4	Identifies whether the XIPControlWord4 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPDDRMode	Identifies whether the XIPDDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPSDRCommandDDRData	Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.

k) Data Types and Constants

	Name	Description
	SQI_ADDR_BYTES	Defines the list of SQI address bytes.
	SQI_BD_CTRL_WORD	Defines the list of SQI Buffer Descriptor control word.
	SQI_BD_STATE	Defines the list of SQI Buffer Descriptor states.
	SQI_CLK_DIV	Defines the list of SQI Clock Divider values.
	SQI_CS_NUM	Defines the list of SQI Chip Selects.
	SQI_CS_OEN	Defines the list of SQI Chip Selects on which output is enable.
	SQI_DATA_FORMAT	Defines the Data Format Options available (LSBF/MSBF).
	SQI_DATA_MODE	Defines the list of SQI Data modes.
	SQI_DATA_OEN	Defines the list of SQI Data pins on which output is enabled.
	SQI_DATA_TYPE	Data type defining the SQI Data size.
	SQI_DUMMY_BYTES	Defines the list of SQI dummy bytes.
	SQI_INTERRUPTS	Defines the list of SQI interrupts.
	SQI_LANE_MODE	Defines the list of SQI Lane modes (single/dual/quad).
	SQI_MODE_BYTES	Defines the list of SQI mode bytes.
	SQI_MODULE_ID	Identifies the supported SQI modules.
	SQI_XFER_CMD	Defines the list of SQI transfer commands.
	SQI_XFER_MODE	Defines the list of SQI Transfer modes.

Description

This section describes the Application Programming Interface (API) functions of the SQI Peripheral Library.

Refer to each section for a detailed description.

a) Mode Configuration Management Functions

PLIB_SQI_BurstEnable Function

Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.

File

plib_sqi.h

C

```
void PLIB_SQI_BurstEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables burst mode for higher throughput. Burst mode should always be enabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_BurstEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_BurstEnable (SQI_MODULE_ID index)

PLIB_SQI_ClockDisable Function

Disables the SQI transfer clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQI transfer clock (divided clock).

Remarks

SQICLK is disabled.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ClockDisable (SQI_MODULE_ID index)

PLIB_SQI_ControlBufferThresholdGet Function

Returns the transmit buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_ControlBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Control buffer threshold space.

Description

This function returns the threshold value for the control buffer in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t conBufThres = PLIB_SQI_ControlBufferThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ControlBufferThresholdGet(SQI_MODULE_ID index)

PLIB_SQI_ControlBufferThresholdSet Function

Sets the control buffer threshold value.

File

plib_sqi.h

C

```
void PLIB_SQI_ControlBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the control buffer threshold value in bytes, that is used to signal control buffer threshold interrupts.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_CONBUF_THRESHOLD is the threshold value.
PLIB_SQI_ControlBufferThresholdSet(MY_SQI_INSTANCE, MY_CONBUF_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Control buffer threshold

Function

void PLIB_SQI_ControlBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_ControlWordGet Function

Get the SQI Control Word.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ControlWordGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI Control Word.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t sqiCon = PLIB_SQI_ControlWordGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ControlWordGet (SQI_MODULE_ID index)

PLIB_SQI_ControlWordSet Function

Sets the SQI Control Word.

File

plib_sqi.h

C

```
void PLIB_SQI_ControlWordSet(SQI_MODULE_ID index, bool csDeassert, SQI_CS_NUM csNum, SQI_LANE_MODE
laneMode, SQI_XFER_CMD command, uint16_t xferCount);
```

Returns

None.

Description

This function sets the SQI Control Word. This function is a combination of several functions in case the driver plans to write the complete Control Word. In PIO mode, the Control word is before each transfer

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                        1,
                        SQI_CS_1,
                        SQI_LANE_QUAD,
                        SQI_CMD_TRANSMIT,
                        5
                        );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

csDeassert	Chip select deassert after transfer
csNum	Active Chip Select number (0/1)
laneMode	SQI lane mode (QUAL/DUAL/SINGLE)
command	Transfer command (TRANSMIT/RECIEVE/IDLE)
xferCount	Number of bytes to be transmitted/received

Function

void PLIB_SQI_ControlWordSet(SQI_MODULE_ID index,
bool csDeassert,
SQI_CS_NUM csNum,
SQI_LANE_MODE laneMode,
SQI_XFER_CMD command,
uint16_t xferCount
)

PLIB_SQI_CSOutputEnableSelect Function

Selects the output enables on SQI Chip Select pins.

File

plib_sqi.h

C

```
void PLIB_SQI_CSOutputEnableSelect(SQI_MODULE_ID index, SQI_CS_OEN csPins);
```

Returns

None.

Description

This function enables the selected SQI chip selects as outputs.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_CSOutputEnableSelect(MY_SQI_INSTANCE, SQI_CS_OEN_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csPins	Chip select pins for which outputs are enabled

Function

void PLIB_SQI_CSOutputEnableSelect(SQI_MODULE_ID index, SQI_CS_OEN csPins)

PLIB_SQI_DataFormatGet Function

Returns the data format.

File

plib_sqi.h

C

```
SQI_DATA_FORMAT PLIB_SQI_DataFormatGet(SQI_MODULE_ID index);
```

Returns

- true - Data Format is LSBF (i.e., LITTLE-ENDIAN)
- false - Data Format is MSBF

Description

This function returns the SQI data format (LSBF or MSBF).

Remarks

Typical data format in most of systems is LITTLE-ENDIAN.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_DATA_FORMAT dataFormat = PLIB_SQI_DataFormatGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_DATA_FORMAT PLIB_SQI_DataFormatGet(SQI_MODULE_ID index)

PLIB_SQI_DataFormatSet Function

Sets the data format to Least Significant Bit First (LSBF)..

File

plib_sqi.h

C

```
void PLIB_SQI_DataFormatSet(SQI_MODULE_ID index, SQI_DATA_FORMAT dataformat);
```

Returns

None.

Description

This function sets the SQI data format to LSBF (i.e., LITTLE-ENDIAN).

Remarks

Typical data format in most of the Systems is LITTLE ENDIAN.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DataFormatSet(MY_SQI_INSTANCE, SQI_DATA_FORMAT_LSBF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DataFormatSet(SQI_MODULE_ID index, SQI_DATA_FORMAT dataformat)

PLIB_SQI_DataModeSet Function

Sets the SQI data mode of operation.

File

plib_sqi.h

C

```
void PLIB_SQI_DataModeSet(SQI_MODULE_ID index, SQI_DATA_MODE mode);
```

Returns

None.

Description

This function sets the data mode to be SPI Mode 0, SPI Mode 3, or Serial Flash.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_DATA_MODE_3 is an enum element.
PLIB_SQI_DataModeSet(MY_SQI_INSTANCE, SQI_DATA_MODE_SF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Data mode (Mode 0/Mode 3)

Function

void PLIB_SQI_DataModeSet(SQI_MODULE_ID index, SQI_DATA_MODE mode)

PLIB_SQI_DataOutputEnableSelect Function

Selects the output enables on SQI data outputs.

File

plib_sqi.h

C

```
void PLIB_SQI_DataOutputEnableSelect(SQI_MODULE_ID index, SQI_DATA_OEN dataPins);
```

Returns

None.

Description

This function enables the selected SQI data lines as outputs.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DataOutputEnableSelect(MY_SQI_INSTANCE, SQI_DATA_OEN_DUAL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPins	Data pins for which outputs are enabled

Function

void PLIB_SQI_DataOutputEnableSelect (SQI_MODULE_ID index, SQI_DATA_OEN dataPins)

PLIB_SQI_Disable Function

Disables the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_Disable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQI module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_Disable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_Disable (SQI_MODULE_ID index)

PLIB_SQI_Enable Function

Enables the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_Enable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQI module.

Remarks

The SQICLK, SQICSx, SQID0, SQID1, SQID2, and SQID3 pins must be assigned to available RPn pins before use.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_Enable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_Enable(SQI_MODULE_ID index)

PLIB_SQI_HoldClear Function

Clears the hold function to be disabled on SQID3 in single and dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_HoldClear(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets SQID3 to be controlled by SQI for normal data operation.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_HoldClear(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_HoldClear (SQI_MODULE_ID index)

PLIB_SQI_HoldGet Function

Gets the status of HOLD function on SQID3 pin.

File

plib_sqi.h

C

```
bool PLIB_SQI_HoldGet(SQI_MODULE_ID index);
```

Returns

SQID3 as HOLD pin value.

Description

This function gets the status of HOLD function on SQID3 pin.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool holdStatus = PLIB_SQI_HoldGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_HoldGet (SQI_MODULE_ID index)

PLIB_SQI_HoldSet Function

Sets the hold function to be enabled on SQID3 in single or dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_HoldSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets the SQID3 pin to HIGH/LOW to be be used for hold function in single and dual lane modes for supported Flash memories.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_HoldSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_HoldSet (SQI_MODULE_ID index)

PLIB_SQI_LaneModeGet Function

Returns the lane mode (single/dual/quad).

File

plib_sqi.h

C

```
SQI_LANE_MODE PLIB_SQI_LaneModeGet(SQI_MODULE_ID index);
```

Returns

Lane mode (single/dual/quad).

Description

This function returns the number of lanes (single/dual/quad) used for transfers.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_LANE_MODE laneMode = PLIB_SQI_LaneModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_LANE_MODE PLIB_SQI_LaneModeGet (SQI_MODULE_ID index)

PLIB_SQI_LaneModeSet Function

Sets the data lane mode (single/dual/quad).

File

plib_sqi.h

C

```
void PLIB_SQI_LaneModeSet(SQI_MODULE_ID index, SQI_LANE_MODE mode);
```

Returns

None.

Description

This function sets the number of lanes (single/dual/quad) used for transfers.

Remarks

None.

Preconditions

Make sure the output enable is selected on the data lines using PLIB_SQI_DataOutputEnableSelect. The device needs to be programmed to the same mode that the SQI controller is set to (may require special commands).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_LaneModeSet(MY_SQI_INSTANCE, SQI_LANE_QUAD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Lane mode (single/dual/quad)

Function

void PLIB_SQI_LaneModeSet (SQI_MODULE_ID index, SQI_LANE_MODE mode)

PLIB_SQI_NumberOfReceiveBufferReads Function

Returns the number of receive buffer reads.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_NumberOfReceiveBufferReads(SQI_MODULE_ID index);
```

Returns

Number of Receive Buffer Reads.

Description

This function returns the number of receive buffer reads for debug purpose.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufReads = PLIB_SQI_ReceiveBufferReadsGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ReceiveBufferReadsGet(SQI_MODULE_ID index)

PLIB_SQI_ReceiveData Function

Reads the data from the receive buffer.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ReceiveData(SQI_MODULE_ID index);
```

Returns

None.

Description

This function reads the data from the SQI receive buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
```

```
// application developer.
uint32_t receivedData= PLIB_SQI_ReceiveData(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ReceiveData (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchDisable Function

Disables the receive latch so receive data is discarded when in transmit mode.

File

plib_sqi.h

C

```
void PLIB_SQI_ReceiveLatchDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the receive latch, which disables the receive data to be latched in transmit mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ReceiveLatchDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ReceiveLatchDisable (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchEnable Function

Enables the receive latch so receive data is latched during transmit mode.

File

plib_sqi.h

C

```
void PLIB_SQI_ReceiveLatchEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the receive latch, which latches receive data in transmit mode. Otherwise, receive data in transmit mode is discarded.

Remarks

As most of the SQI communication is half-duplex, enable this function only when it is absolutely required.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ReceiveLatchEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ReceiveLatchEnable (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchGet Function

Returns the receive latch status in transmit mode.

File

plib_sqi.h

C

```
bool PLIB_SQI_ReceiveLatchGet(SQI_MODULE_ID index);
```

Returns

- true - Receive latch is set
- false - Receive latch is not set

Description

This function returns the receive latch status in transmit mode. Returns true, if latch is set (enabling latching of receive buffer data), false if latch is not set (disabling the latching of the receive buffer data).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool rxLatch = PLIB_SQI_ReceiveLatchGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ReceiveLatchGet (SQI_MODULE_ID index)

PLIB_SQI_SoftReset Function

Issues a soft reset to the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_SoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a software reset to the SQI module clearing all the read/write register, internal state machines and data buffers.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_SoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_SoftReset (SQI_MODULE_ID index)

PLIB_SQI_TransferModeGet Function

Returns the SQI transfer mode of operation.

File

plib_sqi.h

C

```
SQI_XFER_MODE PLIB_SQI_TransferModeGet(SQI_MODULE_ID index);
```

Returns

Transfer mode (PIO/DMA/XIP).

Description

This function returns the SQI transfer mode of operation: PIO, DMA, or XIP.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_XFER_MODE xferMode = PLIB_SQI_TransferModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.

Function

SQI_XFER_MODE PLIB_SQI_TransferModeGet (SQI_MODULE_ID index)

PLIB_SQI_TransferModeSet Function

Sets the SQI transfer mode of operation.

File

plib_sqi.h

C

```
void PLIB_SQI_TransferModeSet(SQI_MODULE_ID index, SQI_XFER_MODE mode);
```

Returns

None.

Description

This function sets the SQI transfer mode of operation, (PIO, DMA, or XIP.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_XFER_MODE_DMA is an enum element.
PLIB_SQI_TransferModeSet(MY_SQI_INSTANCE, SQI_XFER_MODE_DMA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Transfer mode (PIO/DMA/XIP)

Function

void PLIB_SQI_TransferModeSet (SQI_MODULE_ID index, SQI_XFER_MODE mode)

PLIB_SQI_TransmitData Function

Writes data into the SQI transmit buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_TransmitData(SQI_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function writes the data into the SQI transmit buffer, which will be eventually sent out on SQI bus.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TRANSMIT_DATA is the data to be sent.
PLIB_SQI_TransmitData(MY_SQI_INSTANCE, MY_TRANSMIT_DATA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

data	Data to be transmitted

Function

void PLIB_SQI_TransmitData (SQI_MODULE_ID index, uint32_t data)

PLIB_SQI_WriteProtectClear Function

Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_WriteProtectClear(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQID2 pin to be used for write-protect.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_WriteProtectClear(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_WriteProtectClear (SQI_MODULE_ID index)

PLIB_SQI_WriteProtectGet Function

Gets the state of the write-protect feature on SQID2.

File

plib_sqi.h

C

```
bool PLIB_SQI_WriteProtectGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the write-protect feature status on the SQID2 pin.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool wpStatus = PLIB_SQI_WriteProtectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_WriteProtectGet (SQI_MODULE_ID index)

PLIB_SQI_WriteProtectSet Function

Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.

File

plib_sqi.h

C

```
void PLIB_SQI_WriteProtectSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQID2 pin to be used for write-protect in single and dual lane modes for supported Flash memories.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_WriteProtectSet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_WriteProtectSet (SQI_MODULE_ID index)

PLIB_SQI_ConBufferSoftReset Function

Issues a soft reset to the SQI control buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_ConBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI control buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ConBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ConBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_RxBufferSoftReset Function

Issues a soft reset to the SQI receive buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI receive buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_RxBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_RxBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_TxBufferSoftReset Function

Issues a soft reset to the SQI transmit buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI transmit buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_TxBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_TxBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_TapDelaySet Function

Sets the tap delays.

File

plib_sqi.h

C

```
void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index, uint8_t sdrClkInDly, uint8_t dataOutDly, uint8_t clkOutDly);
```

Returns

SQI command (transmit/receive/idle) that is currently being executed.

Description

Sets the tap delays that might be required at the higher interface speeds to handle data set-up and hold delays.

Remarks

This function should be used only when the SQI returns failures at maximum frequency.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index,
                          8,
                          0,
                          2
                          );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index,

uint8_t sdrClkInDly,
uint8_t dataOutDly,
uint8_t clkOutDly
);

PLIB_SQI_DDRModeGet Function

Return the SQI data rate mode (single/double) value.

File

plib_sqi.h

C

```
bool PLIB_SQI_DDRModeGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI communication mode value.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool pioDDRMode = PLIB_SQI_DDRModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DDRModeGet(SQI_MODULE_ID index)

PLIB_SQI_DDRModeSet Function

Sets SQI communication to DDR mode.

File

plib_sqi.h

C

```
void PLIB_SQI_DDRModeSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets SQI communication to DDR mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DDRModeSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DDRModeSet(SQI_MODULE_ID index)

b) Clock Configuration Management Functions

PLIB_SQI_ClockDividerSet Function

Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockDividerSet(SQI_MODULE_ID index, SQI_CLK_DIV clkDivider);
```

Returns

None.

Description

This function sets the SQI clock divider value.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockDividerSet(MY_SQI_INSTANCE, CLK_DIV_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clkDivider	Clock divider value

Function

void PLIB_SQI_ClockDividerSet (SQI_MODULE_ID index, SQI_CLK_DIV clkDivider)

PLIB_SQI_ClockEnable Function

Enables the SQI transfer clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQI transfer clock (divided clock).

Remarks

SQICLK is enabled.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ClockEnable (SQI_MODULE_ID index)

PLIB_SQI_ClockIsStable Function

Returns SQI transfer clock state.

File

plib_sqi.h

C

```
bool PLIB_SQI_ClockIsStable(SQI_MODULE_ID index);
```

Returns

True if clock is stable and false if it is not.

Description

This function returns the SQI transfer clock state.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool clockState = PLIB_SQI_ClockIsStable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ClockIsStable (SQI_MODULE_ID index)

c) XIP Configuration Management Functions

PLIB_SQI_XIPAddressBytesGet Function

Returns the number of address bytes.

File

plib_sqi.h

C

```
SQI_ADDR_BYTES PLIB_SQI_XIPAddressBytesGet(SQI_MODULE_ID index);
```

Returns

Number of Address Bytes.

Description

This function returns the number of address bytes to be sent to the flash.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t addressBytes = PLIB_SQI_XIPAddressBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_ADDR_BYTES PLIB_SQI_XIPAddressBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPAddressBytesSet Function

Sets the number of address bytes.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPAddressBytesSet(SQI_MODULE_ID index, SQI_ADDR_BYTES bytes);
```

Returns

None.

Description

This function sets the number of address bytes to be sent to the flash. Typical flash address bytes are 3 (24-bit address).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPAddressBytesSet(MY_SQI_INSTANCE, ADDR_BYTES_3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of address bytes

Function

void PLIB_SQI_XIPAddressBytesSet (SQI_MODULE_ID index, SQI_ADDR_BYTES bytes)

PLIB_SQI_XIPChipSelectGet Function

Returns the current active Chip Select.

File

plib_sqi.h

C

```
SQI_CS_NUM PLIB_SQI_XIPChipSelectGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the active Chip Select in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_CS_NUM xipCSActive = (PLIB_SQI_XIPChipSelectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_CS_NUM PLIB_SQI_XIPChipSelectGet (SQI_MODULE_ID index)

PLIB_SQI_XIPChipSelectSet Function

Activates the Chip Select in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPChipSelectSet(SQI_MODULE_ID index, SQI_CS_NUM csNum);
```

Returns

None.

Description

This function sets the Chip Select that is active in XIP mode.

Remarks

None.

Preconditions

Make sure the Chip Select output enable is selected on the CS lines (PLIB_SQI_CSOutputEnableSelect).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPChipSelectSet(MY_SQI_INSTANCE, SQI_CS_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Function

void PLIB_SQI_XIPChipSelectSet (SQI_MODULE_ID index, SQI_CS_NUM csNum)

PLIB_SQI_XIPControlWord1Get Function

Get the XIP mode Control Word 1.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord1Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 1.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon1 = PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord1Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord1Set Function

Sets the XIP mode Control Word 1.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord1Set(SQI_MODULE_ID index, SQI_DUMMY_BYTES dummyBytes, SQI_ADDR_BYTES
addressBytes, uint8_t readOpcode, SQI_LANE_MODE dataLaneMode, SQI_LANE_MODE dummyLaneMode, SQI_LANE_MODE
modeCodeLaneMode, SQI_LANE_MODE addressLaneMode, SQI_LANE_MODE cmdLaneMode);
```

Returns

None.

Description

This function sets XIP mode Control Word 1. This function combines work of multiple PLIB APIs and can be used by the driver where complete XIP Control Word 1 is being modified.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE,
                             DUMMY_BYTES_2,
                             ADDR_BYTES_3,
                             0x0B,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dummyBytes	Number of dummy bytes (0-7)
addressBytes	Number of address bytes (0-4)
readOpcode	Quad flash read opcode (ex: 0x0B)
dataLaneMode	Number of SQI data lanes used for sending data bytes
dummyLaneMode	Number of SQI data lanes used for sending dummy bytes
modeCodeLaneMode	Number of SQI data lanes used for sending mode code
addressLaneMode	Number of SQI data lanes used for sending address
cmdLaneMode	Number of SQI data lanes used for sending command

Function

```
void PLIB_SQI_XIPControlWord1Set (  SQI_MODULE_ID   index,
                                   SQI_DUMMY_BYTES dummyBytes,
                                   SQI_ADDR_BYTES  addressBytes,
uint8_t        readOpcode,
                                   SQI_LANE_MODE   dataLaneMode,
                                   SQI_LANE_MODE   dummyLaneMode,
                                   SQI_LANE_MODE   modeCodeLaneMode,
                                   SQI_LANE_MODE   addressLaneMode,
                                   SQI_LANE_MODE   cmdLaneMode
)
```

PLIB_SQI_XIPControlWord2Get Function

Gets the XIP mode Control Word 2.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord2Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 2.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon2 = PLIB_SQI_XIPControlWord2Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord2Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord2Set Function

Sets the XIP mode Control Word 2.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord2Set(SQI_MODULE_ID index, SQI_CS_NUM devSelect, SQI_MODE_BYTES modeBytes,
uint8_t modeCode);
```

Returns

None.

Description

This function sets XIP mode Control Word 2. This function combines work of multiple PLIB APIs and can be used by the driver where complete XIP Control Word 2 is being modified.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE,
                             SQI_CS_0,
                             MODE_BYTES_0,
                             0x0
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
modeCode	Mode code used for supported flash devices

modeBytes	Number of mode code bytes
devSelect	Chip select for XIP mode

Function

void PLIB_SQI_XIPControlWord2Set (SQI_MODULE_ID index,
SQI_CS_NUM devSelect,
SQI_MODE_BYTES modeBytes,
uint8_t modeCode
)

PLIB_SQI_XIPDummyBytesGet Function

Sets the number of dummy bytes.

File

plib_sqi.h

C

```
SQI_DUMMY_BYTES PLIB_SQI_XIPDummyBytesGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t dummyBytes = PLIB_SQI_XIPDummyBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_DUMMY_BYTES PLIB_SQI_XIPDummyBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPDummyBytesSet Function

Sets the number of dummy bytes.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPDummyBytesSet(SQI_MODULE_ID index, SQI_DUMMY_BYTES bytes);
```

Returns

None.

Description

This function sets the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPDummyBytesSet(MY_SQI_INSTANCE, DUMMY_BYTE_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of dummy bytes

Function

void PLIB_SQI_XIPDummyBytesSet (SQI_MODULE_ID index, SQI_DUMMY_BYTES bytes)

PLIB_SQI_XIPLaneModeSet Function

Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPLaneModeSet(SQI_MODULE_ID index, SQI_LANE_MODE dataLanes, SQI_LANE_MODE dummyLanes,
SQI_LANE_MODE modeLanes, SQI_LANE_MODE addrLanes, SQI_LANE_MODE cmdLanes);
```

Returns

None.

Description

This function selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.

Remarks

This function can't be called when in XIP mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPLaneModeSet(MY_SQI_INSTANCE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE
                       );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataLanes	Data lane mode (single/dual/quad)
dummyLanes	Dummy lane mode (single/dual/quad)
modeLanes	Mode lane mode (single/dual/quad)
addrLanes	Address lane mode (single/dual/quad)
cmdLanes	Command lane mode (single/dual/quad)

Function

void PLIB_SQI_XIPLaneModeSet (SQI_MODULE_ID index,
SQI_LANE_MODE dataLanes,
SQI_LANE_MODE dummyLanes,
SQI_LANE_MODE modeLanes,
SQI_LANE_MODE addrLanes,
SQI_LANE_MODE cmdLanes
)

PLIB_SQI_XIPModeBytesGet Function

Returns the number of bytes used for mode code command.

File

plib_sqi.h

C

```
SQI_MODE_BYTES PLIB_SQI_XIPModeBytesGet(SQI_MODULE_ID index);
```

Returns

Number of Bytes used for Mode Code Command.

Description

This function returns the number of bytes for the mode code command in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_MODE_BYTES modeCodeBytes = PLIB_SQI_XIPModeBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_MODE_BYTES PLIB_SQI_XIPModeBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPModeBytesSet Function

Allocates the bytes for mode code command.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPModeBytesSet(SQI_MODULE_ID index, SQI_MODE_BYTES bytes);
```

Returns

None.

Description

This function sets the number of bytes for the mode code command in XIP mode.

Remarks

Refer to serial device data sheet for the details on this command.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPModeBytesSet(MY_SQI_INSTANCE, MODE_BYTES_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of bytes of Mode code

Function

void PLIB_SQI_XIPModeBytesSet (SQI_MODULE_ID index, SQI_MODE_BYTES bytes)

PLIB_SQI_XIPModeCodeGet Function

Returns the mode code op-code.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_XIPModeCodeGet(SQI_MODULE_ID index);
```

Returns

Mode Code Opcode.

Description

This function returns the mode code command (opcode) in XIP mode for the devices that support it.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t modeCode = PLIB_SQI_XIPModeCodeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_XIPModeCodeGet (SQI_MODULE_ID index)

PLIB_SQI_XIPModeCodeSet Function

Sets the mode code command.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPModeCodeSet(SQI_MODULE_ID index, uint8_t code);
```

Returns

None.

Description

This function sets the mode code command in XIP mode for the supported flash devices.

Remarks

Some of the devices seems to support this command, refer to specific serial device data sheet for op-code and sequence details.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPModeCodeSet(MY_SQI_INSTANCE, MY_MODE_CODE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
code	Mode code (byte)

Function

void PLIB_SQI_XIPModeCodeSet (SQI_MODULE_ID index, uint8_t code)

PLIB_SQI_XIPReadOpcodeGet Function

Returns the read op code in XIP mode.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_XIPReadOpcodeGet(SQI_MODULE_ID index);
```

Returns

Read Opcode.

Description

This function returns the read op code used in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t readOpcode = PLIB_SQI_XIPReadOpcodeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_XIPReadOpcodeGet (SQI_MODULE_ID index)

PLIB_SQI_XIPReadOpcodeSet Function

Sets the read op code in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPReadOpcodeSet(SQI_MODULE_ID index, uint8_t opCode);
```

Returns

None.

Description

This function sets the flash read opcode in XIP mode. Value of read op code depends on the Flash device attached.

Remarks

Refer to the Flash device data sheet for supported read op codes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_READ_OPCODE is the op code dependent on
// attached serial device.
PLIB_SQI_XIPReadOpcodeSet(MY_SQI_INSTANCE, MY_READ_OPCODE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
opCode	Flash read op code

Function

void PLIB_SQI_XIPReadOpcodeSet (SQI_MODULE_ID index, uint8_t opCode)

PLIB_SQI_XIPControlWord3Get Function

Gets the XIP mode Control Word 3.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord3Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 3.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
uint32_t xipCon3 = PLIB_SQI_XIPControlWord3Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord3Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord3Set Function

Sets the XIP mode Control Word 3.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord3Set(SQI_MODULE_ID index, bool initStatCheck, uint8_t initCmdCount,
SQI_LANE_MODE initCmdType, uint8_t initCmd3, uint8_t initCmd2, uint8_t initCmd1);
```

Returns

None.

Description

This function sets XIP mode Control Word 3. This function is used to set multiple commands in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE,
                             0,
                             3,
                             SQI_LANE_QUAD,
                             0x06,              //WEN
                             0x99,              //RST
                             0x66               //RSTEN
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
initStatCheck	Flash status check at the end of initialization commands
initCmdCount	Number of mode code bytes
initCmdType	Chip select for XIP mode
initCmd3	Command 3
initCmd2	Command 2
initCmd1	Command 1

Function

void PLIB_SQI_XIPControlWord3Set (SQI_MODULE_ID index,
bool initStatCheck,
uint8_t initCmdCount,
SQI_LANE_MODE initCmdType,
uint8_t initCmd3,
uint8_t initCmd2,

uint8_t initCmd1
)

PLIB_SQI_XIPControlWord4Get Function

Gets the XIP mode Control Word 4.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord4Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 4.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon4 = PLIB_SQI_XIPControlWord4Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord4Get (SQI_MODULE_ID index)

PLIB_SQI_XIPDDRModeSet Function

Selects the rate mode (SDR/DDR) for different transactions in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPDDRModeSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function selects the double data rate mode for different transactions (command, address, dummy and data etc.,) in XIP mode of operation

Remarks

This function can't be called when in XIP mode. The device is set in SDR mode if this function is not invoked.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
PLIB_SQI_XIPDDRModeSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_XIPDDRModeSet (SQI_MODULE_ID index)

PLIB_SQI_XIPSDRCommandDDRDataGet Function

Returns the SQI command in SDR mode and Data in DDR mode bit value.

File

plib_sqi.h

C

```
bool PLIB_SQI_XIPSDRCommandDDRDataGet(SQI_MODULE_ID index);
```

Returns

Returns true if the bit is set and false if it's cleared.

Description

This function returns the command to be transferred in SDR mode and data in DDR mode bit value. Some serial flash devices require the sequence to send command in SDR mode and the rest of the data (including address bytes) in DDR mode

Remarks

This function can't be called when in XIP mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool cmdsdrValue = PLIB_SQI_XIPSDRCommandDDRDataGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_XIPSDRCommandDDRDataGet (SQI_MODULE_ID index)

PLIB_SQI_XIPSDRCommandDDRDataSet Function

Sets the SQI command in SDR mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPSDRCommandDDRDataSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets the command to be transferred in SDR mode and data in DDR mode. Some serial flash devices require the sequence to send command in SDR mode and the rest of the data (including address bytes) in DDR mode

Remarks

This function can't be called when in XIP mode. This function has no affect if the set-up is working in single lane mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPSDRCommandDDRDataSet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_XIPSDRCommandDDRDataSet (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord4Set Function

Sets the XIP mode Control Word 4.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord4Set(SQI_MODULE_ID index, bool initStatCheck, uint8_t initCmdCount,
SQI_LANE_MODE initCmdType, uint8_t initCmd3, uint8_t initCmd2, uint8_t initCmd1);
```

Returns

None.

Description

This function sets XIP mode Control Word 4. This function is used to set multiple commands in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord4Set (MY_SQI_INSTANCE,
                             0,
                             1,
                             SQI_LANE_QUAD,
                             0x00,
                             0x00,
                             0xC7              //Chip Erase
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
initStatCheck	Flash status check at the end of initialization commands
initCmdCount	Number of mode code bytes
initCmdType	Chip select for XIP mode
initCmd3	Command 3
initCmd2	Command 2

initCmd1	Command 1

Function

void PLIB_SQI_XIPControlWord4Set (SQI_MODULE_ID index,
bool initStatCheck,
uint8_t initCmdCount,
SQI_LANE_MODE initCmdType,
uint8_t initCmd3,
uint8_t initCmd2,
uint8_t initCmd1
)

d) PIO Mode Transfer Management Functions

PLIB_SQI_ByteCountGet Function

Returns the current transmit/receive count.

File

plib_sqi.h

C

```
uint16_t PLIB_SQI_ByteCountGet(SQI_MODULE_ID index);
```

Returns

Transfer Count.

Description

This function returns the transmit/receive count, which is set by software and is actively controlled and maintained by hardware.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint16_t xferCount = PLIB_SQI_ByteCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Function

uint16_t PLIB_SQI_ByteCountGet (SQI_MODULE_ID index)

PLIB_SQI_ByteCountSet Function

Sets the transmit/receive count.

File

plib_sqi.h

C

```
void PLIB_SQI_ByteCountSet(SQI_MODULE_ID index, uint16_t xferCount);
```

Returns

None.

Description

This function sets the number of bytes to transmit or receive, which is set by software and is actively controlled and maintained by hardware.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_XFER_COUNT is the transfer count.
PLIB_SQI_ByteCountSet(MY_SQI_INSTANCE, MY_XFER_COUNT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Function

void PLIB_SQI_ByteCountSet (SQI_MODULE_ID index, uint16_t xferCount)

PLIB_SQI_ChipSelectDeassertDisable Function

Clears the Chip Select deassert.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectDeassertDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the Chip Select deassert. Chip Select stays asserted after transmission or reception of specified number of bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ChipSelectDeassertDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ChipSelectDeassertDisable(SQI_MODULE_ID index)

PLIB_SQI_ChipSelectDeassertEnable Function

Sets the Chip Select deassert.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectDeassertEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables Chip Select deassert. Chip Select is deasserted after transmission or reception of the specified number of bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ChipSelectDeassertEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ChipSelectDeassertEnable (SQI_MODULE_ID index)

PLIB_SQI_ChipSelectGet Function

Returns the Chip Select that is currently active.

File

plib_sqi.h

C

```
SQI_CS_NUM PLIB_SQI_ChipSelectGet(SQI_MODULE_ID index);
```

Returns

Chip Select (2-bit) - Current Chip Select active (0/1).

Description

This function returns the Chip Select that is currently active.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_CS_NUM csNum = PLIB_SQI_ChipSelectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_CS_NUM PLIB_SQI_ChipSelectGet(SQI_MODULE_ID index)

PLIB_SQI_ChipSelectSet Function

Activates the Chip Select.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectSet(SQI_MODULE_ID index, SQI_CS_NUM csNum);
```

Returns

None.

Description

This function sets the Chip Select to be activated on the next transaction.

Remarks

None.

Preconditions

Make sure the Chip Select output enable is selected on the CS lines (PLIB_SQI_CSOutputEnableSelect).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_CS_NUM_0 is the enum element.
PLIB_SQI_ChipSelectSet(MY_SQI_INSTANCE, SQI_CS_NUM_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Function

void PLIB_SQI_ChipSelectSet (SQI_MODULE_ID index, SQI_CS_NUM csNum)

PLIB_SQI_ConfigWordGet Function

Gets the SQI Configuration Word.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ConfigWordGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI Configuration Word.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t sqiCfg = PLIB_SQI_ConfigWordGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ConfigWordGet (SQI_MODULE_ID index)

PLIB_SQI_ConfigWordSet Function

Sets SQI Configuration Word.

File

plib_sqi.h

C

```
void PLIB_SQI_ConfigWordSet(SQI_MODULE_ID index, bool sqiEnable, SQI_CS_OEN csPins, SQI_DATA_OEN dataPins,
bool reset, bool burstEn, bool hold, bool writeProtect, bool rxLatch, SQI_DATA_FORMAT lsbf, SQI_DATA_MODE
dataMode, SQI_XFER_MODE xferMode);
```

Returns

None.

Description

This function sets the SQI Configuration Word. This function is a combination of several function in case the driver plans to write the complete Configuration Word.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
                       1,
                       SQI_CS_OEN_0,
                       SQI_DATA_OEN_QUAD,
                       0,
                       1,
                       0,
                       0,
                       0,
                       SQI_DATA_FORMAT_LSBF,
                       SQI_DATA_MODE_3,
                       SQI_XFER_MODE_PIO
                       );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
sqiEnable	Enables/Disables the SQI module
csPins	Chip Select Output Enable

dataPins	Data Output Enable
reset	Resets control, transmit, receive buffers and state machines
burstEn	Burst Enable (always set to '1')
hold	SQID2 to act as HOLD# signal in single and dual lane modes
writeProtect	SQID3 to act as WP# signal in single and dual lane modes
rxLatch	Activates receive latch in transmit mode
lsbf	Sets data endian mode to least significant bit first (LSBF)
dataMode	Sets data mode to mode 0/mode 1/Serial Flash mode
xferMode	Sets transfer mode to XIP/DMA/PIO mode

Function

void PLIB_SQI_ConfigWordSet(SQI_MODULE_ID index,
bool sqiEnable,
SQI_CS_OEN csPins,
SQI_DATA_OEN dataPins,
bool reset,
bool burstEn,
bool hold,
bool writeProtect,
bool rxLatch,
SQI_DATA_FORMAT lsbf,
SQI_DATA_MODE dataMode,
SQI_XFER_MODE xferMode
)

PLIB_SQI_ReceiveBufferIsUnderrun Function

Returns the status of receive buffer.

File

plib_sqi.h

C

```
bool PLIB_SQI_ReceiveBufferIsUnderrun(SQI_MODULE_ID index);
```

Returns

- true - Receive Buffer is underrun
- false - Receive Buffer is not underrun

Description

This function returns the status of the receive buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool rxun = PLIB_SQI_ReceiveBufferIsUnderrun(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ReceiveBufferIsUnderrun (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdGet Function

Returns the receive command threshold.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_RxBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Receive Buffer Threshold value.

Description

This function returns the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufThres = PLIB_SQI_RxCommandThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_RxBufferThresholdGet (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdIntGet Function

Sets the receive buffer threshold interrupt.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_RxBufferThresholdIntGet(SQI_MODULE_ID index);
```

Returns

Receive Buffer Threshold value (used to trigger an interrupt).

Description

This function returns the receive buffer threshold used to set an interrupt.

Remarks

This is a 5-bit field and bits 7, 6, and 5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufIntThres = PLIB_SQI_RxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive interrupt (buffer) threshold

Function

uint8_t PLIB_SQI_RxBufferThresholdIntGet (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdIntSet Function

Sets the receive buffer threshold for interrupt.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferThresholdIntSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the receive buffer threshold used to trigger an interrupt. Sets an interrupt condition when receive buffer count is larger than or equal to the receive interrupt threshold bytes.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_RX_INT_THRESHOLD is the threshold value.
PLIB_SQI_RxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_RX_INT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive buffer threshold for interrupt

Function

void PLIB_SQI_RxBufferThresholdIntSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_RxBufferThresholdSet Function

Sets the receive command threshold.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Remarks

Valid threshold values are 0-31.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_RECEIVE_THRESHOLD is the receive threshold value.
PLIB_SQI_RxCommandThresholdSet(MY_SQI_INSTANCE, MY_RECEIVE_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive command (buffer) threshold

Function

void PLIB_SQI_RxBufferThresholdSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_TransferDirectionGet Function

Returns the transfer command.

File

plib_sqi.h

C

```
SQI_XFER_CMD PLIB_SQI_TransferDirectionGet(SQI_MODULE_ID index);
```

Returns

Transfer Command (Idle/Receive/Transmit).

Description

This function returns the transfer command that is active currently.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_XFER_CMD xferDirection = PLIB_SQI_TransferDirectionGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_XFER_CMD PLIB_SQI_TransferDirectionGet (SQI_MODULE_ID index)

PLIB_SQI_TransferDirectionSet Function

Sets the transfer command.

File

plib_sqi.h

C

```
void PLIB_SQI_TransferDirectionSet(SQI_MODULE_ID index, SQI_XFER_CMD command);
```

Returns

None.

Description

This function sets the transfer command to Idle/Transmit/Receive.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_CMD_TRANSMIT is an enum element.
PLIB_SQI_TransferDirectionSet(MY_SQI_INSTANCE, SQI_CMD_TRANSMIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
command	Transfer command

Function

void PLIB_SQI_TransferDirectionSet (SQI_MODULE_ID index, SQI_XFER_CMD command)

PLIB_SQI_TransmitBufferFreeSpaceGet Function

Returns the number of transmit buffer words available.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TransmitBufferFreeSpaceGet(SQI_MODULE_ID index);
```

Returns

Amount of transmit buffer space free in bytes.

Description

This function returns the number of transmit buffer bytes available.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txBufFreeSpace = PLIB_SQI_TransmitBufferFreeSpaceGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TransmitBufferFreeSpaceGet(SQI_MODULE_ID index)

PLIB_SQI_TransmitBufferHasOverflowed Function

Returns the current status of the transmit buffer.

File

plib_sqi.h

C

```
bool PLIB_SQI_TransmitBufferHasOverflowed(SQI_MODULE_ID index);
```

Returns

- true - Transmit Buffer has overflowed
- false - Transmit Buffer has not overflowed

Description

This function returns the current state of the transmit buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool txOv = PLIB_SQI_TransmitBufferHasOverflowed(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_TransmitBufferHasOverflowed (SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdGet Function

Returns the transmit command threshold value.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TxBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer threshold value.

Description

This function returns the transmit command threshold value that is used to monitor transmits based on the transmit buffer space availability.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txThreshold = PLIB_SQI_TxBufferThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TxBufferThresholdGet(SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdIntGet Function

Returns the value to trigger the transmit buffer threshold interrupt.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TxBufferThresholdIntGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer threshold for interrupt.

Description

This function returns the transmit buffer threshold used to set an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txBufIntThres = PLIB_SQI_TxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TxBufferThresholdIntGet (SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdIntSet Function

Sets the value to trigger the transmit buffer threshold interrupt.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferThresholdIntSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the transmit buffer threshold used for an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TX_INT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_TX_INT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit interrupt (buffer) threshold

Function

void PLIB_SQI_TxBufferThresholdIntSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_TxBufferThresholdSet Function

Sets the transmit command threshold.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the transmit command threshold, which is used to control transmits based on the transmit buffer space availability.

Remarks

Valid threshold values are 0-31.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TRANSMIT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdSet(MY_SQI_INSTANCE, MY_TRANSMIT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit command (buffer) threshold

Function

void PLIB_SQI_TxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold)

e) Interrupt Control and Status Management Functions

PLIB_SQI_InterruptDisable Function

Disables the interrupt source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptDisable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables the interrupt source passed into the function.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and TXFULL is an enum element.
PLIB_SQI_InterruptDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Function

void PLIB_SQI_InterruptDisable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptEnable Function

Enables the passed interrupt source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptEnable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables the interrupt source passed into the function.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Function

void PLIB_SQI_InterruptEnable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptFlagGet Function

Return SQI Interrupt flag status.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptFlagGet(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

Interrupt status.

Description

This function returns the SQI interrupt source flag status (set/cleared).

Remarks

None.

Preconditions

None.

Example

```
if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_INT_ANY) )
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_TXFULL) )
    {
       ...
    }
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_RXFULL) )
    {
        ...
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt flag of interest

Function

bool PLIB_SQI_InterruptFlagGet(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptIsEnabled Function

Returns the interrupt state.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptIsEnabled(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupt is enabled
- false - Interrupt is disabled

Description

This function returns whether or not the interrupt state is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (PLIB_SQI_InterruptIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))
{
    ..
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt under check

Function

bool PLIB_SQI_InterruptIsEnabled (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalDisable Function

Disables the interrupt signal source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptSignalDisable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables the interrupt signals source passed into the function, thus prohibiting it from reaching to the external interrupt controller.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptSignalDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Function

void PLIB_SQI_InterruptSignalDisable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalEnable Function

Enables the passed interrupt signal source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptSignalEnable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables the interrupt signal source to be passed into the function, which allows it to go out to the external Interrupt Controller.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptSignalEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Function

void PLIB_SQI_InterruptSignalEnable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalIsEnabled Function

Returns the interrupt signal state.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptSignalIsEnabled(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupt signal is enabled
- false - Interrupt signal is disabled

Description

This function returns whether the interrupt signal state is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
if (PLIB_SQI_InterruptSignalIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))
{
    ..
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt signal under check

Function

bool PLIB_SQI_InterruptSignalIsEnabled (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_DataLineStatus Function

Returns the logical status of the SQI data lines.

File

plib_sqi.h

C

```
bool PLIB_SQI_DataLineStatus(SQI_MODULE_ID index, uint8_t dataPin);
```

Returns

SQIDx Status (High/Low).

Description

This function returns the logical status of the data lines (0/1).

Remarks

Parsing values other than 0/1/2/3 returns SQID0 pin status.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool sqiDataLineStatus = PLIB_SQI_DataLineStatus(MY_SQI_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPin	Data pin for which status will be returned (0/1/2/3)

Function

bool PLIB_SQI_DataLineStatus(SQI_MODULE_ID index, uint8_t dataPin)

PLIB_SQI_CommandStatusGet Function

Returns the SQI command (transmit/receive/idle).

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_CommandStatusGet(SQI_MODULE_ID index);
```

Returns

SQI command (transmit/receive/idle) that is currently being executed.

Description

This function returns the SQI command (transmit/receive/idle) that is currently being executed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t cmdStat = PLIB_SQI_CommandStatusGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_CommandStatusGet(SQI_MODULE_ID index)

PLIB_SQI_ConBufWordsAvailable Function

Returns the number of control buffer words available.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_ConBufWordsAvailable(SQI_MODULE_ID index);
```

Returns

Number of words available.

Description

This function returns the number of control buffer words available.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t numConBuf = PLIB_SQI_ConBufWordsAvailable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ConBufWordsAvailable(SQI_MODULE_ID index)

f) DMA Buffer Address Management Functions

PLIB_SQI_DMABDBaseAddressGet Function

Returns the address of the base buffer descriptor.

File

plib_sqi.h

C

```
void* PLIB_SQI_DMABDBaseAddressGet(SQI_MODULE_ID index);
```

Returns

Base Buffer Descriptor address.

Description

This function returns the address of the base DMA buffer descriptor.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
void *baseBDAddress;

If (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
{
   baseBDAddress = PLIB_SQI_DMABDBaseAddressGet(MY_SQI_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void * PLIB_SQI_DMABDBaseAddressGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDBaseAddressSet Function

Sets the address of the base buffer descriptor.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDBaseAddressSet(SQI_MODULE_ID index, void * baseBDAddress);
```

Returns

None.

Description

This function writes the address of the base (first/only) buffer descriptor into the buffer descriptor base address register.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
    PLIB_SQI_DMABDBaseAddressSet(MY_SQI_INSTANCE, (void *) (&MY_BD_STRUCT));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseBDAddress	Base buffer descriptor address

Function

void PLIB_SQI_DMABDBaseAddressSet (SQI_MODULE_ID index, void *baseBDAddress)

PLIB_SQI_DMABDCurrentAddressGet Function

Returns the address of the current buffer descriptor in process.

File

plib_sqi.h

C

```
void* PLIB_SQI_DMABDCurrentAddressGet(SQI_MODULE_ID index);
```

Returns

Current Buffer Descriptor Address.

Description

This function returns the address of the DMA buffer descriptor that is currently in progress.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t currentBDAddress;

If (PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
{
   void* currentBDAddress = PLIB_SQI_DMABDCurrentAddressGet(MY_SQI_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void* PLIB_SQI_DMABDCurrentAddressGet (SQI_MODULE_ID index)

g) DMA Buffer Descriptor Management Functions

PLIB_SQI_DMABDControlWordGet Function

Returns Current Buffer Descriptor Control Word Information.

File

plib_sqi.h

C

```
uint16_t PLIB_SQI_DMABDControlWordGet(SQI_MODULE_ID index);
```

Returns

Control Word - DMA Buffer Descriptor Control Word

Description

This function returns current buffer descriptor Control Word information excluding buffer length. This information is returned in transmit and receive status functions.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint16_t dmabdconword;

dmabdconword =  PLIB_SQI_DMABDControlWordGet(MY_SQI_INSTANCE);

switch (dmabdconword)
{
    case BD_ENABLED: ...;
    case BD_DISABLED: ...;
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_SQI_DMABDControlWordGet (SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveBufferCountGet Function

Returns the receive buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferCountGet(SQI_MODULE_ID index);
```

Returns

Receive buffer space in bytes.

Description

This function returns the current receive buffer space in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdRxBufCount = PLIB_SQI_DMABDReceiveBufferCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDReceiveBufferCountGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveBufferLengthGet Function

Returns the receive length in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferLengthGet(SQI_MODULE_ID index);
```

Returns

Receive buffer space in bytes.

Description

This function returns the current receive length in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdRxBufLength = PLIB_SQI_DMABDReceiveBufferLengthGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDReceiveBufferLengthGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDReceiveStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdrxState = PLIB_SQI_DMABDReceiveStateGet(MY_SQI_INSTANCE);

switch (bdRxState)
{
   case BD_IDLE: ...;
   case BD_STATE_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE uint8_t PLIB_SQI_DMABDReceiveStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitBufferCountGet Function

Returns the transmit buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDTransmitBufferCountGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer space in bytes.

Description

This function returns the current transmit buffer space in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdTxBufCount = PLIB_SQI_DMABDTransmitBufferCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDTransmitBufferCountGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitBufferLengthGet Function

Returns the transmit length in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDTransmitBufferLengthGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer space in bytes.

Description

This function returns the current transmit length in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdTxBufLength = PLIB_SQI_DMABDTransmitBufferLengthGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDTransmitBufferLengthGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDTransmitStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
```

```
// application developer.
uint8_t bdTxState = PLIB_SQI_DMABDTransmitStateGet(MY_SQI_INSTANCE);

switch (bdTxState)
{
   case BD_IDLE: ...;
   case BD_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE PLIB_SQI_DMABDTransmitStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDFetchStop Function

Stops the DMA buffer descriptor fetch process.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDFetchStop(SQI_MODULE_ID index);
```

Returns

None.

Description

This function stops the DMA buffer descriptor fetch process.

Remarks

None.

Preconditions

PLIB_SQI_DMABDFetchStart is called.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDFetchStop(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDFetchStop (SQI_MODULE_ID index)

h) DMA Control and Status Management Functions

PLIB_SQI_DMABDFetchStart Function

Starts the DMA buffer descriptor fetch process.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDFetchStart(SQI_MODULE_ID index);
```

Returns

None.

Description

This function starts the DMA buffer descriptor fetch process.

Remarks

None.

Preconditions

Make sure the buffer descriptors are set up and the buffer descriptor base address register is pointing to the first/only buffer descriptor. Also ensure any previous buffer descriptor processing is fixed.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDFetchStart(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDFetchStart (SQI_MODULE_ID index)

PLIB_SQI_DMABDIsBusy Function

Returns whether or not the DMA Buffer Descriptor is busy.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMABDIsBusy(SQI_MODULE_ID index);
```

Returns

- true - DMA Buffer Descriptor is busy
- false - DMA Buffer Descriptor is not busy

Description

This function returns whether or not the DMA buffer descriptor process is busy.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool bdBusy =  PLIB_SQI_DMABDIsBusy(MY_SQI_INSTANCE)
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMABDIsBusy (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollCounterSet Function

Sets the poll counter value.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollCounterSet(SQI_MODULE_ID index, uint16_t pollCount);
```

Returns

None.

Description

This function sets the poll counter value that indicates the number of cycles the DMA would wait before fetching the next descriptor word, if the current descriptor fetched was disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE)
{
    PLIB_SQI_DMABDPollCounterSet(MY_SQI_INSTANCE, MY_POLL_VALUE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pollControl	Polling value

Function

void PLIB_SQI_DMABDPollCounterSet(SQI_MODULE_ID index, uint16_t pollCount)

PLIB_SQI_DMABDPollDisable Function

Disables the buffer descriptor polling.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the buffer descriptor polling.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDPollDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDPollDisable (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollEnable Function

Enables the buffer descriptor polling.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the buffer descriptor polling and works in tandem with poll control register.

Remarks

Enable this control bit only when you are planning to have dead descriptors in the linked list.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDPollEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDPollEnable (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollIsEnabled Function

Returns whether or not the DMA buffer descriptor poll is enabled.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMABDPollIsEnabled(SQI_MODULE_ID index);
```

Returns

- true - The DMA Poll Control is enabled

- false - The DMA Poll Control is disabled

Description

This function returns whether or not the DMA buffer descriptor poll is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
If (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE))
{
   PLIB_SQI_PollControlSet(MY_SQI_INSTANCE, MY_POLL_CONTROL_VALUE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMABDPollIsEnabled (SQI_MODULE_ID index)

PLIB_SQI_DMABDStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdState = PLIB_SQI_DMABDStateGet(MY_SQI_INSTANCE);

switch (bdState)
{
   case BD_IDLE: ...;
   case BD_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE PLIB_SQI_DMABDStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMADisable Function

Disables the built-in DMA logic.

File

plib_sqi.h

C

```
void PLIB_SQI_DMADisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the built-in DMA logic for data transfer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMADisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMADisable (SQI_MODULE_ID index)

PLIB_SQI_DMAEnable Function

Enables the built-in DMA logic.

File

plib_sqi.h

C

```
void PLIB_SQI_DMAEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the built-in DMA logic for data transfer.

Remarks

None.

Preconditions

DMA buffer descriptors need to be setup before enabling the DMA.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMAEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMAEnable (SQI_MODULE_ID index)

PLIB_SQI_DMAHasStarted Function

Returns whether or no the DMA process has started.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMAHasStarted(SQI_MODULE_ID index);
```

Returns

- true - The DMA process has started
- false - The DMA process has not started

Description

This function returns whether or not the DMA process has started.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool dmaStarted =  PLIB_SQI_DMAHasStarted (MY_SQI_INSTANCE)
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMAHasStarted (SQI_MODULE_ID index)

PLIB_SQI_DMAIsEnabled Function

Returns whether or not DMA is enabled.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMAIsEnabled(SQI_MODULE_ID index);
```

Returns

- true - DMA is enabled
- false - DMA is disabled

Description

This function returns whether or not the DMA is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
If (PLIB_SQI_DMAIsEnabled(MY_SQI_INSTANCE))
{
   ...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMAIsEnabled (SQI_MODULE_ID index)

i) Other Functions

PLIB_SQI_StatusCheckSet Function

Sets the Flash status check related control bits.

File

plib_sqi.h

C

```
void PLIB_SQI_StatusCheckSet(SQI_MODULE_ID index, uint16_t statCheckCmd, uint8_t numStatBytes,
SQI_LANE_MODE statCmdType, bool statBitPos);
```

Returns

None.

Description

This function sets the Flash status check related control bits and enables the status check for PIO mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DDRModeSet(MY_SQI_INSTANCE,
                    0x05,
                    1,
                    SQI_LANE_QUAD,
                    0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
statCheckCmd	Status check command to be sent to the Flash
numStatBytes	Number of status command bytes
statCmdType	Lane mode (Single/Dual/Quad) for status command

Function

void PLIB_SQI_StatusCheckSet(SQI_MODULE_ID index,
uint16_t statCheckCmd,
uint8_t numStatBytes,
SQI_LANE_MODE statCmdType,
bool statBitPos
)

j) Feature Existence Functions

PLIB_SQI_ExistsBDBaseAddress Function

Identifies whether the BDBaseAddress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDBaseAddress(SQI_MODULE_ID index);
```

Returns

- true - The BDBaseAddress feature is supported on the device
- false - The BDBaseAddress feature is not supported on the device

Description

This function identifies whether the BDBaseAddress feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDBaseAddressSet
- PLIB_SQI_DMABDBaseAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDBaseAddress(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDControlWord Function

Identifies whether the BDControlWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDControlWord(SQI_MODULE_ID index);
```

Returns

- true - The BDControlWord feature is supported on the device
- false - The BDControlWord feature is not supported on the device

Description

This function identifies whether the BDControlWord feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDControlWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDControlWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDCurrentAddress Function

Identifies whether the BDCurrentAddress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDCurrentAddress(SQI_MODULE_ID index);
```

Returns

- true - The BDCurrentAddress feature is supported on the device
- false - The BDCurrentAddress feature is not supported on the device

Description

This function identifies whether the BDCurrentAddress feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDCurrentAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDCurrentAddress(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDPollCount Function

Identifies whether the BDPollCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDPollCount(SQI_MODULE_ID index);
```

Returns

- true - The BDPollCount feature is supported on the device
- false - The BDPollCount feature is not supported on the device

Description

This function identifies whether the BDPollCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDPollCounterSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDPollCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDPollingEnable Function

Identifies whether the BDPollingEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDPollingEnable(SQI_MODULE_ID index);
```

Returns

- true - The BDPollingEnable feature is supported on the device
- false - The BDPollingEnable feature is not supported on the device

Description

This function identifies whether the BDPollingEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDPollEnable
- PLIB_SQI_DMABDPollDisable
- PLIB_SQI_DMABDPollIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDPollingEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDProcessState Function

Identifies whether the BDProcessState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDProcessState(SQI_MODULE_ID index);
```

Returns

- true - The BDProcessState feature is supported on the device
- false - The BDProcessState feature is not supported on the device

Description

This function identifies whether the BDProcessState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDProcessState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxBufCount Function

Identifies whether the BDRxBufCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxBufCount(SQI_MODULE_ID index);
```

Returns

- true - The BDRxBufCount feature is supported on the device
- false - The BDRxBufCount feature is not supported on the device

Description

This function identifies whether the BDRxBufCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveBufferCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxBufCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxLength Function

Identifies whether the BDRxLength feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxLength(SQI_MODULE_ID index);
```

Returns

- true - The BDRxLength feature is supported on the device
- false - The BDRxLength feature is not supported on the device

Description

This function identifies whether the BDRxLength feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveBufferLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxLength(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxState Function

Identifies whether the BDRxState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxState(SQI_MODULE_ID index);
```

Returns

- true - The BDRxState feature is supported on the device
- false - The BDRxState feature is not supported on the device

Description

This function identifies whether the BDRxState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxBufCount Function

Identifies whether the BDTxBufCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxBufCount(SQI_MODULE_ID index);
```

Returns

- true - The BDTxBufCount feature is supported on the device
- false - The BDTxBufCount feature is not supported on the device

Description

This function identifies whether the BDTxBufCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitBufferCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxBufCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxLength Function

Identifies whether the BDTxLength feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxLength(SQI_MODULE_ID index);
```

Returns

- true - The BDTxLength feature is supported on the device
- false - The BDTxLength feature is not supported on the device

Description

This function identifies whether the BDTxLength feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitBufferLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxLength(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxState Function

Identifies whether the BDTxState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxState(SQI_MODULE_ID index);
```

Returns

- true - The BDTxState feature is supported on the device
- false - The BDTxState feature is not supported on the device

Description

This function identifies whether the BDTxState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBurstControl Function

Identifies whether the BurstControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBurstControl(SQI_MODULE_ID index);
```

Returns

- true - The BurstControl feature is supported on the device
- false - The BurstControl feature is not supported on the device

Description

This function identifies whether the BurstControl feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_BurstEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBurstControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsChipSelect Function

Identifies whether the ChipSelect feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsChipSelect(SQI_MODULE_ID index);
```

Returns

- true - The ChipSelect feature is supported on the device
- false - The ChipSelect feature is not supported on the device

Description

This function identifies whether the ChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ChipSelectSet
- PLIB_SQI_ChipSelectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsChipSelect(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockControl Function

Identifies whether the ClockControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockControl(SQI_MODULE_ID index);
```

Returns

- true - The ClockControl feature is supported on the device
- false - The ClockControl feature is not supported on the device

Description

This function identifies whether the ClockControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ClockEnable
- PLIB_SQI_ClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockDivider Function

Identifies whether the ClockDivider feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockDivider(SQI_MODULE_ID index);
```

Returns

- true - The ClockDivider feature is supported on the device
- false - The ClockDivider feature is not supported on the device

Description

This function identifies whether the ClockDivider feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ClockDividerSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockDivider(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockReady Function

Identifies whether the ClockReady feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockReady(SQI_MODULE_ID index);
```

Returns

- true - The ClockReady feature is supported on the device
- false - The ClockReady feature is not supported on the device

Description

This function identifies whether the ClockReady feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ClockIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockReady(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufThreshold Function

Identifies whether the ConBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The ConBufThreshold feature is supported on the device
- false - The ConBufThreshold feature is not supported on the device

Description

This function identifies whether the ConBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ControlBufferThresholdSet
- PLIB_SQI_ControlBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsConfigWord Function

Identifies whether the ConfigWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConfigWord(SQI_MODULE_ID index);
```

Returns

- true - The ConfigWord feature is supported on the device
- false - The ConfigWord feature is not supported on the device

Description

This function identifies whether the ConfigWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ConfigWordSet
- PLIB_SQI_ConfigWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConfigWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsControlWord Function

Identifies whether the ControlWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsControlWord(SQI_MODULE_ID index);
```

Returns

- true - The ControlWord feature is supported on the device
- false - The ControlWord feature is not supported on the device

Description

This function identifies whether the ControlWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ControlWordSet
- PLIB_SQI_ControlWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsControlWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsCSDeassert Function

Identifies whether the CSDeassert feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCSDeassert(SQI_MODULE_ID index);
```

Returns

- true - The CSDeassert feature is supported on the device
- false - The CSDeassert feature is not supported on the device

Description

This function identifies whether the CSDeassert feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ChipSelectDeassertEnable
- PLIB_SQI_ChipSelectDeassertDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCSDeassert(SQI_MODULE_ID index)

PLIB_SQI_ExistsCSOutputEnable Function

Identifies whether the CSOutputEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCSOutputEnable(SQI_MODULE_ID index);
```

Returns

- true - The CSOutputEnable feature is supported on the device
- false - The CSOutputEnable feature is not supported on the device

Description

This function identifies whether the CSOutputEnable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_CSOutputEnableSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCSOutputEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataFormat Function

Identifies whether the DataFormat feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataFormat(SQI_MODULE_ID index);
```

Returns

- true - The DataFormat feature is supported on the device
- false - The DataFormat feature is not supported on the device

Description

This function identifies whether the DataFormat feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DataFormatSet
- PLIB_SQI_DataFormatGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataFormat(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataModeControl Function

Identifies whether the DataModeControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataModeControl(SQI_MODULE_ID index);
```

Returns

- true - The DataModeControl feature is supported on the device
- false - The DataModeControl feature is not supported on the device

Description

This function identifies whether the DataModeControl feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataModeControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataOutputEnable Function

Identifies whether the DataOutputEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataOutputEnable(SQI_MODULE_ID index);
```

Returns

- true - The DataOutputEnable feature is supported on the device
- false - The DataOutputEnable feature is not supported on the device

Description

This function identifies whether the DataOutputEnable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataOutputEnableSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataOutputEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataPinStatus Function

Identifies whether the DataPinStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataPinStatus(SQI_MODULE_ID index);
```

Returns

- true - The DataPinStatus feature is supported on the device
- false - The DataPinStatus feature is not supported on the device

Description

This function identifies whether the DataPinStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataLineStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataPinStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsDmaEnable Function

Identifies whether the DMAEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDmaEnable(SQI_MODULE_ID index);
```

Returns

- true - The DMAEnable feature is supported on the device
- false - The DMAEnable feature is not supported on the device

Description

This function identifies whether the DMAEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMAEnable
- PLIB_SQI_DMADisable
- PLIB_SQI_DMAIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDmaEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMAEngineBusy Function

Identifies whether the DMAEngineBusy feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMAEngineBusy(SQI_MODULE_ID index);
```

Returns

- true - The DMAEngineBusy feature is supported on the device
- false - The DMAEngineBusy feature is not supported on the device

Description

This function identifies whether the DMAEngineBusy feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMAEngineBusy(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMAProcessInProgress Function

Identifies whether the DMAProcessInProgress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMAProcessInProgress(SQI_MODULE_ID index);
```

Returns

- true - The DMAProcessInProgress feature is supported on the device
- false - The DMAProcessInProgress feature is not supported on the device

Description

This function identifies whether the DMAProcessInProgress feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMAHasStarted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMAProcessInProgress(SQI_MODULE_ID index)

PLIB_SQI_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsEnableControl(SQI_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_Enable
- PLIB_SQI_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsEnableControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsHoldPinControl Function

Identifies whether the HoldPinControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsHoldPinControl(SQI_MODULE_ID index);
```

Returns

- true - The HoldPinControl feature is supported on the device
- false - The HoldPinControl feature is not supported on the device

Description

This function identifies whether the HoldPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_HoldSet
- PLIB_SQI_HoldClear
- PLIB_SQI_HoldGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsHoldPinControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptControl Function

Identifies whether the InterruptControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptControl(SQI_MODULE_ID index);
```

Returns

- true - The InterruptControl feature is supported on the device
- false - The InterruptControl feature is not supported on the device

Description

This function identifies whether the InterruptControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_InterruptEnable
- PLIB_SQI_InterruptDisable
- PLIB_SQI_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptSignalControl Function

Identifies whether the InterruptSignalControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptSignalControl(SQI_MODULE_ID index);
```

Returns

- true - The InterruptSignalControl feature is supported on the device
- false - The InterruptSignalControl feature is not supported on the device

Description

This function identifies whether the InterruptSignalControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_InterruptSignalEnable
- PLIB_SQI_InterruptSignalDisable
- PLIB_SQI_InterruptSignalIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptSignalControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptStatus Function

Identifies whether the InterruptStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptStatus(SQI_MODULE_ID index);
```

Returns

- true - The InterruptStatus feature is supported on the device
- false - The InterruptStatus feature is not supported on the device

Description

This function identifies whether the InterruptStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_InterruptFlagGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsLaneMode Function

Identifies whether the LaneMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsLaneMode(SQI_MODULE_ID index);
```

Returns

- true - The LaneMode feature is supported on the device
- false - The LaneMode feature is not supported on the device

Description

This function identifies whether the LaneMode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_LaneModeSet

- PLIB_SQI_LaneModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsLaneMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsReceiveLatch Function

Identifies whether the ReceiveLatch feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsReceiveLatch(SQI_MODULE_ID index);
```

Returns

- true - The ReceiveLatch feature is supported on the device
- false - The ReceiveLatch feature is not supported on the device

Description

This function identifies whether the ReceiveLatch feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ReceiveLatchEnable
- PLIB_SQI_ReceiveLatchDisable
- PLIB_SQI_ReceiveLatchGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsReceiveLatch(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufferCount Function

Identifies whether the RxBufferCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufferCount(SQI_MODULE_ID index);
```

Returns

- true - The RxBufferCount feature is supported on the device

- false - The RxBufferCount feature is not supported on the device

Description

This function identifies whether the RxBufferCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_NumberOfReceiveBufferReads

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufferCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufIntThreshold Function

Identifies whether the RxBufIntThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufIntThreshold(SQI_MODULE_ID index);
```

Returns

- true - The RxBufIntThreshold feature is supported on the device
- false - The RxBufIntThreshold feature is not supported on the device

Description

This function identifies whether the RxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_RxBufferThresholdIntSet
- PLIB_SQI_RxBufferThresholdIntGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufIntThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufThreshold Function

Identifies whether the RxBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The RxBufThreshold feature is supported on the device
- false - The RxBufThreshold feature is not supported on the device

Description

This function identifies whether the RxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_RxBufferThresholdSet
- PLIB_SQI_RxBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxData Function

Identifies whether the RxData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxData(SQI_MODULE_ID index);
```

Returns

- true - The RxData feature is supported on the device
- false - The RxData feature is not supported on the device

Description

This function identifies whether the RxData feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ReceiveData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxData(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxUnderRun Function

Identifies whether the RxUnderRun feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxUnderRun(SQI_MODULE_ID index);
```

Returns

- true - The RxUnderRun feature is supported on the device
- false - The RxUnderRun feature is not supported on the device

Description

This function identifies whether the RxUnderRun feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ReceiveBufferIsUnderrun

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxUnderRun(SQI_MODULE_ID index)

PLIB_SQI_ExistsSoftReset Function

Identifies whether the SoftReset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The SoftReset feature is supported on the device
- false - The SoftReset feature is not supported on the device

Description

This function identifies whether the SoftReset feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_SoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferCommand Function

Identifies whether the TransferCommand feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferCommand(SQI_MODULE_ID index);
```

Returns

- true - The TransferCommand feature is supported on the device
- false - The TransferCommand feature is not supported on the device

Description

This function identifies whether the TransferCommand feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TransferDirectionSet
- PLIB_SQI_TransferDirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferCommand(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferCount Function

Identifies whether the TransferCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferCount(SQI_MODULE_ID index);
```

Returns

- true - The TransferCount feature is supported on the device
- false - The TransferCount feature is not supported on the device

Description

This function identifies whether the TransferCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ByteCountSet
- PLIB_SQI_ByteCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferModeControl Function

Identifies whether the TransferModeControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferModeControl(SQI_MODULE_ID index);
```

Returns

- true - The TransferModeControl feature is supported on the device
- false - The TransferModeControl feature is not supported on the device

Description

This function identifies whether the TransferModeControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TransferModeSet
- PLIB_SQI_TransferModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferModeControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufferFree Function

Identifies whether the TxBufferFree feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufferFree(SQI_MODULE_ID index);
```

Returns

- true - The TxBufferFree feature is supported on the device
- false - The TxBufferFree feature is not supported on the device

Description

This function identifies whether the TxBufferFree feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitBufferFreeSpaceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufferFree(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufIntThreshold Function

Identifies whether the TxBufIntThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufIntThreshold(SQI_MODULE_ID index);
```

Returns

- true - The TxBufIntThreshold feature is supported on the device
- false - The TxBufIntThreshold feature is not supported on the device

Description

This function identifies whether the TxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TxBufferThresholdIntSet
- PLIB_SQI_TxBufferThresholdIntGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufIntThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufThreshold Function

Identifies whether the TxBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The TxBufThreshold feature is supported on the device
- false - The TxBufThreshold feature is not supported on the device

Description

This function identifies whether the TxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TxBufferThresholdSet
- PLIB_SQI_TxBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxData Function

Identifies whether the TxData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxData(SQI_MODULE_ID index);
```

Returns

- true - The TxData feature is supported on the device
- false - The TxData feature is not supported on the device

Description

This function identifies whether the TxData feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxData(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxOverFlow Function

Identifies whether the TxOverFlow feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxOverFlow(SQI_MODULE_ID index);
```

Returns

- true - The TxOverFlow feature is supported on the device
- false - The TxOverFlow feature is not supported on the device

Description

This function identifies whether the TxOverFlow feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitBufferHasOverflowed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxOverFlow(SQI_MODULE_ID index)

PLIB_SQI_ExistsWPPinControl Function

Identifies whether the WPPinControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsWPPinControl(SQI_MODULE_ID index);
```

Returns

- true - The WPPinControl feature is supported on the device
- false - The WPPinControl feature is not supported on the device

Description

This function identifies whether the WPPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_WriteProtectSet
- PLIB_SQI_WriteProtectClear
- PLIB_SQI_WriteProtectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsWPPinControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPChipSelect Function

Identifies whether the XIPChipSelect feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPChipSelect(SQI_MODULE_ID index);
```

Returns

- true - The XIPChipSelect feature is supported on the device
- false - The XIPChipSelect feature is not supported on the device

Description

This function identifies whether the XIPChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPChipSelectSet
- PLIB_SQI_XIPChipSelectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPChipSelect(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord1 Function

Identifies whether the XIPControlWord1 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord1(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord1 feature is supported on the device
- false - The XIPControlWord1 feature is not supported on the device

Description

This function identifies whether the XIPControlWord1 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord1Set
- PLIB_SQI_XIPControlWord1Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord1(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord2 Function

Identifies whether the XIPControlWord2 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord2(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord2 feature is supported on the device
- false - The XIPControlWord2 feature is not supported on the device

Description

This function identifies whether the XIPControlWord2 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord2Set
- PLIB_SQI_XIPControlWord2Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord2(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPLaneMode Function

Identifies whether the XIPLaneMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPLaneMode(SQI_MODULE_ID index);
```

Returns

- true - The XIPLaneMode feature is supported on the device
- false - The XIPLaneMode feature is not supported on the device

Description

This function identifies whether the XIPLaneMode feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_XIPLaneModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPLaneMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPModeBytes Function

Identifies whether the XIPModeBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPModeBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPModeBytes feature is supported on the device
- false - The XIPModeBytes feature is not supported on the device

Description

This function identifies whether the XIPModeBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPModeBytesSet
- PLIB_SQI_XIPModeBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPModeBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPModeCode Function

Identifies whether the XIPModeCode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPModeCode(SQI_MODULE_ID index);
```

Returns

- true - The XIPModeCode feature is supported on the device
- false - The XIPModeCode feature is not supported on the device

Description

This function identifies whether the XIPModeCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPModeCodeSet
- PLIB_SQI_XIPModeCodeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPModeCode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPNumberOfAddressBytes Function

Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPNumberOfAddressBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPNumberOfAddressBytes feature is supported on the device
- false - The XIPNumberOfAddressBytes feature is not supported on the device

Description

This function identifies whether the XIPNumberOfAddressBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPAddressBytesSet
- PLIB_SQI_XIPAddressBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPNumberOfAddressBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPNumberOfDummyBytes Function

Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPNumberOfDummyBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPNumberOfDummyBytes feature is supported on the device
- false - The XIPNumberOfDummyBytes feature is not supported on the device

Description

This function identifies whether the XIPNumberOfDummyBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPDummyBytesSet
- PLIB_SQI_XIPDummyBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPNumberOfDummyBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPReadOpCode Function

Identifies whether the XIPReadOpCode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPReadOpCode(SQI_MODULE_ID index);
```

Returns

- true - The XIPReadOpCode feature is supported on the device
- false - The XIPReadOpCode feature is not supported on the device

Description

This function identifies whether the XIPReadOpCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPReadOpcodeSet
- PLIB_SQI_XIPReadOpcodeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPReadOpCode(SQI_MODULE_ID index)

PLIB_SQI_ExistsCommandStatus Function

Identifies whether the CommandStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCommandStatus(SQI_MODULE_ID index);
```

Returns

- true - The CommandStatus feature is supported on the device
- false - The CommandStatus feature is not supported on the device

Description

This function identifies whether the CommandStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_CommandStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCommandStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufAvailable Function

Identifies whether the ConBufWordsAvailable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufAvailable(SQI_MODULE_ID index);
```

Returns

- true - The ConBufWordsAvailable feature is supported on the device
- false - The ConBufWordsAvailable feature is not supported on the device

Description

This function identifies whether the ConBufWordsAvailable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ConBufWordsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufAvailable(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufferSoftReset Function

Identifies whether the control buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The control buffer soft reset feature is supported on the device
- false - The control buffer soft reset feature is not supported on the device

Description

This function identifies whether the control buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_ConBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsDDRMode Function

Identifies whether the DDRMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDDRMode(SQI_MODULE_ID index);
```

Returns

- true - The DDRModeSet and DDRModeGet feature is supported on the device
- false - The DDRModeSet and DDRModeGet feature is not supported on the device

Description

This function identifies whether the DDRModeSet feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DDRModeSet
- PLIB_SQI_DDRModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDDRMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMABDFetch Function

Identifies whether the DMABDFetch feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMABDFetch(SQI_MODULE_ID index);
```

Returns

- true - The DMABDFetch feature is supported on the device
- false - The DMABDFetch feature is not supported on the device

Description

This function identifies whether the DMABDFetch feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDFetchStart
- PLIB_SQI_DMABDFetchStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMABDFetch(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufferSoftReset Function

Identifies whether the receive buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The receive buffer soft reset feature is supported on the device
- false - The receive buffer soft reset feature is not supported on the device

Description

This function identifies whether the receive buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_RxBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsStatusCheck Function

Identifies whether the StatusCheckSet feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsStatusCheck(SQI_MODULE_ID index);
```

Returns

- true - The StatusCheckSet feature is supported on the device
- false - The StatusCheckSet feature is not supported on the device

Description

This function identifies whether the StatusCheck feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_StatusCheckSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsStatusCheck(SQI_MODULE_ID index)

PLIB_SQI_ExistsTapDelay Function

Identifies whether the TapDelay feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTapDelay(SQI_MODULE_ID index);
```

Returns

- true - The TapDelaySet feature is supported on the device
- false - The TapDelaySet feature is not supported on the device

Description

This function identifies whether the TapDelay feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TapDelaySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTapDelay(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufferSoftReset Function

Identifies whether the transmit buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The transmit buffer soft reset feature is supported on the device
- false - The transmit buffer soft reset feature is not supported on the device

Description

This function identifies whether the transmit buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_TxBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord3 Function

Identifies whether the XIPControlWord3 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord3(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord3Set and XIPControlWord3Get feature is supported on the device
- false - The XIPControlWord3Set and XIPControlWord3Get feature is not supported on the device

Description

This function identifies whether the XIPControlWord3 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord3Set

- PLIB_SQI_XIPControlWord3Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord3(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord4 Function

Identifies whether the XIPControlWord4 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord4(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord4Set and XIPControlWord4Get feature is supported on the device
- false - The XIPControlWord4Set and XIPControlWord4Get feature is not supported on the device

Description

This function identifies whether the XIPControlWord4 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord4Set
- PLIB_SQI_XIPControlWord4Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord4(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPDDRMode Function

Identifies whether the XIPDDRMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPDDRMode(SQI_MODULE_ID index);
```

Returns

- true - The XIPDDRModeSet feature is supported on the device
- false - The XIPDDRModeSet feature is not supported on the device

Description

This function identifies whether the XIPDDRMode feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_XIPDDRModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPDDRMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPSDRCommandDDRData Function

Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPSDRCommandDDRData(SQI_MODULE_ID index);
```

Returns

- true - The XIPSDRCommandDDRDataSet and XIPSDRCommandDDRDataSetfeatureGet is supported on the device
- false - The XIPSDRCommandDDRDataSet and XIPSDRCommandDDRDataGet not supported on the device

Description

This function identifies whether the XIPSDRCommandDDRData feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPSDRCommandDDRDataSet
- PLIB_SQI_XIPSDRCommandDDRDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPSDRCommandDDRData(SQI_MODULE_ID index)

k) Data Types and Constants

SQI_ADDR_BYTES Enumeration

Defines the list of SQI address bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  ADDR_BYTES_4,
  ADDR_BYTES_3,
  ADDR_BYTES_2,
  ADDR_BYTES_1,
  ADDR_BYTES_0
} SQI_ADDR_BYTES;
```

Members

Members	Description
ADDR_BYTES_4	SQI is set to Transmit 4 Address Bytes
ADDR_BYTES_3	SQI is set to Transmit 3 Address Bytes
ADDR_BYTES_2	SQI is set to Transmit 2 Address Bytes
ADDR_BYTES_1	SQI is set to Transmit 1 Address Bytes
ADDR_BYTES_0	SQI is set to Transmit No Address Bytes

Description

SQI Number of Address Bytes

This macro defines the list of SQI address bytes.

SQI_BD_CTRL_WORD Enumeration

Defines the list of SQI Buffer Descriptor control word.

File

help_plib_sqi.h

C

```
typedef enum {
  BD_ENABLE,
  BD_DISABLE,
  CS_1_ACTIVE,
  CS_0_ACTIVE,
  FLASH_SC_ENABLE,
  FLASH_SC_DISABLE,
  SET_LSBF,
  SET_MSBF,
  DMA_LANE_QUAD,
  DMA_LANE_DUAL,
  DMA_LANE_SINGLE,
  DMA_IN_TRANSMIT,
  DMA_IN_RECEIVE,
  LAST_BD,
  NOT_LAST_BD,
  LAST_PKT,
  NOT_LAST_PKT,
  PKT_INT_ENABLE,
  PKT_INT_DISABLE,
  BD_DONE_INT_ENABLE,
  BD_DONE_INT_DISABLE,
  BD_BUF_LENGTH
} SQI_BD_CTRL_WORD;
```

Members

Members	Description
BD_ENABLE	Current Buffer Descriptor is Enabled
BD_DISABLE	Current Buffer Descriptor is Disabled
CS_1_ACTIVE	Chip Select 1 is Active
CS_0_ACTIVE	Chip Select 0 is Active
FLASH_SC_ENABLE	Automatic Status Check Enabled
FLASH_SC_DISABLE	Automatic Status Check Disabled
SET_LSBF	Least Significant Bit First
SET_MSBF	Most Significant Bit First

DMA_LANE_QUAD	DMA writes/reads in Quad Lane Mode
DMA_LANE_DUAL	DMA writes/reads in Dual Lane Mode
DMA_LANE_SINGLE	DMA writes/reads in Single Lane Mode
DMA_IN_TRANSMIT	DMA is Transmitting
DMA_IN_RECEIVE	DMA is Receiving
LAST_BD	Indicates Last Buffer Descriptor in the List
NOT_LAST_BD	Indicates Non Last Buffer Descriptor in the List
LAST_PKT	Indicates Last Packet of the Data Chunk
NOT_LAST_PKT	Indicates Non Last Packet of the Data Chunk
PKT_INT_ENABLE	Indicates Packet Interrupt is Enabled
PKT_INT_DISABLE	Indicates Packet Interrupt is Disabled
BD_DONE_INT_ENABLE	Indicates Buffer Descriptor Done Interrupt is Enabled
BD_DONE_INT_DISABLE	Indicates Buffer Descriptor Done Interrupt is Disabled
BD_BUF_LENGTH	Indicates Buffer Length

Description

SQI Buffer Descriptor Control Words

This macro defines the list of SQI Buffer Descriptor control word.

SQI_BD_STATE Enumeration

Defines the list of SQI Buffer Descriptor states.

File

help_plib_sqi.h

C

```
typedef enum {
  BD_STATE_DISABLED,
  BD_STATE_DONE,
  BD_STATE_PROCESSING_DATA,
  BD_STATE_BEING_FETCHED,
  BD_STATE_FETCH_REQ_PENDING,
  BD_STATE_IDLE
} SQI_BD_STATE;
```

Members

Members	Description
BD_STATE_DISABLED	Fetched Buffer Descriptor is Disabled
BD_STATE_DONE	Fetched Buffer Descriptor Processed
BD_STATE_PROCESSING_DATA	Fetched Buffer Descriptor is in Data transfer phase
BD_STATE_BEING_FETCHED	In the process of Fetching the Buffer Descriptor
BD_STATE_FETCH_REQ_PENDING	Buffer Descriptor Fetch Request Pending
BD_STATE_IDLE	Buffer Descriptor process is idle

Description

SQI Buffer Descriptor (BD) State

This macro defines the list of SQI Buffer Descriptor states.

SQI_CLK_DIV Enumeration

Defines the list of SQI Clock Divider values.

File

help_plib_sqi.h

C

```
typedef enum {
  CLK_DIV_256,
  CLK_DIV_128,
```

```
  CLK_DIV_64,
  CLK_DIV_32,
  CLK_DIV_16,
  CLK_DIV_8,
  CLK_DIV_4,
  CLK_DIV_2,
  CLK_DIV_1
} SQI_CLK_DIV;
```

Members

Members	Description
CLK_DIV_256	SQI Clock is divided by 256
CLK_DIV_128	SQI Clock is divided by 128
CLK_DIV_64	SQI Clock is divided by 64
CLK_DIV_32	SQI Clock is divided by 32
CLK_DIV_16	SQI Clock is divided by 16
CLK_DIV_8	SQI Clock is divided by 8
CLK_DIV_4	SQI Clock is divided by 4
CLK_DIV_2	SQI Clock is divided by 2
CLK_DIV_1	SQI Clock is not divided

Description

SQI Clock Divider

This macro defines the list of SQI Clock Divider values.

SQI_CS_NUM Enumeration

Defines the list of SQI Chip Selects.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CS_1,
  SQI_CS_0
} SQI_CS_NUM;
```

Members

Members	Description
SQI_CS_1	SQI Chip Select 1
SQI_CS_0	SQI Chip Select 0

Description

SQI Device Select or Chip Select

This macro defines the list of SQI Chip Selects.

SQI_CS_OEN Enumeration

Defines the list of SQI Chip Selects on which output is enable.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CS_OEN_BOTH,
  SQI_CS_OEN_1,
  SQI_CS_OEN_0,
  SQI_CS_OEN_NONE
} SQI_CS_OEN;
```

Members

Members	Description
SQI_CS_OEN_BOTH	SQI chip select 0 and 1 are enabled
SQI_CS_OEN_1	SQI chip select 1 is enabled but chip select 0 is disabled
SQI_CS_OEN_0	SQI chip select 0 is enabled but chip select 1 is disabled
SQI_CS_OEN_NONE	SQI chip select 0 and 1 are disabled

Description

SQI Chip Select Output Enable

This macro defines the list of SQI Chip Selects on which output is enabled.

SQI_DATA_FORMAT Enumeration

Defines the Data Format Options available (LSBF/MSBF).

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_FORMAT_LSBF,
  SQI_DATA_FORMAT_MSBF
} SQI_DATA_FORMAT;
```

Members

Members	Description
SQI_DATA_FORMAT_LSBF	SQI Data is Least Significant Bit First Formatted
SQI_DATA_FORMAT_MSBF	SQI Data is Most Significant Bit First Formatted

Description

SQI Data Mode

This macro defines the Data Formats (LSBF/MSBF).

SQI_DATA_MODE Enumeration

Defines the list of SQI Data modes.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_MODE_3,
  SQI_DATA_MODE_0
} SQI_DATA_MODE;
```

Members

Members	Description
SQI_DATA_MODE_3	SQI is in SPI mode 3
SQI_DATA_MODE_0	SQI is in SPI mode 0

Description

SQI Data Mode

This macro defines the list of SQI Data modes.

SQI_DATA_OEN Enumeration

Defines the list of SQI Data pins on which output is enabled.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_OEN_QUAD,
  SQI_DATA_OEN_DUAL,
  SQI_DATA_OEN_SINGLE
} SQI_DATA_OEN;
```

Members

Members	Description
SQI_DATA_OEN_QUAD	SQI data outputs 3 to 0 are enabled (quad lane mode)
SQI_DATA_OEN_DUAL	SQI data outputs 1 and 0 are enabled (dual lane mode)
SQI_DATA_OEN_SINGLE	SQI data output 0 is enabled (single lane mode)

Description

SQI Data Output Enable

This macro defines the list of SQI Data Output pins on which output is enabled.

SQI_DATA_TYPE Type

Data type defining the SQI Data size.

File

help_plib_sqi.h

C

```
typedef uint32_t SQI_DATA_TYPE;
```

Description

SQI Data Type definition

This data type defines the SQI Data size

SQI_DUMMY_BYTES Enumeration

Defines the list of SQI dummy bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  DUMMY_BYTES_7,
  DUMMY_BYTES_6,
  DUMMY_BYTES_5,
  DUMMY_BYTES_4,
  DUMMY_BYTES_3,
  DUMMY_BYTES_2,
  DUMMY_BYTE_1,
  DUMMY_BYTES_0
} SQI_DUMMY_BYTES;
```

Members

Members	Description
DUMMY_BYTES_7	SQI is set to Transmit 7 Dummy Bytes
DUMMY_BYTES_6	SQI is set to Transmit 6 Dummy Bytes
DUMMY_BYTES_5	SQI is set to Transmit 5 Dummy Bytes
DUMMY_BYTES_4	SQI is set to Transmit 4 Dummy Bytes
DUMMY_BYTES_3	SQI is set to Transmit 3 Dummy Bytes
DUMMY_BYTES_2	SQI is set to Transmit 2 Dummy Bytes

DUMMY_BYTE_1	SQI is set to Transmit 1 Dummy Bytes
DUMMY_BYTES_0	SQI is set to Transmit No Dummy Bytes

Description

SQI Number of Dummy Bytes

This macro defines the list of SQI dummy bytes after address bytes.

SQI_INTERRUPTS Enumeration

Defines the list of SQI interrupts.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_PKTCOMP,
  SQI_BDDONE,
  SQI_CONTHR,
  SQI_CONEMPTY,
  SQI_CONFULL,
  SQI_RXTHR,
  SQI_RXFULL,
  SQI_RXEMPTY,
  SQI_TXTHR,
  SQI_TXFULL,
  SQI_TXEMPTY
} SQI_INTERRUPTS;
```

Members

Members	Description
SQI_PKTCOMP	SQI Packet Complete Interrupt (used in DMA mode)
SQI_BDDONE	SQI Buffer Descriptor Done Interrupt (used in DMA mode)
SQI_CONTHR	SQI Control Buffer Threshold Interrupt
SQI_CONEMPTY	SQI Control Buffer Empty Interrupt
SQI_CONFULL	SQI Control Buffer Full Interrupt
SQI_RXTHR	SQI Receive Buffer Threshold Interrupt
SQI_RXFULL	SQI Receive Buffer Empty Interrupt
SQI_RXEMPTY	SQI Receive Buffer Full Interrupt
SQI_TXTHR	SQI Transmit Buffer Threshold Interrupt
SQI_TXFULL	SQI Transmit Buffer Empty Interrupt
SQI_TXEMPTY	SQI Transmit Buffer Full Interrupt

Description

SQI Interrupt List

This macro defines the list of SQI interrupts.

SQI_LANE_MODE Enumeration

Defines the list of SQI Lane modes (single/dual/quad).

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_LANE_QUAD,
  SQI_LANE_DUAL,
  SQI_LANE_SINGLE
} SQI_LANE_MODE;
```

Members

Members	Description
SQI_LANE_QUAD	SQI is set into Quad Lane Mode
SQI_LANE_DUAL	SQI is set into Dual Lane Mode
SQI_LANE_SINGLE	SQI is set into Single Lane Mode

Description

SQI Lane Mode

This macro defines the list of SQI Lane modes.

SQI_MODE_BYTES Enumeration

Defines the list of SQI mode bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  MODE_BYTES_3,
  MODE_BYTES_2,
  MODE_BYTES_1,
  MODE_BYTES_0
} SQI_MODE_BYTES;
```

Members

Members	Description
MODE_BYTES_3	SQI is set to Transmit 3 Mode Bytes
MODE_BYTES_2	SQI is set to Transmit 2 Mode Bytes
MODE_BYTES_1	SQI is set to Transmit 1 Mode Bytes
MODE_BYTES_0	SQI is set to Transmit No Mode Bytes

Description

SQI Number of Mode Bytes

This macro defines the list of SQI mode bytes allocated for mode code.

SQI_MODULE_ID Enumeration

Identifies the supported SQI modules.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_ID_1,
  SQI_NUMBER_OF_MODULES
} SQI_MODULE_ID;
```

Members

Members	Description
SQI_ID_1	SQI Module 1 ID
SQI_NUMBER_OF_MODULES	Number of available SQI modules

Description

SQI Module ID

This enumeration identifies the SQI modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

SQI_XFER_CMD Enumeration

Defines the list of SQI transfer commands.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CMD_RECEIVE,
  SQI_CMD_TRANSMIT,
  SQI_CMD_IDLE
} SQI_XFER_CMD;
```

Members

Members	Description
SQI_CMD_RECEIVE	SQI issues a Receive Command
SQI_CMD_TRANSMIT	SQI issues a Transmit Command
SQI_CMD_IDLE	SQI is Idle

Description

SQI Transfer Command

This macro defines the list of SQI transfer commands.

SQI_XFER_MODE Enumeration

Defines the list of SQI Transfer modes.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_XFER_MODE_XIP,
  SQI_XFER_MODE_DMA,
  SQI_XFER_MODE_PIO
} SQI_XFER_MODE;
```

Members

Members	Description
SQI_XFER_MODE_XIP	SQI is in XIP mode
SQI_XFER_MODE_DMA	SQI is in DMA mode
SQI_XFER_MODE_PIO	SQI is in PIO mode

Description

SQI Transfer Mode

This macro defines the list of SQI Transfer modes.

Files

Files

Name	Description
plib_sqi.h	SQI Peripheral Library Interface Header for common definitions.
help_plib_sqi.h	Identifies the various enumerations in SQI modules supported.

Description

This section lists the source and header files used by the library.

plib_sqi.h

SQI Peripheral Library Interface Header for common definitions.

Functions

	Name	Description
	PLIB_SQI_BurstEnable	Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.
	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
	PLIB_SQI_ChipSelectDeassertDisable	Clears the Chip Select deassert.
	PLIB_SQI_ChipSelectDeassertEnable	Sets the Chip Select deassert.
	PLIB_SQI_ChipSelectGet	Returns the Chip Select that is currently active.
	PLIB_SQI_ChipSelectSet	Activates the Chip Select.
	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
	PLIB_SQI_ClockDividerSet	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
	PLIB_SQI_ClockEnable	Enables the SQI transfer clock.
	PLIB_SQI_ClockIsStable	Returns SQI transfer clock state.
	PLIB_SQI_CommandStatusGet	Returns the SQI command (transmit/receive/idle).
	PLIB_SQI_ConBufferSoftReset	Issues a soft reset to the SQI control buffer.
	PLIB_SQI_ConBufWordsAvailable	Returns the number of control buffer words available.
	PLIB_SQI_ConfigWordGet	Gets the SQI Configuration Word.
	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
	PLIB_SQI_ControlWordGet	Get the SQI Control Word.
	PLIB_SQI_ControlWordSet	Sets the SQI Control Word.
	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI Chip Select pins.
	PLIB_SQI_DataFormatGet	Returns the data format.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First (LSBF)..
	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation.
	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_DDRModeGet	Return the SQI data rate mode (single/double) value.
	PLIB_SQI_DDRModeSet	Sets SQI communication to DDR mode.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_DMABDBaseAddressGet	Returns the address of the base buffer descriptor.
	PLIB_SQI_DMABDBaseAddressSet	Sets the address of the base buffer descriptor.
	PLIB_SQI_DMABDControlWordGet	Returns Current Buffer Descriptor Control Word Information.
	PLIB_SQI_DMABDCurrentAddressGet	Returns the address of the current buffer descriptor in process.
	PLIB_SQI_DMABDFetchStart	Starts the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDFetchStop	Stops the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns whether or not the DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollIsEnabled	Returns whether or not the DMA buffer descriptor poll is enabled.
	PLIB_SQI_DMABDReceiveBufferCountGet	Returns the receive buffer space in bytes.
	PLIB_SQI_DMABDReceiveBufferLengthGet	Returns the receive length in bytes.
	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.

	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns whether or no the DMA process has started.
	PLIB_SQI_DMAIsEnabled	Returns whether or not DMA is enabled.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module.
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxBufCount	Identifies whether the BDTxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxLength	Identifies whether the BDTxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxState	Identifies whether the BDTxState feature exists on the SQI module.
	PLIB_SQI_ExistsBurstControl	Identifies whether the BurstControl feature exists on the SQI module.
	PLIB_SQI_ExistsChipSelect	Identifies whether the ChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsClockControl	Identifies whether the ClockControl feature exists on the SQI module.
	PLIB_SQI_ExistsClockDivider	Identifies whether the ClockDivider feature exists on the SQI module.
	PLIB_SQI_ExistsClockReady	Identifies whether the ClockReady feature exists on the SQI module.
	PLIB_SQI_ExistsCommandStatus	Identifies whether the CommandStatus feature exists on the SQI module.
	PLIB_SQI_ExistsConBufAvailable	Identifies whether the ConBufWordsAvailable feature exists on the SQI module.
	PLIB_SQI_ExistsConBufferSoftReset	Identifies whether the control buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsConBufThreshold	Identifies whether the ConBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsConfigWord	Identifies whether the ConfigWord feature exists on the SQI module.
	PLIB_SQI_ExistsControlWord	Identifies whether the ControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsCSDeassert	Identifies whether the CSDeassert feature exists on the SQI module.
	PLIB_SQI_ExistsCSOutputEnable	Identifies whether the CSOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataFormat	Identifies whether the DataFormat feature exists on the SQI module.
	PLIB_SQI_ExistsDataModeControl	Identifies whether the DataModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsDataOutputEnable	Identifies whether the DataOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataPinStatus	Identifies whether the DataPinStatus feature exists on the SQI module.
	PLIB_SQI_ExistsDDRMode	Identifies whether the DDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsDMABDFetch	Identifies whether the DMABDFetch feature exists on the SQI module.
	PLIB_SQI_ExistsDmaEnable	Identifies whether the DMAEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDMAEngineBusy	Identifies whether the DMAEngineBusy feature exists on the SQI module.
	PLIB_SQI_ExistsDMAProcessInProgress	Identifies whether the DMAProcessInProgress feature exists on the SQI module.
	PLIB_SQI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SQI module.
	PLIB_SQI_ExistsHoldPinControl	Identifies whether the HoldPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptControl	Identifies whether the InterruptControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptSignalControl	Identifies whether the InterruptSignalControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptStatus	Identifies whether the InterruptStatus feature exists on the SQI module.
	PLIB_SQI_ExistsLaneMode	Identifies whether the LaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsReceiveLatch	Identifies whether the ReceiveLatch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferCount	Identifies whether the RxBufferCount feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferSoftReset	Identifies whether the receive buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufIntThreshold	Identifies whether the RxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufThreshold	Identifies whether the RxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxData	Identifies whether the RxData feature exists on the SQI module.
	PLIB_SQI_ExistsRxUnderRun	Identifies whether the RxUnderRun feature exists on the SQI module.

	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module.
	PLIB_SQI_ExistsStatusCheck	Identifies whether the StatusCheckSet feature exists on the SQI module.
	PLIB_SQI_ExistsTapDelay	Identifies whether the TapDelay feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module.
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferSoftReset	Identifies whether the transmit buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module.
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module.
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord3	Identifies whether the XIPControlWord3 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord4	Identifies whether the XIPControlWord4 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPDDRMode	Identifies whether the XIPDDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPSDRCommandDDRData	Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.
	PLIB_SQI_HoldClear	Clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	Gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	Sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the passed interrupt source.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state.
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the passed interrupt signal source.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state.
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).
	PLIB_SQI_NumberOfReceiveBufferReads	Returns the number of receive buffer reads.
	PLIB_SQI_ReceiveBufferIsUnderrun	Returns the status of receive buffer.
	PLIB_SQI_ReceiveData	Reads the data from the receive buffer.
	PLIB_SQI_ReceiveLatchDisable	Disables the receive latch so receive data is discarded when in transmit mode.
	PLIB_SQI_ReceiveLatchEnable	Enables the receive latch so receive data is latched during transmit mode.
	PLIB_SQI_ReceiveLatchGet	Returns the receive latch status in transmit mode.
	PLIB_SQI_RxBufferSoftReset	Issues a soft reset to the SQI receive buffer.
	PLIB_SQI_RxBufferThresholdGet	Returns the receive command threshold.
	PLIB_SQI_RxBufferThresholdIntGet	Sets the receive buffer threshold interrupt.
	PLIB_SQI_RxBufferThresholdIntSet	Sets the receive buffer threshold for interrupt.
	PLIB_SQI_RxBufferThresholdSet	Sets the receive command threshold.
	PLIB_SQI_SoftReset	Issues a soft reset to the SQI module.
	PLIB_SQI_StatusCheckSet	Sets the Flash status check related control bits.
	PLIB_SQI_TapDelaySet	Sets the tap delays.
	PLIB_SQI_TransferDirectionGet	Returns the transfer command.
	PLIB_SQI_TransferDirectionSet	Sets the transfer command.

	PLIB_SQI_TransferModeGet	Returns the SQI transfer mode of operation.
	PLIB_SQI_TransferModeSet	Sets the SQI transfer mode of operation.
	PLIB_SQI_TransmitBufferFreeSpaceGet	Returns the number of transmit buffer words available.
	PLIB_SQI_TransmitBufferHasOverflowed	Returns the current status of the transmit buffer.
	PLIB_SQI_TransmitData	Writes data into the SQI transmit buffer.
	PLIB_SQI_TxBufferSoftReset	Issues a soft reset to the SQI transmit buffer.
	PLIB_SQI_TxBufferThresholdGet	Returns the transmit command threshold value.
	PLIB_SQI_TxBufferThresholdIntGet	Returns the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdIntSet	Sets the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdSet	Sets the transmit command threshold.
	PLIB_SQI_WriteProtectClear	Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	PLIB_SQI_WriteProtectGet	Gets the state of the write-protect feature on SQID2.
	PLIB_SQI_WriteProtectSet	Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.
	PLIB_SQI_XIPAddressBytesGet	Returns the number of address bytes.
	PLIB_SQI_XIPAddressBytesSet	Sets the number of address bytes.
	PLIB_SQI_XIPChipSelectGet	Returns the current active Chip Select.
	PLIB_SQI_XIPChipSelectSet	Activates the Chip Select in XIP mode.
	PLIB_SQI_XIPControlWord1Get	Get the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord1Set	Sets the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord2Get	Gets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord2Set	Sets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord3Get	Gets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord3Set	Sets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord4Get	Gets the XIP mode Control Word 4.
	PLIB_SQI_XIPControlWord4Set	Sets the XIP mode Control Word 4.
	PLIB_SQI_XIPDDRModeSet	Selects the rate mode (SDR/DDR) for different transactions in XIP mode.
	PLIB_SQI_XIPDummyBytesGet	Sets the number of dummy bytes.
	PLIB_SQI_XIPDummyBytesSet	Sets the number of dummy bytes.
	PLIB_SQI_XIPLaneModeSet	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	PLIB_SQI_XIPModeBytesGet	Returns the number of bytes used for mode code command.
	PLIB_SQI_XIPModeBytesSet	Allocates the bytes for mode code command.
	PLIB_SQI_XIPModeCodeGet	Returns the mode code op-code.
	PLIB_SQI_XIPModeCodeSet	Sets the mode code command.
	PLIB_SQI_XIPReadOpcodeGet	Returns the read op code in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read op code in XIP mode.
	PLIB_SQI_XIPSDRCommandDDRDataGet	Returns the SQI command in SDR mode and Data in DDR mode bit value.
	PLIB_SQI_XIPSDRCommandDDRDataSet	Sets the SQI command in SDR mode.

Description

Serial Quad Interface (SQI) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SQI Peripheral Library.

File Name

plib_sqi.h

Company

Microchip Technology Inc.

help_plib_sqi.h

Identifies the various enumerations in SQI modules supported.

Enumerations

	Name	Description
	SQI_ADDR_BYTES	Defines the list of SQI address bytes.
	SQI_BD_CTRL_WORD	Defines the list of SQI Buffer Descriptor control word.
	SQI_BD_STATE	Defines the list of SQI Buffer Descriptor states.
	SQI_CLK_DIV	Defines the list of SQI Clock Divider values.
	SQI_CS_NUM	Defines the list of SQI Chip Selects.
	SQI_CS_OEN	Defines the list of SQI Chip Selects on which output is enable.
	SQI_DATA_FORMAT	Defines the Data Format Options available (LSBF/MSBF).
	SQI_DATA_MODE	Defines the list of SQI Data modes.
	SQI_DATA_OEN	Defines the list of SQI Data pins on which output is enabled.
	SQI_DUMMY_BYTES	Defines the list of SQI dummy bytes.
	SQI_INTERRUPTS	Defines the list of SQI interrupts.
	SQI_LANE_MODE	Defines the list of SQI Lane modes (single/dual/quad).
	SQI_MODE_BYTES	Defines the list of SQI mode bytes.
	SQI_MODULE_ID	Identifies the supported SQI modules.
	SQI_XFER_CMD	Defines the list of SQI transfer commands.
	SQI_XFER_MODE	Defines the list of SQI Transfer modes.

Types

	Name	Description
	SQI_DATA_TYPE	Data type defining the SQI Data size.

Description

This enumeration identifies the SQI modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

File Name

help_plib_sqi.h

Company

Microchip Technology Inc.

Timer Peripheral Library

This section describes the Timer Peripheral Library.

Introduction

This library provides a low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The timer function is one of the basic features of a microcontroller. Timers are useful for generating accurate time-based periodic interrupts for software application or real-time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions or producing precise time delays. Some timers can also be used to control timing of other peripherals.

Using the Library

This topic describes the basic architecture of the Timer Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_tmr.h`

The interface to the Timer Peripheral Library is defined in the `plib_tmr.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Timer Peripheral Library must include `peripheral.h`.

Library File:

The Timer Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available timers on the device. Refer to the **"Timer"** chapters in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Timer module on your device.

Description

Microchip timers can be classified as:

- Overflow-based
- Period match-based

These timers are essentially counters of a specific size (8-, 16-, or 32-bits wide) that increment based on the clock cycle and the timer prescaler. An application can monitor these counters to determine how much time has elapsed.

The prescaler determines the timer granularity (resolution). It is a mechanism for generating the clock for the timer by the CPU clock. Every CPU has a clock source and the frequency of this source decides the rate at which instructions are executed by the processor. Most of the timers have the option of providing input clock prescalers, while some have the optional postscalers. The prescaler is used to divide the clock frequency of the Timer module and produce a clock for the timer.

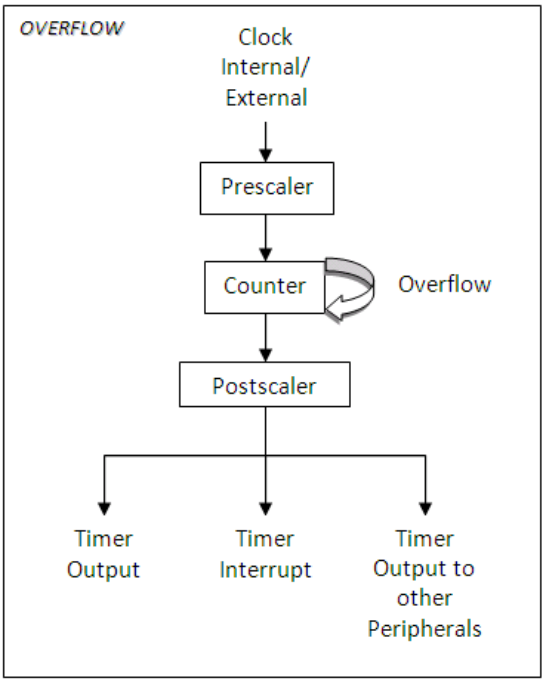
Each timer can be configured with a different source (internal or external) meaning, the source of the timer's input clock is selectable. Both internal and external sources are available, depending on the part in use, with some timers providing a clock synchronization mechanisms with reference to external source.

On some timers the clock edge, either the rising or the falling edge, on which the increment occurs, is selectable.

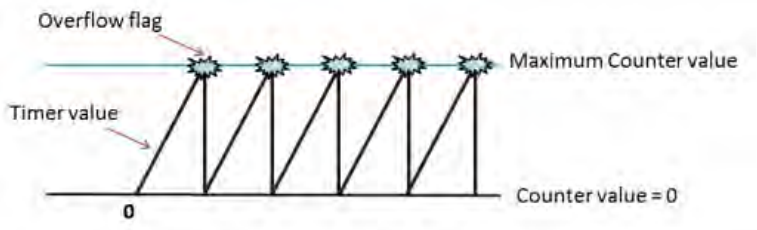
Overflow-based Timers

Overflow-based timers provide timer outputs or generate interrupts after a rollover from the maximum possible timer count.

Overflow Timer Abstraction Diagram



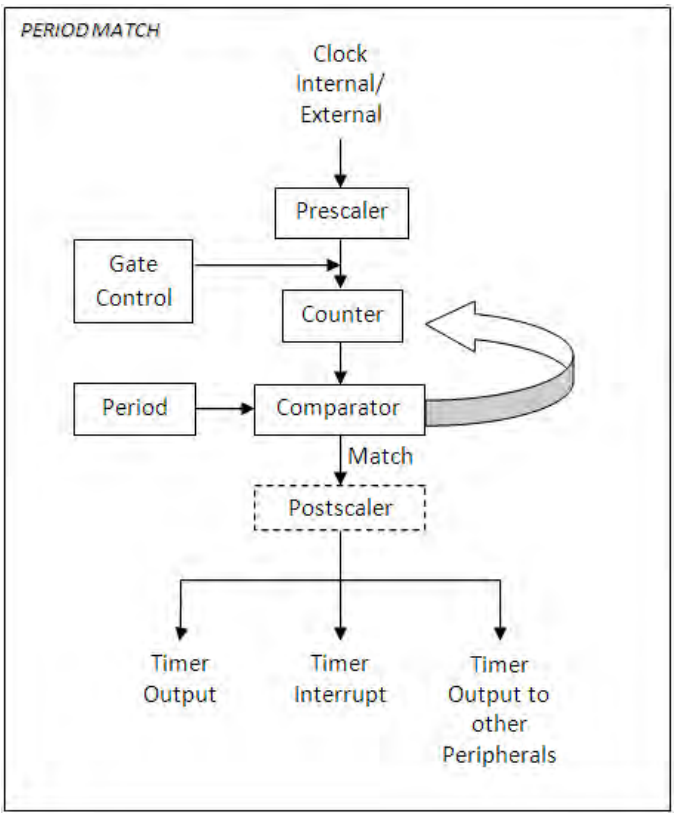
An overflow interrupt is triggered whenever the timer register overflows (i.e., reaches its maximum value based on its size as depicted by the following diagram).

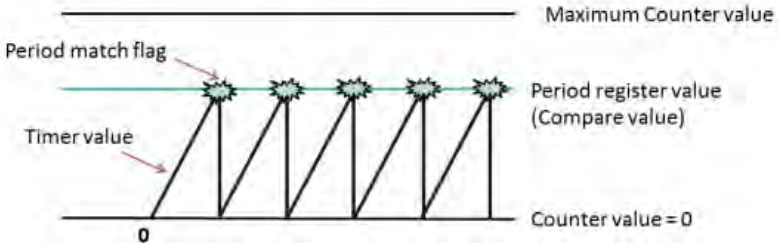


Period Match-based Timers

Period match-based timers provide the timer outputs or generate interrupts upon a match between the counter's count value against the preprogrammed period.

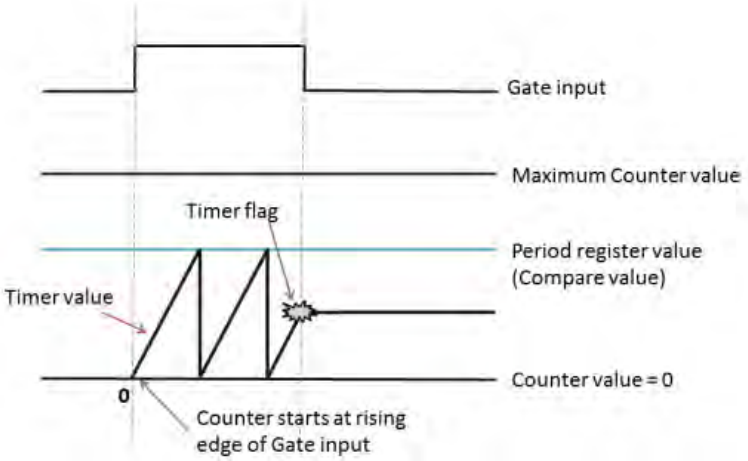
Period Match Timer Abstraction Diagram





Gated Time Accumulation/Gated Control Mode

The Gated Time Accumulation/Gate control mode allows the internal timer to increment based upon the duration of the high time applied to the timer input pin. In the Gated Time Accumulation mode, the timer clock source is derived from the internal system clock. When the input clock pin state is high, the timer will count up until a period match has occurred, or the input pin state is changed to a low state. A pin state transition from high-to-low will trigger an interrupt.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Timer module.

Library Interface Section	Description
General Setup Functions	This section provides a set of functions that are used for: • Starting and stopping the timer • Setting up Single-shot or Continuous mode of operation • Timer oscillator control • Selecting if the external event can reset the counter • Timer assignment to various modules as the clock source • Setting up the counter resolution
Power Control Functions	This section provides functions that are required for controlling the timer operation in power-saving modes such as Idle and Sleep.
Clock Source Control Functions	This section provides a set of functions that are used for: • Selecting the input clock source • Setting up external clock synchronization • Selecting the source edge to increment the counter • Getting the source edge information
Gate Control Functions	This section provides a set of functions for controlling the gating logic of the timers.
Preprocessing Functions	This section provides a set of functions that are used for prescaler setup.
Period Control Functions	This section provides a set of functions for reading and writing the period.
Counter Control Functions	This section provides a set of functions for: • Reading and writing the counter values • Asynchronous write control
Post-processing Functions	This section provides a set of functions for controlling how the Timer module operates with other peripherals. This section only covers the peripherals that can be controlled from inside of the Timer module. For peripherals that have a dedicated timer, or the control lies within the peripheral, refer to that peripheral for the interface into the timers. This section can also provide a postscaling option, if available, on the microcontroller.
Miscellaneous Functions	This section provides a set of functions that are required to read status information.

How the Library Works

Timer modules can operate in the following modes:

- Synchronous Clock Counter
- Synchronous External Clock Counter
- Asynchronous External Clock Counter
- Gated Timer

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Synchronous Internal Clock Counter

This section provides information and examples for setting up Synchronous Internal Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Description

Synchronous Clock Counter operation provides the following capabilities:

- Elapsed time measurements
- Time delays
- Periodic timer interrupts

Setup

Use the following steps for synchronous internal clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. Set the start value of the timer using PLIB_TMR_Counter16BitSet or PLIB_TMR_Counter32BitSet.
3. The period register value can be set using PLIB_TMR_Period8BitSet, PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
4. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
5. In this mode, the input clock source for the timer is the internal clock, and is selected using PLIB_TMR_ClockSourceSelect, depending on which clock is used as the internal clock in the timer module of the device.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Synchronous Counter (Internal Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as internal
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_TMRX_VALUE_1);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

Timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// Clear the timer
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0ff0)
{
   PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Synchronous External Clock Counter

This section provides information and examples for Synchronous External Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Description

Synchronous external clock counter operation provides the following capabilities:

- Counting periodic or non-periodic pulses
- Use external clock as time base for timers

Setup

Use the following steps for synchronous external clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.

2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet or PLIB_TMR_Counter32BitSet.
4. Some timer modules, can select an external clock as its input source. Use PLIB_TMR_ClockSourceSelect to select the external clock source.
5. The clock source is synchronized with the internal clock for synchronous operation. Use PLIB_TMR_ClockSourceExternalSyncEnable to synchronize the external clock source.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Synchronous Counter (External Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and enable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_256);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// read the current timer value
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0ff0)
{
   PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Asynchronous Counter

This section provides information and examples for setting up Asynchronous Counter operation.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Description

Asynchronous Counter operation provides the following capabilities:

- The timer can operate in Sleep mode and can generate an interrupt on a period match
- The processor can wake up from Sleep on the timer interrupt
- The timer can be clocked from the Secondary Oscillator for real-time clock applications

Setup

Use the following steps for Asynchronous Counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet and PLIB_TMR_Counter32BitSet.

4. Some timer modules can select an external clock as its input source. Use PLIB_TMR_ClockSourceSelect to select the external clock source.
5. For Asynchronous mode operation, the clock source is not synchronized with the internal clock. Use PLIB_TMR_ClockSourceExternalSyncDisable to disable the synchronization of the external clock source.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Asynchronous Counter Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Gated Timer

This section provides information and examples for setting up Gated Timer operation.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Description

Gated Timer operation can be used to allow measurement of an external signal.

Setup

Use the following steps for Gated Timer setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet and PLIB_TMR_Counter32BitSet.
4. In this mode, the input clock source for the timer is the internal clock, and is selected using PLIB_TMR_ClockSourceSelect, depending on which clock is used as the internal clock in the timer module of the device.
5. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired pre scaler into the function from the enumeration TMR_PRESCALE.
6. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
7. Gated operation can be enabled using the function PLIB_TMR_GateEnable.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Gated Timer Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
```

```
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F00);

// Enable the gate timer
PLIB_TMR_GateEnable(MY_TIMER_INSTANCE);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, on the falling edge of the gate signal.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Other Features

This section provides information on additional features.

Description

Each of the following modes can additionally support some or all of the features listed. Please refer to the specific device data sheet or family reference manual for details.

Operation During Sleep Mode

When the Timer module is configured to operate in Asynchronous mode (with an external clock source), it will continue to increment each timer clock (or prescale multiple of clocks) during Sleep mode.

Operation During Idle Mode

A Timer module can continue to operate in Idle mode by calling PLIB_TMR_StopInIdleDisable, or can stop operation in Idle mode by calling PLIB_TMR_StopInIdleEnable.

Timer Mode Control

- PLIB_TMR_Mode16BitEnable: This interface enables 16-bit mode and disables 32-bit mode
- PLIB_TMR_Mode32BitEnable: This interface enables 32-bit mode and disables 16-bit mode

Timer Usage Mode

Some timers operate in either Period Match mode or Overflow mode. The mode state can be retrieved from the Timer Peripheral Library using PLI_TMR_IsPeriodMatchBased. This function returns 'true' Period Match mode and 'false' for Overflow mode.

Timer Prescaler Information

Prescaler divisor information associated with the respective prescaler value selected through TMR_PRESCALE can be retrieved through PLIB_TMR_PrescaleDivisorGet.

The true prescaler value can be obtained by using PLIB_TMR_PrescaleGet.

Configuring the Library

The library is configured for the supported Timer modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.

b) Power Control Functions

	Name	Description
	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

c) Clock Source Control Functions

	Name	Description
	PLIB_TMR_ClockSourceExternalSyncDisable	Disables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceExternalSyncEnable	Enables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceSelect	Selects the clock source.

d) Gate Control Functions

	Name	Description
	PLIB_TMR_GateDisable	Enables counting regardless of the corresponding timer gate function.
	PLIB_TMR_GateEnable	Enables counting controlled by the corresponding gate function.

e) Preprocessing Functions

	Name	Description
	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
	PLIB_TMR_PrescaleGet	Gets the prescaler divisor value.
	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.

f) Period Control Functions

	Name	Description
	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.

g) Counter Control Functions

	Name	Description
	PLIB_TMR_Counter16BitClear	Clears the 16-bit timer value.
	PLIB_TMR_Counter16BitGet	Gets the 16-bit timer value.
	PLIB_TMR_Counter16BitSet	Sets the 16-bit timer value.
	PLIB_TMR_Counter32BitClear	Clears the 32-bit timer value.
	PLIB_TMR_Counter32BitGet	Gets the 32-bit timer value.
	PLIB_TMR_Counter32BitSet	Sets the 32-bit timer value.
	PLIB_TMR_CounterAsyncWriteDisable	Disables the writes to the counter register until the pending write operation completes.
	PLIB_TMR_CounterAsyncWriteEnable	Enables back-to-back writes with legacy asynchronous timer functionality.
	PLIB_TMR_CounterAsyncWriteInProgress	Returns the status of the counter write in Asynchronous mode.

i) Miscellaneous Functions

	Name	Description
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.

j) Feature Existence Functions

	Name	Description
	PLIB_TMR_ExistsClockSource	Identifies whether the ClockSource feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceSync	Identifies whether the ClockSourceSync feature exists on the Timer module.
	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.

	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.
	PLIB_TMR_ExistsMode16Bit	Identifies whether the Mode16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode32Bit	Identifies whether the Mode32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod16Bit	Identifies whether the Period16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod32Bit	Identifies whether the Period32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPrescale	Identifies whether the Prescale feature exists on the Timer module.
	PLIB_TMR_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOperationMode	Identifies whether the TimerOperationMode feature exists on the Timer module.

k) Data Types and Constants

	Name	Description
	TMR_CLOCK_SOURCE	Data type defining the different clock sources.
	TMR_MODULE_ID	Identifies the supported Timer modules.
	TMR_PRESCALE	Defines the list of possible prescaler values.

Description

This section describes the Application Programming Interface (API) functions of the Timer Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_TMR_Mode16BitEnable Function

Enables the Timer module for 16-bit operation and disables all other modes.

File

plib_tmr.h

C

```
void PLIB_TMR_Mode16BitEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the Timer module for 16-bit operation and disables all other modes.

Remarks

Calling this function disables the operation of the Timer module 8-bit mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsMode16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode16BitEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Mode16BitEnable(TMR_MODULE_ID index)

PLIB_TMR_Mode32BitEnable Function

Enables 32-bit operation on the Timer module combination.

File

plib_tmr.h

C

```
void PLIB_TMR_Mode32BitEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the Timer module and the index +1 Timer module to act as one 32-bit timer module and disables all other modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsMode32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode32BitEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Mode32BitEnable(TMR_MODULE_ID index)

PLIB_TMR_Start Function

Starts/enables the indexed timer.

File

plib_tmr.h

C

```
void PLIB_TMR_Start(TMR_MODULE_ID index);
```

Returns

None.

Description

This function starts/enables the indexed timer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
```

```
PLIB_TMR_Start(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Start(TMR_MODULE_ID index)

PLIB_TMR_Stop Function

Stops/disables the indexed timer.

File

plib_tmr.h

C

```
void PLIB_TMR_Stop(TMR_MODULE_ID index);
```

Returns

None.

Description

This function stops/disables the indexed timer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Stop(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Stop(TMR_MODULE_ID index)

b) Power Control Functions

PLIB_TMR_StopInIdleDisable Function

Continue module operation when the device enters Idle mode.

File

plib_tmr.h

C

```
void PLIB_TMR_StopInIdleDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function continues module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_StopInIdleDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_StopInIdleDisable(TMR_MODULE_ID index)

PLIB_TMR_StopInIdleEnable Function

Discontinues module operation when the device enters Idle mode.

File

plib_tmr.h

C

```
void PLIB_TMR_StopInIdleEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function discontinues module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_StopInIdleEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_StopInIdleEnable(TMR_MODULE_ID index)

c) Clock Source Control Functions

PLIB_TMR_ClockSourceExternalSyncDisable Function

Disables the clock synchronization of the external input.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceExternalSyncDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function disables the clock synchronization of the external input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSourceSync in your application to determine whether this feature is available.

Preconditions

The timer module must be configured to use the external clock using the function PLIB_TMR_ClockSourceSelect with the clock source as TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN. If the function PLIB_TMR_ClockSourceSelect configures the clock with some other enumeration, this function will have no effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_ClockSourceExternalSyncDisable(TMR_MODULE_ID index)

PLIB_TMR_ClockSourceExternalSyncEnable Function

Enables the clock synchronization of the external input.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceExternalSyncEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the clock synchronization of the external input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSourceSync in your application to determine whether this feature is available.

Preconditions

The timer module must be configured to use the external clock using the function PLIB_TMR_ClockSourceSelect with the clock source as TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN. If the function PLIB_TMR_ClockSourceSelect configures the clock with some other enumeration, this function will have no effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_ClockSourceExternalSyncEnable(TMR_MODULE_ID index)

PLIB_TMR_ClockSourceSelect Function

Selects the clock source.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceSelect(TMR_MODULE_ID index, TMR_CLOCK_SOURCE source);
```

Returns

None.

Description

This function selects the clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceSelect(MY_TMR_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values of TMR_CLOCK_SOURCE

Function

void PLIB_TMR_ClockSourceSelect(TMR_MODULE_ID index, TMR_CLOCK_SOURCE source)

d) Gate Control Functions

PLIB_TMR_GateDisable Function

Enables counting regardless of the corresponding timer gate function.

File

plib_tmr.h

C

```
void PLIB_TMR_GateDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables counting regardless of the gate function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsGatedTimeAccumulation in your application to determine whether this feature is available.

Preconditions

The timer must be enabled for this function to take effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_GateDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_GateDisable(TMR_MODULE_ID index)

PLIB_TMR_GateEnable Function

Enables counting controlled by the corresponding gate function.

File

plib_tmr.h

C

```
void PLIB_TMR_GateEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables counting controlled by the gate function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsGatedTimeAccumulation in your application to determine whether this feature is available.

Preconditions

The timer must be enabled, for this function to take effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_GateEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_GateEnable(TMR_MODULE_ID index)

e) Preprocessing Functions

PLIB_TMR_PrescaleDivisorGet Function

Gets the prescaler divisor information.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_PrescaleDivisorGet(TMR_MODULE_ID index, TMR_PRESCALE prescale);
```

Returns

prescale divisor value

Description

This function returns the prescaler divisor information.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t div = PLIB_TMR_PrescaleDivisorGet (MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Function

uint16_t PLIB_TMR_PrescaleDivisorGet(TMR_MODULE_ID index, TMR_PRESCALE prescale)

PLIB_TMR_PrescaleGet Function

Gets the prescaler divisor value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_PrescaleGet(TMR_MODULE_ID index);
```

Returns

Prescaler divisor value.

Description

This function returns the prescaler divisor value. The value returned will be direct number and not the enum equivalent.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
```

```
// application developer.
uint16_t prescale;
prescale = PLIB_TMR_PrescaleGet (MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_PrescaleGet (TMR_MODULE_ID index)

PLIB_TMR_PrescaleSelect Function

Selects the clock prescaler.

File

plib_tmr.h

C

```
void PLIB_TMR_PrescaleSelect(TMR_MODULE_ID index, TMR_PRESCALE prescale);
```

Returns

None.

Description

This function selects the clock prescaler.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_PrescaleSelect(MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_8);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Function

void PLIB_TMR_PrescaleSelect(TMR_MODULE_ID index, TMR_PRESCALE prescale)

f) Period Control Functions

PLIB_TMR_Period16BitGet Function

Gets the 16-bit period value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_Period16BitGet(TMR_MODULE_ID index);
```

Returns

period - 16-bit period value

Description

This function gets the 16-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t period = PLIB_TMR_Period16BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_Period16BitGet(TMR_MODULE_ID index)

PLIB_TMR_Period16BitSet Function

Sets the 16-bit period value.

File

plib_tmr.h

C

```
void PLIB_TMR_Period16BitSet(TMR_MODULE_ID index, uint16_t period);
```

Returns

None.

Description

This function sets the 16-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 16-bit value which needs to be stored in the
// period register.
PLIB_TMR_Period16BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	16-bit period register value

Function

void PLIB_TMR_Period16BitSet(TMR_MODULE_ID index, uint16_t period)

PLIB_TMR_Period32BitGet Function

Gets the 32-bit period value.

File

plib_tmr.h

C

```
uint32_t PLIB_TMR_Period32BitGet(TMR_MODULE_ID index);
```

Returns

period - 32-bit period value

Description

This function gets the 32-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint32_t period = PLIB_TMR_Period32BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_TMR_Period32BitGet(TMR_MODULE_ID index)

PLIB_TMR_Period32BitSet Function

Sets the 32-bit period value.

File

plib_tmr.h

C

```
void PLIB_TMR_Period32BitSet(TMR_MODULE_ID index, uint32_t period);
```

Returns

None.

Description

This function sets the 32-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 32-bit value which needs to be stored in the
// period register.
PLIB_TMR_Period32BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	32-bit period register value

Function

void PLIB_TMR_Period32BitSet(TMR_MODULE_ID index, uint32_t period)

g) Counter Control Functions

PLIB_TMR_Counter16BitClear Function

Clears the 16-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter16BitClear(TMR_MODULE_ID index);
```

Returns

None.

Description

This function clears the 16-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter16BitClear(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_TMR_Counter16BitClear(TMR_MODULE_ID index)

PLIB_TMR_Counter16BitGet Function

Gets the 16-bit timer value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_Counter16BitGet(TMR_MODULE_ID index);
```

Returns

16-bit timer value.

Description

This function gets the 16-bit timer value.

Remarks

PLIB_TMR_Counter16BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t timerValue = PLIB_TMR_Counter16BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_Counter16BitGet(TMR_MODULE_ID index)

PLIB_TMR_Counter16BitSet Function

Sets the 16-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter16BitSet(TMR_MODULE_ID index, uint16_t value);
```

Returns

None.

Description

This function sets the 16-bit timer value.

Remarks

When the timer register is written to, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter16BitSet(MY_TMR_INSTANCE, 0x100);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	16-bit value to be set

Function

void PLIB_TMR_Counter16BitSet(TMR_MODULE_ID index, uint16_t value)

PLIB_TMR_Counter32BitClear Function

Clears the 32-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter32BitClear(TMR_MODULE_ID index);
```

Returns

None.

Description

This function clears the 32-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter32BitClear(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_TMR_Counter32BitClear(TMR_MODULE_ID index)

PLIB_TMR_Counter32BitGet Function

Gets the 32-bit timer value.

File

plib_tmr.h

C

```
uint32_t PLIB_TMR_Counter32BitGet(TMR_MODULE_ID index);
```

Returns

32 bit timer value.

Description

This function gets the 32-bit timer value.

Remarks

PLIB_TMR_Counter32BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint32_t timerValue = PLIB_TMR_Counter32BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_TMR_Counter32BitGet(TMR_MODULE_ID index)

PLIB_TMR_Counter32BitSet Function

Sets the 32-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter32BitSet(TMR_MODULE_ID index, uint32_t value);
```

Returns

None.

Description

This function sets the 32-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter32BitSet(MY_TMR_INSTANCE, 0x1000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	32-bit value to be set

Function

void PLIB_TMR_Counter32BitSet(TMR_MODULE_ID index, uint32_t value)

PLIB_TMR_CounterAsyncWriteDisable Function

Disables the writes to the counter register until the pending write operation completes.

File

plib_tmr.h

C

```
void PLIB_TMR_CounterAsyncWriteDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function disables the writes to the counter register until the pending write operation completes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounterAsyncWriteControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_CounterAsyncWriteDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_CounterAsyncWriteDisable(TMR_MODULE_ID index)

PLIB_TMR_CounterAsyncWriteEnable Function

Enables back-to-back writes with legacy asynchronous timer functionality.

File

plib_tmr.h

C

```
void PLIB_TMR_CounterAsyncWriteEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables back-to-back writes with legacy asynchronous timer functionality.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_TMR_ExistsCounterAsyncWriteControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_CounterAsyncWriteEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_CounterAsyncWriteEnable(TMR_MODULE_ID index)

PLIB_TMR_CounterAsyncWriteInProgress Function

Returns the status of the counter write in Asynchronous mode.

File

plib_tmr.h

C

```
bool PLIB_TMR_CounterAsyncWriteInProgress(TMR_MODULE_ID index);
```

Returns

- true - Write is in progress
- false - Write is complete

Description

This function returns the status of the counter write in Asynchronous mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounterAsyncWriteInProgress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool inProgress = PLIB_TMR_CounterAsyncWriteInProgress(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_TMR_CounterAsyncWriteInProgress(TMR_MODULE_ID index)

h) Post-processing Functions

i) Miscellaneous Functions

PLIB_TMR_IsPeriodMatchBased Function

Gets the operating mode state of the Timer module based on Period Match or Overflow mode.

File

plib_tmr.h

C

```
bool PLIB_TMR_IsPeriodMatchBased(TMR_MODULE_ID index);
```

Returns

- true - Operation in Period Match mode
- false - Operation in Overflow mode

Description

This function gets the operating mode state of the Timer module based on Period Match or Overflow mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsTimerOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool status = PLIB_TMR_IsPeriodMatchBased(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_TMR_IsPeriodMatchBased (TMR_MODULE_ID index)

j) Feature Existence Functions

PLIB_TMR_ExistsClockSource Function

Identifies whether the ClockSource feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsClockSource(TMR_MODULE_ID index);
```

Returns

- true - The ClockSource feature is supported on the device
- false - The ClockSource feature is not supported on the device

Description

This function identifies whether the ClockSource feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_ClockSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsClockSource(TMR_MODULE_ID index)

PLIB_TMR_ExistsClockSourceSync Function

Identifies whether the ClockSourceSync feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsClockSourceSync(TMR_MODULE_ID index);
```

Returns

- true - The ClockSourceSync feature is supported on the device
- false - The ClockSourceSync feature is not supported on the device

Description

This function identifies whether the ClockSourceSync feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_ClockSourceExternalSyncEnable
- PLIB_TMR_ClockSourceExternalSyncDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsClockSourceSync(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounter16Bit Function

Identifies whether the Counter16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounter16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Counter16Bit feature is supported on the device
- false - The Counter16Bit feature is not supported on the device

Description

This function identifies whether the Counter16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Counter16BitSet
- PLIB_TMR_Counter16BitGet
- PLIB_TMR_Counter16BitClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounter16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounter32Bit Function

Identifies whether the Counter32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounter32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Counter32Bit feature is supported on the device
- false - The Counter32Bit feature is not supported on the device

Description

This function identifies whether the Counter32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Counter32BitSet
- PLIB_TMR_Counter32BitGet
- PLIB_TMR_Counter32BitClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounter32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounterAsyncWriteControl Function

Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounterAsyncWriteControl(TMR_MODULE_ID index);
```

Returns

- true - The CounterAsyncWriteControl feature is supported on the device
- false - The CounterAsyncWriteControl feature is not supported on the device

Description

This function identifies whether the CounterAsyncWriteControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_CounterAsyncWriteEnable
- PLIB_TMR_CounterAsyncWriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounterAsyncWriteControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounterAsyncWriteInProgress Function

Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounterAsyncWriteInProgress(TMR_MODULE_ID index);
```

Returns

- true - The CounterAsyncWriteInProgress feature is supported on the device
- false - The CounterAsyncWriteInProgress feature is not supported on the device

Description

This function identifies whether the CounterAsyncWriteInProgress feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_CounterAsyncWriteInProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounterAsyncWriteInProgress(TMR_MODULE_ID index)

PLIB_TMR_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsEnableControl(TMR_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Start
- PLIB_TMR_Stop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsEnableControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsGatedTimeAccumulation Function

Identifies whether the GatedTimeAccumulation feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsGatedTimeAccumulation(TMR_MODULE_ID index);
```

Returns

- true - The GatedTimeAccumulation feature is supported on the device
- false - The GatedTimeAccumulation feature is not supported on the device

Description

This function identifies whether the GatedTimeAccumulation feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_GateEnable
- PLIB_TMR_GateDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsGatedTimeAccumulation(TMR_MODULE_ID index)

PLIB_TMR_ExistsMode16Bit Function

Identifies whether the Mode16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsMode16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Mode16Bit feature is supported on the device
- false - The Mode16Bit feature is not supported on the device

Description

This function identifies whether the Mode16Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_Mode16BitEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsMode16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsMode32Bit Function

Identifies whether the Mode32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsMode32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Mode32Bit feature is supported on the device
- false - The Mode32Bit feature is not supported on the device

Description

This function identifies whether the Mode32Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_Mode32BitEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsMode32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPeriod16Bit Function

Identifies whether the Period16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPeriod16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Period16Bit feature is supported on the device
- false - The Period16Bit feature is not supported on the device

Description

This function identifies whether the Period16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Period16BitSet
- PLIB_TMR_Period16BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPeriod16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPeriod32Bit Function

Identifies whether the Period32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPeriod32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Period32Bit feature is supported on the device
- false - The Period32Bit feature is not supported on the device

Description

This function identifies whether the Period32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Period32BitSet
- PLIB_TMR_Period32BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPeriod32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPrescale Function

Identifies whether the Prescale feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPrescale(TMR_MODULE_ID index);
```

Returns

- true - The Prescale feature is supported on the device
- false - The Prescale feature is not supported on the device

Description

This function identifies whether the Prescale feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_PrescaleSelect
- PLIB_TMR_PrescaleGet
- PLIB_TMR_PrescaleDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPrescale(TMR_MODULE_ID index)

PLIB_TMR_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsStopInIdleControl(TMR_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_StopInIdleEnable
- PLIB_TMR_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsStopInIdleControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsTimerOperationMode Function

Identifies whether the TimerOperationMode feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsTimerOperationMode(TMR_MODULE_ID index);
```

Returns

- true - The TimerOperationMode feature is supported on the device
- false - The TimerOperationMode feature is not supported on the device

Description

This function identifies whether the TimerOperationMode feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_IsPeriodMatchBased

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsTimerOperationMode(TMR_MODULE_ID index)

k) Data Types and Constants

TMR_CLOCK_SOURCE Enumeration

Data type defining the different clock sources.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK,
  TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN
} TMR_CLOCK_SOURCE;
```

Members

Members	Description
TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK	Clock source is the internal clock = FOSC/n (where n, is processor specific)
TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN	Clock source is the external input pin

Description

Clock sources selection enumeration

This data type defines the different clock sources.

TMR_MODULE_ID Enumeration

Identifies the supported Timer modules.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_ID_0,
  TMR_ID_1,
  TMR_ID_2,
  TMR_ID_3,
  TMR_ID_4,
  TMR_ID_5,
  TMR_ID_6,
  TMR_ID_7,
  TMR_ID_8,
  TMR_ID_9,
  TMR_NUMBER_OF_MODULES
} TMR_MODULE_ID;
```

Members

Members	Description
TMR_ID_0	Timer 0
TMR_ID_1	Timer 1
TMR_ID_2	Timer 2
TMR_ID_3	Timer 3
TMR_ID_4	Timer 4
TMR_ID_5	Timer 5
TMR_ID_6	Timer 6
TMR_ID_7	Timer 7
TMR_ID_8	Timer 8
TMR_ID_9	Timer 9
TMR_NUMBER_OF_MODULES	Max number of timers

Description

TMR Module ID

This enumeration identifies the available Timer modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

TMR_PRESCALE Enumeration

Defines the list of possible prescaler values.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_PRESCALE_VALUE_1,
  TMR_PRESCALE_VALUE_2,
  TMR_PRESCALE_VALUE_4,
  TMR_PRESCALE_VALUE_8,
  TMR_PRESCALE_VALUE_16,
  TMR_PRESCALE_VALUE_32,
  TMR_PRESCALE_VALUE_64,
  TMR_PRESCALE_VALUE_256
} TMR_PRESCALE;
```

Members

Members	Description
TMR_PRESCALE_VALUE_1	Timer Prescaler Value 1:1
TMR_PRESCALE_VALUE_2	Timer Prescaler Value 1:2
TMR_PRESCALE_VALUE_4	Timer Prescaler Value 1:4
TMR_PRESCALE_VALUE_8	Timer Prescaler Value 1:8
TMR_PRESCALE_VALUE_16	Timer Prescaler Value 1:16
TMR_PRESCALE_VALUE_32	Timer Prescaler Value 1:32
TMR_PRESCALE_VALUE_64	Timer Prescaler Value 1:64
TMR_PRESCALE_VALUE_256	Timer Prescaler Value 1:256

Description

Prescaler Values

This macro defines the list of possible prescaler values.

Files

Files

Name	Description
plib_tmr.h	Timer Peripheral Library interface header.
help_plib_tmr.h	This is file help_plib_tmr.h.

Description

This section lists the source and header files used by the library.

plib_tmr.h

Timer Peripheral Library interface header.

Functions

	Name	Description
	PLIB_TMR_ClockSourceExternalSyncDisable	Disables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceExternalSyncEnable	Enables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceSelect	Selects the clock source.
	PLIB_TMR_Counter16BitClear	Clears the 16-bit timer value.
	PLIB_TMR_Counter16BitGet	Gets the 16-bit timer value.
	PLIB_TMR_Counter16BitSet	Sets the 16-bit timer value.
	PLIB_TMR_Counter32BitClear	Clears the 32-bit timer value.
	PLIB_TMR_Counter32BitGet	Gets the 32-bit timer value.
	PLIB_TMR_Counter32BitSet	Sets the 32-bit timer value.
	PLIB_TMR_CounterAsyncWriteDisable	Disables the writes to the counter register until the pending write operation completes.
	PLIB_TMR_CounterAsyncWriteEnable	Enables back-to-back writes with legacy asynchronous timer functionality.
	PLIB_TMR_CounterAsyncWriteInProgress	Returns the status of the counter write in Asynchronous mode.
	PLIB_TMR_ExistsClockSource	Identifies whether the ClockSource feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceSync	Identifies whether the ClockSourceSync feature exists on the Timer module.

	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.
	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.
	PLIB_TMR_ExistsMode16Bit	Identifies whether the Mode16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode32Bit	Identifies whether the Mode32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod16Bit	Identifies whether the Period16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod32Bit	Identifies whether the Period32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPrescale	Identifies whether the Prescale feature exists on the Timer module.
	PLIB_TMR_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOperationMode	Identifies whether the TimerOperationMode feature exists on the Timer module.
	PLIB_TMR_GateDisable	Enables counting regardless of the corresponding timer gate function.
	PLIB_TMR_GateEnable	Enables counting controlled by the corresponding gate function.
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.
	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
	PLIB_TMR_PrescaleGet	Gets the prescaler divisor value.
	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.
	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

Description

Timer Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer Peripheral Library.

File Name

plib_tmr.h

Company

Microchip Technology Inc.

help_plib_tmr.h

Enumerations

	Name	Description
	TMR_CLOCK_SOURCE	Data type defining the different clock sources.
	TMR_MODULE_ID	Identifies the supported Timer modules.
	TMR_PRESCALE	Defines the list of possible prescaler values.

Description

This is file help_plib_tmr.h.

USART Peripheral Library

This section describes the USART Peripheral Library.

Introduction

This library provides a low-level abstraction of the USART Peripheral Library on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module is the key component of the serial communications sub-system of a computer. The USART takes the data and transmits the individual bits in a sequential fashion. While receiving, the USART reassembles the bits into complete bytes. The data can be **8-bit** or **9-bit**.

There are two primary forms of serial transmission: Asynchronous and Synchronous.

Asynchronous

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters (i.e., baud rate) in advance and special bits are added to each word, which are used to synchronize the sending and receiving units.

When a word is given to the USART for asynchronous transmissions, a bit called the "Start" bit is added to the beginning of each word that is to be transmitted. The Start bit is used to alert the receiver that a word of data is about to be sent. After the Start bit, the individual bits of the word of data are sent, with the Least Significant bit (LSb) being sent first. When the entire data word has been sent, the transmitter may add a Parity bit that the transmitter generates. The Parity bit may be used by the receiver to perform simple error checking. Then, at least one Stop bit is sent by the transmitter.

The Start bit is always a space and the Stop bits are always marks. Each transmitted bit persists for a period of 1/(Baud Rate). An on-chip dedicated Baud Rate Generator (BRG) is used to derive standard baud rate frequencies.

When the receiver has received all of the bits in the data word, it may check for the Parity bits (both sender and receiver must agree on whether a Parity bit is to be used), and then the receiver looks for a Stop bit. If the Stop bit does not appear when it is supposed to, the USART module considers the entire word to be unintelligible and will report a framing error to the host processor when the data word is read.

Note: Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Synchronous

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to "read" the next bit of the data. This mode is not available on all microcontrollers.

It is typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system.

There are two signal lines a bidirectional data line and a clock line. Since the data line is bidirectional, synchronous operation is only half-duplex.

Start and Stop bits are not used in synchronous transmissions.

Using the Library

This topic describes the basic architecture of the USART Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usart.h`

The interface to the USART Peripheral library is defined in the `plib_usart.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USART Peripheral library must include `peripheral.h`.

Library File:

The USART Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

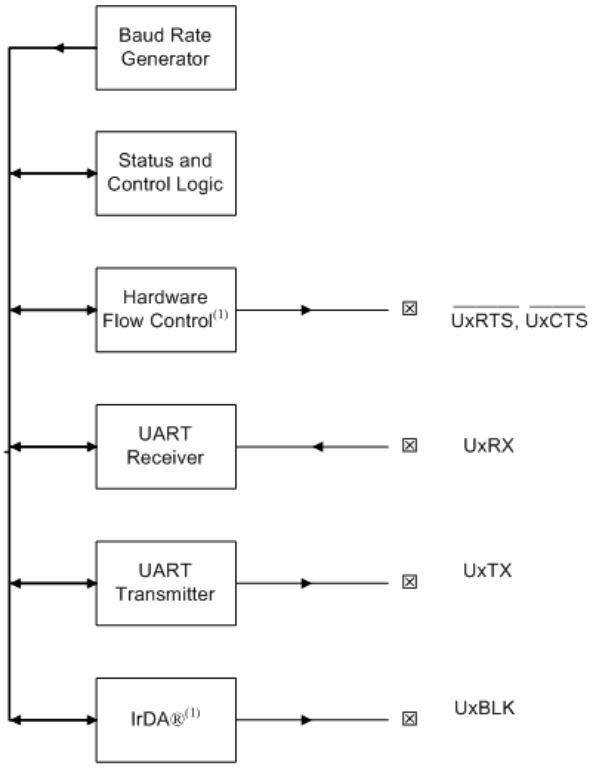
Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This section describes the hardware abstraction model.

Description

Hardware Abstraction Model



Note 1: This feature is not available on all devices. Refer to the **"USART"** chapter in the specific device data sheet to determine availability.

Hardware Abstraction Model

The USART Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

The **Baud Rate Controller** controls the timing in Asynchronous mode, the desired baud rate can be programmed in the baud rate controller or, the baud rate controller can be configured to determine the baud rate of the transmitting device automatically.

The **Transmitter** and **Receiver** are capable of handling 8-bit or 9-bit data, which can be programmed in the control logic. The 9th bit is used to transfer the address or the parity. Enable 9-bit mode if the hardware is to be used for address or parity detection.

The **Status and Control logic**, provide the ability to control different ways the transmitter, receiver, and the baud rate controller can function. It also can provides status about the transmitter, receiver, or the baud rate controller.

The **Hardware Flow Control** uses CTS and the RTS lines to perform handshaking. Flow control is the ability to stop, and then restart the flow without any loss of bytes. Flow control is needed for modems and other hardware to allow a jump in instantaneous flow rates.

The USART module provides two types of infrared USART support: one is the IrDA clock output to support the external IrDA encoder and decoder, and the other is the full implementation of the IrDA encoder and decoder.

Note: These features are not available on all devices. Please refer to the specific device data sheet to determine availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART module.

Library Interface Section	Description
General Functions	Provides setup and configuration interface routines for • IrDA • Hardware Flow Control • Operation in power saving modes. • and other general setup

Baud Rate Generator Functions	Provides setup and configuration interface routines together with status routines for Baud Rate Generator (BRG).
Transmitter Functions	Provides setup, data transfer, error and the status interface routines for the transmitter.
Receiver Functions	Provides setup, data transfer, error and the status interface routines for the receiver.
Feature Existence Functions	Provides interface routines to determine existence of features.

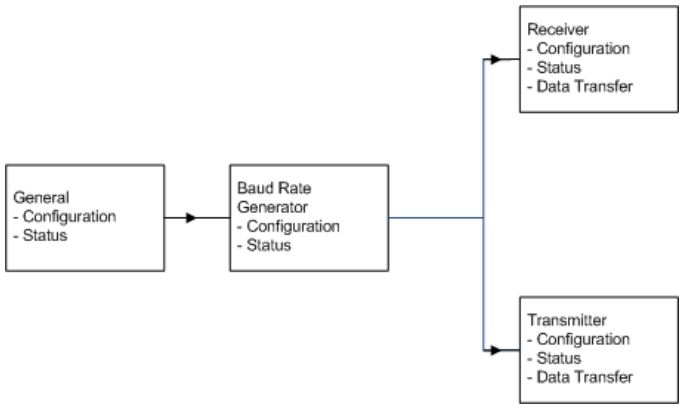
How the Library Works

Provides information on how the library works.

Description

This section provides information on using the peripheral library with the USART module. The usage model for different scenarios is also described.

Usage Model



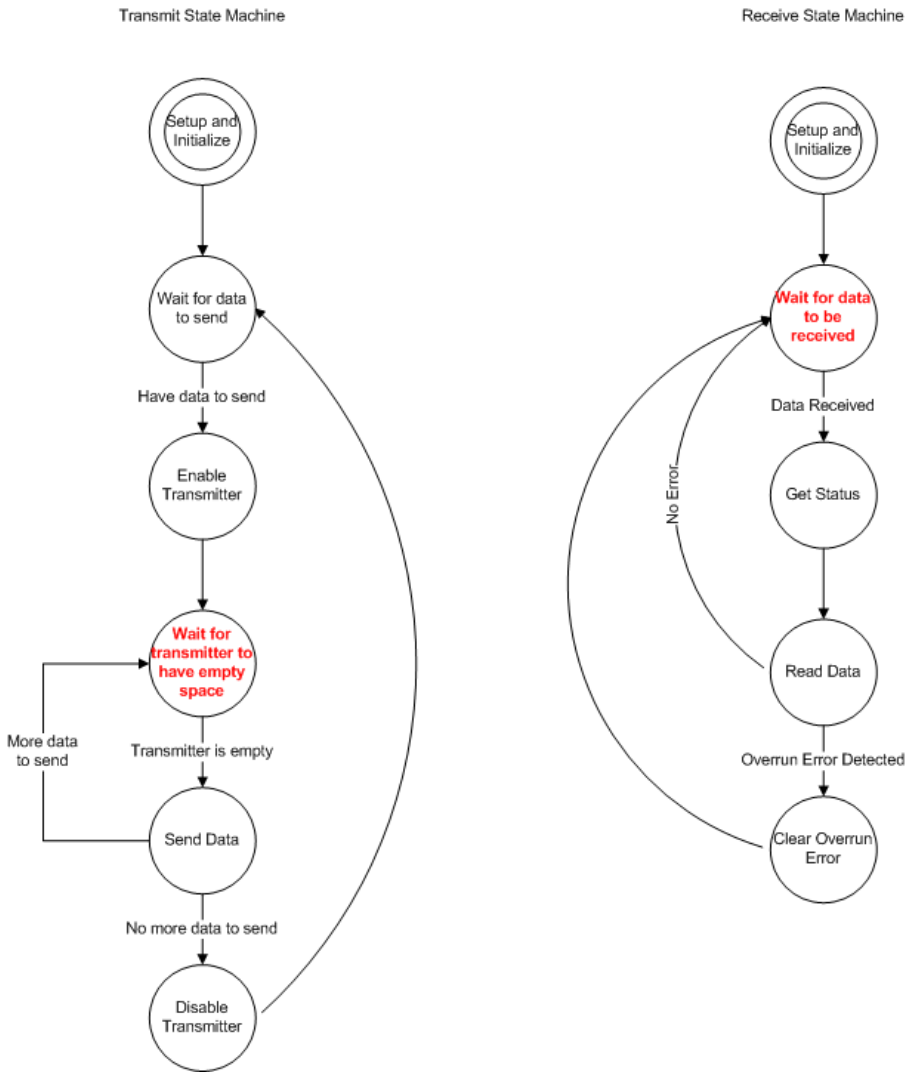
State Machine

This section describes the transmission and reception state machine.

Description

The following state machine is for the transmission and reception during normal operation. This state machine provides a general idea of the usage model of the USART Peripheral Library. Refer to the usage model for more detailed steps for the scenario that is being used.

Transmit and Receive State machine



Notes:

1. The wait states shown in red in the diagram can either be achieved using polled functions or using interrupts.
2. Refer to the specific USART mode in use for the setup and initialization steps.
3. Refer to the following table for the functions that can be used at each state.

State	Associated Function
Setup and initialization	Refer to the specific USART mode for detailed setup instructions.
Wait for data to send	Once the USART module has been appropriately set up and initialized, the state machine waits for the application to issue one of the send data functions.
Wait for data to be received	When the application is using Polled mode, the application can use the function PLIB_USART_ReceiverDataIsAvailable. When the application is using Interrupt mode, the application should wait for a receive interrupt.
Enable transmitter	Use the function PLIB_USART_TransmitterEnable to enable the transmitter
Wait for transmitter to have empty space	When the application is using polled mode, the application can use the function PLIB_USART_TransmitterBufferIsFull. When the application is using Interrupt mode, the application should wait for a transmit interrupt.
Send data	Use the function PLIB_USART_TransmitterByteSend to send data. If an address needs to be transmitted or if a byte needs to be transmitted with a Parity bit use the function PLIB_USART_Transmitter9BitsSend.
Disable transmitter	Use the function PLIB_USART_TransmitterDisable to disable the transmitter.
Get status	To get the status of the hardware module, use the set of functions, PLIB_USART_ReceiverFramingErrorHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverOverrunHasOccurred.

Read data	Use the function PLIB_USART_ReceiverByteReceive to read data.
Clear overrun error	Clear the overrun error using the function PLIB_USART_ReceiverOverrunErrorClear.

Asynchronous USART Mode

This section provides processes for setting up asynchronous transmission and reception.

Description

Asynchronous Transmission

Setup

1. Set the desired baud rate using either PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device,* enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the transmit interrupt.
   - Ensure that the interrupts are enabled for the system and the peripherals
   - Select the interrupt priority
   - Set up the transmit FIFO interrupt mode using PLIB_USART_TransmitterInterruptModeSelect. Refer to USART_TRANSMIT_INTR_MODE for a list of possible values.
6. Enable the USART module using the function PLIB_USART_Enable

Example: Asynchronous Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, baudRate);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.
```

Enable Transmission

1. Enable the transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full, before attempting to write into the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.
3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8-bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting an address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Asynchronous Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait for
//an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Asynchronous Transmission of Break Characters

A break characters consists of a **Start Bit** followed by twelve bits of "0" and a **Stop Bit.**

Setup and Transmission

1. Configure the desired mode, using the previous steps described.
2. Enable the transmitter using PLIB_USART_TransmitterEnable.
3. Transmit the break character using PLIB_USART_TransmitterBreakSend function.
4. Check if the break transmission is complete using PLIB_USART_TransmitterBreakSendIsComplete.
5. Transmission of the next bytes can start.

Example: USART Break and Sync Transmission

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);

// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// Start the transmission
// Transmit the break character
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// Transmit the byte
while(PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));

PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, 0x55);
```

Refer to the following steps for setting up asynchronous receptions and how to receive data.

Asynchronous Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the receive interrupt.
   - Ensure that the interrupts are enabled for the system and the peripherals
   - Select the interrupt priority
   - Set up the receive FIFO interrupt mode using PLIB_USART_ReceiverInterruptModeSelect. Refer to USART_RECEIVE_INTR_MODE for a list of possible values.
6. *Optional:* If inverted receive polarity is desired, use PLIB_USART_ReceiverIdleStateLowEnable.
7. *If available on the device*, enable the reception using PLIB_USART_ReceiverEnable.
8. Enable the USART module using PLIB_USART_Enable.

Example: Asynchronous Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being
```

```
// used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.
2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.

Example: Asynchronous Reception

```
// TODO - Either wait for Receive buffer to have data or wait for
// a receive interrupt.
// read the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Asynchronous Reception with Address Detection Mode

A typical multi-processor communication protocol will differentiate between data bytes and address/control bytes. A common method is to use a 9th data bit to identify whether a data byte is address or data information. If the 9th bit is set, the data is processed as address or control information. If the 9th bit is cleared, the received data word is processed as data associated with the previous address/control byte.

The protocol operates as follows:

- The master device transmits a data word with 9th bit set. The data word contains the address of a slave device.
- All slave devices in the communication chain receive the address word and check the slave address.
- The slave device that was addressed will receive and process subsequent data bytes sent by the master device. All other slave devices will discard subsequent data bytes until a new address word(9th bit set) is received.

This mode will typically be used in RS-485 systems.

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the receive interrupt.
   - Ensure that the interrupts are enabled for the system and the peripherals. Also select the interrupt priority.
   - Set up the receive FIFO interrupt mode by using the function PLIB_USART_ReceiverInterruptModeSelect. Refer to USART_RECEIVE_INTR_MODE for a list of possible values.
   - For Address Detect mode, configure PLIB_USART_ReceiverInterruptModeSelect with the receive FIFO interrupt mode as USART_RECEIVE_FIFO_ONE_CHAR.
6. *Optional:* If inverted receive polarity is desired, use PLIB_USART_ReceiverIdleStateLowEnable.
7. *If available on the device*, enable the reception using PLIB_USART_ReceiverEnable.
8. Enable address detection using PLIB_USART_ReceiverAddressDetectEnable.
9. Enable the USART module using PLIB_USART_Enable.

Note: This feature is not available on all devices. On devices where automatic address detection is available, configure automatic address detection using PLIB_USART_ReceiverAddressAutoDetectEnable. Skip steps 2-6 in the following Reception section during the reception phase.

Example: Asynchronous Reception Setup (with address detect enable)

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
// being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Set the reception to 9 bits to enable address byte.
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_9N1);

// Enable address detection.
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed or poll using PLIB_USART_ReceiverDataIsAvailable.
2. Read the address using PLIB_USART_ReceiverByteReceive.
3. The application determines if this is the device address. If the device is addressed, disable address detect using PLIB_USART_ReceiverAddressDetectDisable to allow subsequent data bytes to be received; otherwise, discard the received word.
4. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
5. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.
6. If the device has been addressed, disable the address detection using PLIB_USART_ReceiverAddressDetectDisable, which will allow all received data into the receive buffer and generate interrupts if desired.
7. *Optional (if available on the device):* If a long packet is expected, the receive interrupt mode could be changed to buffer more than one data word between interrupts. Use PLIB_USART_ReceiverInterruptModeSelect with USART_RECEIVE_FIFO_HALF_FULL or USART_RECEIVE_FIFO_3B4FULL.
8. When the last data byte has been received, enable the address detect using PLIB_USART_ReceiverAddressDetectEnable, so that only address bytes will be received.
9. *Optional (if available on the device):* Ensure that the receive interrupt is configured to be triggered after each received word using PLIB_USART_ReceiverInterruptModeSelect with USART_RECEIVE_FIFO_ONE_CHAR.

Example: Asynchronous Reception (with Address Detect)

```
// Either wait for Receive buffer to have data or wait for a receive interrupt.
int8_t my_address;

// Receive the address
if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    my_address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
    // where, my_address is the address received
}

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}

if(my_address == MY_DEVICE_ADDRESS)
{
    // The device has been addressed.
    PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
}
```

```
// TODO - Receive the data bytes for the packet. Either wait for RX buffer
//to have data or wait for a receive interrupt.

// When all the bytes for the packet are received.

// Enable address detection mode
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);
```

Synchronous Master Mode

This section provides processes for setting up synchronous master transmission and reception.

Note: The features described in this section are not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

Synchronous Master Transmission

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Disable receive using PLIB_USART_ReceiverDisable.
5. *Optional:* If desired set the number of data bits to 9 (to send address byte or to enable parity check) using PLIB_USART_LineControlModeSelect.
6. *Optional:* If desired, enable the transmit interrupt, ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Master Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO Call the appropriate function.
```

Transmission

1. Enable transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full before attempting to write onto the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.

Note: If using interrupts, make sure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8 - bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Master Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO Either wait for Transmit buffer to be not full or wait for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Synchronous Master Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate IO direction and corresponding to RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using PLIB_USART_ReceiverDisable.

Example: Synchronous Master Reception Setup

```
uint32_t baudRate = 9600;
// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
                                    USART_HANDSHAKE_MODE_SIMPLEX);
// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Enable receive mode
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.

Note: If using interrupts, make sure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using the function PLIB_USART_ReceiverOverrunErrorClear.

Example: Synchronous Master Reception

```
// Read the byte
// where, my_byte is the 8 bit data to be transmitted
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Synchronous Slave Mode

This section provides processes for setting up synchronous slave transmission and reception.

Note: This feature is not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

Synchronous Slave Transmission

Setup

1. Set the desired baud rate using either PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Enable the synchronous slave serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.

4. Disable receive mode using PLIB_USART_ReceiverDisable.
5. *Optional:* If desired, enable the transmit interrupt,ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Slave Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware
//peripheral MY_CLOCK_FREQUENCY - is hardware clock frequency which is
//currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable slave clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Disable receive mode
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);

// TODO - If using interrupts Enable the transmit interrupt
// TODO - Call the appropriate function for the transmit enable interrupt,
// TODO - Also, make sure that the interrupts are enabled for system and the peripherals.
```

Transmission

1. Enable transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full before attempting to write into the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8-bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting address if transmitting a byte with parity)
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Slave Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait
//for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_word);
// where, my_word is the 9 bit data to be transmitted
```

Synchronous Slave Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using PLIB_USART_ReceiverEnable.

Example: Synchronous Slave Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
```

```
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.

Example: Synchronous Slave Reception

```
// Transmit the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
 PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Other Features

This section describes additional features of the USART Peripheral Library.

Note: The features described in this section are not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

IrDA Support

Support for IrDA is available in some microcontrollers.

1. Enable IrDA support using PLIB_USART_IrDAEnable. IrDA support can be disabled using PLIB_USART_IrDADisable.
2. *Optional:* If using the external IrDA encoder and decoder, use the function PLIB_USART_OperationModeSelect with USART_ENABLE_TX_RX_BCLK_USED to output the 16x baud clock.
3. *Optional:* IrDA Transmit Polarity can be inverted using PLIB_USART_TransmitterIdleIsLowEnable.
4. *Optional:* IrDA Receive Polarity can be inverted using PLIB_USART_ReceiverIdleStateLowEnable.
5. Enable the USART using PLIB_USART_Enable.

Loopback Support

This is the mode in which the transmission is internally connected to reception.

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using PLIB_USART_LineControlModeSelect. Refer to USART_LINECONTROL_MODE for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the transmit interrupt, also make sure that the interrupts are enabled for the system and the peripherals.
5. *Optional:* Also select the interrupt priority.
6. *Optional:* Also, set up the transmit FIFO interrupt mode using PLIB_USART_TransmitterInterruptModeSelect. Refer to USART_TRANSMIT_INTR_MODE for a list of possible values.

7. Enable the USART module using PLIB_USART_Enable and enable the transmission using PLIB_USART_TransmitterEnable.
8. Enable loopback using PLIB_USART_LoopbackEnable. Loopback can be disabled at any time using PLIB_USART_LoopbackDisable
9. Continue with transmission and/or reception using the same steps as for asynchronous transmission/reception.

Auto Baud Support

1. Enable the auto baud detection using the function PLIB_USART_BaudRateAutoDetectEnable, which requires a reception of the Sync character (0x55).
2. *Optional:* Poll PLIB_USART_BaudRateAutoDetectIsComplete to check if the baud rate is detected.
3. *Optional:* Call PLIB_USART_BaudRateGet to get the baud rate that was detected.
4. Clear the receive interrupt.
5. On some microcontrollers, reading the data using PLIB_USART_ReceiverByteReceive may be required.

Flow Control Support

The USART can use Simplex or Flow Control modes:

- Use PLIB_USART_HandshakeModeSelect to enable flow control (USART_HANDSHAKE_MODE_FLOW_CONTROL) or use simplex mode (USART_HANDSHAKE_MODE_SIMPLEX) selected from USART_HANDSHAKE_MODE.
- To configure the USART line control features, use PLIB_USART_LineControlModeSelect selected from USART_LINECONTROL_MODE.
- Additionally, the USART pins may need to be configured using PLIB_USART_OperationModeSelect and the appropriate values from USART_OPERATION_MODE.

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the USART modules are shut down. If the device enters Sleep mode in the middle of a USART transmission or reception operation, the operation is aborted and the USART pins are driven to default state.

A Start bit, when detected, can wake up the device.

- The application needs to enable the wake-up on start bit using PLIB_USART_WakeOnStartEnable just before entering Sleep mode, to wake-up the device
- For the device to wake-up from sleep, reception of the Sync character or Wake-up Signal character is required
- *Optional:* If using interrupts, a receive interrupt is generated

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

- USART operation during Idle mode can be stopped using PLIB_USART_StopInIdleEnable. When the device enters Idle mode, the USART module operation is similar to that as in Sleep mode(Refer to Operation in Sleep Mode).
- By default, the USART module continues to operate in Idle mode and provide full functionality

USART Operation With DMA

The DMA module can be used to transfer data between the USART and the CPU, without CPU assistance.

1. Set the desired baud rate using the function PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using the function PLIB_USART_LineControlModeSelect. Refer to USART_LINECONTROL_MODE for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Enable the USART module using PLIB_USART_Enable and enable the transmission using PLIB_USART_TransmitterEnable.
5. Enable the Transmit, Receive, and Error interrupts, ensuring that the interrupts are enabled for the system and the peripherals.
6. Select the interrupt priority.
7. Set up the Transmit FIFO Interrupt mode and Receive FIFO Interrupt mode using the PLIB_USART_TransmitterInterruptModeSelect and PLIB_USART_ReceiverInterruptModeSelect functions. To generate interrupts for every character received, use USART_RECEIVE_FIFO_ONE_CHAR from USART_RECEIVE_INTR_MODE.
8. Configure the DMA to be used with the USART receiver and transmitter.

Configuring the Library

The library is configured for the supported USART modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Functions

	Name	Description
	PLIB_USART_Disable	Disables the specific USART module
	PLIB_USART_Enable	Enables the specific USART module.

	PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
	PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
	PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
	PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
	PLIB_USART_LoopbackDisable	Disables Loopback mode.
	PLIB_USART_LoopbackEnable	Enables Loopback mode.
	PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
	PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.
	PLIB_USART_ErrorsGet	Return the status of all errors in the specified USART module.
	PLIB_USART_InitializeModeGeneral	Enables or disables general features of the USART module.
	PLIB_USART_InitializeOperation	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.
	PLIB_USART_AddressGet	Gets the address for the Address Detect mode.
	PLIB_USART_AddressMaskGet	Gets the address mask for the Address Detect mode.
	PLIB_USART_AddressMaskSet	Sets the address mask for the Address Detect mode.
	PLIB_USART_AddressSet	Sets the address for the Address Detect mode.
	PLIB_USART_ModuleIsBusy	Returns the USART module's running status.
	PLIB_USART_RunInOverflowDisable	Disables the Run in overflow condition mode.
	PLIB_USART_RunInOverflowEnable	Enables the USART module to continue to operate when an overflow error condition has occurred.
	PLIB_USART_RunInOverflowIsEnabled	Gets the status of the Run in Overflow condition.
	PLIB_USART_RunInSleepModeDisable	Turns off the USART module's BRG clock during Sleep mode.
	PLIB_USART_RunInSleepModeEnable	Allows the USART module's BRG clock to run when the device enters Sleep mode.
	PLIB_USART_RunInSleepModeIsEnabled	Gets the status of Run in Sleep mode.

b) Baud Rate Generator Functions

	Name	Description
	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
	PLIB_USART_BaudRateGet	Gets the baud rate current in use.
	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudSetAndEnable	Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.
	PLIB_USART_BRGClockSourceGet	Gets the BRG clock source of the USART module.
	PLIB_USART_BRGClockSourceSelect	Configures the BRG clock source of the USART module.

c) Transmitter Functions

	Name	Description
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.

	PLIB_USART_TransmitterInterruptModeSelect	Sets the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_TransmitterAddressGet	Returns the address of the USART TX register

d) Receiver Functions

	Name	Description
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART vverrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.
	PLIB_USART_ReceiverAddressGet	Returns the address of the USART RX register
	PLIB_USART_Receiver9BitsReceive	Data to be received in the byte mode with the 9th bit.

e) Feature Existence Functions

	Name	Description
	PLIB_USART_ExistsBaudRate	Identifies whether the BaudRate feature exists on the USART module.
	PLIB_USART_ExistsBaudRateAutoDetect	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsBaudRateHigh	Identifies whether the BaudRateHigh feature exists on the USART module.
	PLIB_USART_ExistsEnable	Identifies whether the EnableControl feature exists on the USART module.
	PLIB_USART_ExistsHandshakeMode	Identifies whether the HandShakeMode feature exists on the USART module.
	PLIB_USART_ExistsIrDA	Identifies whether the IrDAControl feature exists on the USART module.
	PLIB_USART_ExistsLineControlMode	Identifies whether the LineControlMode feature exists on the USART module.
	PLIB_USART_ExistsLoopback	Identifies whether the Loopback feature exists on the USART module.
	PLIB_USART_ExistsOperationMode	Identifies whether the OperationMode feature exists on the USART module.
	PLIB_USART_ExistsReceiver	Identifies whether the Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressAutoDetect	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressDetect	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverDataAvailableStatus	Identifies whether the ReceiverDataAvailable feature exists on the USART module
	PLIB_USART_ExistsReceiverEnable	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	PLIB_USART_ExistsReceiverFramingErrorStatus	Identifies whether the ReceiverFramingError feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStateLowEnable	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStatus	Identifies whether the ReceiverIdle feature exists on the USART module.
	PLIB_USART_ExistsReceiverInterruptMode	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsReceiverOverrunStatus	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	PLIB_USART_ExistsReceiverParityErrorStatus	Identifies whether the ReceiverParityError feature exists on the USART module.

	PLIB_USART_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USART module.
	PLIB_USART_ExistsTransmitter	Identifies whether the Transmitter feature exists on the USART module.
	PLIB_USART_ExistsTransmitter9BitsSend	Identifies whether the Transmitter9Bits feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBreak	Identifies whether the TransmitterBreak feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBufferFullStatus	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEmptyStatus	Identifies whether the TransmitterEmpty feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEnable	Identifies whether the TransmitterEnableControl feature exists on the USART module
	PLIB_USART_ExistsTransmitterIdleIsLow	Identifies whether the TransmitterIdleIsLow feature exists on the USART module.
	PLIB_USART_ExistsTransmitterInterruptMode	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsWakeOnStart	Identifies whether the WakeOnStart feature exists on the USART module.
	PLIB_USART_ExistsReceiver9Bits	Identifies whether the 9 Bits Receiver feature exists on the USART module.
	PLIB_USART_ExistsBRGClockSourceSelect	Identifies whether the BRG Clock source select feature exists on the USART module.
	PLIB_USART_ExistsModuleBusyStatus	Identifies whether the module running status feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddress	Identifies whether the Receiver Address feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressMask	Identifies whether the Receiver Address Mask feature exists on the USART module.
	PLIB_USART_ExistsRunInOverflow	Identifies whether the Run in overflow condition feature exists on the USART module.
	PLIB_USART_ExistsRunInSleepMode	Identifies whether the Run in Sleep mode feature exists on the USART module.

f) Data Types and Constants

	Name	Description
	USART_HANDSHAKE_MODE	Lists the USART handshake modes.
	USART_LINECONTROL_MODE	Data type defining the different configurations by which the USART data flow can be configured.
	USART_MODULE_ID	Enumeration: USART_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	_USART_MODULE_ID	
	USART_OPERATION_MODE	Data type defining the different configurations by which the USART can be enabled.
	USART_RECEIVE_INTR_MODE	Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.
	USART_TRANSMIT_INTR_MODE	Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Description

This section describes the Application Programming Interface (API) functions of the USART Peripheral Library.

Refer to each section for a detailed description.

a) General Functions

PLIB_USART_Disable Function

Disables the specific USART module

File

plib_usart.h

C

```
void PLIB_USART_Disable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific USART module.

Remarks

The default state after any reset for a USART module is the disable state. If the USART is disabled, all of the related pins are in control of the general purpose I/O logic.

Disabling the USART module resets the buffers to empty states. Any data characters in the buffers are lost and the baud rate is reset. All error and status bits are also reset.

Disabling the USART while the USART is active, will abort all pending transmissions and receptions. Re-enabling the USART will restart the module in the same configuration.

When disabled, the USART power consumption is minimal. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Disable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_Disable (USART_MODULE_ID index)

PLIB_USART_Enable Function

Enables the specific USART module.

File

plib_usart.h

C

```
void PLIB_USART_Enable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific USART module.

Remarks

By calling this function, the USART pins are controlled by the USART module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Enable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_Enable (USART_MODULE_ID index)

PLIB_USART_HandshakeModeSelect Function

Sets the data flow configuration.

File

plib_usart.h

C

```
void PLIB_USART_HandshakeModeSelect(USART_MODULE_ID index, USART_HANDSHAKE_MODE handshakeConfig);
```

Returns

None.

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsHandshakeMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
                                    USART_HANDSHAKE_MODE_SIMPLEX);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_HANDSHAKE_MODE

Function

void PLIB_USART_HandshakeModeSelect(USART_MODULE_ID index,
USART_HANDSHAKE_MODE handshakeConfig)

PLIB_USART_IrDADisable Function

Disables the IrDA encoder and decoder.

File

plib_usart.h

C

```
void PLIB_USART_IrDADisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the IrDA encoder and decoder.

Remarks

By default, the IrDA Encoder and Decoder are disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsIrDA in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_IrDADisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_IrDADisable (USART_MODULE_ID index)

PLIB_USART_IrDAEnable Function

Enables the IrDA encoder and decoder.

File

plib_usart.h

C

```
void PLIB_USART_IrDAEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the IrDA encoder and decoder.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsIrDA in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_IrDAEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_IrDAEnable (USART_MODULE_ID index)

PLIB_USART_LineControlModeSelect Function

Sets the data flow configuration.

File

plib_usart.h

C

```
void PLIB_USART_LineControlModeSelect(USART_MODULE_ID index, USART_LINECONTROL_MODE dataFlowConfig);
```

Returns

None.

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLineControlMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE,
                                 USART_8N1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_LINECONTROL_MODE

Function

void PLIB_USART_LineControlModeSelect(USART_MODULE_ID index,
USART_LINECONTROL_MODE dataFlowConfig)

PLIB_USART_LoopbackDisable Function

Disables Loopback mode.

File

plib_usart.h

C

```
void PLIB_USART_LoopbackDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables Loopback mode.

Remarks

By default, Loopback mode is disabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLoopback in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_LoopbackDisable (USART_MODULE_ID index)

PLIB_USART_LoopbackEnable Function

Enables Loopback mode.

File

plib_usart.h

C

```
void PLIB_USART_LoopbackEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables Loopback mode.

Remarks

By default, Loopback mode is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLoopback in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_LoopbackEnable (USART_MODULE_ID index)

PLIB_USART_OperationModeSelect Function

Configures the operation mode of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_OperationModeSelect(USART_MODULE_ID index, USART_OPERATION_MODE operationmode);
```

Returns

None.

Description

This function configures the operation mode of the USART (i.e., controls the pins used by the USART module). Refer to USART_OPERATION_MODE for the possible settings.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_OperationModeSelect(MY_USART_INSTANCE, USART_ENABLE_TX_RX_BCLK_USED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
operationmode	One of the possible values from USART_OPERATION_MODE

Function

void PLIB_USART_OperationModeSelect(USART_MODULE_ID index,
USART_OPERATION_MODE operationmode)

PLIB_USART_StopInIdleDisable Function

Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).

File

plib_usart.h

C

```
void PLIB_USART_StopInIdleDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Stop in Idle mode. The USART module continues operation when the device is in Idle mode.

Remarks

By default, the USART module will continue operation in Idle mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_StopInIdleDisable (USART_MODULE_ID index)

PLIB_USART_StopInIdleEnable Function

Discontinues operation when the device enters Idle mode.

File

plib_usart.h

C

```
void PLIB_USART_StopInIdleEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module to discontinue operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_StopInIdleEnable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartDisable Function

Disables the wake-up on start bit detection feature during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_WakeOnStartDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the wake-up on start bit detection feature during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_WakeOnStartDisable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartEnable Function

Enables the wake-up on start bit detection feature during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_WakeOnStartEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the wake-up on start feature during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_WakeOnStartEnable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartIsEnabled Function

Gets the state of the sync break event completion.

File

plib_usart.h

C

```
bool PLIB_USART_WakeOnStartIsEnabled(USART_MODULE_ID index);
```

Returns

None.

Description

This function returns the status of the sync break event, when called after enabling using PLIB_USART_WakeOnStartEnable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

//Call the interface just prior to entering the sleep mode.
PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);

// Check the status if the Sync break event is over.
if(PLIB_USART_WakeOnStartIsEnabled(MY_USART_INSTANCE))
{
   // Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_WakeOnStartIsEnabled (USART_MODULE_ID index)

PLIB_USART_ErrorsGet Function

Return the status of all errors in the specified USART module.

File

plib_usart.h

C

```
USART_ERROR PLIB_USART_ErrorsGet(USART_MODULE_ID index);
```

Returns

Returns a bitmap of USART error status.

Description

This function returns status of all errors in the specified USART module. The return value can be bitwise AND'ed with a USART_ERROR type to know the status of a specific error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the followings functions return true:

- PLIB_USART_ExistsReceiverFramingErrorStatus
- PLIB_USART_ExistsReceiverParityErrorStatus
- PLIB_USART_ExistsReceiverOverrunStatus

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

USART_ERROR error;

// Get the status of all errors.
error = PLIB_USART_ErrorsGet(MY_USART_INSTANCE);

// Check if parity error is active
if(error & USART_ERROR_PARITY)
{
    // Parity error is active.
}
else if(error & USART_ERROR_FRAMING)
```

```
{
    // Framing error is active.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

USART_ERROR PLIB_USART_ErrorsGet (USART_MODULE_ID index)

PLIB_USART_InitializeModeGeneral Function

Enables or disables general features of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_InitializeModeGeneral(USART_MODULE_ID index, bool autobaud, bool loopBackMode, bool
wakeFromSleep, bool irdaMode, bool stopInIdle);
```

Returns

None.

Description

This function enables or disables general features of the USART module.

Remarks

Enabling the wake from sleep feature will cause the first character that is received by the USART module to be discarded. This feature should only be enabled if the CPU is to placed in power saving mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsLoopback
- PLIB_USART_ExistsBaudRateAutoDetect
- PLIB_USART_ExistsWakeOnStart
- PLIB_USART_ExistsIrDA
- PLIB_USART_ExistsStopInIdle

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Enable loopback, disable IrDA, disable auto baud detection and disable
// wake from sleep. Enable stop in idle
PLIB_USART_InitializeModeGeneral(MY_USART_INSTANCE, false, true,
                                                    false, false, true);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
autobaud	If true, auto baud rate detection is enabled. If false the feature is disabled.
loopBackMode	If true, loop back is enabled. If false the feature is disabled.
wakeFromSleep	If true, the USART module will wake up the CPU from sleep mode on USART activity. If false the feature is disabled.
irdaMode	If true, the IrDA mode is enabled. If false the feature is disabled.
stopInIdle	If true, module will stop functioning when CPU enters Idle mode. If false, the feature is disabled.

Function

void PLIB_USART_InitializeModeGeneral(USART_MODULE_ID index, bool autobaud,
bool loopBackMode, bool wakeFromSleep, bool irdaMode, bool stopInIdle);

PLIB_USART_InitializeOperation Function

Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.

File

plib_usart.h

C

```
void PLIB_USART_InitializeOperation(USART_MODULE_ID index, USART_RECEIVE_INTR_MODE receiveInterruptMode,
USART_TRANSMIT_INTR_MODE transmitInterruptMode, USART_OPERATION_MODE operationMode);
```

Returns

None.

Description

This function configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsReceiverInterruptMode
- PLIB_USART_ExistsTransmitterInterruptMode
- PLIB_USART_ExistsOperationMode

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Set receive FIFO to interrupt when FIFO is 3/4 level full
// Set Transmit FIFO to interrupt when FIFO is empty
// USART module will only use RX and TX hardware lines

PLIB_USART_InitializeOperation(MY_USART_INSTANCE, USART_RECEIVE_FIFO_3B4FULL,
    USART_TRANSMIT_FIFO_EMPTY  , USART_ENABLE_TX_RX_USED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
receiveInterruptMode	Receiver FIFO interrupt level
transmitInterruptMode	Transmit FIFO interrupt level
operationMode	Hardware lines to be used by the USART.

Function

void PLIB_USART_InitializeOperation(USART_MODULE_ID index ,
USART_RECEIVE_INTR_MODE receiveInterruptMode,
USART_TRANSMIT_INTR_MODE transmitInterruptMode,
USART_OPERATION_MODE operationMode);

PLIB_USART_AddressGet Function

Gets the address for the Address Detect mode.

File

plib_usart.h

C

```
uint8_t PLIB_USART_AddressGet(USART_MODULE_ID index);
```

Returns

- address - The address being used

Description

This function returns the address value being used for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t address = 0;
address = PLIB_USART_AddressGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_USART_AddressGet

(

USART_MODULE_ID index

)

PLIB_USART_AddressMaskGet Function

Gets the address mask for the Address Detect mode.

File

plib_usart.h

C

```
uint8_t PLIB_USART_AddressMaskGet(USART_MODULE_ID index);
```

Returns

- mask - Address mask being used

Description

This function returns the address mask value being used for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t mask = 0;
mask = PLIB_USART_AddressMaskGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_USART_AddressMaskGet
(
USART_MODULE_ID index
)

PLIB_USART_AddressMaskSet Function

Sets the address mask for the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_AddressMaskSet(USART_MODULE_ID index, uint8_t mask);
```

Returns

None.

Description

This function sets the address mask for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t mask = 0x0F;
PLIB_USART_AddressMaskSet (MY_USART_INSTANCE, mask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mask	Address match mask bits

Function

void PLIB_USART_AddressMaskSet
(
USART_MODULE_ID index,
uint8_t mask
)

PLIB_USART_AddressSet Function

Sets the address for the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_AddressSet(USART_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the address for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t address = 0x02;
PLIB_USART_AddressSet (MY_USART_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	Address

Function

void PLIB_USART_AddressSet

(

USART_MODULE_ID index,

uint8_t address

)

PLIB_USART_ModuleIsBusy Function

Returns the USART module's running status.

File

plib_usart.h

C

```
bool PLIB_USART_ModuleIsBusy(USART_MODULE_ID index);
```

Returns

- true - USART module is busy
- false - USART module is idle

Description

This function checks if the USART module status is busy. The following functions should not be used when the module status is busy:

- PLIB_USART_LineControlModeSelect
- PLIB_USART_BaudRateHighSet
- PLIB_USART_BaudRateHighEnable
- PLIB_USART_BaudRateHighDisable
- PLIB_USART_ReceiverIdleStateLowEnable
- PLIB_USART_ReceiverIdleStateLowDisable
- PLIB_USART_BaudRateAutoDetectEnable
- PLIB_USART_LoopbackEnable
- PLIB_USART_LoopbackDisable
- PLIB_USART_WakeOnStartEnable
- PLIB_USART_WakeOnStartDisable
- PLIB_USART_OperationModeSelect
- PLIB_USART_HandshakeModeSelect

- PLIB_USART_IrDAEnable
- PLIB_USART_IrDADisable
- PLIB_USART_StopInIdleEnable
- PLIB_USART_StopInIdleDisable
- PLIB_USART_RunInOverflowEnable
- PLIB_USART_RunInOverflowDisable
- PLIB_USART_BRGClockSourceSelect
- PLIB_USART_RunInSleepModeEnable
- PLIB_USART_RunInSleepModeDisable

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsModuleBusyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
bool moduleStatus;

moduleStatus = PLIB_USART_ModuleIsBusy (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_ModuleIsBusy (USART_MODULE_ID index)

PLIB_USART_RunInOverflowDisable Function

Disables the Run in overflow condition mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInOverflowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the USART module from accepting new data when an overflow error condition is detected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInOverflowDisable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInOverflowDisable (USART_MODULE_ID index)

PLIB_USART_RunInOverflowEnable Function

Enables the USART module to continue to operate when an overflow error condition has occurred.

File

plib_usart.h

C

```
void PLIB_USART_RunInOverflowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module to continue to operate when an overflow error condition has occurred.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInOverflowEnable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInOverflowEnable (USART_MODULE_ID index)

PLIB_USART_RunInOverflowIsEnabled Function

Gets the status of the Run in Overflow condition.

File

plib_usart.h

C

```
bool PLIB_USART_RunInOverflowIsEnabled(USART_MODULE_ID index);
```

Returns

- true - Run in overflow condition is enabled
- false - Run in overflow condition is disabled

Description

This function indicates if the USART module has been enabled to run in an overflow condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool status;
status = PLIB_USART_RunInOverflowIsEnabled (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_RunInOverflowIsEnabled (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeDisable Function

Turns off the USART module's BRG clock during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInSleepModeDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function turns off the USART module's BRG clock during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInSleepModeDisable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInSleepModeDisable (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeEnable Function

Allows the USART module's BRG clock to run when the device enters Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInSleepModeEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module's BRG clock to continue operation when the device enters the Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInSleepModeEnable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInSleepModeEnable (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeIsEnabled Function

Gets the status of Run in Sleep mode.

File

plib_usart.h

C

```
bool PLIB_USART_RunInSleepModeIsEnabled(USART_MODULE_ID index);
```

Returns

- true - Run in Sleep mode is enabled
- false - Run in Sleep mode is disabled

Description

This function indicates if the USART module has been enabled to run in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool status;
status = PLIB_USART_RunInSleepModeIsEnabled (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_RunInSleepModeIsEnabled (USART_MODULE_ID index)

b) Baud Rate Generator Functions

PLIB_USART_BaudRateAutoDetectEnable Function

Enables baud rate measurement on the next character, which requires reception of the Sync character.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateAutoDetectEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the baud rate measurement on the next character, which requires reception of the Sync character.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);

// Wait until the baud rate is detected.
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateAutoDetectEnable (USART_MODULE_ID index)

PLIB_USART_BaudRateAutoDetectIsComplete Function

Gets the state of the automatic baud detection.

File

plib_usart.h

C

```
bool PLIB_USART_BaudRateAutoDetectIsComplete(USART_MODULE_ID index);
```

Returns

- true - Baud rate detection is not complete
- false - Baud rate detection is complete

Description

This function gets the state of the automatic baud detection and returns a '0' if the baud rate auto detection is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);

// Wait until the baud rate is detected.
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_BaudRateAutoDetectIsComplete (USART_MODULE_ID index)

PLIB_USART_BaudRateGet Function

Gets the baud rate current in use.

File

plib_usart.h

C

```
uint32_t PLIB_USART_BaudRateGet(USART_MODULE_ID index, int32_t clockFrequency);
```

Returns

- BaudRate - Baud rate value

Description

This function gets the baud rate that is currently in use. The clock frequency needs to be passed to the function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRate ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRate = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock Frequency

Function

uint32_t PLIB_USART_BaudRateGet (USART_MODULE_ID index,
int32_t clockFrequency);

PLIB_USART_BaudRateHighDisable Function

Disables the high baud rate selection.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the high baud rate selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateHighDisable (USART_MODULE_ID index)

PLIB_USART_BaudRateHighEnable Function

Enables high baud rate selection.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables high baud rate selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateHighEnable (USART_MODULE_ID index)

PLIB_USART_BaudRateHighSet Function

Sets the baud rate to the desired value.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighSet(USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value, it is the baud rate value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudRateHighSet (USART_MODULE_ID index,

uint32_t clockFrequency, uint32_t baudRate);

PLIB_USART_BaudRateSet Function

Sets the baud rate to the desired value.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateSet(USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudRateSet (USART_MODULE_ID index, uint32_t clockFrequency,

uint32_t baudRate);

PLIB_USART_BaudSetAndEnable Function

Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.

File

plib_usart.h

C

```
void PLIB_USART_BaudSetAndEnable(USART_MODULE_ID index, uint32_t systemClock, uint32_t baud);
```

Returns

None.

Description

This function sets the baud rate to the desired value and enables the USART receiver, USART transmitter and USART module.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsBaudRate
- PLIB_USART_ExistsTransmitterEnable
- PLIB_USART_ExistsReceiverEnable
- PLIB_USART_ExistsEnable

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudSetAndEnable(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudSetAndEnable (USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);

PLIB_USART_BRGClockSourceGet Function

Gets the BRG clock source of the USART module.

File

plib_usart.h

C

```
USART_BRG_CLOCK_SOURCE PLIB_USART_BRGClockSourceGet(USART_MODULE_ID index);
```

Returns

One of the possible values of USART_BRG_CLOCK_SOURCE

Description

This function returns the BRG Clock source of the USART. Refer to USART_BRG_CLOCK_SOURCE for the possible clock sources.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBRGClockSourceSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

USART_BRG_CLOCK_SOURCE brgClockSource;
brgClockSource = PLIB_USART_BRGClockSourceGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

USART_BRG_CLOCK_SOURCE PLIB_USART_BRGClockSourceGet (USART_MODULE_ID index)

PLIB_USART_BRGClockSourceSelect Function

Configures the BRG clock source of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_BRGClockSourceSelect(USART_MODULE_ID index, USART_BRG_CLOCK_SOURCE brgClockSource);
```

Returns

None.

Description

This function configures the BRG Clock source of the USART. Refer to USART_BRG_CLOCK_SOURCE for the possible clock sources.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBRGClockSourceSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BRGClockSourceSelect (MY_USART_INSTANCE, USART_BRG_CLOCK_SOURCE_FRC_IN_SLEEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
brgClockSource	One of the possible values from USART_BRG_CLOCK_SOURCE

Function

void PLIB_USART_BRGClockSourceSelect
(
    USART_MODULE_ID index,
USART_BRG_CLOCK_SOURCE brgClockSource
)

c) Transmitter Functions

PLIB_USART_Transmitter9BitsSend Function

Data to be transmitted in the byte mode with the 9th bit.

File

plib_usart.h

C

```
void PLIB_USART_Transmitter9BitsSend(USART_MODULE_ID index, int8_t data, bool Bit9th);
```

Returns

None.

Description

The data is transmitted in the byte mode for the specified USART module, with 9th bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitter9BitsSend in your application to determine whether this feature is available.

Preconditions

The USART module should be configured to use the 9 data bits.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint8_t data = 'a';

if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))
{
    PLIB_USART_Transmitter9BitsSend(MY_USART_INSTANCE, data, false);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.
9thBit	9th bit of the data to be transmitted.

Function

void PLIB_USART_Transmitter9BitsSend (USART_MODULE_ID index,

int8_t data, bool 9thBit)

PLIB_USART_TransmitterBreakSend Function

Transmits the break character.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterBreakSend(USART_MODULE_ID index);
```

Returns

None.

Description

This function transmits the break character.

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBreak in your application to determine whether this feature is available.

Preconditions

The application should wait for the transmitter to be idle, before calling this function.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterBreakSend (USART_MODULE_ID index)

PLIB_USART_TransmitterBreakSendIsComplete Function

Returns the status of the break transmission

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterBreakSendIsComplete(USART_MODULE_ID index);
```

Returns

- true - Transmit break on the next transmission
- false - Break transmission completed or not started

Description

The function returns the status of the break transmission.

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBreak in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterBreakSendIsComplete (USART_MODULE_ID index)

PLIB_USART_TransmitterBufferIsFull Function

Gets the transmit buffer full status.

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterBufferIsFull(USART_MODULE_ID index);
```

Returns

- true - The transmit buffer is full
- false - The transmit buffer is not full, at least one more character can be written

Description

This function gets the transmit status of the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBufferFullStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterBufferIsFull (USART_MODULE_ID index)

PLIB_USART_TransmitterByteSend Function

Data to be transmitted in the Byte mode.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterByteSend(USART_MODULE_ID index, int8_t data);
```

Returns

None.

Description

The data is transmitted in the Byte mode for the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitter in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint8_t data = 'a';
if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))
{
    PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, data);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.

Function

void PLIB_USART_TransmitterByteSend (USART_MODULE_ID index, int8_t data)

PLIB_USART_TransmitterDisable Function

Disables the specific USART module transmitter.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific USART module transmitter.

Remarks

Disabling the transmitter during a transmission will cause the transmission to be aborted.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterDisable (USART_MODULE_ID index)

PLIB_USART_TransmitterEnable Function

Enables the specific USART module transmitter.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific USART module transmitter.

Remarks

The transmitter should not be enabled until the USART is enabled. The transmissions will not be enabled otherwise.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEnable in your application to determine whether this feature is available.

Preconditions

The USART module should be enabled using the function PLIB_USART_Enable before this function is called.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterEnable (USART_MODULE_ID index)

PLIB_USART_TransmitterIdleIsLowDisable Function

Disables the Transmit Idle Low state.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterIdleIsLowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Transmit Idle Low state. In USART Synchronous mode, this function configures that the TX polarity the idle state is high. When IrDA is enabled, this function sets the IrDA encoded Transmit Idle state to a '0'.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterIdleIsLow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterIdleIsLowDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterIdleIsLowDisable (USART_MODULE_ID index)

PLIB_USART_TransmitterIdleIsLowEnable Function

Enables the Transmit Idle Low state.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterIdleIsLowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the Transmit Idle Low state. In the USART Synchronous mode, this function configures that the TX polarity, the idle state is low.

When IrDA is enabled, this function sets that IrDA encoded Transmit Idle state to a '1'.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterIdleIsLow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterIdleIsLowEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterIdleIsLowEnable (USART_MODULE_ID index)

PLIB_USART_TransmitterInterruptModeSelect Function

Sets the USART transmitter interrupt mode.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterInterruptModeSelect(USART_MODULE_ID index, USART_TRANSMIT_INTR_MODE fifolevel);
```

Returns

None.

Description

This function sets the condition in which the USART module should generate an interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterInterruptMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterInterruptModeSelect(MY_USART_INSTANCE,
    USART_TRANSMIT_FIFO_EMPTY );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptMode	Interrupt mode; for possible configurations, refer to USART_TRANSMIT_INTR_MODE

Function

void PLIB_USART_TransmitterInterruptModeSelect(USART_MODULE_ID index,
USART_TRANSMIT_INTR_MODE interruptMode)

PLIB_USART_TransmitterIsEmpty Function

Gets the transmit shift register empty status.

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterIsEmpty(USART_MODULE_ID index);
```

Returns

- true - The transmit shift register is empty
- false - The transmit shift register is not empty

Description

This function gets the transmit shift register empty status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEmptyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(!PLIB_USART_TransmitterIsEmpty(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterIsEmpty (USART_MODULE_ID index)

PLIB_USART_TransmitterAddressGet Function

Returns the address of the USART TX register

File

plib_usart.h

C

```
void* PLIB_USART_TransmitterAddressGet(USART_MODULE_ID index);
```

Returns

Address of the USART TX register

Description

This function returns the address of the USART TX register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_TransmitterAddressGet(USART_MODULE_ID index)

d) Receiver Functions

PLIB_USART_ReceiverAddressAutoDetectDisable Function

Disables the automatic Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressAutoDetectDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the automatic Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressAutoDetectDisable (
USART_MODULE_ID index)

PLIB_USART_ReceiverAddressAutoDetectEnable Function

Setup the automatic Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressAutoDetectEnable(USART_MODULE_ID index, int8_t Mask);
```

Returns

None.

Description

This function configures the automatic Address Detect mode. Uses the mask as the address character for automatic address detection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectEnable(MY_USART_INSTANCE,
                                           MY_DEVICE_ADDRESS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
Mask	Address character to be used, when enabled

Function

void PLIB_USART_ReceiverAddressAutoDetectEnable(USART_MODULE_ID index,

int8_t Mask)

PLIB_USART_ReceiverAddressDetectDisable Function

Enables the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressDetectDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Remarks

All bytes are received, and the 9th bit can be used as the parity bit. By default, the address detect is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressDetectDisable (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressDetectEnable Function

Enables the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressDetectEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Remarks

If 9 data bits are not selected, this bit has no effect.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressDetectEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressIsReceived Function

Checks and return if the data received is an address.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverAddressIsReceived(USART_MODULE_ID index);
```

Returns

- true - if the data received has the 9th bit set
- false - if the address received has the 9th bit cleared

Description

Checks and return if the data received is an address. The address has the 9th bit set. If data received has the 9th bit set, the function returns true; otherwise, the function returns false.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

The USART module should be configured to use the 9 data bits.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t address;

if(PLIB_USART_ReceiverAddressIsReceived(MY_USART_INSTANCE))
{
    address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverAddressIsReceived (USART_MODULE_ID index)

PLIB_USART_ReceiverByteReceive Function

Data to be received in the Byte mode.

File

plib_usart.h

C

```
int8_t PLIB_USART_ReceiverByteReceive(USART_MODULE_ID index);
```

Returns

- data - Data to be received

Description

The data to be received in Byte mode from the specified USART module. Call the functions PLIB_USART_ReceiverFramingErrorHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred and PLIB_USART_ReceiverOverrunHasOccurred to get any error that occurred.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiver in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool isError;
uint8_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}

isError = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE)     |
              PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE) |
              PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE);

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

int8_t PLIB_USART_ReceiverByteReceive (USART_MODULE_ID index)

PLIB_USART_ReceiverDataIsAvailable Function

Identifies if the receive data is available for the specified USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverDataIsAvailable(USART_MODULE_ID index);
```

Returns

- true - The data is available
- false - The data is not available

Description

This function identifies if the receive data is available for the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverDataAvailableStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverDataIsAvailable (USART_MODULE_ID index)

PLIB_USART_ReceiverDisable Function

Disables the USART receiver.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the USART receiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverDisable (USART_MODULE_ID index)

PLIB_USART_ReceiverEnable Function

Enables the USART receiver.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART receiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverFramingErrorHasOccurred Function

Gets the framing error status.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverFramingErrorHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The framing error was detected on the current character
- false - The framing error was not detected on the current character

Description

This function gets the framing error status.

Remarks

Reading the error clears the error.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverFramingErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverFramingErrorHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverIdleStateLowDisable Function

Disables receive polarity inversion.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverIdleStateLowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is not inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStateLowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverIdleStateLowDisable (USART_MODULE_ID index);

PLIB_USART_ReceiverIdleStateLowEnable Function

Enables receive polarity inversion.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverIdleStateLowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStateLowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverIdleStateLowEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverInterruptModeSelect Function

Sets the USART receiver FIFO level.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverInterruptModeSelect(USART_MODULE_ID index, USART_RECEIVE_INTR_MODE interruptMode);
```

Returns

None.

Description

This function sets the USART receiver FIFO level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverInterruptMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverInterruptModeSelect(MY_USART_INSTANCE,
    USART_RECEIVE_FIFO_ONE_CHAR );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
fifolevel	For possible configurations, refer to USART_RECEIVE_INTR_MODE

Function

void PLIB_USART_ReceiverInterruptModeSelect(USART_MODULE_ID index,
USART_RECEIVE_INTR_MODE interruptMode)

PLIB_USART_ReceiverIsIdle Function

Identifies if the receiver is idle.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverIsIdle(USART_MODULE_ID index);
```

Returns

- true - The receive buffer is idle
- false - The receive buffer is not idle

Description

This function identifies if the receiver is idle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t mydata;

if(PLIB_USART_ReceiverIsIdle(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverIsIdle (USART_MODULE_ID index)

PLIB_USART_ReceiverOverrunErrorClear Function

Clears a USART vverrun error.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverOverrunErrorClear(USART_MODULE_ID index);
```

Returns

None.

Description

This function clears an overrun error. Clearing the error, resets the receive buffer.

Remarks

WARNING: Calling this API will clear all of the previously received data.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverOverrunStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverOverrunErrorClear (USART_MODULE_ID index)

PLIB_USART_ReceiverOverrunHasOccurred Function

Identifies if there was a receiver overrun error.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverOverrunHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The receive buffer has overflowed
- false - The receive buffer has not overflowed

Description

This function identifies if there was a receiver overrun error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverOverrunStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverOverrunHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverParityErrorHasOccurred Function

Gets the parity error status.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverParityErrorHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The parity error was detected on the current character
- false - The parity error was not detected on the current character

Description

This function gets the parity error status.

Remarks

Reading the error clears the error. A Parity error is irrelevant in case of 9-bit mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverParityErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverParityErrorHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressGet Function

Returns the address of the USART RX register

File

plib_usart.h

C

```
void* PLIB_USART_ReceiverAddressGet(USART_MODULE_ID index);
```

Returns

Address of the USART RX register

Description

This function returns the address of the USART RX register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ReceiverAddressGet(USART_MODULE_ID index)

PLIB_USART_Receiver9BitsReceive Function

Data to be received in the byte mode with the 9th bit.

File

plib_usart.h

C

```
int16_t PLIB_USART_Receiver9BitsReceive(USART_MODULE_ID index);
```

Returns

- data - Data to be received

Description

The data to be received in Byte mode from the specified USART module. with the 9th bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiver9Bits in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint16_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_Receiver9BitsReceive (MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

int16_t PLIB_USART_Receiver9BitsReceive (USART_MODULE_ID index)

e) Feature Existence Functions

PLIB_USART_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRate(USART_MODULE_ID index);
```

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateSet
- PLIB_USART_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRate(USART_MODULE_ID index)

PLIB_USART_ExistsBaudRateAutoDetect Function

Identifies whether the BaudRateAutoDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRateAutoDetect(USART_MODULE_ID index);
```

Returns

- true - The BaudRateAutoDetect feature is supported on the device
- false - The BaudRateAutoDetect feature is not supported on the device

Description

This function identifies whether the BaudRateAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateAutoDetectEnable
- PLIB_USART_BaudRateAutoDetectIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRateAutoDetect(USART_MODULE_ID index)

PLIB_USART_ExistsBaudRateHigh Function

Identifies whether the BaudRateHigh feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRateHigh(USART_MODULE_ID index);
```

Returns

- true - The BaudRateHigh feature is supported on the device
- false - The BaudRateHigh feature is not supported on the device

Description

This function identifies whether the BaudRateHigh feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateHighSet
- PLIB_USART_BaudRateHighDisable
- PLIB_USART_BaudRateHighEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRateHigh(USART_MODULE_ID index)

PLIB_USART_ExistsEnable Function

Identifies whether the EnableControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsEnable(USART_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the USART module. When this function returns true, these functions are

supported on the device:

- PLIB_USART_Disable
- PLIB_USART_Enable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsEnable(USART_MODULE_ID index)

PLIB_USART_ExistsHandshakeMode Function

Identifies whether the HandShakeMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsHandshakeMode(USART_MODULE_ID index);
```

Returns

- true - The HandShakeMode feature is supported on the device
- false - The HandShakeMode feature is not supported on the device

Description

This function identifies whether the HandShakeMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_HandshakeModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsHandshakeMode(USART_MODULE_ID index)

PLIB_USART_ExistsIrDA Function

Identifies whether the IrDAControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsIrDA(USART_MODULE_ID index);
```

Returns

- true - The IrDAControl feature is supported on the device

- false - The IrDAControl feature is not supported on the device

Description

This function identifies whether the IrDAControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_IrDADisable
- PLIB_USART_IrDAEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsIrDA(USART_MODULE_ID index)

PLIB_USART_ExistsLineControlMode Function

Identifies whether the LineControlMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsLineControlMode(USART_MODULE_ID index);
```

Returns

- true - The LineControlMode feature is supported on the device
- false - The LineControlMode feature is not supported on the device

Description

This function identifies whether the LineControlMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_LineControlModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsLineControlMode(USART_MODULE_ID index)

PLIB_USART_ExistsLoopback Function

Identifies whether the Loopback feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsLoopback(USART_MODULE_ID index);
```

Returns

- true - The Loopback feature is supported on the device
- false - The Loopback feature is not supported on the device

Description

This function identifies whether the Loopback feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_LoopbackEnable
- PLIB_USART_LoopbackDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsLoopback(USART_MODULE_ID index)

PLIB_USART_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsOperationMode(USART_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_OperationModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsOperationMode(USART_MODULE_ID index)

PLIB_USART_ExistsReceiver Function

Identifies whether the Receiver feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiver(USART_MODULE_ID index);
```

Returns

- true - The Receiver feature is supported on the device
- false - The Receiver feature is not supported on the device

Description

This function identifies whether the Receiver feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverByteReceive
- PLIB_USART_ReceiverAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiver(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressAutoDetect Function

Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressAutoDetect(USART_MODULE_ID index);
```

Returns

- true - The ReceiverAddressAutoDetect feature is supported on the device
- false - The ReceiverAddressAutoDetect feature is not supported on the device

Description

This function identifies whether the ReceiverAddressAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverAddressAutoDetectEnable
- PLIB_USART_ReceiverAddressAutoDetectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressAutoDetect(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressDetect Function

Identifies whether the ReceiverAddressDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressDetect(USART_MODULE_ID index);
```

Returns

- true - The ReceiverAddressDetect feature is supported on the device
- false - The ReceiverAddressDetect feature is not supported on the device

Description

This function identifies whether the ReceiverAddressDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverAddressDetectEnable
- PLIB_USART_ReceiverAddressDetectDisable
- PLIB_USART_ReceiverAddressIsReceived

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressDetect(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverDataAvailableStatus Function

Identifies whether the ReceiverDataAvailable feature exists on the USART module

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverDataAvailableStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverDataAvailable feature is supported on the device
- false - The ReceiverDataAvailable feature is not supported on the device

Description

This function identifies whether the ReceiverDataAvailable feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverDataIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverDataAvailableStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverEnable Function

Identifies whether the ReceiverEnableControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverEnable(USART_MODULE_ID index);
```

Returns

- true - The ReceiverEnableControl feature is supported on the device
- false - The ReceiverEnableControl feature is not supported on the device

Description

This function identifies whether the ReceiverEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverEnable
- PLIB_USART_ReceiverDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverEnable(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverFramingErrorStatus Function

Identifies whether the ReceiverFramingError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverFramingErrorStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverFramingError feature is supported on the device
- false - The ReceiverFramingError feature is not supported on the device

Description

This function identifies whether the ReceiverFramingError feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverFramingErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverFramingErrorStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverIdleStateLowEnable Function

Identifies whether the ReceiverPolarityInvert feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverIdleStateLowEnable(USART_MODULE_ID index);
```

Returns

- true - The ReceiverPolarityInvert feature is supported on the device
- false - The ReceiverPolarityInvert feature is not supported on the device

Description

This function identifies whether the ReceiverPolarityInvert feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverIdleStateLowEnable
- PLIB_USART_ReceiverIdleStateLowDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverIdleStateLowEnable(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverIdleStatus Function

Identifies whether the ReceiverIdle feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverIdleStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverIdle feature is supported on the device
- false - The ReceiverIdle feature is not supported on the device

Description

This function identifies whether the ReceiverIdle feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverIsIdle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverIdleStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverInterruptMode Function

Identifies whether the ReceiverInterruptMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverInterruptMode(USART_MODULE_ID index);
```

Returns

- true - The ReceiverInterruptMode feature is supported on the device
- false - The ReceiverInterruptMode feature is not supported on the device

Description

This function identifies whether the ReceiverInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverInterruptMode(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverOverrunStatus Function

Identifies whether the ReceiverOverrunError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverOverrunStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverOverrunError feature is supported on the device
- false - The ReceiverOverrunError feature is not supported on the device

Description

This function identifies whether the ReceiverOverrunError feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverOverrunErrorClear
- PLIB_USART_ReceiverOverrunHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverOverrunStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverParityErrorStatus Function

Identifies whether the ReceiverParityError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverParityErrorStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverParityError feature is supported on the device
- false - The ReceiverParityError feature is not supported on the device

Description

This function identifies whether the ReceiverParityError feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverParityErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverParityErrorStatus(USART_MODULE_ID index)

PLIB_USART_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsStopInIdle(USART_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_StopInIdleEnable
- PLIB_USART_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsStopInIdle(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitter Function

Identifies whether the Transmitter feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitter(USART_MODULE_ID index);
```

Returns

- true - The Transmitter feature is supported on the device
- false - The Transmitter feature is not supported on the device

Description

This function identifies whether the Transmitter feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterByteSend
- PLIB_USART_TransmitterAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitter(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitter9BitsSend Function

Identifies whether the Transmitter9Bits feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitter9BitsSend(USART_MODULE_ID index);
```

Returns

- true - The Transmitter9Bits feature is supported on the device
- false - The Transmitter9Bits feature is not supported on the device

Description

This function identifies whether the Transmitter9Bits feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_Transmitter9BitsSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitter9BitsSend(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterBreak Function

Identifies whether the TransmitterBreak feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterBreak(USART_MODULE_ID index);
```

Returns

- true - The TransmitterBreak feature is supported on the device
- false - The TransmitterBreak feature is not supported on the device

Description

This function identifies whether the TransmitterBreak feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterBreakSend
- PLIB_USART_TransmitterBreakSendIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterBreak(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterBufferFullStatus Function

Identifies whether the TransmitterBufferFull feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterBufferFullStatus(USART_MODULE_ID index);
```

Returns

- true - The TransmitterBufferFull feature is supported on the device
- false - The TransmitterBufferFull feature is not supported on the device

Description

This function identifies whether the TransmitterBufferFull feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterBufferFullStatus(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterEmptyStatus Function

Identifies whether the TransmitterEmpty feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterEmptyStatus(USART_MODULE_ID index);
```

Returns

- true - The TransmitterEmpty feature is supported on the device
- false - The TransmitterEmpty feature is not supported on the device

Description

This function identifies whether the TransmitterEmpty feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterEmptyStatus(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterEnable Function

Identifies whether the TransmitterEnableControl feature exists on the USART module

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterEnable(USART_MODULE_ID index);
```

Returns

- true - The TransmitterEnableControl feature is supported on the device
- false - The TransmitterEnableControl feature is not supported on the device

Description

This function identifies whether the TransmitterEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterEnable
- PLIB_USART_TransmitterDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterEnable(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterIdleIsLow Function

Identifies whether the TransmitterIdleIsLow feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterIdleIsLow(USART_MODULE_ID index);
```

Returns

- true - The TransmitterIdleIsLow feature is supported on the device
- false - The TransmitterIdleIsLow feature is not supported on the device

Description

This function identifies whether the TransmitterIdleIsLow feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterIdleIsLowDisable
- PLIB_USART_TransmitterIdleIsLowEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterIdleIsLow(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterInterruptMode Function

Identifies whether the TransmitterInterruptMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterInterruptMode(USART_MODULE_ID index);
```

Returns

- true - The TransmitterInterruptMode feature is supported on the device
- false - The TransmitterInterruptMode feature is not supported on the device

Description

This function identifies whether the TransmitterInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterInterruptMode(USART_MODULE_ID index)

PLIB_USART_ExistsWakeOnStart Function

Identifies whether the WakeOnStart feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsWakeOnStart(USART_MODULE_ID index);
```

Returns

- true - The WakeOnStart feature is supported on the device
- false - The WakeOnStart feature is not supported on the device

Description

This function identifies whether the WakeOnStart feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_WakeOnStartEnable
- PLIB_USART_WakeOnStartDisable
- PLIB_USART_WakeOnStartIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsWakeOnStart(USART_MODULE_ID index)

PLIB_USART_ExistsReceiver9Bits Function

Identifies whether the 9 Bits Receiver feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiver9Bits(USART_MODULE_ID index);
```

Returns

- true - The feature is supported on the device
- false - The feature is not supported on the device

Description

This function identifies whether the 9 Bits Receiver feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_Receiver9BitsReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiver9Bits (USART_MODULE_ID index)

PLIB_USART_ExistsBRGClockSourceSelect Function

Identifies whether the BRG Clock source select feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBRGClockSourceSelect(USART_MODULE_ID index);
```

Returns

- true - The BRG clock source select feature is supported on the device
- false - The BRG clock source select feature is not supported on the device

Description

This function identifies whether the BRG Clock source select feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BRGClockSourceSelect
- PLIB_USART_BRGClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBRGClockSourceSelect (USART_MODULE_ID index)

PLIB_USART_ExistsModuleBusyStatus Function

Identifies whether the module running status feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsModuleBusyStatus(USART_MODULE_ID index);
```

Returns

- true - The Module running status feature is supported on the device
- false - The Module running status feature is not supported on the device

Description

This function identifies whether the module running status feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ModuleIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsModuleBusyStatus (USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddress Function

Identifies whether the Receiver Address feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddress(USART_MODULE_ID index);
```

Returns

- true - The Receiver address feature is supported on the device
- false - The Receiver address feature is not supported on the device

Description

This function identifies whether the Receiver Address feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_AddressSet
- PLIB_USART_AddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddress (USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressMask Function

Identifies whether the Receiver Address Mask feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressMask(USART_MODULE_ID index);
```

Returns

- true - The Receiver address mask feature is supported on the device
- false - The Receiver address mask feature is not supported on the device

Description

This function identifies whether the Receiver Address Mask feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_AddressMaskSet
- PLIB_USART_AddressMaskGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressMask (USART_MODULE_ID index)

PLIB_USART_ExistsRunInOverflow Function

Identifies whether the Run in overflow condition feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsRunInOverflow(USART_MODULE_ID index);
```

Returns

- true - The Run in Overflow condition feature is supported on the device
- false - The Run in Overflow condition feature is not supported on the device

Description

This function identifies whether the Run in Overflow condition feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_RunInOverflowEnable
- PLIB_USART_RunInOverflowDisable
- PLIB_USART_RunInOverflowIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsRunInOverflow (USART_MODULE_ID index)

PLIB_USART_ExistsRunInSleepMode Function

Identifies whether the Run in Sleep mode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsRunInSleepMode(USART_MODULE_ID index);
```

Returns

- true - The Run in Sleep mode feature is supported on the device
- false - The Run in Sleep mode feature is not supported on the device

Description

This function identifies whether the Run in Sleep mode feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_RunInSleepModeEnable
- PLIB_USART_RunInSleepModeDisable

- PLIB_USART_RunInSleepModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsRunInSleepMode (USART_MODULE_ID index)

f) Data Types and Constants

USART_HANDSHAKE_MODE Enumeration

Lists the USART handshake modes.

File

plib_usart_help.h

C

```
typedef enum {
  USART_HANDSHAKE_MODE_FLOW_CONTROL,
  USART_HANDSHAKE_MODE_SIMPLEX
} USART_HANDSHAKE_MODE;
```

Members

Members	Description
USART_HANDSHAKE_MODE_FLOW_CONTROL	Enables flow control
USART_HANDSHAKE_MODE_SIMPLEX	Enables simplex mode communication, no flow control

Description

USART Handshake modes

This enumeration lists the USART handshake modes.

Remarks

None.

USART_LINECONTROL_MODE Enumeration

Data type defining the different configurations by which the USART data flow can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_8N1,
  USART_8E1,
  USART_8O1,
  USART_8N2,
  USART_8E2,
  USART_8O2,
  USART_9N1,
  USART_9N2
} USART_LINECONTROL_MODE;
```

Members

Members	Description
USART_8N1	8 Data Bits, No Parity,one Stop Bit
USART_8E1	8 Data Bits, Even Parity, 1 stop bit
USART_8O1	8 Data Bits, odd Parity, 1 stop bit
USART_8N2	8 Data Bits, No Parity,two Stop Bits
USART_8E2	8 Data Bits, Even Parity, 2 stop bits
USART_8O2	8 Data Bits, odd Parity, 2 stop bits
USART_9N1	9 Data Bits, No Parity, 1 stop bit
USART_9N2	9 Data Bits, No Parity, 2 stop bits

Description

Data Flow configuration

This data type defines the different configurations by which the USART can be configured for the data flow.

Remarks

None.

USART_MODULE_ID Enumeration

File

plib_usart_help.h

C

```
typedef enum {
  USART_ID_1,
  USART_ID_2,
  USART_ID_3,
  USART_ID_4,
  USART_ID_5,
  USART_ID_6,
  USART_NUMBER_OF_MODULES
} USART_MODULE_ID;
```

Members

Members	Description
USART_ID_1	USART 1
USART_ID_2	USART 2
USART_ID_3	USART 3
USART_ID_4	USART 4
USART_ID_5	USART 5
USART_ID_6	USART 6
USART_NUMBER_OF_MODULES	Total number of USART modules available

Description

Enumeration: USART_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

_USART_MODULE_ID Macro

File

plib_usart_help.h

C

```
#define _USART_MODULE_ID
```

USART_OPERATION_MODE Enumeration

Data type defining the different configurations by which the USART can be enabled.

File

plib_usart_help.h

C

```
typedef enum {
  USART_ENABLE_TX_RX_BCLK_USED,
  USART_ENABLE_TX_RX_CTS_RTS_USED,
  USART_ENABLE_TX_RX_RTS_USED,
  USART_ENABLE_TX_RX_USED
} USART_OPERATION_MODE;
```

Members

Members	Description
USART_ENABLE_TX_RX_BCLK_USED	TX, RX and BCLK pins are used by USART module
USART_ENABLE_TX_RX_CTS_RTS_USED	TX, RX, CTS and RTS pins are used by USART module
USART_ENABLE_TX_RX_RTS_USED	TX, RX and RTS pins are used by USART module
USART_ENABLE_TX_RX_USED	TX and RX pins are used by USART module

Description

Enable configuration

This data type defines the different configurations by which the USART can be enabled.

Remarks

The actual definition of this enumeration is device-specific.

USART_RECEIVE_INTR_MODE Enumeration

Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_RECEIVE_FIFO_HALF_FULL,
  USART_RECEIVE_FIFO_3B4FULL,
  USART_RECEIVE_FIFO_ONE_CHAR
} USART_RECEIVE_INTR_MODE;
```

Members

Members	Description
USART_RECEIVE_FIFO_HALF_FULL	Interrupt when receive buffer is half full
USART_RECEIVE_FIFO_3B4FULL	Interrupt when receive buffer is 3/4 full
USART_RECEIVE_FIFO_ONE_CHAR	Interrupt when a character is received

Description

Receive Interrupt mode configuration

This data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.

Remarks

None.

USART_TRANSMIT_INTR_MODE Enumeration

Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_TRANSMIT_FIFO_EMPTY,
  USART_TRANSMIT_FIFO_IDLE,
  USART_TRANSMIT_FIFO_NOT_FULL
} USART_TRANSMIT_INTR_MODE;
```

Members

Members	Description
USART_TRANSMIT_FIFO_EMPTY	Interrupt when the transmit buffer becomes empty
USART_TRANSMIT_FIFO_IDLE	Interrupt when all characters are transmitted
USART_TRANSMIT_FIFO_NOT_FULL	Interrupt when at least one location is empty in the transmit buffer

Description

Transmit Interrupt mode configuration

This data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Remarks

None.

Files

Files

Name	Description
plib_usart.h	USART Peripheral Library interface header.
plib_usart_help.h	

Description

This section lists the source and header files used by the library.

plib_usart.h

USART Peripheral Library interface header.

Functions

	Name	Description
	PLIB_USART_AddressGet	Gets the address for the Address Detect mode.
	PLIB_USART_AddressMaskGet	Gets the address mask for the Address Detect mode.
	PLIB_USART_AddressMaskSet	Sets the address mask for the Address Detect mode.
	PLIB_USART_AddressSet	Sets the address for the Address Detect mode.
	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
	PLIB_USART_BaudRateGet	Gets the baud rate current in use.
	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudSetAndEnable	Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.
	PLIB_USART_BRGClockSourceGet	Gets the BRG clock source of the USART module.
	PLIB_USART_BRGClockSourceSelect	Configures the BRG clock source of the USART module.
	PLIB_USART_Disable	Disables the specific USART module
	PLIB_USART_Enable	Enables the specific USART module.

	PLIB_USART_ErrorsGet	Return the status of all errors in the specified USART module.
	PLIB_USART_ExistsBaudRate	Identifies whether the BaudRate feature exists on the USART module.
	PLIB_USART_ExistsBaudRateAutoDetect	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsBaudRateHigh	Identifies whether the BaudRateHigh feature exists on the USART module.
	PLIB_USART_ExistsBRGClockSourceSelect	Identifies whether the BRG Clock source select feature exists on the USART module.
	PLIB_USART_ExistsEnable	Identifies whether the EnableControl feature exists on the USART module.
	PLIB_USART_ExistsHandshakeMode	Identifies whether the HandShakeMode feature exists on the USART module.
	PLIB_USART_ExistsIrDA	Identifies whether the IrDAControl feature exists on the USART module.
	PLIB_USART_ExistsLineControlMode	Identifies whether the LineControlMode feature exists on the USART module.
	PLIB_USART_ExistsLoopback	Identifies whether the Loopback feature exists on the USART module.
	PLIB_USART_ExistsModuleBusyStatus	Identifies whether the module running status feature exists on the USART module.
	PLIB_USART_ExistsOperationMode	Identifies whether the OperationMode feature exists on the USART module.
	PLIB_USART_ExistsReceiver	Identifies whether the Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiver9Bits	Identifies whether the 9 Bits Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddress	Identifies whether the Receiver Address feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressAutoDetect	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressDetect	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressMask	Identifies whether the Receiver Address Mask feature exists on the USART module.
	PLIB_USART_ExistsReceiverDataAvailableStatus	Identifies whether the ReceiverDataAvailable feature exists on the USART module
	PLIB_USART_ExistsReceiverEnable	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	PLIB_USART_ExistsReceiverFramingErrorStatus	Identifies whether the ReceiverFramingError feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStateLowEnable	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStatus	Identifies whether the ReceiverIdle feature exists on the USART module.
	PLIB_USART_ExistsReceiverInterruptMode	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsReceiverOverrunStatus	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	PLIB_USART_ExistsReceiverParityErrorStatus	Identifies whether the ReceiverParityError feature exists on the USART module.
	PLIB_USART_ExistsRunInOverflow	Identifies whether the Run in overflow condition feature exists on the USART module.
	PLIB_USART_ExistsRunInSleepMode	Identifies whether the Run in Sleep mode feature exists on the USART module.
	PLIB_USART_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USART module.
	PLIB_USART_ExistsTransmitter	Identifies whether the Transmitter feature exists on the USART module.
	PLIB_USART_ExistsTransmitter9BitsSend	Identifies whether the Transmitter9Bits feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBreak	Identifies whether the TransmitterBreak feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBufferFullStatus	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEmptyStatus	Identifies whether the TransmitterEmpty feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEnable	Identifies whether the TransmitterEnableControl feature exists on the USART module
	PLIB_USART_ExistsTransmitterIdleIsLow	Identifies whether the TransmitterIdleIsLow feature exists on the USART module.
	PLIB_USART_ExistsTransmitterInterruptMode	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsWakeOnStart	Identifies whether the WakeOnStart feature exists on the USART module.
	PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
	PLIB_USART_InitializeModeGeneral	Enables or disables general features of the USART module.
	PLIB_USART_InitializeOperation	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.

	PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
	PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
	PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
	PLIB_USART_LoopbackDisable	Disables Loopback mode.
	PLIB_USART_LoopbackEnable	Enables Loopback mode.
	PLIB_USART_ModuleIsBusy	Returns the USART module's running status.
	PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
	PLIB_USART_Receiver9BitsReceive	Data to be received in the byte mode with the 9th bit.
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressGet	Returns the address of the USART RX register
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART vverrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.
	PLIB_USART_RunInOverflowDisable	Disables the Run in overflow condition mode.
	PLIB_USART_RunInOverflowEnable	Enables the USART module to continue to operate when an overflow error condition has occurred.
	PLIB_USART_RunInOverflowIsEnabled	Gets the status of the Run in Overflow condition.
	PLIB_USART_RunInSleepModeDisable	Turns off the USART module's BRG clock during Sleep mode.
	PLIB_USART_RunInSleepModeEnable	Allows the USART module's BRG clock to run when the device enters Sleep mode.
	PLIB_USART_RunInSleepModeIsEnabled	Gets the status of Run in Sleep mode.
	PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterAddressGet	Returns the address of the USART TX register
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Sets the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.

Description

USART Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USART

Peripheral Library for all families of Microchip microcontrollers. The functions in this file are common to the USART module.

File Name

plib_usart.h

Company

Microchip Technology Inc.

plib_usart_help.h

Enumerations

	Name	Description
	USART_HANDSHAKE_MODE	Lists the USART handshake modes.
	USART_LINECONTROL_MODE	Data type defining the different configurations by which the USART data flow can be configured.
	USART_MODULE_ID	Enumeration: USART_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	USART_OPERATION_MODE	Data type defining the different configurations by which the USART can be enabled.
	USART_RECEIVE_INTR_MODE	Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.
	USART_TRANSMIT_INTR_MODE	Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Macros

	Name	Description
	_USART_MODULE_ID	

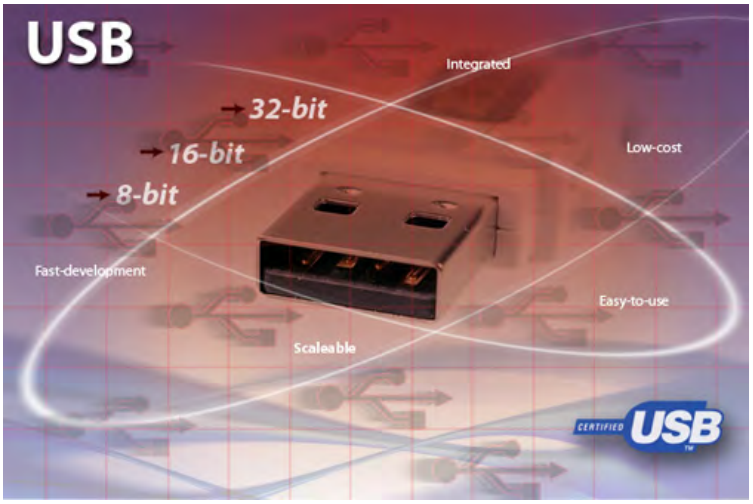
USB Peripheral Library

This section describes the USB Peripheral Library.

Introduction

This library provides a low-level abstraction of the USB module on Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description



USB Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host). There are three possible module implementations: host, device and OTG dual role. The user should have an understanding of the USB documents available on the USB implementers web site (www.usb.org).

Device

The USB device accepts data from the host and responds to requests for data. It performs some peripheral functions, such as a mouse or data storage device.

- Functionality may be class or vendor-specific
- Draws 100 mA or less before configuration
- Can draw up to 500 mA after successful negotiation with the host
- Can support low-speed, full-speed or high-speed protocol. Hi-speed support requires implementation of full-speed. (*Low, Full, and Hi-Speed supported by this module.*)
- Supports control transfers. Supports data transfers required for implementation
- Optionally supports Session Request Protocol (SRP)
- Can be bus-powered or self-powered

Host

The host is the master in a USB system and is responsible for identifying all devices connected to it (enumeration), initiating all transfers, allocating bus bandwidth and supplying power to any bus-powered USB devices connected directly to it. There are two types of hosts.

USB Standard Host:

- A large variety of devices are supported
- This host supports all USB transfer types
- USB hubs are supported to allow connection of multiple devices simultaneously
- Device drivers can be updated to support new devices
- A type ‘A’ receptacle is used for each port
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured
- device, and optionally, up to 500 mA for a configured device
- Full-Speed and Low-Speed must be supported. Hi-Speed can be supported.

- This is a typical personal computer implementation

Embedded Host

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL)
- This type of host is only required to support transfer types required by devices in the TPL
- USB hub support is optional (*Provided in this module*)
- Device drivers are not required to be pocketable
- A type ‘A’ receptacle is used for each port
- Only speeds required by devices in the TLP must be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- This is a typical implementation for a microcontroller

USB On-the-Go (OTG)

The OTG dual role device supports both USB host and device functionality. OTG dual role devices use a micro-AB receptacle. This allows a micro-A or a micro-B plug to be attached. Both the micro-A and micro-B plugs have an additional pin, the ID pin, to signify the connection type. The plug type, micro-A or micro-B, determines the default role of the OTG device, host or USB device. An OTG device will perform the role of a host when a micro-A plug is detected. When a micro-B plug is detected, the role of a USB device is performed.

When an OTG device is directly connected to another OTG device using an OTG cable, micro-A to micro-B, Host Negotiation Protocol (HNP) can be used to swap the roles of the host and USB device between the two without disconnecting and reconnecting cabling. To differentiate between the two OTG devices, the term, "A-device", is used to refer to the device connected to the micro-A plug and "B-device" is used to refer to the OTG device connected to the micro-B plug.

OTG dual role operating as a host (A-device):

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL). Generic class support is not allowed.
- Only required to support transaction types required by devices in the TPL
- USB hub support is optional. (*Multi-point support provided by this module.*)
- Device drivers are not required to be pocketable
- Only a single micro-AB receptacle is used
- Only Full-Speed must be supported. Hi-Speed and/or Low-Speed can be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- The USB port must be able to deliver a minimum of 8 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- Supports Host Negotiation Protocol (HNP) and Session Request Protocol (SRP). The host can switch roles to become a device. The initial role as a host or device is determined by the plug type, micro-A or micro-B, inserted into the micro-AB receptacle
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

OTG dual role operating as a USB device (B-device):

- Class or vendor-specific functionality
- Draws 8 mA or less before configuration
- Is typically self-powered due to low-current requirements, but can draw up to 500 mA after successful negotiation with the host
- Only a single micro-AB receptacle is used
- Must support Full-Speed. Support of Low-Speed and/or Hi-Speed is optional
- Supports control transactions. Supports data transactions required for implementation.
- Supports Session Request Protocol (SRP) and/or Host Negotiation Protocol (HNP). (*This module supports both SRP and HNP.*)
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

Additional Features of the Hi-Speed USB Module

- Operates either as a function controller of a Hi-Speed/Full-Speed USB device or as the host/device in a point-to-point or multi-point communications with other USB function
- Supports OTG communications with on or more Hi-Speed, Full-Speed, or Low-Speed devices
- Provides soft connect/disconnect.
- In addition to Endpoint Zero, supports seven transmit and seven receive endpoints
- Dynamic FIFO sizing for Endpoints 1-7. (Endpoint Zero FIFO fixed at 64 bytes.) FIFOs use module-internal SRAM.
- Module-internal eight channel DMA with access to all FIFOs
- All host transaction scheduling supported in hardware
- Supports Link Power Management

Using the Library

This topic describes the basic architecture of the USB Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usb.h`

The interface to the USB Peripheral Library is defined in the `plib_usb.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USB Peripheral Library must include `peripheral.h`.

Library File:

The USB Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Models

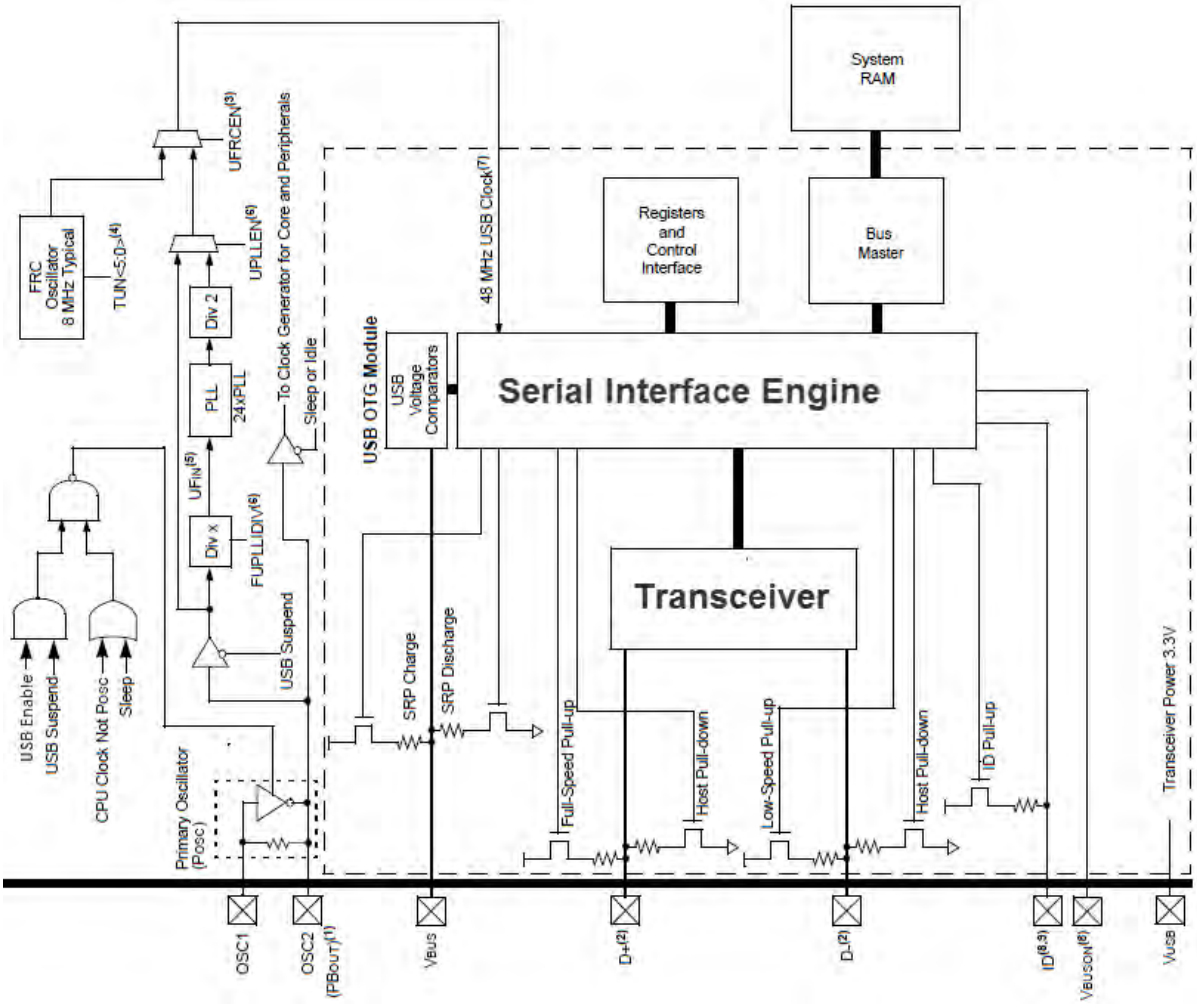
This section describes the hardware abstraction model for the USB Peripheral Library.

Description

Hardware Abstraction Model

The following figure shows the USB Module for the PIC32 family of devices.

PIC32 USB Module



Note 1: PB clock is only available on this pin for select EC modes.
2: Pins can be used as digital inputs when USB is not enabled.
3: This bit field is contained in the OSCCON register.
4: This bit field is contained in the OSCTRM register.
5: USB PLL UF_{IN} requirements: 4 MHz $\leq UF_{IN} \leq$ 5 MHz.
6: This bit field is contained in the DEVCFG2 register.
7: A 48 MHz clock is required for proper USB operation.
8: Pins can be used as GPIO when the USB OTG module is disabled.
9: Pin is pulled high internally when USB OTG module is enabled.

The USB OTG module contains analog and digital components to provide a USB 2.0 full-speed and low-speed embedded host, full-speed device, or On-The-Go (OTG) implementation with a minimum of external components. This module in Host mode is intended for use as an embedded host and therefore does not implement a Universal Host Controller Interface (UHCI) or a Open Host Controller Interface (OHCI).

The USB OTG module consists of the clock generator, the USB voltage comparators, the transceiver, the Serial Interface Engine (SIE), a dedicated USB Bus Master, pull-up and pull-down resistors and the register interface.

The clock generator provides the 48 MHz clock, which is required for USB full-speed and low-speed communication. The voltage comparators monitor the voltage on the VBUS pin to determine the state of the bus. The transceiver provides the analog translation between the USB bus and the digital logic. The SIE is a state machine that transfers data to and from the endpoint buffers, and generates the hardware protocol for data transfers. The USB Bus Master transfers data between the data buffers in RAM and the SIE. The integrated pull-up and pull-down resistors eliminate the need for external signaling components. The register interface allows the CPU to configure and communicate with the module.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USB module.

Library Interface Section	Description
USB Setup Functions	This section provides functions to perform general USB peripheral setup such as functions to set USB Speed, control on-chip pull ups, etc.
Buffer Descriptor Table Functions	This section provides functions that allow the application to setup, configure and access/modify the Buffer Descriptors in the Buffer Descriptor Table.
USB Activity Functions	This section provides function that allow the application to monitor bus conditions on the USB.
USB Bus Signaling Functions	This section provides functions that allow the application to generate Reset and Resume Signaling.
Last Transaction Status Functions	This section provides functions that allow the application to query conditions of the USB peripheral.
Endpoints Functions	This section provides functions that allow the application to manage endpoints.
Interrupts Functions	This section provides functions that allow the application to enable, disable and query the status interrupts in the USB peripheral.
Host Functions	This section provides functions that are required to operate the USB module while in Host mode.
On-The-Go (OTG) Functions	This section provides functions that are required to operate the USB module while in OTG mode.
External Transceiver Support Functions	This section provides function that are required to operate to enable/disable the external transceiver interface.

VBUS Support Functions	This section provides functions that allow VBUS level monitoring and VBUS boost PWM module control.
Test Support Functions	This section provides functions that enable/disable test signals needed for eye pattern measurement.

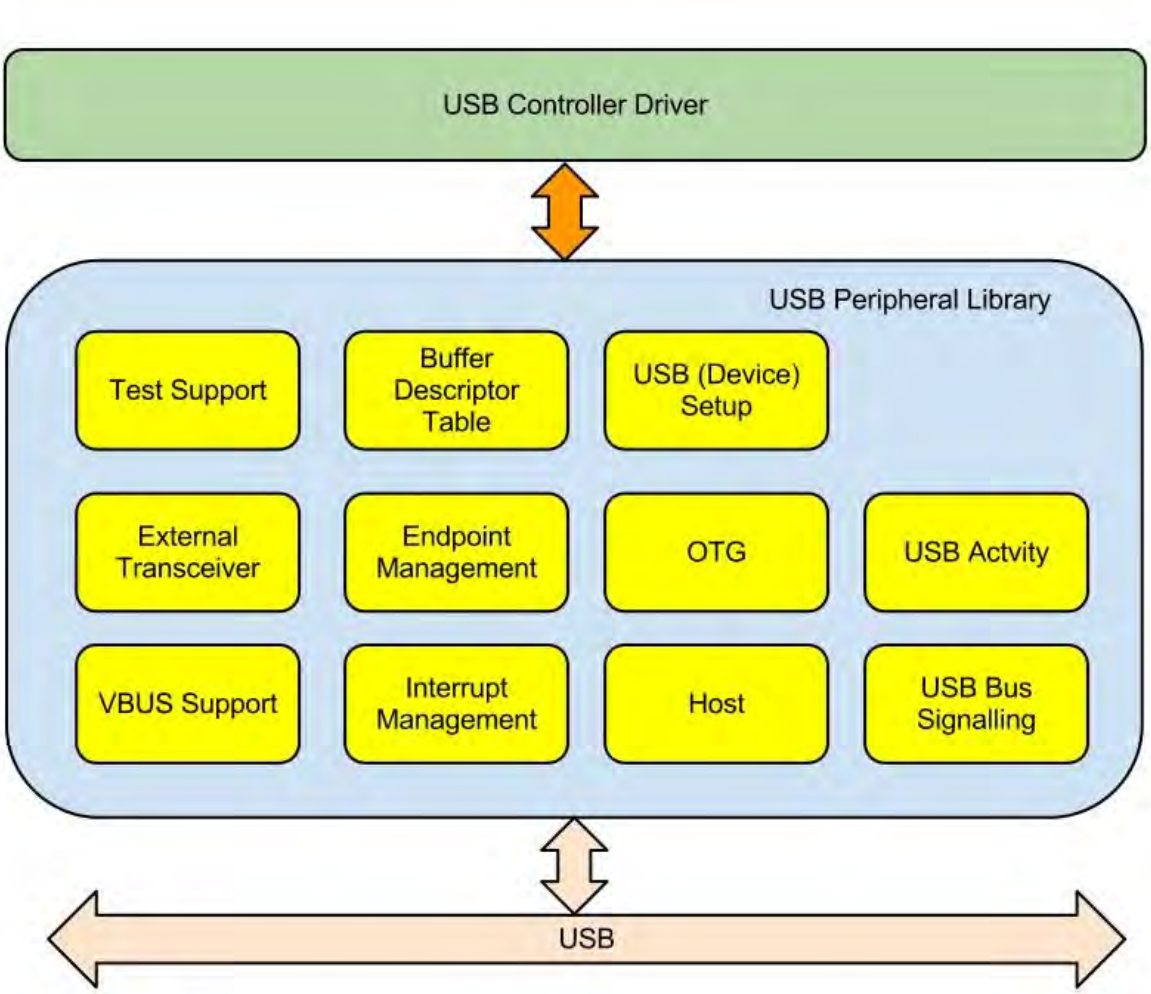
How the Library Works

Provides information on how the library works.

Description

The following figure describes the abstraction model employed by the USB Peripheral Library.

USB Peripheral Library Abstraction Model



USB Buffers and the Buffer Descriptor Table (BDT)

This section describes USB buffers and the Buffer Descriptor Table.

Description

All USB endpoints are implemented using buffers and control bits in RAM. Both software and the USB module have access to these buffers and control bits. To arbitrate this access a semaphore flag system is used.

Each endpoint can be configured for transmit only, receive only or transmit *and* receive. Transmit and receive functions have separate buffers. For each buffer there is a Buffer Descriptor (BD) in the Buffer Descriptor Table (BDT). On most devices the BDT has room for two transmit and two receive buffers for each endpoint, supporting ping-pong buffering. Buffer descriptors must be aligned on 512 byte boundaries to enable the USB module to correctly read each entry.

The starting address of the Buffer Descriptor Table is set by the PLIB_USB_BDTBaseAddressSet.

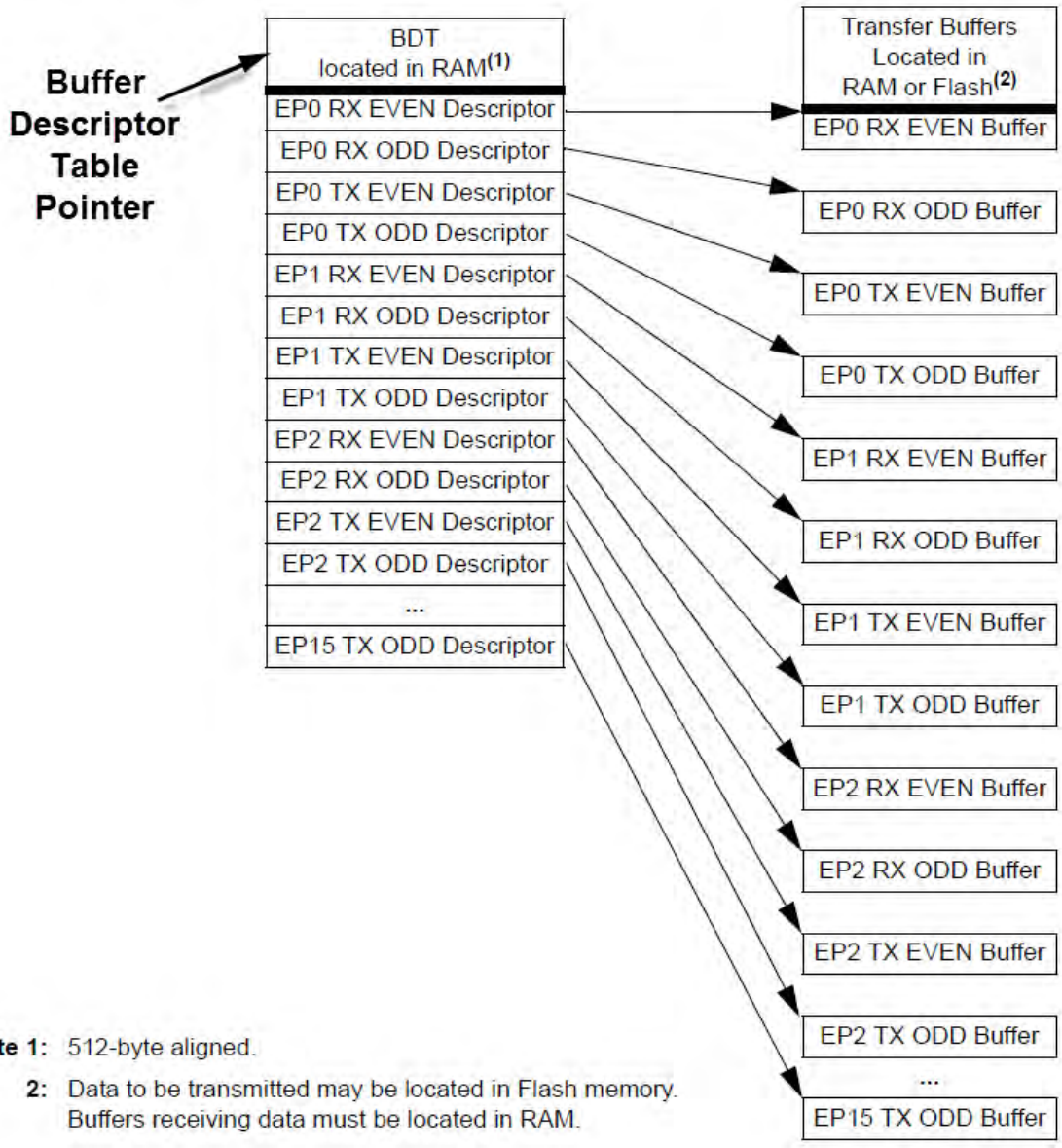
```
#define USB_MAX_EP_NUMBER 15
```

```
#define BDT_NUM_ENTRIES      (((USB_MAX_EP_NUMBER + 1) * 4)-2)

volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__ (( aligned (512) ));
```

Each Buffer Descriptor entry in the BDT has a **bufferAddress** pointer to the Buffer Descriptor's buffer in memory (RAM or FLASH).

The following figure shows the structure for PIC32 devices. These devices support ping-pong buffering for all endpoints.

Buffer Descriptor Table and Endpoint Buffers



With 16 endpoints (Endpoints 0 - 15) there are 64 possible entries in the Buffer Descriptor Table:

```
//                                          (Even,Odd) or (Ping,Pong)
//                                             (RX,TX) |
//                            (EP0      to       EPn)  |  |
volatile USB_BDT_ENTRY myBDT[(USB_MAX_EP_NUMBER + 1)][2][2] __attribute__ (( aligned (512) ));
```

Peripheral library functions are provided to support reading each BD while hiding the details of BDT addressing that can change when ping-pong buffering is disabled for one or more endpoints. The format of Buffer Descriptors vary from family to family and it is not important to know the differences, since peripheral library functions hide these details. To understand the information stored in each Buffer Descriptor Table entry it is informative to look at the C language description for one family of devices. (All device families support the same information but with different ways of packing the data into memory.)

For PIC32 devices each BD is 8 bytes long and is described by this C code:

```
    typedef union _USB_BDT_ENTRY __attribute__ ((packed))
    {
        uint64_t          dlValue;        //Double Long Integer Value
        uint32_t           lValue[2];     //Long  Integer Values
        uint16_t           sValue[4];     //Short Integer Values
```

```
        struct __attribute__ ((packed))
        {
            USB_BD_STATUS  bufferStatus;  //(RW) Buffer Status
            uint16_t       byteCount:10;  //(RW) Byte Count
            uint8_t                : 6;  //(  ) Reserved
            uint32_t       bufferAddress; //(RW) Buffer Address in Data Ram or Flash
        };
    } USB_BDT_ENTRY;
```

For transmit buffers, byteCount, is set to the length of the buffer using PLIB_USB_BufferByteCountSet. For receive buffers PLIB_USB_BufferByteCountSet is used to set the maximum allowable receive data length. PLIB_USB_BufferByteCountGet is used to determine how many bytes were actually transmitted or received. The memory address of each buffer is found in bufferAddress. The functions PLIB_USB_BufferAddressSet and PLIB_USB_BufferAddressGet support this field in the Buffer Descriptor.

Buffer status is stored in the 16 least significant bits of each BDT entry:

```
    typedef union _USB_BD_STATUS __attribute__ ((packed))
    {
        uint16_t       sValue;                  //Short Integer Value
        struct __attribute__ ((packed))
        {
            uint8_t                          :2; //(  )Reserved
            uint8_t    stallEnable           :1; //( W) Buffer Stall Enable
            uint8_t    dataToggleSyncEnable :1; //( W) Data Toggle Synch Enable
            uint8_t                          :1; //(  ) Reserved
            uint8_t                          :1; //(  ) Reserved
            uint8_t    dataToggle            :1; //(RW) Data Toggle Synch Value
            uint8_t    uSBOwnsBuffer         :1; //(RW) USB Ownership of buffer and BDT entry
        };
        struct __attribute__ ((packed))
        {
            uint8_t                          :2; //(  ) Reserved
            uint8_t    packetID              :4; //(R ) Packet Identifier (PID)
        };
    } USB_BD_STATUS;
```

The fields stallEnable and dataToggleSyncEnable in the Buffer Descriptor status structure are writeable but not readable. The function PLIB_USB_BufferStallEnable sets the stallEnable bit. (Hardware will clear the bit after receiving a SETUP token from the host.) The functions PLIB_USB_BufferDataToggleSyncEnable and PLIB_USB_BufferDataToggleSyncDisable manipulate dataToggleSyncEnable .

On a read from the status structure stallEnable and dataToggleSyncEnable and the next two (reserved) bits are replaced by the Packet Identifier (PID), called packetID in the structure. The function PLIB_USB_BufferPIDGet reads packetID.

The rest of the status structure is writeable and readable: dataToggle and uSBOwnsBuffer.

The dataToggle (DATA0 or DATA1) of a receive buffer is determined by using PLIB_USB_BufferDataToggleGet. For transmit buffers the dataToggle value is set using PLIB_USB_BufferDataToggleSelect .

The field uSBOwnsBuffer is used by both software and the USB module as a semaphore. The function PLIB_USB_BufferReleaseToUSB releases the buffer to the USB module and the function PLIB_USB_BufferReleasedToSW returns a Boolean true when the buffer is released back to software by the USB module. As long as PLIB_USB_BufferReleasedToSW returns false, software should not change the Buffer Descriptor Table entry or its associated buffer.

USB Setup Example

This section provides example code for setting up the USB module.

Description

Following is an example of how to set up the USB module:

```
#define USB_MAX_EP_NUMBER 15

#define BDT_NUM_ENTRIES      (((USB_MAX_EP_NUMBER + 1) * 4)-2)

volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__ (( aligned (512) ));

unsigned int bufferTransactionCount[BDT_NUM_ENTRIES];

    // Turn off USB module
    PLIB_USB_Disable( usbID );

    // Set up the Hardware
    if ( usbModuleSetup.StopInIdle )
    {
        PLIB_USB_StopInIdleEnable( usbID );
```

```
    }
    else
    {
        PLIB_USB_StopInIdleDisable( usbID );
    }

    if ( usbModuleSetup.SuspendInSleep )
    {
        PLIB_USB_AutoSuspendEnable( usbID );
    }
    else
    {
        PLIB_USB_AutoSuspendDisable( usbID );
    }

    PLIB_USB_OperatingModeSelect( usbID, usbModuleSetup.OpMode );

    PLIB_USB_PingPongModeSelect( usbID, usbModuleSetup.ppMode );

    // Reset all ping pong buffers to "Even"
    PLIB_USB_PingPongFreeze( usbID );
    PLIB_USB_PingPongUnfreeze( usbID );

    // Interrupt flag cleared on the safer side
    IFS1bits.USBIF = 0;

    // Enable USB module interrupts
    PLIB_USB_InterruptEnable( usbID, DRV_USB_GEN_INT_ENABLES );

    // Disable all OTG interrupts
    PLIB_USB_OTG_InterruptDisable( usbID, DRV_USB_OTB_INT_ENABLES );

    // Enable all Error interrupts
    PLIB_USB_ErrorInterruptEnable( usbID, DRV_USB_ERR_INT_ENABLES );

    // Enable the interrupt source in case of interrupt mode
    IEC1bits.USBIE = 1;

    // Setting the Interrupt Priority in case of interrupt mode.
    IPC11bits.USBIP = 4;

    // Initialize BDT
    for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )
    {
        gBDT[iEntry].lValue[0] = 0ul;
        gBDT[iEntry].lValue[1] = 0ul;

        bufferTransactionCount[iEntry] = 0;
    }//end for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )

    // Inform USB module of BDT's address in memory
    PLIB_USB_BDTBaseAddressSet( usbID , (void *)((uint32_t)KVA_TO_PA(gBDT)) );

    /***********************/
    /* SET UP ENDPOINT ZERO */
    /***********************/
    PLIB_USB_BufferAddressSet( usbID, (void*)gBDT, PLIB_USB_PingPongModeGet( usbID ),
                               0, USB_BUFFER_RX, USB_BUFFER_EVEN, pSetupRcvBuffer );

    // Configure Endpoint Zero control register
    PLIB_USB_EP0LSDirectConnectDisable( usbID );   // For Hosts, included for completeness
    PLIB_USB_EP0NakRetryEnable( usbID );           // For Hosts, included for completeness
    PLIB_USB_EPnControlTransferEnable( usbID, 0 ); // Enable control transfers
    PLIB_USB_EPnRxSelect( usbID, 0, USB_EP_RX );   // Enable receive
    PLIB_USB_EPnHandshakeEnable( usbID, 0 );       // Enable handshaking

    // Set up Endpoint Zero's receive buffer BDT entries
    ppMode = PLIB_USB_PingPongModeGet( usbID ); // Ping Pong Mode needed for BDT entry indexing
```

```
    // Clear PID bits
    PLIB_USB_BufferPIDBitsClear(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Disable buffer stall
    PLIB_USB_BufferStallDisable(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Enable Data Toggle Synchronization
    PLIB_USB_BufferDataToggleSyncEnable(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Set Data0/1 to Data0
    PLIB_USB_BufferDataToggleSelect(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN,USB_BUFFER_DATA
0);
//Rx0

    // Setup packets have 8 bytes of data payload
    PLIB_USB_BufferByteCountSet(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN,8);

    // Release buffers to USB module
    PLIB_USB_BufferReleaseToUSB(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN);

    PLIB_USB_Enable( gDrvUSBObj[hDriver].usbID ); // Turn on USB module
```

Configuring the Library

The library is configured for the supported USB module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) USB Setup Functions

	Name	Description
	PLIB_USB_AllInterruptEnable	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	PLIB_USB_AutoSuspendDisable	Disables USB OTG Auto-suspend mode.
	PLIB_USB_AutoSuspendEnable	Enables USB Auto-suspend mode.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.
	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB_UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB_UOEMonitorEnable	Enables the OE signal output.

b) Buffer Descriptor Table Functions

	Name	Description
	PLIB_USB_BDTBaseAddressGet	Returns the base address of the Buffer Descriptor Table.
	PLIB_USB_BDTBaseAddressSet	Sets the base address for the Buffer Descriptor Table for PIC32 devices.
	PLIB_USB_BufferAddressGet	Gets the memory address of an endpoint buffer.
	PLIB_USB_BufferAddressSet	Sets the endpoint buffer address.

	PLIB_USB_BufferAllCancelReleaseToUSB	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	PLIB_USB_BufferByteCountGet	Returns the endpoint buffer byte count.
	PLIB_USB_BufferByteCountSet	Sets the buffer byte count.
	PLIB_USB_BufferCancelReleaseToUSB	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	PLIB_USB_BufferClearAll	Clears (zeros out) entries in the Buffer Descriptor Table.
	PLIB_USB_BufferClearAllDTSEnable	Clears the endpoint descriptor entry and enables data toggle synchronization.
	PLIB_USB_BufferDataToggleGet	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	PLIB_USB_BufferDataToggleSelect	Sets the endpoint buffer to DATA0 or DATA1.
	PLIB_USB_BufferDataToggleSyncDisable	Disables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferDataToggleSyncEnable	Enables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferEP0RxStatusInitialize	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	PLIB_USB_BufferIndexGet	Gets the Buffer Descriptor Table index for a buffer.
	PLIB_USB_BufferPIDBitsClear	Clears the Buffer Status bits in the Buffer Descriptor Table.
	PLIB_USB_BufferPIDGet	Returns the token packet ID (PID) from the endpoint buffer status.
	PLIB_USB_BufferReleasedToSW	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	PLIB_USB_BufferReleaseToUSB	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	PLIB_USB_BufferSchedule	Hands over a buffer to the USB module along with the buffer address and byte count.
	PLIB_USB_BufferStallDisable	Disables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallEnable	Enables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallGet	Returns the buffer stall status for an endpoint/direction/ping-pong.

c) Endpoints Functions

	Name	Description
	PLIB_USB_EP0HostSetup	Sends token to the specified address.
	PLIB_USB_EP0LSDirectConnectDisable	Disables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0LSDirectConnectEnable	Enables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0NakRetryDisable	Disables retrying of NAKed transactions.
	PLIB_USB_EP0NakRetryEnable	Enables retrying NAK'd transactions for Endpoint 0.
	PLIB_USB_EPnAttributesClear	Clears the set attributes of the specified endpoint.
	PLIB_USB_EPnAttributesSet	Configures attributes of the endpoint such as direction, handshake capability and direction.
	PLIB_USB_EPnControlTransferDisable	Disables endpoint control transfers.
	PLIB_USB_EPnControlTransferEnable	Enables endpoint control transfers.
	PLIB_USB_EPnDirectionDisable	Disables the specified endpoint direction.
	PLIB_USB_EPnHandshakeDisable	Disables endpoint handshaking.
	PLIB_USB_EPnHandshakeEnable	Enables endpoint handshaking.
	PLIB_USB_EPnIsStalled	Tests whether the endpoint epValue is stalled.
	PLIB_USB_EPnRxDisable	Disables an endpoint's ability to process IN tokens.
	PLIB_USB_EPnRxEnable	Enables an endpoint to process IN tokens.
	PLIB_USB_EPnRxSelect	Selects receive capabilities of an endpoint.
	PLIB_USB_EPnStallClear	Clears an endpoint's stalled flag.
	PLIB_USB_EPnTxDisable	Disables an endpoint's ability to process OUT tokens.
	PLIB_USB_EPnTxEnable	Enables an endpoint to process OUT tokens.
	PLIB_USB_EPnTxRxSelect	Selects transmit and/or receive capabilities of an endpoint.
	PLIB_USB_EPnTxSelect	Selects transmit capabilities of an endpoint.

d) Interrupts Functions

	Name	Description
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.

	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.

e) Error Interrupts Functions

	Name	Description
	PLIB_USB_ErrorInterruptDisable	Disables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptEnable	Enables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptFlagAllGet	Returns a logically ORed bit map of active error interrupt flags.
	PLIB_USB_ErrorInterruptFlagClear	Clears an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.

f) Last Transaction Status Functions

	Name	Description
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.

g) Host Functions

	Name	Description
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.
	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB_TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB_TokenEPSet	Sets the Endpoint address for a host transaction.
	PLIB_USB_TokenPIDGet	Returns the token transaction type.
	PLIB_USB_TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB_TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.

h) USB Bus Signaling Functions

	Name	Description
	PLIB_USB_ResetSignalDisable	Disables reset signaling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signaling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.

i) On-The-Go (OTG) Functions

	Name	Description
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.
	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.

	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.

j) OTG Interrupts Functions

	Name	Description
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.

k) USB Activity Functions

	Name	Description
	PLIB_USB_ActivityPending	Returns whether or not USB activity is pending.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_JStateIsActive	Live differential receiver J State flag.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.
	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.

l) External Transceiver Support Functions

	Name	Description
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB_TransceiverEnable	Enables the on-chip transceiver.

m) VBUS Support Functions

	Name	Description
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolaritiyActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB_VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB_VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB_VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB_VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB_VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

n) Test Support Functions

	Name	Description
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.

o) Other Functions

	Name	Description
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.

	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_TokenSend	Sends token to the specified address.

p) Feature Existence Functions

	Name	Description
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.
	PLIB_USB_ExistsModuleBusy	Identifies whether the ModuleBusy feature exists on the USB module.
	PLIB_USB_ExistsModulePower	Identifies whether the ModulePower feature exists on the USB module.
	PLIB_USB_ExistsNextTokenSpeed	Identifies whether the NextTokenSpeed feature exists on the USB module.
	PLIB_USB_ExistsOnChipPullup	Identifies whether the OnChipPullup feature exists on the USB module.
	PLIB_USB_ExistsOnChipTransceiver	Identifies whether the OnChipTransceiver feature exists on the USB module.
	PLIB_USB_ExistsOpModeSelect	Identifies whether the OpModeSelect feature exists on the USB module.
	PLIB_USB_ExistsOTG_ASessionValid	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_BSessionEnd	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	PLIB_USB_ExistsOTG_IDPinState	Identifies whether the OTG_IDPinState feature exists on the USB module.
	PLIB_USB_ExistsOTG_Interrupt	Identifies whether the OTG_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsOTG_InterruptStatus	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsOTG_LineState	Identifies whether the OTG_LineState feature exists on the USB module.
	PLIB_USB_ExistsOTG_PullUpPullDown	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	PLIB_USB_ExistsOTG_SessionValid	Identifies whether the OTG_SessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusCharge	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusDischarge	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusPowerOnOff	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	PLIB_USB_ExistsPacketTransfer	Identifies whether the PacketTransfer feature exists on the USB module.
	PLIB_USB_ExistsPingPongMode	Identifies whether the PingPongMode feature exists on the USB module.
	PLIB_USB_ExistsResumeSignaling	Identifies whether the ResumeSignaling feature exists on the USB module.
	PLIB_USB_ExistsSleepEntryGuard	Identifies whether the SleepEntryGuard feature exists on the USB module.
	PLIB_USB_ExistsSOFThreshold	Identifies whether the SOFThreshold feature exists on the USB module.
	PLIB_USB_ExistsSpeedControl	Identifies whether the SpeedControl feature exists on the USB module.

	PLIB_USB_ExistsStartOfFrames	Identifies whether the StartOfFrames feature exists on the USB module.
	PLIB_USB_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USB module.
	PLIB_USB_ExistsSuspend	Identifies whether the Suspend feature exists on the USB module.
	PLIB_USB_ExistsTokenEP	Identifies whether the TokenEP feature exists on the USB module.
	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.

q) Data Types and Constants

	Name	Description
	USB_BUFFER_DATA01	Provides enumeration data toggle for a buffer.
	USB_BUFFER_DIRECTION	Provides enumeration transmit/receive direction for a buffer.
	USB_BUFFER_PING_PONG	Enumerates the ping-pong buffer (Even vs. Odd).
	USB_BUFFER_SCHEDULE_DATA01	Provides enumeration data toggle for a buffer.
	USB_EP_TXRX	Provides enumeration transmit/receive setup for an endpoint.
	USB_OPMODES	Provides enumeration of operating modes supported by USB.
	USB_OTG_INTERRUPTS	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	USB_OTG_PULL_UP_PULL_DOWN	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	USB_PID	Legal PID values.
	USB_PING_PONG_MODE	Supports the four modes of ping-pong buffering.
	USB_PING_PONG_STATE	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	USB_TOKEN_SPEED	Provides enumeration of available token speeds.
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

This section describes the Application Programming Interface (API) functions of the USB Peripheral Library.

Refer to each section for a detailed description.

a) USB Setup Functions

PLIB_USB_AllInterruptEnable Function

Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.

File

plib_usb.h

C

```
void PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
USB_ERROR_INTERRUPTS usbErrorInterruptsFlag, USB_OTG_INTERRUPTS otgInterruptFlag);
```

Returns

None.

Description

This function configures the USB peripheral general interrupts, error interrupts and OTG interrupts.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsALL_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// This code snippet disables all OTG interrupts, disables
// the SOF interrupt and enables all error interrupts.
USB_OTG_INTERRUPTS otgInterruptEnables = ~USB_OTG_INT_ALL ;
USB_INTERRUPTS generalInterruptEnables = USB_INT_ALL & ~USB_INT_SOF ;
```

```
USB_ERROR_INTERRUPTS errorInterruptEnables = USB_ERR_INT_ALL ;

PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
    USB_ERROR_INTERRUPTS usbErrorInterruptsFlag,
    USB_OTG_INTERRUPTS otgInterruptFlag);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
usbInterruptsFlag	General interrupts to be configured
usbErrorInterruptsFlag	USB Error interrupts to be configured
otgInterruptFlag	OTG interrupts to be configured

Function

void PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
USB_ERROR_INTERRUPTS usbErrorInterruptsFlag, USB_OTG_INTERRUPTS otgInterruptFlag);

PLIB_USB_AutoSuspendDisable Function

Disables USB OTG Auto-suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_AutoSuspendDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables USB OTG Auto-suspend mode. The USB OTG module will operate normally and does not automatically suspend upon entry to Sleep mode. Software must use PLIB_USB_SuspendEnable to suspend the module, including the USB 48 MHz clock

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsAutomaticSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_AutoSuspendDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_AutoSuspendDisable (USB_MODULE_ID index)

PLIB_USB_AutoSuspendEnable Function

Enables USB Auto-suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_AutoSuspendEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB Auto-suspend mode. The USB module automatically suspends upon entry to Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsAutomaticSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_AutoSuspendEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_AutoSuspendEnable (USB_MODULE_ID index)

PLIB_USB_DeviceAddressGet Function

Returns the address of the USB module in Device mode.

File

plib_usb.h

C

```
uint8_t PLIB_USB_DeviceAddressGet(USB_MODULE_ID index);
```

Returns

Device Address

Description

This function returns the address of the USB module in Device mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsDeviceAddressin your application to determine whether this feature is available.

Preconditions

None.

Example

```
myUSBAddress = PLIB_USB_DeviceAddressGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_DeviceAddressGet (USB_MODULE_ID index)

PLIB_USB_DeviceAddressSet Function

Sets the USB Device's address.

File

plib_usb.h

C

```
void PLIB_USB_DeviceAddressSet(USB_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the USB Device's address as part of enumeration.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsDeviceAddressin your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
uint8_t myUSBAddress = ....;
PLIB_USB_DeviceAddressSet(MY_USB_INSTANCE, myUSBAddress);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	USB address

Function

void PLIB_USB_DeviceAddressSet (USB_MODULE_ID index, uint8_t address)

PLIB_USB_Disable Function

Disables (powers down) the USB module.

File

plib_usb.h

C

```
void PLIB_USB_Disable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables (powers down) the USB module.

Remarks

For PIC32 devices, the USB module must be in Device mode before the USB module is powered down.

For PIC32 devices, all reads or writes to module registers after powering down the module will be invalid until PLIB_USB_ModuleIsBusy (MY_USB_INSTANCE) == false.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModulePower in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#if defined(__PIC32MX__)
  // Disable Host, Device, or OTG before powering down
  PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_NONE );
```

```
    // Turn off USB
    PLIB_USB_Disable(MY_USB_INSTANCE);
    // For PIC32, wait until module is no longer busy before trying to
    // access any USB module registers.
    while ( PLIB_USB_ModuleIsBusy (MY_USB_INSTANCE) )
    {
        //wait
    }
#endif
    // Can now read or modify USB module status
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_Disable (USB_MODULE_ID index)

PLIB_USB_Enable Function

Enables (powers up) the USB module.

File

plib_usb.h

C

```
void PLIB_USB_Enable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables (powers up) the USB module.

Remarks

See also PLIB_USB_ModuleIsBusy.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModulePoweryour application to determine whether this feature is available.

Preconditions

None.

Example

```
    // Complete Needed setup for the module
    PLIB_USB_Enable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_Enable (USB_MODULE_ID index)

PLIB_USB_FullSpeedDisable Function

Forces the USB module to operate at low speed.

File

plib_usb.h

C

```
void PLIB_USB_FullSpeedDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function forces the USB module to operate at low speed.

Remarks

For PIC32 devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSpeedControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_FullSpeedDisable (USB_MODULE_ID index)

PLIB_USB_FullSpeedEnable Function

Enables the USB to operate at full speed.

File

plib_usb.h

C

```
void PLIB_USB_FullSpeedEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB to operate at full speed.

Remarks

For PIC32 devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSpeedControl in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_FullSpeedEnable (USB_MODULE_ID index)

PLIB_USB_OnChipPullUpDisable Function

Disables on-chip pull-ups.

File

plib_usb.h

C

```
void PLIB_USB_OnChipPullUpDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables on-chip pull-ups.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipPullup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OnChipPullUpDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OnChipPullUpDisable (USB_MODULE_ID index)

PLIB_USB_OnChipPullUpEnable Function

Enables on-chip pull-ups.

File

plib_usb.h

C

```
void PLIB_USB_OnChipPullUpEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables on-chip pull-ups. Pull-up on D+ in Full-Speed mode. Pull-up on D- in Low-Speed mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipPullup in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_OnChipPullUpEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OnChipPullUpEnable (USB_MODULE_ID index)

PLIB_USB_OperatingModeSelect Function

Selects the operating mode of the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OperatingModeSelect(USB_MODULE_ID index, USB_OPMODES opMode);
```

Returns

None.

Description

This function selects the operating mode of the USB module, either Host, Device, or OTG.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOpModeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_DEVICE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
opMode	Selected operating mode: USB_OPMODE_DEVICE, USB_OPMODE_HOST, or USB_OPMODE_OTG

Function

void PLIB_USB_OperatingModeSelect(USB_MODULE_ID index, USB_OPMODES opMode)

PLIB_USB_PingPongModeGet Function

Returns the Ping-Pong Configuration setting.

File

plib_usb.h

C

```
USB_PING_PONG_MODE PLIB_USB_PingPongModeGet(USB_MODULE_ID index);
```

Returns

Ping-Pong Mode - One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Description

This function returns the Ping-Pong Configuration setting.

Remarks

None.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPingPongMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
ppConfig = PLIB_USB_PingPongModeGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PING_PONG_MODE PLIB_USB_PingPongModeGet (USB_MODULE_ID index)

PLIB_USB_PingPongModeSelect Function

Selects the Ping-Pong Configuration setting.

File

plib_usb.h

C

```
void PLIB_USB_PingPongModeSelect(USB_MODULE_ID index, USB_PING_PONG_MODE ppConfig);
```

Returns

None.

Description

This function selects the Ping-Pong Configuration setting.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPingPongMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_PingPongModeSelect(MY_USB_INSTANCE,USB_PING_PONG__ALL_BUT_EP0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
ppConfig	Ping-Pong configuration selection. One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Function

void PLIB_USB_PingPongModeSelect (USB_MODULE_ID index, USB_PING_PONG_MODE ppConfig)

PLIB_USB_SleepGuardDisable Function

File

plib_usb.h

C

```
void PLIB_USB_SleepGuardDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables Sleep Guard. Entry into Sleep mode is immediate.

Remarks

Not available on all PIC32 devices. Refer to the specific device data sheet for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSleepEntryGuard in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SleepGuardDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SleepGuardDisable (USB_MODULE_ID index)

Summary.

Disables Sleep Guard. Entry into Sleep mode is immediate.

PLIB_USB_SleepGuardEnable Function

Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.

File

plib_usb.h

C

```
void PLIB_USB_SleepGuardEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function block entry into Sleep mode if bus activity is detected or if an interrupt is pending.

Remarks

Not available on all PIC32 devices. Refer to the specific device data sheet for details. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSleepEntryGuard in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SleepGuardEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SleepGuardEnable (USB_MODULE_ID index)

PLIB_USB_StopInIdleDisable Function

Allows the USB module to continue operation when the device enters Idle mode.

File

plib_usb.h

C

```
void PLIB_USB_StopInIdleDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function allows the USB module to continue operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_StopInIdleDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_StopInIdleDisable (USB_MODULE_ID index)

PLIB_USB_StopInIdleEnable Function

Enables USB module operation to stop when the device enters Idle mode.

File

plib_usb.h

C

```
void PLIB_USB_StopInIdleEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB module operation to stop when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_StopInIdleEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_StopInIdleEnable (USB_MODULE_ID index)

PLIB_USB_SuspendDisable Function

Disables USB OTG Suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_SuspendDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables USB OTG Suspend mode. The USB OTG module will operate normally.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SuspendDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SuspendDisable (USB_MODULE_ID index)

PLIB_USB_SuspendEnable Function

Enables USB Suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_SuspendEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB Suspend mode. The 48 MHz USB clock will be gated off. The transceiver is placed in a low-power state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SuspendEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SuspendEnable (USB_MODULE_ID index)

PLIB_USB_UOEMonitorDisable Function

Disables the OE signal output.

File

plib_usb.h

C

```
void PLIB_USB_UOEMonitorDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the OE signal output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsUOEMonitor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Disable the OE output.
PLIB_USB_UOEMonitorDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_UOEMonitorDisable(USB_MODULE_ID index);

PLIB_USB_UOEMonitorEnable Function

Enables the OE signal output.

File

plib_usb.h

C

```
void PLIB_USB_UOEMonitorEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the OE signal output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsUOEMonitor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Enable the OE output.
PLIB_USB_UOEMonitorEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_UOEMonitorEnable(USB_MODULE_ID index);

b) Buffer Descriptor Table Functions

PLIB_USB_BDTBaseAddressGet Function

Returns the base address of the Buffer Descriptor Table.

File

plib_usb.h

C

```
void* PLIB_USB_BDTBaseAddressGet(USB_MODULE_ID index);
```

Returns

None.

Description

This function returns the base address of the Buffer Descriptor Table.

Remarks

Must be set for PIC32 devices using PLIB_USB_BDTBaseAddressSet.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTBaseAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
void * pMyBDT;
pMyBDT = PLIB_USB_BDTBaseAddressGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void * PLIB_USB_BDTBaseAddressGet (USB_MODULE_ID index)

PLIB_USB_BDTBaseAddressSet Function

Sets the base address for the Buffer Descriptor Table for PIC32 devices.

File

plib_usb.h

C

```
void PLIB_USB_BDTBaseAddressSet(USB_MODULE_ID index, void* address);
```

Returns

None.

Description

This function sets the base address for the Buffer Descriptor Table. This function is only available on PIC32 devices.

Remarks

The address of the Buffer Descriptor Table must be 512 byte-aligned. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTBaseAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#if defined(__PIC32MX__)
  // For PIC32
  PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)((uint32_t)KVA_TO_PA(&myBDT)) );
#else
  // Everybody else
  PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)(&myBDT) );
#endif
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	Physical memory address in RAM of Buffer Descriptor Table

Function

void PLIB_USB_BDTBaseAddressSet (USB_MODULE_ID index, void * address)

PLIB_USB_BufferAddressGet Function

Gets the memory address of an endpoint buffer.

File

plib_usb.h

C

```
void* PLIB_USB_BufferAddressGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Buffer address in memory.

Description

This function gets the memory address of an endpoint buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint Value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void * PLIB_USB_BufferAddressGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferAddressSet Function

Sets the endpoint buffer address.

File

plib_usb.h

C

```
void PLIB_USB_BufferAddressSet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, void* bufferAddress);
```

Returns

None.

Description

This function sets the endpoint buffer address.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	address in memory of endpoint transmit or receive buffer

Function

void PLIB_USB_BufferAddressSet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
void * bufferAddress)

PLIB_USB_BufferAllCancelReleaseToUSB Function

Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.

File

plib_usb.h

C

```
void PLIB_USB_BufferAllCancelReleaseToUSB(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, int nEndpoints);
```

Returns

None.

Description

This function cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Cancel all buffer releases to USB.
//BDT has 3 Endpoints.

PLIB_USB_BufferAllCancelReleaseToUSB(MY_USB_INSTANCE, pBDT,
                                USB_PING_PONG_NO_PING_PONG, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to the Buffer Descriptor Table
ppMode	Buffer Descriptor Table Ping-Pong mode
nEndpoints	Number of endpoints in the Buffer-Descriptor table

Function

void PLIB_USB_BufferAllCancelReleaseToUSB(USB_MODULE_ID index,
void * pBDT, USB_PING_PONG_MODE ppMode, int nEndpoints);

PLIB_USB_BufferByteCountGet Function

Returns the endpoint buffer byte count.

File

plib_usb.h

C

```
uint16_t PLIB_USB_BufferByteCountGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Endpoint buffer byte count.

Description

This function returns the endpoint buffer byte count, the actual number of bytes transmitted or received.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint16_t PLIB_USB_BufferByteCountGet (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferByteCountSet Function

Sets the buffer byte count.

File

plib_usb.h

C

```
void PLIB_USB_BufferByteCountSet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, uint16_t
bufferByteCount);
```

Returns

None.

Description

This function sets the number of bytes to be transmitted or the maximum number of bytes to be received.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferByteCount	number of bytes to be transmitted or the maximum number of bytes to be received

Function

PLIB_USB_BufferByteCountSet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
uint16_t bufferByteCount)

PLIB_USB_BufferCancelReleaseToUSB Function

Cancels release of the endpoint buffer by software, allowing software to again access the buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferCancelReleaseToUSB(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function cancels the release of the endpoint buffer by software, allowing software to again access the buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferCancelReleaseToUSB (USB_MODULE_ID index,
void * pBDT,

USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferClearAll Function

Clears (zeros out) entries in the Buffer Descriptor Table.

File

plib_usb.h

C

```
void PLIB_USB_BufferClearAll(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function clears (zeros out) the entries in the Buffer Descriptor Table.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferClearAll(USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferClearAllDTSEnable Function

Clears the endpoint descriptor entry and enables data toggle synchronization.

File

plib_usb.h

C

```
void PLIB_USB_BufferClearAllDTSEnable(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection);
```

Returns

None.

Description

This function clears the endpoint descriptor entry and enables data toggle synchronization.

Remarks

None.

Preconditions

None.

Example

```
//Clear endpoint 6 buffer descriptor transmit entry and
//enable data toggle synchronization.

PLIB_USB_BufferClearAllDTSEnable ( MY_USB_INSTANCE, pBDT,
    USB_PING_PONG_NO_PING_PONG, 6, USB_BUFFER_TX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode.
epvalue	Endpoint to be be affected
bufferDirection	Endpoint direction

Function

void PLIB_USB_BufferClearAllDTSEnable(USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection);

PLIB_USB_BufferDataToggleGet Function

Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.

File

plib_usb.h

C

```
USB_BUFFER_DATA01 PLIB_USB_BufferDataToggleGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Data Toggle value, USB_BUFFER_DATA0 or USB_BUFFER_DATA1, for the buffer

Description

This function returns data synchronization (DATA0 or DATA1) for the endpoint buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

USB_BUFFER_DATA01

PLIB_USB_BufferDataToggleGet (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferDataToggleSelect Function

Sets the endpoint buffer to DATA0 or DATA1.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSelect(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, USB_BUFFER_DATA01
bufferData01);
```

Returns

None.

Description

This function sets the endpoint buffer to DATA0 or DATA1.

Remarks

See PLIB_USB_BufferDataToggleGet to determine the received data toggle setting. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferData01	USB_BUFFER_DATA0 or USB_BUFFER_DATA1

Function

void PLIB_USB_BufferDataToggleSelect (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
USB_BUFFER_DATA01 bufferData01)

PLIB_USB_BufferDataToggleSyncDisable Function

Disables DATA0/DATA1 synchronization between the device and host.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSyncDisable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function disables DATA0/DATA1 synchronization between the device and host.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferDataToggleSyncDisable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferDataToggleSyncEnable Function

Enables DATA0/DATA1 synchronization between the device and host.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSyncEnable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function enables DATA0/DATA1 synchronization between the device and host.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferDataToggleSyncEnable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferEP0RxStatusInitialize Function

Initializes the Endpoint 0 RX endpoint buffer descriptors.

File

plib_usb.h

C

```
void PLIB_USB_BufferEP0RxStatusInitialize(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
USB_BUFFER_PING_PONG pingpong, uint16_t bufferByteCount);
```

Returns

None.

Description

This function initializes the Endpoint 0 RX endpoint buffer descriptors. This function will clear the Endpoint 0 RX Buffer Descriptor status field, load the endpoint size and release the buffer to the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Initialize EP0 RX even buffer descriptor and release back to
//USB. Buffer size is 64
```

```
PLIB_USB_BufferEP0RxStatusInitialize ( MY_USB_INSTANCE, pBDT,
    USB_PING_PONG_NO_PING_PONG, USB_BUFFER_EVEN, 64);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode
bufferByteCount	size of the EP0 RX buffer in bytes

Function

void PLIB_USB_BufferEP0RxStatusInitialize (USB_MODULE_ID index,
void* pBDT,
USB_PING_PONG_MODE ppMode,
USB_BUFFER_PING_PONG pingpong,
uint16_t bufferByteCount);

PLIB_USB_BufferIndexGet Function

Gets the Buffer Descriptor Table index for a buffer.

File

plib_usb.h

C

```
uint8_t PLIB_USB_BufferIndexGet(USB_MODULE_ID index, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Buffer index into the Buffer Descriptor Table.

Description

This function gets the Buffer Descriptor Table index for a buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint8_t PLIB_USB_BufferIndexGet (USB_MODULE_ID index,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferPIDBitsClear Function

Clears the Buffer Status bits in the Buffer Descriptor Table.

File

plib_usb.h

C

```
void PLIB_USB_BufferPIDBitsClear(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function clears the Buffer Status bits in the Buffer Descriptor Table.

Remarks

Call PLIB_USB_BufferPIDBitsClear before setting buffer control bits. This is equivalent to: PLIB_USB_BufferCancelReleaseToUSB(...) PLIB_USB_BufferDataToggleSelect(...,USB_BUFFER_DATA0) PLIB_USB_BufferDataToggleSyncDisable(...) PLIB_USB_BufferStallDisable(...) This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

The associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true'.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferPIDBitsClear (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferPIDGet Function

Returns the token packet ID (PID) from the endpoint buffer status.

File

plib_usb.h

C

```
uint8_t PLIB_USB_BufferPIDGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Endpoint buffer packet ID (PID).

Description

This function returns the token packet ID (PID) from the endpoint buffer status.

Remarks

There is no equivalent "Set" routine, since this field is read-only in the buffer status register within the Buffer Descriptor Table. It is set when the buffer has been transmitted or received by the USB module and the usbOwnsBuffer field has been cleared by the USB module, releasing the buffer for software access. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint8_t PLIB_USB_BufferPIDGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferReleasedToSW Function

Returns the boolean flag value of 'true' when the buffer has been released by the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_BufferReleasedToSW(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

- true - The buffer has been released by hardware
- false - The buffer is still controlled by hardware

Description

This function returns the boolean flag value of 'true' when the buffer has been released by the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

bool PLIB_USB_BufferReleasedToSW (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferReleaseToUSB Function

Releases the endpoint buffer by software, allowing the USB module access to the buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferReleaseToUSB(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function releases the endpoint buffer by software, allowing the USB module access to buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferReleaseToUSB (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferSchedule Function

Hands over a buffer to the USB module along with the buffer address and byte count.

File

plib_usb.h

C

```
void PLIB_USB_BufferSchedule(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, void * bufferAddress, int16_t
bufferByteCount, USB_BUFFER_SCHEDULE_DATA01 bufferData01);
```

Returns

None.

Description

This function sets the endpoint descriptor buffer address, sets the send/receive byte count, and then hands over the buffer to the USB module.

Remarks

This function does the work of three other functions: PLIB_USB_BufferAddressSet, PLIB_USB_BufferByteCountSet, PLIB_USB_BufferReleaseToUSB

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	Address of the application buffer
bufferByteCount	Send or expected receive byte count
bufferData01	USB_BUFFER_SET_DATA0, USB_BUFFER_SET_DATA1, or USB_BUFFER_DONTCHANGE (The last choice leaves the existing DATA0/1 value of the buffer alone.)

Function

void PLIB_USB_BufferSchedule(USB_MODULE_ID index ,
void* pBDT ,
USB_PING_PONG_MODE ppMode ,
uint8_t epValue ,
USB_BUFFER_DIRECTION bufferDirection ,
USB_BUFFER_PING_PONG bufferPingPong ,
void * bufferAddress ,
int16_t bufferByteCount ,
USB_BUFFER_SCHEDULE_DATA01 bufferData01)

PLIB_USB_BufferStallDisable Function

Disables STALL handshaking for the associated endpoint buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferStallDisable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function disables STALL handshaking for the associated endpoint buffer.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available. .

Preconditions

The associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferStallDisable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferStallEnable Function

Enables STALL handshaking for the associated endpoint buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferStallEnable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function enables STALL handshaking for the associated endpoint buffer.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

The associated buffer must have been released by the USB module (i.e. PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferStallEnable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferStallGet Function

Returns the buffer stall status for an endpoint/direction/ping-pong.

File

plib_usb.h

C

```
bool PLIB_USB_BufferStallGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

- true - Buffer stall is enabled
- false - Buffer stall is not enabled

Description

This function returns the buffer stall status for an endpoint/direction/ping-pong.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

the associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

bool PLIB_USB_BufferStallGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,

uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

c) Endpoints Functions

PLIB_USB_EP0HostSetup Function

Sends token to the specified address.

File

plib_usb.h

C

```
void PLIB_USB_EP0HostSetup(USB_MODULE_ID index);
```

Returns

None.

Description

This function configures endpoint 0 for typical host operation. Control transfers are enable. Transmit and Receive is enabled. Handshaking is enabled. Low Speed connection is disabled. NAK retry is disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Configure endpoint 0 for host operation

PLBIB_USB_EP0HostSetup(USB_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0HostSetup(USB_MODULE_ID index);

PLIB_USB_EP0LSDirectConnectDisable Function

Disables direct connection to a low-speed device for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0LSDirectConnectDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables direct connection to a low-speed device for Endpoint 0.

Remarks

Host mode and U1EP0 only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0LowSpeedConnect in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
PLIB_USB_EP0LSDirectConnectDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0LSDirectConnectDisable (USB_MODULE_ID index)

PLIB_USB_EP0LSDirectConnectEnable Function

Enables direct connection to a low-speed device for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0LSDirectConnectEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables direct connection to a low-speed device for Endpoint 0.

Remarks

Host Mode and U1EP0 only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0LowSpeedConnect in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_EP0LSDirectConnectEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0LSDirectConnectEnable (USB_MODULE_ID index)

PLIB_USB_EP0NakRetryDisable Function

Disables retrying of NAKed transactions.

File

plib_usb.h

C

```
void PLIB_USB_EP0NakRetryDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables retrying of NAKed transactions.

Remarks

Host/OTG only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0NAKRetry in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host or OTG modes.

Example

```
PLIB_USB_EP0NakRetryDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0NakRetryDisable (USB_MODULE_ID index)

PLIB_USB_EP0NakRetryEnable Function

Enables retrying NAK'd transactions for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0NakRetryEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables retrying NAK'd transactions for Endpoint 0.

Remarks

Host/OTG only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0NAKRetry in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host or OTG modes.

Example

```
PLIB_USB_EP0NakRetryEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0NakRetryEnable (USB_MODULE_ID index)

PLIB_USB_EPnAttributesClear Function

Clears the set attributes of the specified endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnAttributesClear(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

Clears the set attributes of the specified endpoint. The endpoint transmit receive, handshake and setup packet handling capability is disabled.

Remarks

None.

Preconditions

None.

Example

```
// This clears up the endpoint 0 attributes and thus disables
// the endpoints
PLIB_USB_EPnAttributesClear(MY_USB_INSTANCE, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnAttributesClear (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnAttributesSet Function

Configures attributes of the endpoint such as direction, handshake capability and direction.

File

plib_usb.h

C

```
void PLIB_USB_EPnAttributesSet(USB_MODULE_ID index, uint8_t epValue, int direction, bool isControl, bool handshake);
```

Returns

None.

Description

Configures attributes of the endpoint such as direction, handshake capability and direction. If the isControl flag is true, then the direction and handshake parameters are ignored and the endpoint is configured for control transfers.

Remarks

None.

Preconditions

None.

Example

```
// This enables endpoint 0 for control transfers.
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 0, 0, true, true);
// This enables endpoint 2 for non control transfer, direction
// is RX and handshake enable.
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 2, 1, false, true);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	Endpoint direction, if 1 then RX and if 0 then TX.

isControl	If true endpoint is configured for control transfers.
handshake	If true, then handshake is enabled on the endpoint else it is disabled.

Function

void PLIB_USB_EPnAttributesSet (USB_MODULE_ID index,
uint8_t epValue, int direction, bool isControl, bool handshake)

PLIB_USB_EPnControlTransferDisable Function

Disables endpoint control transfers.

File

plib_usb.h

C

```
void PLIB_USB_EPnControlTransferDisable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function disables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnControlTransferDisable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnControlTransferDisable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnControlTransferEnable Function

Enables endpoint control transfers.

File

plib_usb.h

C

```
void PLIB_USB_EPnControlTransferEnable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function enables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnControlTransferEnable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnControlTransferEnable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnDirectionDisable Function

Disables the specified endpoint direction.

File

plib_usb.h

C

```
void PLIB_USB_EPnDirectionDisable(USB_MODULE_ID index, uint8_t epValue, int direction);
```

Returns

None.

Description

Disables the specified endpoint direction.

Remarks

None.

Preconditions

None.

Example

```
// This function disables the TX direction of endpoint 1
PLIB_USB_EPnDirectionDisable(MY_USB_INSTANCE, 1, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	If 1, then TX direction is disabled. If 0 RX direction is disabled.

Function

void PLIB_USB_EPnDirectionDisable (USB_MODULE_ID index,

uint8_t epValue, int direction)

PLIB_USB_EPnHandshakeDisable Function

Disables endpoint handshaking.

File

plib_usb.h

C

```
void PLIB_USB_EPnHandshakeDisable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function disables endpoint handshaking. Typically used for Isochronous endpoints.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnHandshakeDisable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnHandshakeDisable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnHandshakeEnable Function

Enables endpoint handshaking.

File

plib_usb.h

C

```
void PLIB_USB_EPnHandshakeEnable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function enables endpoint handshaking.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnHandshakeEnable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnHandshakeEnable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnIsStalled Function

Tests whether the endpoint epValue is stalled.

File

plib_usb.h

C

```
bool PLIB_USB_EPnIsStalled(USB_MODULE_ID index, uint8_t epValue);
```

Returns

- true - The endpoint is stalled
- false - The endpoint is not stalled

Description

This function tests whether the endpoint epValue is stalled.

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )
{
    // Handle the stall
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

bool PLIB_USB_EPnIsStalled (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnRxDisable Function

Disables an endpoint's ability to process IN tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxDisable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function disables an endpoint's ability to process IN tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//De-provision endpoint 3 to process IN Token
PLIB_USB_EPnRxDisable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnRxDisable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnRxEnable Function

Enables an endpoint to process IN tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxEnable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function enables an endpoint to process IN tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Provision endpoint 3 to process IN Token
PLIB_USB_EPnRxEnable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnRxEnable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnRxSelect Function

Selects receive capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects receive capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnRxSelect(MY_USB_INSTANCE, someEP, USB_EP_RX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_RX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnRxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

PLIB_USB_EPnStallClear Function

Clears an endpoint's stalled flag.

File

plib_usb.h

C

```
void PLIB_USB_EPnStallClear(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function clears an endpoint's stalled flag.

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )
{
    // Handle the stall
    PLIB_USB_EPnStallClear(MY_USB_INSTANCE, someEP);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnStallClear (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnTxDisable Function

Disables an endpoint's ability to process OUT tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxDisable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function disables an endpoint's ability to process OUT tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//De-provision endpoint 3 to process OUT Token
PLIB_USB_EPnTxDisable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnTxDisable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnTxEnable Function

Enables an endpoint to process OUT tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxEnable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function enables an endpoint to process OUT tokens.

Remarks

None.

Preconditions

None.

Example

```
//Provision endpoint 3 to process OUT Token
PLIB_USB_EPnTxEnable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnTxEnable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnTxRxSelect Function

Selects transmit and/or receive capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxRxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects transmit and/or receive capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnTxRxSelect(MY_USB_INSTANCE, someEP, USB_EP_TXRX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_RX, USB_EP_TX_RX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnTxRxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

PLIB_USB_EPnTxSelect Function

Selects transmit capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects transmit capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnTxSelect(MY_USB_INSTANCE, someEP, USB_EP_TX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnTxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

d) Interrupts Functions

PLIB_USB_InterruptDisable Function

Disables a general interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptDisable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables a general interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptDisable( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptDisable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptEnable Function

Enables a general interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables general interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptEnable( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptEnableGet Function

Returns the enable/disable status of general USB module interrupts

File

plib_usb.h

C

```
USB_INTERRUPTS PLIB_USB_InterruptEnableGet(USB_MODULE_ID index);
```

Returns

A bit map containing status of enabled interrupts.

Description

Returns the enable/disable status of general USB module interrupts

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_INTERRUPTS enabledInterrupts;
enabledInterrupts = PLIB_USB_InterruptEnableGet( MY_USB_INSTANCE);
if(enabledInterrupts|USB_INT_ATTACH)
{
    // This means Attach interrupt is enabled.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_INTERRUPTS PLIB_USB_InterruptEnableGet(USB_MODULE_ID index)

PLIB_USB_InterruptFlagAllGet Function

Returns a logically ORed bit map of active general USB interrupt flags.

File

plib_usb.h

C

```
USB_INTERRUPTS PLIB_USB_InterruptFlagAllGet(USB_MODULE_ID index);
```

Returns

Returns a logically ORed bit map of active general USB interrupt flags.

Description

This function returns a logically ORed bit map of active general USB interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_INTERRUPTS interruptEnables;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_DEVICE_RESET)
{
    // Device received reset signaling
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_INTERRUPTS PLIB_USB_InterruptFlagAllGet(USB_MODULE_ID index);

PLIB_USB_InterruptFlagClear Function

Clears a general interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptFlagClear(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptFlagClear( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptFlagClear(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptFlagGet Function

Tests a general interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_InterruptFlagGet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_ANY) )
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_ERROR) )
    {
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE,USB_INT_ERROR);
        // Error clean up
    }
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_HOST_DETACH) )
    {
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE,USB_INT_HOST_DETACH);
        // Device detached clean up
    }
```

```
        .
        .
        .
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_InterruptFlagGet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptFlagSet Function

Sets a general interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptFlagSet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptFlagSet( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptFlagSet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptIsEnabled Function

Returns true if interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function returns true if interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_InterruptIsEnabled( MY_USB_INSTANCE, USB_INT_ERROR ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

e) Error Interrupts Functions

PLIB_USB_ErrorInterruptDisable Function

Disables an error interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptDisable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables an error interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptDisable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptDisable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptEnable Function

Enables an error interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptEnable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables error interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptEnable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptEnable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagAllGet Function

Returns a logically ORed bit map of active error interrupt flags.

File

plib_usb.h

C

```
USB_ERROR_INTERRUPTS PLIB_USB_ErrorInterruptFlagAllGet(USB_MODULE_ID index);
```

Returns

Returns a logically ORed bit map of active error interrupt flags.

Description

This function returns a logically ORed bit map of active error interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_ERROR_INTERRUPTS errorInterruptEnables;

errorInterruptEnables = PLIB_USB_ErrorInterruptFlagAllGet(MY_USB_INSTANCE);
```

```
if(errorInterruptEnables | USB_ERR_INT_DEVICE_EOF_ERROR)
{
    // End of frame error occurred.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_ERROR_INTERRUPTS PLIB_USB_ErrorInterruptFlagAllGet(USB_MODULE_ID index);

PLIB_USB_ErrorInterruptFlagClear Function

Clears an error interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptFlagClear(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears an error interrupt source flag for the USB module.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptFlagClear( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptFlagClear(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagGet Function

Tests an error interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ErrorInterruptFlagGet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests an error interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_ANY) )
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE);
        // PID Error Check failure cleanup
    }
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR);
        // EOF error cleanup
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_ErrorInterruptFlagGet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagSet Function

Sets an error interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptFlagSet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets an error interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptFlagSet( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptFlagSet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptIsEnabled Function

Returns true if interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_ErrorInterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function determines whether interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_ErrorInterruptIsEnabled( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_ErrorInterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

f) Last Transaction Status Functions

PLIB_USB_LastTransactionDetailsGet Function

Returns the details of the last completed transaction.

File

plib_usb.h

C

```
void PLIB_USB_LastTransactionDetailsGet(USB_MODULE_ID index, USB_BUFFER_DIRECTION * direction,
USB_PING_PONG_STATE * pingpong, uint8_t * endpoint);
```

Returns

None.

Description

This function returns the details of the last completed transaction.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastTransactionDetails in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_BUFFER_DIRECTION direction;
USB_PING_PONG_STATE pingpong;
uint8_t endpoint;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_TOKEN_DONE)
{
    // Find out details of the token
    PLIB_USB_LastTransactionDetailsGet(MY_USB_INSTANCE, &direction,
                                    &pingpong, &endpoint);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
direction	Return value contains direction of the last transfer
pingpong	Return value contains Ping-Pong indication of the the last transfer
endpoint	Return value contains the endpoint which processed the last transfer

Function

void PLIB_USB_LastTransactionDetailsGet(USB_MODULE_ID index,

USB_BUFFER_DIRECTION * direction,

USB_PING_PONG_STATE * pingpong,

uint8_t * endpoint);

PLIB_USB_LastTransactionDirectionGet Function

Indicates the direction of the last transaction.

File

plib_usb.h

C

```
USB_BUFFER_DIRECTION PLIB_USB_LastTransactionDirectionGet(USB_MODULE_ID index);
```

Returns

USB_LastDirection: USB_RECEIVE_TRANSFER or USB_TRANSMIT_TRANSFER

Description

This function indicates the direction of the last transaction, either transmit or receive.

Remarks

None.

Preconditions

None.

Example

See PLIB_USB_LastTransactionEndPtGet.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_BUFFER_DIRECTION PLIB_USB_LastTransactionDirectionGet (USB_MODULE_ID index)

PLIB_USB_LastTransactionEndPtGet Function

Returns the endpoint number of the last USB transfer.

File

plib_usb.h

C

```
uint8_t PLIB_USB_LastTransactionEndPtGet(USB_MODULE_ID index);
```

Returns

endPoint - Endpoint of last completed USB transfer

Description

This function returns the endpoint number of the last USB transfer, which is actually the index into the Buffer Descriptor Table of the last USB transfer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
while ( !PLIB_USB_INT_FlagGet(MY_USB_INSTANCE,USB_INT_GEN,TOKEN_DONE) )
{
    // Do nothing, wait until completion of next transaction
}

// Retrieve information relating to the last completed transaction
endpoint  = PLIB_USB_LastTransactionEndPtGet(MY_USB_INSTANCE);
direction = PLIB_USB_LastTransactionDirectionGet(MY_USB_INSTANCE);
pingPongState = PLIB_USB_LastTransactionPingPongStateGet(MY_USB_INSTANCE);

// Clearing the Token Processing Done flag advances the status FIFO to
// oldest transaction in the FIFO.  Wait for completion of next transaction
// before using PLIB_USB_Last*Get functions again to read status.
PLIB_USB_INT_FlagClear(MY_USB_INSTANCE,USB_INT_GEN,TOKEN_DONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_LastTransactionEndPtGet (USB_MODULE_ID index)

PLIB_USB_LastTransactionPingPongStateGet Function

Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.

File

plib_usb.h

C

```
USB_PING_PONG_STATE PLIB_USB_LastTransactionPingPongStateGet(USB_MODULE_ID index);
```

Returns

USB_PING_PONG_STATE.

Description

This function indicates whether the last transaction was to an Even buffer or an Odd buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastPingPong in your application to determine whether this feature is available.

Preconditions

None.

Example

See PLIB_USB_LastTransactionEndPtGet.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PING_PONG_STATE PLIB_USB_LastTransactionPingPongStateGet (USB_MODULE_ID index)

g) Host Functions

PLIB_USB_IsBusyWithToken Function

Indicates whether there is a token being executed by the USB module as Host.

File

plib_usb.h

C

```
bool PLIB_USB_IsBusyWithToken(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates whether there is a token being executed by the USB module as Host. Software should check that the previous token is finished before issuing a new token.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostBusyWithToken in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
while( PLIB_USB_IsBusyWithToken(MY_USB_INSTANCE) )
{
    // do nothing
}
// Issue new token
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_IsBusyWithToken (USB_MODULE_ID index)

PLIB_USB_SOFDisable Function

Disables the automatic generation of the SOF token.

File

plib_usb.h

C

```
void PLIB_USB_SOFDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the automatic generation of the SOF token.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStartOfFrames in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_SOFDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SOFDisable (USB_MODULE_ID index)

PLIB_USB_SOFEnable Function

Enables the automatic generation of the SOF token every 1 ms.

File

plib_usb.h

C

```
void PLIB_USB_SOFEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the automatic generation of the SOF token every 1 ms.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStartOfFrames in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_SOFEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SOFEnable (USB_MODULE_ID index)

PLIB_USB_SOFThresholdGet Function

Returns the Start-of-Frame (SOF) Count bits.

File

plib_usb.h

C

```
uint8_t PLIB_USB_SOFThresholdGet(USB_MODULE_ID index);
```

Returns

SOF threshold value.

Description

This function returns the Start-of-Frame (SOF) Count bits. (Value represents 10 + (packet size of n bytes);).

Remarks

Host mode only.

```
    SOF Threshold Value = packet byte count + 10
                        = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                        = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                        = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                        = 0x0001_0010 = 0x12 = 18 for  8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSOFThreshold in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
  thresholdSOF = PLIB_USB_SOFThresholdGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_SOFThresholdGet (USB_MODULE_ID index)

PLIB_USB_SOFThresholdSet Function

Sets the Start-of-Frame (SOF) threshold value.

File

plib_usb.h

C

```
void PLIB_USB_SOFThresholdSet(USB_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the Start-of-Frame (SOF) threshold value.

Remarks

Host mode only.

```
    SOF Threshold Value = packet byte count + 10
                        = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                        = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                        = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                        = 0x0001_0010 = 0x12 = 18 for  8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSOFThreshold in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Set SOF threshold for 64-byte packets
PLIB_USB_SOFThresholdSet(MY_USB_INSTANCE,64+10);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
threshold	SOF threshold

Function

void PLIB_USB_SOFThresholdSet (USB_MODULE_ID index, uint8_t threshold)

PLIB_USB_TokenEPGet Function

Returns the specified Endpoint address.

File

plib_usb.h

C

```
uint8_t PLIB_USB_TokenEPGet(USB_MODULE_ID index);
```

Returns

Endpoint value - 0 <= epValue <= Module Maximum Endpoint.

Description

This function returns the address of the specified Endpoint.

Remarks

Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenEP in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
someEP = PLIB_USB_TokenEPGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_TokenEPGet (USB_MODULE_ID index)

PLIB_USB_TokenEPSet Function

Sets the Endpoint address for a host transaction.

File

plib_usb.h

C

```
void PLIB_USB_TokenEPSet(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function sets the Endpoint address for a host transaction.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenEP in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
uint8_t someEP = 0x03;
PLIB_USB_TokenEPSet(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_TokenEPSet (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_TokenPIDGet Function

Returns the token transaction type.

File

plib_usb.h

C

```
USB_PID PLIB_USB_TokenPIDGet(USB_MODULE_ID index);
```

Returns

Packet ID of token, USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Description

This function returns the token transaction type.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenPID in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
somePID = PLIB_USB_TokenPIDGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PID PLIB_USB_TokenPIDGet (USB_MODULE_ID index)

PLIB_USB_TokenPIDSet Function

Sets the token transaction type to pidValue.

File

plib_usb.h

C

```
void PLIB_USB_TokenPIDSet(USB_MODULE_ID index, USB_PID pidValue);
```

Returns

None.

Description

This function sets the token transaction type to pidValue.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenPID in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
somePID = USB_PID_SETUP;
PLIB_USB_TokenPIDSet (MY_USB_INSTANCE, somePID );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pidValue	USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Function

void PLIB_USB_TokenPIDSet (USB_MODULE_ID index, USB_PID pidValue)

PLIB_USB_TokenSpeedSelect Function

Selects low speed or full speed for subsequent token executions.

File

plib_usb.h

C

```
void PLIB_USB_TokenSpeedSelect(USB_MODULE_ID index, USB_TOKEN_SPEED tokenSpeed);
```

Returns

None.

Description

This function selects low speed or full speed for subsequent token executions.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsNextTokenSpeed in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
PLIB_USB_TokenSpeedSet(MY_USB_INSTANCE,USB_LOWSPEED_TOKENS);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
tokenSpeed	Speed for next token execution: USB_LOWSPEED_TOKENS or USB_FULLSPEED_TOKENS

Function

void PLIB_USB_TokenSpeedSelect (USB_MODULE_ID index, USB_TOKEN_SPEED tokenSpeed)

h) USB Bus Signaling Functions

PLIB_USB_ResetSignalDisable Function

Disables reset signaling on the USB bus.

File

plib_usb.h

C

```
void PLIB_USB_ResetSignalDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables reset signaling on the USB bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostGeneratesReset in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

See PLIB_USB_ResetSignalEnable.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResetSignalDisable (USB_MODULE_ID index)

PLIB_USB_ResetSignalEnable Function

Enables reset signaling on the USB bus.

File

plib_usb.h

C

```
void PLIB_USB_ResetSignalEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables reset signaling on the USB bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostGeneratesReset in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Snippet to perform a software reset:
PLIB_USB_ResetSignalEnable(MY_USB_INSTANCE);
// ... delay 50ms
PLIB_USB_ResetSignalDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResetSignalEnable (USB_MODULE_ID index)

PLIB_USB_ResumeSignalingDisable Function

Disables resume signaling.

File

plib_usb.h

C

```
void PLIB_USB_ResumeSignalingDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables resume signaling.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsResumeSignaling in your application to determine whether this feature is available.

Preconditions

None.

Example

See PLIB_USB_ResumeSignalingEnable.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResumeSignalingDisable (USB_MODULE_ID index)

PLIB_USB_ResumeSignalingEnable Function

Enables resume signaling.

File

plib_usb.h

C

```
void PLIB_USB_ResumeSignalingEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables resume signaling. Resume allows the peripheral to perform a remote wake-up by executing resume signaling.

Remarks

Software must enable resume signaling for 10 ms if the device is in Device mode, or for 25 ms if the device is in Host mode, and then disable resume signaling to enable remote wake-up. In Host mode, the USB module will append a low-speed EOP to the end resume signaling when it is disabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsResumeSignaling in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Perform resume signaling:
PLIB_USB_ResumeSignalingEnable(MY_USB_INSTANCE);
// Delay 10ms
PLIB_USB_ResumeSignalingDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResumeSignalingEnable (USB_MODULE_ID index)

i) On-The-Go (OTG) Functions

PLIB_USB_OTG_BSessionHasEnded Function

Returns the status of the B-Session End Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_BSessionHasEnded(USB_MODULE_ID index);
```

Returns

- true - The VBUS Voltage is below VB_SESS_END on the B-device
- false - The VBUS voltage is above VB_SESS_END on the B-device

Description

This function returns the status of the B-Session End Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_BSessionEnd in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( !PLIB_USB_OTG_BSessionHasEnded(MY_USB_INSTANCE) )
{
    // B session valid
}
else
{
    // B session not valid
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_BSessionHasEnded (USB_MODULE_ID index)

PLIB_USB_OTG_IDPinStateIsTypeA Function

Returns the ID Pin state.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_IDPinStateIsTypeA(USB_MODULE_ID index);
```

Returns

- true - Type A Cable attached,
- false - No cable is attached or a Type B cable is attached

Description

This function returns ID Pin state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_IDPinState in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_IDPinStateIsTypeA(MY_USB_INSTANCE) )
{
    // Type A cable attached
}
else
{
    // No cable or Type B cable attached
};
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_IDPinStateIsTypeA (USB_MODULE_ID index)

PLIB_USB_OTG_LineStateIsStable Function

Returns the status of the Line Stable Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_LineStateIsStable(USB_MODULE_ID index);
```

Returns

- true - The USB line state has been stable for the previous 1 ms
- false - The USB line state has not been stable for the previous 1 ms

Description

This function returns the status of the Line Stable Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_LineState in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if( PLIB_USB_OTG_LineStateIsStable(MY_USB_INSTANCE) ) {
    // Line has been stable
    // ... rest of code ...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_LineStateIsStable (USB_MODULE_ID index)

PLIB_USB_OTG_PullUpPullDownSetup Function

Enables or disables pull-up and pull-down resistors.

File

plib_usb.h

C

```
void PLIB_USB_OTG_PullUpPullDownSetup(USB_MODULE_ID index, USB_OTG_PULL_UP_PULL_DOWN resistor, bool
enableResistor);
```

Returns

None.

Description

This function enables or disables pull-up and pull-down resistors.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_PullUpPullDown in your application to determine whether this feature is available.

Preconditions

USB On-The-Go (OTG) must be enabled.

Example

```
// Enable pull-up resistor for D+
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLUP,true);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
resistor	USB_OTG_DPLUS_PULLUP, USB_OTG_DMINUS_PULLUP, USB_OTG_DPLUS_PULLDN, or USB_OTG_DMINUS_PULLDN
enableResistor	true to enable resistor, false to disable it

Function

void PLIB_USB_OTG_PullUpPullDownSetup(USB_MODULE_ID index,
USB_OTG_PULL_UP_PULL_DOWN resistor,
bool enableResistor)

PLIB_USB_OTG_SessionValid Function

Returns the status of the Session Valid Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_SessionValid(USB_MODULE_ID index);
```

Returns

- true - The VBUS voltage is above Session Valid on the A or B device
- false - The VBUS voltage is below Session Valid on the A or B device

Description

This function returns the status of the Session Valid Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_SessionValid in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_SessionValid(MY_USB_INSTANCE) )
{
    // Session valid
}
else
{
    // Session not valid
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_SessionValid (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeDisable Function

Disables VBUS line charge.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables VBUS line charge.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusCharge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeDisable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeEnable Function

Enables the VBUS line to be charged through a pull-up resistor.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the VBUS line to be charged through a pull-up resistor.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeEnable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeTo3V Function

Sets the VBUS line to charge to 3.3V.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeTo3V(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the VBUS line to charge to 3.3V.

Remarks

Not available on PIC32 devices.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeTo3V(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeTo3V(USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeTo5V Function

Sets the VBUS line to charge to 5V.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeTo5V(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the VBUS line to charge to 5V.

Remarks

Not available on PIC32 devices.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeTo5V (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeTo5V (USB_MODULE_ID index)

PLIB_USB_OTG_VBusDischargeDisable Function

Disables VBUS line discharge.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusDischargeDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables VBUS line discharge.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusDischargeDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusDischargeDisable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusDischargeEnable Function

Enables VBUS line to be discharged through a resistor.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusDischargeEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the VBUS line to be discharged through a resistor.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusCharge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusDischargeEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusDischargeEnable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusPowerOff Function

Turns off power on the VBUS Line.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusPowerOff(USB_MODULE_ID index);
```

Returns

None.

Description

This function turns off power on the VBUS Line.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusPowerOnOff in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusPowerOff(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusPowerOff (USB_MODULE_ID index)

PLIB_USB_OTG_VBusPowerOn Function

Turns on power for the VBUS line.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusPowerOn(USB_MODULE_ID index);
```

Returns

None.

Description

This function turns on power for the VBUS line.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusPowerOnOff in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusPowerOn(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusPowerOn (USB_MODULE_ID index)

PLIB_USB_OTG_VBusValid Function

Returns the status of the A-VBUS valid indicator.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_VBusValid(USB_MODULE_ID index);
```

Returns

- true - The VBUS voltage is above VA_VBUS_VLD on the A-device,
- false - The VBUS voltage is below VA_VBUS_VLD on the A-device

Description

This function returns the status of the A-VBUS valid indicator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_ASessionValid in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_VBusValid(MY_USB_INSTANCE) )
{
    // VBUS voltage above session valid for A device
}
else
{
    // VBUS voltage below session valid for A device
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_VBusValid (USB_MODULE_ID index)

j) OTG Interrupts Functions

PLIB_USB_OTG_InterruptDisable Function

Disables a USB On-The-Go (OTG) Interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptDisable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables a USB On-The-Go (OTG) interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptDisable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptDisable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptEnable Function

Enables a USB On-The-Go (OTG) Interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptEnable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables USB On-The-Go (OTG) interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptEnable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptEnable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagClear Function

Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptFlagClear(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptFlagClear( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptFlagClear(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagGet Function

Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_InterruptFlagGet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_ANY) )
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_ADEVICE_VBUS_VALID );
        // Device A VBUS Valid Change
    }
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END);
        // Device B VBUS Valid Change
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_OTG_InterruptFlagGet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagSet Function

Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptFlagSet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptFlagSet( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptFlagSet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptIsEnabled Function

Returns whether or not interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function returns whether or not interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_OTG_InterruptIsEnabled( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE  ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_OTG_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

k) USB Activity Functions

PLIB_USB_ActivityPending Function

Returns whether or not USB activity is pending.

File

plib_usb.h

C

```
bool PLIB_USB_ActivityPending(USB_MODULE_ID index);
```

Returns

- true - The USB module should not be suspended
- false - No interrupts are pending or module may be suspended or powered down

Description

This function returns whether or not USB bus activity has been detected, an interrupt is pending, or an interrupt is yet to be generated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsActivityPending in your application to determine whether this feature is available.

Preconditions

None.

Example

```
while ( PLIB_USB_ActivityPending(MY_USB_INSTANCE) )
{
    // Wait
}
// Suspend USB module.
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_ActivityPending (USB_MODULE_ID index)

PLIB_USB_FrameNumberGet Function

Returns the USB frame number.

File

plib_usb.h

C

```
uint16_t PLIB_USB_FrameNumberGet(USB_MODULE_ID index);
```

Returns

Current frame number in lower 11 bits.

Description

This function returns the USB frame number in the lower 11 bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsFrameNumber in your application to determine whether this feature is available.

Preconditions

None.

Example

```
frameNumber = PLIB_USB_FrameNumberGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint16_t PLIB_USB_FrameNumberGet (USB_MODULE_ID index)

PLIB_USB_JStateIsActive Function

Live differential receiver J State flag.

File

plib_usb.h

C

```
bool PLIB_USB_JStateIsActive(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates the live JState (differential '0' in low speed, differential '1' in full speed) on the bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLiveJState in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Enable Host Mode
PLIB_USB_OperatingModeSelect(MY_USB_INSTANCE,USB_OPMODE_HOST);

// Enable D+ and D- pull-down resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLDN, true);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DMINUS_PULLDN,true);

// Disable D+ and D- pull-up resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLUP, false);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DMINUS_PULLUP,false);

// Enable SOF Packet generation
PLIB_USB_SOFEnable(MY_USB_INSTANCE);

// Enable the device attach interrupt
PLIB_USB_INT_Enable(MY_USB_INSTANCE,USB_INT_OTG,ACTIVITY_DETECT);

// Wait for the Attach interrupt.
while(!PLIB_USB_INT_FlagGet(MY_USB_INSTANCE,USB_INT_OTG,ACTIVITY_DETECT) )
{
    //Do nothing
}

// Check JState
if( PLIB_USB_JStateIsActive(MY_USB_INSTANCE) )
{
    // Full Speed
    PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
}
else
{
    // Low Speed
    PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_JStateIsActive (USB_MODULE_ID index)

PLIB_USB_PacketTransferDisable Function

Disables the Serial Interface Engine (SIE).

File

plib_usb.h

C

```
void PLIB_USB_PacketTransferDisable(USB_MODULE_ID index);
```

Returns

None.

```
  if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
  {
      // SETUP token received, do the needful operations
      .
      .
      .
      // SETUP handling completed, enable Setup token and packet processing:
      PLIB_USB_PacketTransferDisable(MY_USB_INSTANCE);
  }
```

Description

This function disables the Serial Interface Engine (SIE).

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PacketTransferDisable (USB_MODULE_ID index)

PLIB_USB_PacketTransferEnable Function

Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

File

plib_usb.h

C

```
void PLIB_USB_PacketTransferEnable(USB_MODULE_ID index);
```

Returns

None.

```
  if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
  {
      // SETUP token received, do the needful operations
      .
      .
      .
      // SETUP handling completed, enable Setup token and packet processing:
      PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
  }
```

Description

This function re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PacketTransferEnable (USB_MODULE_ID index)

PLIB_USB_PacketTransferIsDisabled Function

Indicates that a setup token has been received from the Host and that token/packet processing is disabled.

File

plib_usb.h

C

```
bool PLIB_USB_PacketTransferIsDisabled(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates that a setup token has been received from the Host and that the Serial Interface Engine (SIE) has been turned off, disabling token and packet processing.

Remarks

Not valid when USB is Host. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Example

```
if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
{
    // SETUP token received, do the needful operations
    .
    .
    .
    // SETUP handling completed, enable Setup token and packet processing:
    PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_PacketTransferIsDisabled (USB_MODULE_ID index)

PLIB_USB_SE0InProgress Function

Returns whether a single-ended zero event is in progress.

File

plib_usb.h

C

```
bool PLIB_USB_SE0InProgress(USB_MODULE_ID index);
```

Returns

- true - A single ended zero event (SE0) is occurring

- false - A single-ended zero event (SE0) is not occurring

Description

This function returns whether a single-ended zero event is in progress.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLiveSingleEndedZero in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_SE0InProgress(MY_USB_INSTANCE) )
{
    // handle the SE0 event
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_SE0InProgress(USB_MODULE_ID index)

l) External Transceiver Support Functions

PLIB_USB_I2CInterfaceForExtModuleDisable Function

Specifies external module(s) are controlled via dedicated pins.

File

plib_usb.h

C

```
void PLIB_USB_I2CInterfaceForExtModuleDisable(USB_MODULE_ID index);
```

Returns

None.

Description

Specifies that external module(s) are controlled via dedicated pins.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_I2CInterfaceForExtModuleDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_I2CInterfaceForExtModuleDisable (USB_MODULE_ID index)

PLIB_USB_I2CInterfaceForExtModuleEnable Function

Specifies external module(s) are controlled via the I2C interface.

File

plib_usb.h

C

```
void PLIB_USB_I2CInterfaceForExtModuleEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function specifies that external module(s) are controlled via the I2C interface.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_I2CInterfaceForExtModuleEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_I2CInterfaceForExtModuleEnable (USB_MODULE_ID index)

PLIB_USB_TransceiverDisable Function

Disables the on-chip transceiver

File

plib_usb.h

C

```
void PLIB_USB_TransceiverDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the on-chip transceiver and enables the interface to the off-chip transceiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipTransceiver in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_TransceiverDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_TransceiverDisable (USB_MODULE_ID index)

PLIB_USB_TransceiverEnable Function

Enables the on-chip transceiver.

File

plib_usb.h

C

```
void PLIB_USB_TransceiverEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the on-chip transceiver. The interface to the off-chip transceiver is disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipTransceiver in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_TransceiverEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_TransceiverEnable (USB_MODULE_ID index)

m) VBUS Support Functions

PLIB_USB_ExternalComparatorMode2Pin Function

Sets the 2-pin input configuration for VBUS comparators.

File

plib_usb.h

C

```
void PLIB_USB_ExternalComparatorMode2Pin(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the 2-pin input configuration for VBUS Comparators.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_ExternalComparatorMode2Pin(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ExternalComparatorMode2Pin (USB_MODULE_ID index)

PLIB_USB_ExternalComparatorMode3Pin Function

Sets the 3-pin input configuration for VBUS Comparators.

File

plib_usb.h

C

```
void PLIB_USB_ExternalComparatorMode3Pin(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the 3-pin input configuration for VBUS comparators.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_ExternalComparatorMode3Pin(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ExternalComparatorMode3Pin (USB_MODULE_ID index)

PLIB_USB_PWMCounterDisable Function

Disables the PWM counter used to generate the VBUS for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_PWMCounterDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the PWM counter used to generate the VBUS for the USB module.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMDisable(MY_USB_INSTANCE);
PLIB_USB_PWMCounterDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMCounterDisable(USB_MODULE_ID index);

PLIB_USB_PWMCounterEnable Function

Enables the PWM counter used to generate the VBUS for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_PWMCounterEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the PWM counter used to generate the VBUS for the USB module.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);
PLIB_USB_PWMCounterEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMCounterEnable(USB_MODULE_ID index)

PLIB_USB_PWMDisable Function

Disables the PWM Generator.

File

plib_usb.h

C

```
void PLIB_USB_PWMDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the PWM Generator. PWM output held in a reset state defined by PLIB_USB_PWMPolarityActiveHigh or PLIB_USB_PWMPolarityActiveLow.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMDisable (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMDisable (USB_MODULE_ID index)

PLIB_USB_PWMEnable Function

Enables the PWM Generator.

File

plib_usb.h

C

```
void PLIB_USB_PWMEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the PWM Generator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMEnable (USB_MODULE_ID index)

PLIB_USB_PWMPolaritiyActiveLow Function

Sets the PWM output to active-high and resets low.

File

plib_usb.h

C

```
void PLIB_USB_PWMPolaritiyActiveLow(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the PWM output to active-high and resets low.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMPolaritiyActiveLow (USB_MODULE_ID index)

PLIB_USB_PWMPolarityActiveHigh Function

Sets the PWM output to active-low and resets high.

File

plib_usb.h

C

```
void PLIB_USB_PWMPolarityActiveHigh(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the PWM output to active-low and resets high.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMPolaritiy (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMPolarityActiveHigh (USB_MODULE_ID index)

PLIB_USB_VBoostDisable Function

Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.

File

plib_usb.h

C

```
void PLIB_USB_VBoostDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the On-Chip 5V Boost Regulator Circuit Disabled bit.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBoostDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBoostDisable (USB_MODULE_ID index)

PLIB_USB_VBoostEnable Function

Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.

File

plib_usb.h

C

```
void PLIB_USB_VBoostEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the On-Chip 5V Boost Regulator Circuit Enabled bit.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBoostEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBoostEnable (USB_MODULE_ID index)

PLIB_USB_VBUSComparatorDisable Function

Disables the on-chip VBUS Comparator.

File

plib_usb.h

C

```
void PLIB_USB_VBUSComparatorDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the on-chip VBUS Comparator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSComparatorDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSComparatorDisable (USB_MODULE_ID index)

PLIB_USB_VBUSComparatorEnable Function

Enables the on-chip VBUS Comparator.

File

plib_usb.h

C

```
void PLIB_USB_VBUSComparatorEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the on-chip VBUS Comparator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSComparatorEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSComparatorEnable (USB_MODULE_ID index)

PLIB_USB_VBUSPullUpDisable Function

Disables the pull-up on the VBUS pin.

File

plib_usb.h

C

```
void PLIB_USB_VBUSPullUpDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the pull-up on the VBUS pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSPullUpDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSPullUpDisable (USB_MODULE_ID index)

PLIB_USB_VBUSPullUpEnable Function

Enables the pull-up on the VBUS pin.

File

plib_usb.h

C

```
void PLIB_USB_VBUSPullUpEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the pull-up on the VBUS pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSPullUpEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSPullUpEnable (USB_MODULE_ID index)

n) Test Support Functions

PLIB_USB_EyePatternDisable Function

Disables the USB eye pattern test.

File

plib_usb.h

C

```
void PLIB_USB_EyePatternDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the USB eye pattern test.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEyePattern in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EyePatternDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EyePatternDisable (USB_MODULE_ID index)

PLIB_USB_EyePatternEnable Function

Enables USB eye pattern test.

File

plib_usb.h

C

```
void PLIB_USB_EyePatternEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB eye pattern test.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEyePattern in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EyePatternEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EyePatternEnable (USB_MODULE_ID index)

o) Other Functions

PLIB_USB_ModuleIsBusy Function

Indicates if the USB module is not ready to be enabled.

File

plib_usb.h

C

```
bool PLIB_USB_ModuleIsBusy(USB_MODULE_ID index);
```

Returns

- true - USB module is active or disabled, but not ready to be enabled
- false - USB module is not active and is ready to be enabled

Description

This function indicates if the USB module is not ready to be enabled.

Remarks

If PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) == true and the USB module is disabled, and all status returned for the module, all enables/disables for the module will produce undefined results.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModuleBusyin your application to determine whether this feature is available.

Preconditions

None.

Example

```
#ifdef (__PIC32MX__)
  while ( PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) )
  {
      // wait
  }
#endif
  PLIB_USB_Disable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_ModuleIsBusy (USB_MODULE_ID index)

PLIB_USB_PingPongFreeze Function

Resets all Ping-Pong buffer pointers to even buffers.

File

plib_usb.h

C

```
void PLIB_USB_PingPongFreeze(USB_MODULE_ID index);
```

Returns

None.

Description

This function resets all Ping-Pong buffer pointers to even buffers.

Remarks

Buffers remain "frozen" at "Even" until they are unfrozen using PLIB_USB_PingPongUnfreeze.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
  // Reset all ping-pong buffers to "Even"
  PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);
  PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongFreeze (USB_MODULE_ID index)

PLIB_USB_PingPongReset Function

Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.

File

plib_usb.h

C

```
void PLIB_USB_PingPongReset(USB_MODULE_ID index);
```

Returns

None.

Description

This function resets the USB peripheral internal Ping-Pong indicator to point to Even buffers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_PingPongReset(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongReset (USB_MODULE_ID index)

PLIB_USB_PingPongUnfreeze Function

Enables Ping-Pong buffering.

File

plib_usb.h

C

```
void PLIB_USB_PingPongUnfreeze(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables Ping-Pong buffering.

Remarks

See PLIB_USB_PingPongFreeze. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset all Ping-Pong buffers to "Even"
PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);
PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongUnfreeze (USB_MODULE_ID index)

PLIB_USB_TokenSend Function

Sends token to the specified address.

File

plib_usb.h

C

```
void PLIB_USB_TokenSend(USB_MODULE_ID index, USB_PID pidValue, uint8_t endpoint, uint8_t deviceAddress,
bool isLowSpeed);
```

Returns

None.

Description

This function sends the specified token to the specified endpoint and address. The token is placed on the bus at the next available time. The token can be executed at low speed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Send an OUT token to endpoint 1 device address 2 at full speed
PLIB_USB_SendToken(MY_USB_INSTANCE, USB_PID_OUT, 1, 2, false);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pidValue	PID of the token to be placed on the bus.
endpoint	Device endpoint to which the token should be sent.
isLowSpeed	Is true if the token should be executed at low speed.

Function

void PLIB_USB_TokenSend(USB_MODULE_ID index, USB_PID pidValue,

uint8_t endpoint, uint8_t deviceAddress, bool isLowSpeed);

p) Feature Existence Functions

PLIB_USB_ExistsActivityPending Function

Identifies whether the ActivityPending feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsActivityPending(USB_MODULE_ID index);
```

Returns

- true - The ActivityPending feature is supported on the device
- false - The ActivityPending feature is not supported on the device

Description

This function identifies whether the ActivityPending feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_ActivityPending

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsActivityPending(USB_MODULE_ID index)

PLIB_USB_ExistsALL_Interrupt Function

Identifies whether the ALL_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsALL_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The ALL_Interrupt feature is supported on the device
- false - The ALL_Interrupt feature is not supported on the device

Description

This function identifies whether the ALL_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_AllInterruptEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsALL_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsAutomaticSuspend Function

Identifies whether the AutomaticSuspend feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsAutomaticSuspend(USB_MODULE_ID index);
```

Returns

- true - The AutomaticSuspend feature is supported on the device
- false - The AutomaticSuspend feature is not supported on the device

Description

This function identifies whether the AutomaticSuspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_AutoSuspendDisable

- PLIB_USB_AutoSuspendEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsAutomaticSuspend(USB_MODULE_ID index)

PLIB_USB_ExistsBDTBaseAddress Function

Identifies whether the BDTBaseAddress feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBDTBaseAddress(USB_MODULE_ID index);
```

Returns

- true - The BDTBaseAddress feature is supported on the device
- false - The BDTBaseAddress feature is not supported on the device

Description

This function identifies whether the BDTBaseAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_BDTBaseAddressGet
- PLIB_USB_BDTBaseAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBDTBaseAddress(USB_MODULE_ID index)

PLIB_USB_ExistsBDTFunctions Function

Identifies whether the BDTFunctions feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBDTFunctions(USB_MODULE_ID index);
```

Returns

- true - The BDTFunctions feature is supported on the device
- false - The BDTFunctions feature is not supported on the device

Description

This function identifies whether the BDTFunctions feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_BufferAddressGet
- PLIB_USB_BufferAddressSet
- PLIB_USB_BufferByteCountGet
- PLIB_USB_BufferByteCountSet
- PLIB_USB_BufferCancelReleaseToUSB
- PLIB_USB_BufferAllCancelReleaseToUSB
- PLIB_USB_BufferClearAll
- PLIB_USB_BufferDataToggleGet
- PLIB_USB_BufferDataToggleSelect
- PLIB_USB_BufferDataToggleSyncEnable
- PLIB_USB_BufferDataToggleSyncDisable
- PLIB_USB_BufferIndexGet
- PLIB_USB_BufferPIDBitsClear
- PLIB_USB_BufferPIDGet
- PLIB_USB_BufferReleasedToSW
- PLIB_USB_BufferReleaseToUSB
- PLIB_USB_BufferSchedule
- PLIB_USB_BufferStallDisable
- PLIB_USB_BufferStallEnable
- PLIB_USB_BufferStallGet
- PLIB_USB_BufferEP0RxStatusInitialize
- PLIB_USB_BufferClearAllDTSEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBDTFunctions(USB_MODULE_ID index)

PLIB_USB_ExistsBufferFreeze Function

Identifies whether the BufferFreeze feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBufferFreeze(USB_MODULE_ID index);
```

Returns

- true - The BufferFreeze feature is supported on the device
- false - The BufferFreeze feature is not supported on the device

Description

This function identifies whether the BufferFreeze feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PingPongFreeze
- PLIB_USB_PingPongUnfreeze
- PLIB_USB_PingPongReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBufferFreeze(USB_MODULE_ID index)

PLIB_USB_ExistsDeviceAddress Function

Identifies whether the DeviceAddress feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsDeviceAddress(USB_MODULE_ID index);
```

Returns

- true - The DeviceAddress feature is supported on the device
- false - The DeviceAddress feature is not supported on the device

Description

This function identifies whether the DeviceAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_DeviceAddressSet
- PLIB_USB_DeviceAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsDeviceAddress(USB_MODULE_ID index)

PLIB_USB_ExistsEP0LowSpeedConnect Function

Identifies whether the EP0LowSpeedConnect feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEP0LowSpeedConnect(USB_MODULE_ID index);
```

Returns

- true - The EP0LowSpeedConnect feature is supported on the device
- false - The EP0LowSpeedConnect feature is not supported on the device

Description

This function identifies whether the EP0LowSpeedConnect feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EP0LSDirectConnectEnable
- PLIB_USB_EP0LSDirectConnectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEP0LowSpeedConnect(USB_MODULE_ID index)

PLIB_USB_ExistsEP0NAKRetry Function

Identifies whether the EP0NAKRetry feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEP0NAKRetry(USB_MODULE_ID index);
```

Returns

- true - The EP0NAKRetry feature is supported on the device
- false - The EP0NAKRetry feature is not supported on the device

Description

This function identifies whether the EP0NAKRetry feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EP0NakRetryEnable
- PLIB_USB_EP0NakRetryDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEP0NAKRetry(USB_MODULE_ID index)

PLIB_USB_ExistsEPnRxEnable Function

Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEPnRxEnable(USB_MODULE_ID index);
```

Returns

- true - The EPnRxEnableEnhanced feature is supported on the device
- false - The EPnRxEnableEnhanced feature is not supported on the device

Description

This function identifies whether the EPnRxEnableEnhanced feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EPnRxEnable
- PLIB_USB_EPnRxDisable
- PLIB_USB_EPnTxEnable
- PLIB_USB_EPnTxDisable
- PLIB_USB_EPnHandshakeEnable
- PLIB_USB_EPnHandshakeDisable
- PLIB_USB_EPnControlTransferEnable
- PLIB_USB_EPnControlTransferDisable
- PLIB_USB_EPnIsStalled
- PLIB_USB_EPnStallClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEPnRxEnable(USB_MODULE_ID index)

PLIB_USB_ExistsEPnTxRx Function

Identifies whether the EPnTxRx feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEPnTxRx(USB_MODULE_ID index);
```

Returns

- true - The EPnTxRx feature is supported on the device
- false - The EPnTxRx feature is not supported on the device

Description

This function identifies whether the EPnTxRx feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EPnTxSelect
- PLIB_USB_EPnRxSelect
- PLIB_USB_EPnTxRxSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEPnTxRx(USB_MODULE_ID index)

PLIB_USB_ExistsERR_Interrupt Function

Identifies whether the ERR_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsERR_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The ERR_Interrupt feature is supported on the device
- false - The ERR_Interrupt feature is not supported on the device

Description

This function identifies whether the ERR_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ErrorInterruptEnable
- PLIB_USB_ErrorInterruptDisable
- PLIB_USB_ErrorInterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsERR_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsERR_InterruptStatus Function

Identifies whether the ERR_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsERR_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The ERR_InterruptStatus feature is supported on the device
- false - The ERR_InterruptStatus feature is not supported on the device

Description

This function identifies whether the ERR_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ErrorInterruptFlagSet
- PLIB_USB_ErrorInterruptFlagClear

- PLIB_USB_ErrorInterruptFlagGet
- PLIB_USB_ErrorInterruptFlagAllGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsERR_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsEyePattern Function

Identifies whether the EyePattern feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEyePattern(USB_MODULE_ID index);
```

Returns

- true - The EyePattern feature is supported on the device
- false - The EyePattern feature is not supported on the device

Description

This function identifies whether the EyePattern feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EyePatternDisable
- PLIB_USB_EyePatternEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEyePattern(USB_MODULE_ID index)

PLIB_USB_ExistsFrameNumber Function

Identifies whether the FrameNumber feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsFrameNumber(USB_MODULE_ID index);
```

Returns

- true - The FrameNumber feature is supported on the device

- false - The FrameNumber feature is not supported on the device

Description

This function identifies whether the FrameNumber feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_FrameNumberGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsFrameNumber(USB_MODULE_ID index)

PLIB_USB_ExistsGEN_Interrupt Function

Identifies whether the GEN_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsGEN_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The GEN_Interrupt feature is supported on the device
- false - The GEN_Interrupt feature is not supported on the device

Description

This function identifies whether the GEN_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_InterruptEnable
- PLIB_USB_InterruptDisable
- PLIB_USB_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsGEN_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsGEN_InterruptStatus Function

Identifies whether the GEN_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsGEN_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The GEN_InterruptStatus feature is supported on the device
- false - The GEN_InterruptStatus feature is not supported on the device

Description

This function identifies whether the GEN_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_InterruptFlagSet
- PLIB_USB_InterruptFlagClear
- PLIB_USB_InterruptFlagGet
- PLIB_USB_InterruptFlagAllGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsGEN_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsHostBusyWithToken Function

Identifies whether the HostBusyWithToken feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsHostBusyWithToken(USB_MODULE_ID index);
```

Returns

- true - The HostBusyWithToken feature is supported on the device
- false - The HostBusyWithToken feature is not supported on the device

Description

This function identifies whether the HostBusyWithToken feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_IsBusyWithToken

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsHostBusyWithToken(USB_MODULE_ID index)

PLIB_USB_ExistsHostGeneratesReset Function

Identifies whether the HostGeneratesReset feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsHostGeneratesReset(USB_MODULE_ID index);
```

Returns

- true - The HostGeneratesReset feature is supported on the device
- false - The HostGeneratesReset feature is not supported on the device

Description

This function identifies whether the HostGeneratesReset feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ResetSignalEnable
- PLIB_USB_ResetSignalDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsHostGeneratesReset(USB_MODULE_ID index)

PLIB_USB_ExistsLastDirection Function

Identifies whether the LastDirection feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastDirection(USB_MODULE_ID index);
```

Returns

- true - The LastDirection feature is supported on the device
- false - The LastDirection feature is not supported on the device

Description

This function identifies whether the LastDirection feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionDirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastDirection(USB_MODULE_ID index)

PLIB_USB_ExistsLastEndpoint Function

Identifies whether the LastEndpoint feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastEndpoint(USB_MODULE_ID index);
```

Returns

- true - The LastEndpoint feature is supported on the device
- false - The LastEndpoint feature is not supported on the device

Description

This function identifies whether the LastEndpoint feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionEndPtGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastEndpoint(USB_MODULE_ID index)

PLIB_USB_ExistsLastPingPong Function

Identifies whether the LastPingPong feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastPingPong(USB_MODULE_ID index);
```

Returns

- true - The LastPingPong feature is supported on the device
- false - The LastPingPong feature is not supported on the device

Description

This function identifies whether the LastPingPong feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionPingPongStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastPingPong(USB_MODULE_ID index)

PLIB_USB_ExistsLastTransactionDetails Function

Identifies whether the LastTransactionDetails feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastTransactionDetails(USB_MODULE_ID index);
```

Returns

- true - The LastTransactionDetails feature is supported on the device
- false - The LastTransactionDetails feature is not supported on the device

Description

This function identifies whether the LastTransactionDetails feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionDetailsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastTransactionDetails(USB_MODULE_ID index)

PLIB_USB_ExistsLiveJState Function

Identifies whether the LiveJState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLiveJState(USB_MODULE_ID index);
```

Returns

- true - The LiveJState feature is supported on the device
- false - The LiveJState feature is not supported on the device

Description

This function identifies whether the LiveJState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_JStateIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLiveJState(USB_MODULE_ID index)

PLIB_USB_ExistsLiveSingleEndedZero Function

Identifies whether the LiveSingleEndedZero feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLiveSingleEndedZero(USB_MODULE_ID index);
```

Returns

- true - The LiveSingleEndedZero feature is supported on the device
- false - The LiveSingleEndedZero feature is not supported on the device

Description

This function identifies whether the LiveSingleEndedZero feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_SE0InProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLiveSingleEndedZero(USB_MODULE_ID index)

PLIB_USB_ExistsModuleBusy Function

Identifies whether the ModuleBusy feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsModuleBusy(USB_MODULE_ID index);
```

Returns

- true - The ModuleBusy feature is supported on the device
- false - The ModuleBusy feature is not supported on the device

Description

This function identifies whether the ModuleBusy feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ModuleIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsModuleBusy(USB_MODULE_ID index)

PLIB_USB_ExistsModulePower Function

Identifies whether the ModulePower feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsModulePower(USB_MODULE_ID index);
```

Returns

- true - The ModulePower feature is supported on the device
- false - The ModulePower feature is not supported on the device

Description

This function identifies whether the ModulePower feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_Enable
- PLIB_USB_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsModulePower(USB_MODULE_ID index)

PLIB_USB_ExistsNextTokenSpeed Function

Identifies whether the NextTokenSpeed feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsNextTokenSpeed(USB_MODULE_ID index);
```

Returns

- true - The NextTokenSpeed feature is supported on the device
- false - The NextTokenSpeed feature is not supported on the device

Description

This function identifies whether the NextTokenSpeed feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_TokenSpeedSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsNextTokenSpeed(USB_MODULE_ID index)

PLIB_USB_ExistsOnChipPullup Function

Identifies whether the OnChipPullup feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOnChipPullup(USB_MODULE_ID index);
```

Returns

- true - The OnChipPullup feature is supported on the device
- false - The OnChipPullup feature is not supported on the device

Description

This function identifies whether the OnChipPullup feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OnChipPullUpDisable
- PLIB_USB_OnChipPullUpEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOnChipPullup(USB_MODULE_ID index)

PLIB_USB_ExistsOnChipTransceiver Function

Identifies whether the OnChipTransceiver feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOnChipTransceiver(USB_MODULE_ID index);
```

Returns

- true - The OnChipTransceiver feature is supported on the device
- false - The OnChipTransceiver feature is not supported on the device

Description

This function identifies whether the OnChipTransceiver feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TransceiverEnable
- PLIB_USB_TransceiverDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOnChipTransceiver(USB_MODULE_ID index)

PLIB_USB_ExistsOpModeSelect Function

Identifies whether the OpModeSelect feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOpModeSelect(USB_MODULE_ID index);
```

Returns

- true - The OpModeSelect feature is supported on the device
- false - The OpModeSelect feature is not supported on the device

Description

This function identifies whether the OpModeSelect feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OperatingModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOpModeSelect(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_ASessionValid Function

Identifies whether the OTG_ASessionValid feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_ASessionValid(USB_MODULE_ID index);
```

Returns

- true - The OTG_ASessionValid feature is supported on the device
- false - The OTG_ASessionValid feature is not supported on the device

Description

This function identifies whether the OTG_ASessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_VBusValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_ASessionValid(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_BSessionEnd Function

Identifies whether the OTG_BSessionEnd feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_BSessionEnd(USB_MODULE_ID index);
```

Returns

- true - The OTG_BSessionEnd feature is supported on the device
- false - The OTG_BSessionEnd feature is not supported on the device

Description

This function identifies whether the OTG_BSessionEnd feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_BSessionHasEnded

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_BSessionEnd(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_IDPinState Function

Identifies whether the OTG_IDPinState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_IDPinState(USB_MODULE_ID index);
```

Returns

- true - The OTG_IDPinState feature is supported on the device
- false - The OTG_IDPinState feature is not supported on the device

Description

This function identifies whether the OTG_IDPinState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_IDPinStateIsTypeA

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_IDPinState(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_Interrupt Function

Identifies whether the OTG_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The OTG_Interrupt feature is supported on the device
- false - The OTG_Interrupt feature is not supported on the device

Description

This function identifies whether the OTG_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_InterruptEnable
- PLIB_USB_OTG_InterruptDisable
- PLIB_USB_OTG_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_InterruptStatus Function

Identifies whether the OTG_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The OTG_InterruptStatus feature is supported on the device
- false - The OTG_InterruptStatus feature is not supported on the device

Description

This function identifies whether the OTG_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_InterruptFlagSet
- PLIB_USB_OTG_InterruptFlagClear
- PLIB_USB_OTG_InterruptFlagGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_LineState Function

Identifies whether the OTG_LineState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_LineState(USB_MODULE_ID index);
```

Returns

- true - The OTG_LineState feature is supported on the device
- false - The OTG_LineState feature is not supported on the device

Description

This function identifies whether the OTG_LineState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_LineStateIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_LineState(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_PullUpPullDown Function

Identifies whether the OTG_PullUpPullDown feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_PullUpPullDown(USB_MODULE_ID index);
```

Returns

- true - The OTG_PullUpPullDown feature is supported on the device
- false - The OTG_PullUpPullDown feature is not supported on the device

Description

This function identifies whether the OTG_PullUpPullDown feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_PullUpPullDownSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_PullUpPullDown(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_SessionValid Function

Identifies whether the OTG_SessionValid feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_SessionValid(USB_MODULE_ID index);
```

Returns

- true - The OTG_SessionValid feature is supported on the device
- false - The OTG_SessionValid feature is not supported on the device

Description

This function identifies whether the OTG_SessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_SessionValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_SessionValid(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusCharge Function

Identifies whether the OTG_VbusCharge feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusCharge(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusCharge feature is supported on the device
- false - The OTG_VbusCharge feature is not supported on the device

Description

This function identifies whether the OTG_VbusCharge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusChargeEnable
- PLIB_USB_OTG_VBusChargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusCharge(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusDischarge Function

Identifies whether the OTG_VbusDischarge feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusDischarge(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusDischarge feature is supported on the device
- false - The OTG_VbusDischarge feature is not supported on the device

Description

This function identifies whether the OTG_VbusDischarge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusDischargeEnable
- PLIB_USB_OTG_VBusDischargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusDischarge(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusPowerOnOff Function

Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusPowerOnOff(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusPowerOnOff feature is supported on the device
- false - The OTG_VbusPowerOnOff feature is not supported on the device

Description

This function identifies whether the OTG_VbusPowerOnOff feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusPowerOff
- PLIB_USB_OTG_VBusPowerOn

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusPowerOnOff(USB_MODULE_ID index)

PLIB_USB_ExistsPacketTransfer Function

Identifies whether the PacketTransfer feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsPacketTransfer(USB_MODULE_ID index);
```

Returns

- true - The PacketTransfer feature is supported on the device
- false - The PacketTransfer feature is not supported on the device

Description

This function identifies whether the PacketTransfer feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PacketTransferIsDisabled
- PLIB_USB_PacketTransferEnable
- PLIB_USB_PacketTransferDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsPacketTransfer(USB_MODULE_ID index)

PLIB_USB_ExistsPingPongMode Function

Identifies whether the PingPongMode feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsPingPongMode(USB_MODULE_ID index);
```

Returns

- true - The PingPongMode feature is supported on the device
- false - The PingPongMode feature is not supported on the device

Description

This function identifies whether the PingPongMode feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PingPongModeSelect
- PLIB_USB_PingPongModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsPingPongMode(USB_MODULE_ID index)

PLIB_USB_ExistsResumeSignaling Function

Identifies whether the ResumeSignaling feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsResumeSignaling(USB_MODULE_ID index);
```

Returns

- true - The ResumeSignaling feature is supported on the device
- false - The ResumeSignaling feature is not supported on the device

Description

This function identifies whether the ResumeSignaling feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ResumeSignalingEnable
- PLIB_USB_ResumeSignalingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsResumeSignaling(USB_MODULE_ID index)

PLIB_USB_ExistsSleepEntryGuard Function

Identifies whether the SleepEntryGuard feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSleepEntryGuard(USB_MODULE_ID index);
```

Returns

- true - The SleepEntryGuard feature is supported on the device
- false - The SleepEntryGuard feature is not supported on the device

Description

This function identifies whether the SleepEntryGuard feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SleepGuardEnable
- PLIB_USB_SleepGuardDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSleepEntryGuard(USB_MODULE_ID index)

PLIB_USB_ExistsSOFThreshold Function

Identifies whether the SOFThreshold feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSOFThreshold(USB_MODULE_ID index);
```

Returns

- true - The SOFThreshold feature is supported on the device
- false - The SOFThreshold feature is not supported on the device

Description

This function identifies whether the SOFThreshold feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SOFThresholdGet
- PLIB_USB_SOFThresholdSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSOFThreshold(USB_MODULE_ID index)

PLIB_USB_ExistsSpeedControl Function

Identifies whether the SpeedControl feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSpeedControl(USB_MODULE_ID index);
```

Returns

- true - The SpeedControl feature is supported on the device
- false - The SpeedControl feature is not supported on the device

Description

This function identifies whether the SpeedControl feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_FullSpeedEnable
- PLIB_USB_FullSpeedDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSpeedControl(USB_MODULE_ID index)

PLIB_USB_ExistsStartOfFrames Function

Identifies whether the StartOfFrames feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsStartOfFrames(USB_MODULE_ID index);
```

Returns

- true - The StartOfFrames feature is supported on the device
- false - The StartOfFrames feature is not supported on the device

Description

This function identifies whether the StartOfFrames feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SOFEnable
- PLIB_USB_SOFDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsStartOfFrames(USB_MODULE_ID index)

PLIB_USB_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsStopInIdle(USB_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_StopInIdleEnable
- PLIB_USB_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsStopInIdle(USB_MODULE_ID index)

PLIB_USB_ExistsSuspend Function

Identifies whether the Suspend feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSuspend(USB_MODULE_ID index);
```

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Description

This function identifies whether the Suspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SuspendEnable
- PLIB_USB_SuspendDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSuspend(USB_MODULE_ID index)

PLIB_USB_ExistsTokenEP Function

Identifies whether the TokenEP feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsTokenEP(USB_MODULE_ID index);
```

Returns

- true - The TokenEP feature is supported on the device
- false - The TokenEP feature is not supported on the device

Description

This function identifies whether the TokenEP feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TokenEPGet
- PLIB_USB_TokenEPSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsTokenEP(USB_MODULE_ID index)

PLIB_USB_ExistsTokenPID Function

Identifies whether the TokenPID feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsTokenPID(USB_MODULE_ID index);
```

Returns

- true - The TokenPID feature is supported on the device
- false - The TokenPID feature is not supported on the device

Description

This function identifies whether the TokenPID feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TokenPIDGet
- PLIB_USB_TokenPIDSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsTokenPID(USB_MODULE_ID index)

PLIB_USB_ExistsUOEMonitor Function

Identifies whether the UOEMonitor feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsUOEMonitor(USB_MODULE_ID index);
```

Returns

- true - The UOEMonitor feature is supported on the device
- false - The UOEMonitor feature is not supported on the device

Description

This function identifies whether the UOEMonitor feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_UOEMonitorEnable
- PLIB_USB_UOEMonitorDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsUOEMonitor(USB_MODULE_ID index)

q) Data Types and Constants

USB_BUFFER_DATA01 Enumeration

Provides enumeration data toggle for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_DATA0,
  USB_BUFFER_DATA1
} USB_BUFFER_DATA01;
```

Members

Members	Description
USB_BUFFER_DATA0	DATA0/1 = 0
USB_BUFFER_DATA1	DATA0/1 = 1

Description

USB Endpoint Buffer Data Toggle Enumeration

This data type provides enumeration data toggle for a buffer.

Remarks

None.

USB_BUFFER_DIRECTION Enumeration

Provides enumeration transmit/receive direction for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_RX,
  USB_BUFFER_TX
} USB_BUFFER_DIRECTION;
```

Members

Members	Description
USB_BUFFER_RX	Receive
USB_BUFFER_TX	Transmit

Description

USB Endpoint Buffer Direction Enumeration

This data type provides enumeration transmit/receive direction for a buffer.

Remarks

None.

USB_BUFFER_PING_PONG Enumeration

Enumerates the ping-pong buffer (Even vs. Odd).

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_EVEN,
  USB_BUFFER_ODD
} USB_BUFFER_PING_PONG;
```

Members

Members	Description
USB_BUFFER_EVEN	Even Buffer
USB_BUFFER_ODD	Odd Buffer

Description

Enumeration of USB Buffer Ping-Pong

This data type enumerates the ping-pong buffer (Even vs. Odd).

Remarks

None.

USB_BUFFER_SCHEDULE_DATA01 Enumeration

Provides enumeration data toggle for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_DONTCHANGE,
  USB_BUFFER_SET_DATA0,
```

```
  USB_BUFFER_SET_DATA1
} USB_BUFFER_SCHEDULE_DATA01;
```

Members

Members	Description
USB_BUFFER_DONTCHANGE	Don't Change DATA0/1
USB_BUFFER_SET_DATA0	DATA0/1 = 0
USB_BUFFER_SET_DATA1	DATA0/1 = 1

Description

USB Endpoint Buffer Data Toggle Enumeration for Buffer Schedulint

This data type provides enumeration data toggle for a buffer.

Remarks

None.

USB_EP_TXRX Enumeration

Provides enumeration transmit/receive setup for an endpoint.

File

plib_usb.h

C

```
typedef enum {
  USB_EP_NOTXRX,
  USB_EP_RX,
  USB_EP_TX,
  USB_EP_TX_RX
} USB_EP_TXRX;
```

Members

Members	Description
USB_EP_NOTXRX	Nothing enabled for endpoint
USB_EP_RX	Receive enabled for endpoint
USB_EP_TX	Transmit enabled for endpoint
USB_EP_TX_RX	Transmit and Receive enabled for endpoint

Description

Enumeration of USB Endpoint Transmit/Receive Setup

This data type provides enumeration transmit/receive setup for an endpoint.

Remarks

None.

USB_OPMODES Enumeration

Provides enumeration of operating modes supported by USB.

File

plib_usb.h

C

```
typedef enum {
  USB_OPMODE_NONE,
  USB_OPMODE_DEVICE,
  USB_OPMODE_HOST,
  USB_OPMODE_OTG
} USB_OPMODES;
```

Members

Members	Description
USB_OPMODE_NONE	None
USB_OPMODE_DEVICE	Device
USB_OPMODE_HOST	Host
USB_OPMODE_OTG	OTG

Description

USB Operating Modes Enumeration

This data type provides enumeration of the operating modes supported by the USB module.

Remarks

None.

USB_OTG_INTERRUPTS Enumeration

Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.

File

plib_usb.h

C

```
typedef enum {
  USB_OTG_INT_ADEVICE_VBUS_VALID,
  USB_OTG_INT_OTG_RESERVED,
  USB_OTG_INT_BDEVICE_SESSION_END,
  USB_OTG_INT_SESSION_VALID,
  USB_OTG_INT_ACTIVITY_DETECT,
  USB_OTG_INT_STABLE_LINE_STATE,
  USB_OTG_INT_ONE_MS_TIMEOUT,
  USB_OTG_INT_ID_STATE_CHANGE,
  USB_OTG_INT_ANY,
  USB_OTG_INT_ALL
} USB_OTG_INTERRUPTS;
```

Members

Members	Description
USB_OTG_INT_ADEVICE_VBUS_VALID	State of (VBUS > Va_vbus_vld) on the A device has changed
USB_OTG_INT_OTG_RESERVED	Reserved. Don't use.
USB_OTG_INT_BDEVICE_SESSION_END	State of (VBUS < Vb_sess_end) on the B device has changed
USB_OTG_INT_SESSION_VALID	State of (VBUS > Va_sess_vld) on the A or B devices has changed
USB_OTG_INT_ACTIVITY_DETECT	Activity detected on the D+, D-, ID, or VBUS lines
USB_OTG_INT_STABLE_LINE_STATE	USB line state has been stable for 1 ms, but different from last time
USB_OTG_INT_ONE_MS_TIMEOUT	One millisecond timer has expired
USB_OTG_INT_ID_STATE_CHANGE	Change in state of ID pin detected.
USB_OTG_INT_ANY	All or Any of the above
USB_OTG_INT_ALL	All or Any of the above

Description

USB OTG Interrupts Enumeration

This data type provides enumeration of interrupts related to the USB OTG module.

Remarks

Not applicable if the USB OTG module is not enabled.

USB_OTG_PULL_UP_PULL_DOWN Enumeration

USB OTG pull-Up and pull-Down resistors for D+ and D- .

File

plib_usb.h

C

```
typedef enum {
  USB_OTG_DPLUS_PULLUP,
  USB_OTG_DMINUS_PULLUP,
  USB_OTG_DPLUS_PULLDN,
  USB_OTG_DMINUS_PULLDN
} USB_OTG_PULL_UP_PULL_DOWN;
```

Members

Members	Description
USB_OTG_DPLUS_PULLUP	D+ Pull-Up
USB_OTG_DMINUS_PULLUP	D- Pull-Up
USB_OTG_DPLUS_PULLDN	D+ Pull-Down
USB_OTG_DMINUS_PULLDN	D- Pull-Down

Description

Enumeration of Pull-Up and Pull-Down Resistors for OTG

This data type enumerates the OTG Pull-Up and Pull-Down resistors for D+ and D- .

Remarks

None.

USB_PID Enumeration

Legal PID values.

File

plib_usb.h

C

```
typedef enum {
  USB_PID_SETUP,
  USB_PID_IN,
  USB_PID_OUT
} USB_PID;
```

Members

Members	Description
USB_PID_SETUP	Setup token
USB_PID_IN	IN token
USB_PID_OUT	OUT token

Description

Enumeration of Legal Packet IDs (PIDs)

This data type enumerates the valid (i.e., legal) PID values. While the PID field is four bits long, only these values are legal and should be used. The use of any other values may cause unpredictable results.

USB_PING_PONG_MODE Enumeration

Supports the four modes of ping-pong buffering.

File

plib_usb.h

C

```
typedef enum {
  USB_PING_PONG_ALL_BUT_EP0,
  USB_PING_PONG_FULL_PING_PONG,
  USB_PING_PONG_EP0_OUT_ONLY,
```

```
  USB_PING_PONG_NO_PING_PONG
} USB_PING_PONG_MODE;
```

Members

Members	Description
USB_PING_PONG_ALL_BUT_EP0	Ping-Pong buffering on all endpoints except Endpoint Zero
USB_PING_PONG_FULL_PING_PONG	Ping-Pong buffering on all endpoints
USB_PING_PONG_EP0_OUT_ONLY	Ping-Pong buffering on just Endpoint Zero transmit
USB_PING_PONG_NO_PING_PONG	No ping-pong buffering

Description

Enumeration of USB Ping-Pong Modes

This data type supports the four modes of ping-pong buffering.

Remarks

None.

USB_PING_PONG_STATE Enumeration

Decodes which buffer (Even vs. Odd) was used for the last transaction.

File

plib_usb.h

C

```
typedef enum {
  USB_PING_PONG_EVEN,
  USB_PING_PONG_ODD
} USB_PING_PONG_STATE;
```

Members

Members	Description
USB_PING_PONG_EVEN	Last transaction on Even Buffer
USB_PING_PONG_ODD	Last transaction on Odd Buffer

Description

Enumeration of USB Ping-Pong Indicator

This data type decodes which buffer (Even vs. Odd) was used for the last transaction.

Remarks

None.

USB_TOKEN_SPEED Enumeration

Provides enumeration of available token speeds.

File

plib_usb.h

C

```
typedef enum {
  USB_LOWSPEED_TOKENS,
  USB_FULLSPEED_TOKENS
} USB_TOKEN_SPEED;
```

Members

Members	Description
USB_LOWSPEED_TOKENS	Low Speed Tokens
USB_FULLSPEED_TOKENS	Full Speed Tokens

Description

USB Token Speeds Enumeration

This data type provides enumeration of available token speeds.

Remarks

For Host mode only.

USB_MAX_EP_NUMBER Macro

Maximum number of endpoints supported (not including EP0).

File

plib_usb.h

C

```
#define USB_MAX_EP_NUMBER 15
```

Description

Maximum number of endpoints

This constant defines the maximum number of endpoints supported (not including EP0). It is used in dimensioning the Buffer Descriptor Table (BDT) array.

Files

Files

Name	Description
plib_usb.h	USB Peripheral Library Interface Header for common definitions

Description

This section lists the source and header files used by the library.

plib_usb.h

USB Peripheral Library Interface Header for common definitions

Enumerations

	Name	Description
	USB_BUFFER_DATA01	Provides enumeration data toggle for a buffer.
	USB_BUFFER_DIRECTION	Provides enumeration transmit/receive direction for a buffer.
	USB_BUFFER_PING_PONG	Enumerates the ping-pong buffer (Even vs. Odd).
	USB_BUFFER_SCHEDULE_DATA01	Provides enumeration data toggle for a buffer.
	USB_EP_TXRX	Provides enumeration transmit/receive setup for an endpoint.
	USB_OPMODES	Provides enumeration of operating modes supported by USB.
	USB_OTG_INTERRUPTS	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	USB_OTG_PULL_UP_PULL_DOWN	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	USB_PID	Legal PID values.
	USB_PING_PONG_MODE	Supports the four modes of ping-pong buffering.
	USB_PING_PONG_STATE	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	USB_TOKEN_SPEED	Provides enumeration of available token speeds.

Functions

	Name	Description
	PLIB_USB_ActivityPending	Returns whether or not USB activity is pending.
	PLIB_USB_AllInterruptEnable	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	PLIB_USB_AutoSuspendDisable	Disables USB OTG Auto-suspend mode.
	PLIB_USB_AutoSuspendEnable	Enables USB Auto-suspend mode.

	PLIB_USB_BDTBaseAddressGet	Returns the base address of the Buffer Descriptor Table.
	PLIB_USB_BDTBaseAddressSet	Sets the base address for the Buffer Descriptor Table for PIC32 devices.
	PLIB_USB_BufferAddressGet	Gets the memory address of an endpoint buffer.
	PLIB_USB_BufferAddressSet	Sets the endpoint buffer address.
	PLIB_USB_BufferAllCancelReleaseToUSB	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	PLIB_USB_BufferByteCountGet	Returns the endpoint buffer byte count.
	PLIB_USB_BufferByteCountSet	Sets the buffer byte count.
	PLIB_USB_BufferCancelReleaseToUSB	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	PLIB_USB_BufferClearAll	Clears (zeros out) entries in the Buffer Descriptor Table.
	PLIB_USB_BufferClearAllDTSEnable	Clears the endpoint descriptor entry and enables data toggle synchronization.
	PLIB_USB_BufferDataToggleGet	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	PLIB_USB_BufferDataToggleSelect	Sets the endpoint buffer to DATA0 or DATA1.
	PLIB_USB_BufferDataToggleSyncDisable	Disables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferDataToggleSyncEnable	Enables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferEP0RxStatusInitialize	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	PLIB_USB_BufferIndexGet	Gets the Buffer Descriptor Table index for a buffer.
	PLIB_USB_BufferPIDBitsClear	Clears the Buffer Status bits in the Buffer Descriptor Table.
	PLIB_USB_BufferPIDGet	Returns the token packet ID (PID) from the endpoint buffer status.
	PLIB_USB_BufferReleasedToSW	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	PLIB_USB_BufferReleaseToUSB	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	PLIB_USB_BufferSchedule	Hands over a buffer to the USB module along with the buffer address and byte count.
	PLIB_USB_BufferStallDisable	Disables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallEnable	Enables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallGet	Returns the buffer stall status for an endpoint/direction/ping-pong.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_EP0HostSetup	Sends token to the specified address.
	PLIB_USB_EP0LSDirectConnectDisable	Disables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0LSDirectConnectEnable	Enables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0NakRetryDisable	Disables retrying of NAKed transactions.
	PLIB_USB_EP0NakRetryEnable	Enables retrying NAK'd transactions for Endpoint 0.
	PLIB_USB_EPnAttributesClear	Clears the set attributes of the specified endpoint.
	PLIB_USB_EPnAttributesSet	Configures attributes of the endpoint such as direction, handshake capability and direction.
	PLIB_USB_EPnControlTransferDisable	Disables endpoint control transfers.
	PLIB_USB_EPnControlTransferEnable	Enables endpoint control transfers.
	PLIB_USB_EPnDirectionDisable	Disables the specified endpoint direction.
	PLIB_USB_EPnHandshakeDisable	Disables endpoint handshaking.
	PLIB_USB_EPnHandshakeEnable	Enables endpoint handshaking.
	PLIB_USB_EPnIsStalled	Tests whether the endpoint epValue is stalled.
	PLIB_USB_EPnRxDisable	Disables an endpoint's ability to process IN tokens.
	PLIB_USB_EPnRxEnable	Enables an endpoint to process IN tokens.
	PLIB_USB_EPnRxSelect	Selects receive capabilities of an endpoint.
	PLIB_USB_EPnStallClear	Clears an endpoint's stalled flag.
	PLIB_USB_EPnTxDisable	Disables an endpoint's ability to process OUT tokens.
	PLIB_USB_EPnTxEnable	Enables an endpoint to process OUT tokens.
	PLIB_USB_EPnTxRxSelect	Selects transmit and/or receive capabilities of an endpoint.
	PLIB_USB_EPnTxSelect	Selects transmit capabilities of an endpoint.
	PLIB_USB_ErrorInterruptDisable	Disables an error interrupt for the USB module.

	PLIB_USB_ErrorInterruptEnable	Enables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptFlagAllGet	Returns a logically ORed bit map of active error interrupt flags.
	PLIB_USB_ErrorInterruptFlagClear	Clears an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.
	PLIB_USB_ExistsModuleBusy	Identifies whether the ModuleBusy feature exists on the USB module.
	PLIB_USB_ExistsModulePower	Identifies whether the ModulePower feature exists on the USB module.
	PLIB_USB_ExistsNextTokenSpeed	Identifies whether the NextTokenSpeed feature exists on the USB module.
	PLIB_USB_ExistsOnChipPullup	Identifies whether the OnChipPullup feature exists on the USB module.
	PLIB_USB_ExistsOnChipTransceiver	Identifies whether the OnChipTransceiver feature exists on the USB module.
	PLIB_USB_ExistsOpModeSelect	Identifies whether the OpModeSelect feature exists on the USB module.
	PLIB_USB_ExistsOTG_ASessionValid	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_BSessionEnd	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	PLIB_USB_ExistsOTG_IDPinState	Identifies whether the OTG_IDPinState feature exists on the USB module.
	PLIB_USB_ExistsOTG_Interrupt	Identifies whether the OTG_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsOTG_InterruptStatus	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsOTG_LineState	Identifies whether the OTG_LineState feature exists on the USB module.
	PLIB_USB_ExistsOTG_PullUpPullDown	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	PLIB_USB_ExistsOTG_SessionValid	Identifies whether the OTG_SessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusCharge	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusDischarge	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusPowerOnOff	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	PLIB_USB_ExistsPacketTransfer	Identifies whether the PacketTransfer feature exists on the USB module.
	PLIB_USB_ExistsPingPongMode	Identifies whether the PingPongMode feature exists on the USB module.
	PLIB_USB_ExistsResumeSignaling	Identifies whether the ResumeSignaling feature exists on the USB module.
	PLIB_USB_ExistsSleepEntryGuard	Identifies whether the SleepEntryGuard feature exists on the USB module.
	PLIB_USB_ExistsSOFThreshold	Identifies whether the SOFThreshold feature exists on the USB module.
	PLIB_USB_ExistsSpeedControl	Identifies whether the SpeedControl feature exists on the USB module.
	PLIB_USB_ExistsStartOfFrames	Identifies whether the StartOfFrames feature exists on the USB module.

	PLIB_USB_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USB module.
	PLIB_USB_ExistsSuspend	Identifies whether the Suspend feature exists on the USB module.
	PLIB_USB_ExistsTokenEP	Identifies whether the TokenEP feature exists on the USB module.
	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.
	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.
	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_JStateIsActive	Live differential receiver J State flag.
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.
	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.
	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolaritiyActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_ResetSignalDisable	Disables reset signaling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signaling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.
	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB_TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB_TokenEPSet	Sets the Endpoint address for a host transaction.
	PLIB_USB_TokenPIDGet	Returns the token transaction type.
	PLIB_USB_TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB_TokenSend	Sends token to the specified address.
	PLIB_USB_TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.
	PLIB_USB_TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB_TransceiverEnable	Enables the on-chip transceiver.
	PLIB_USB_UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB_UOEMonitorEnable	Enables the OE signal output.
	PLIB_USB_VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB_VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB_VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB_VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB_VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB_VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

Macros

	Name	Description
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

USB Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Peripheral Library.

File Name

plib_usb.h

Company

Microchip Technology Inc.

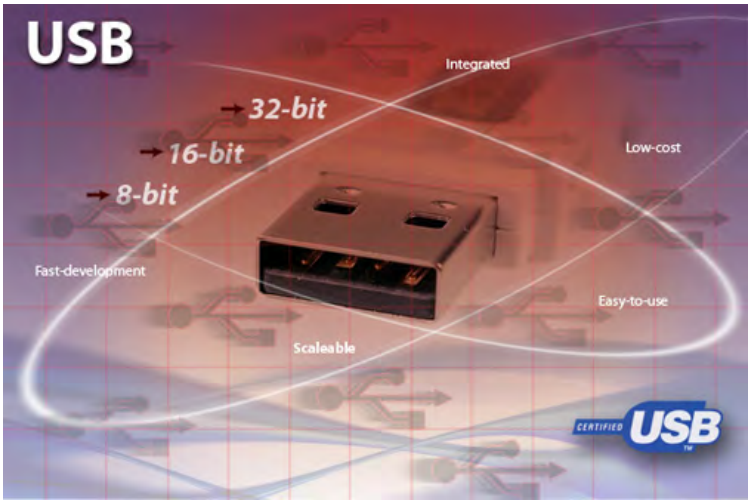
USBHS Peripheral Library

This section describes the Hi-Speed USB (USBHS) Peripheral Library.

Introduction

This library provides a low-level abstraction of the USBHS module on Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description



USB Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host). There are three possible module implementations: host, device and OTG dual role. The user should have an understanding of the USB documents available on the USB implementers web site (www.usb.org).

Features of the Hi-Speed USB (USBHS) Module

- Operates either as a function controller of a Hi-Speed/Full-Speed USB device or as the host/device in a point-to-point or multi-point communications with other USB function
- Supports OTG communications with on or more Hi-Speed, Full-Speed, or Low-Speed devices
- Provides soft connect/disconnect.
- In addition to Endpoint Zero, supports seven transmit and seven receive endpoints
- Dynamic FIFO sizing for Endpoints 1-7. (Endpoint Zero FIFO fixed at 64 bytes.) FIFOs use module-internal SRAM.
- Module-internal eight channel DMA with access to all FIFOs
- All host transaction scheduling supported in hardware
- Supports Link Power Management

Using the Library

This topic describes the basic architecture of the USBHS Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usbhs.h`

The interface to the USBHS Peripheral Library is defined in the `plib_usbhs.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USBHS Peripheral Library must include `peripheral.h`.

Library File:

The USBHS Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USBHS module.

Library Interface Section	Description
USBHS Setup Functions	This section provides functions to perform general USBHS peripheral setup such as functions to set USB Speed, control on-chip pull ups, etc.
Endpoints Functions	This section provides functions that allow the application to manage endpoints.
Interrupts Functions	This section provides functions that allow the application to enable, disable and query the status interrupts in the USBHS peripheral.
Device Functions	This section provides functions that are required to operate the USBHS module while in Device mode.
Host Functions	This section provides functions that are required to operate the USBHS module while in Host mode.
Status Functions	This section provides functions that read the status of the USBHS module.
VBUS Support Functions	This section provides functions that allow VBUS level monitoring and VBUS boost PWM module control.
Feature Existence Functions	This section provides functions that identify whether a particular feature exists on the USBHS module.

How the Library Works

Provides information on how the library works.

Configuring the Library

The library is configured for the supported High-Speed USB module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) USBHS Setup Functions

	Name	Description
	PLIB_USBHS_HighSpeedDisable	This is function PLIB_USBHS_HighSpeedDisable.
	PLIB_USBHS_HighSpeedEnable	Sets the operation speed of the USB Module.
	PLIB_USBHS_ResetDisable	This is function PLIB_USBHS_ResetDisable.
	PLIB_USBHS_ResetEnable	This is function PLIB_USBHS_ResetEnable.
	PLIB_USBHS_ResumeDisable	This is function PLIB_USBHS_ResumeDisable.
	PLIB_USBHS_ResumeEnable	This is function PLIB_USBHS_ResumeEnable.
	PLIB_USBHS_SessionDisable	This is function PLIB_USBHS_SessionDisable.
	PLIB_USBHS_SessionEnable	This is function PLIB_USBHS_SessionEnable.
	PLIB_USBHS_SoftResetDisable	Disables soft reset.
	PLIB_USBHS_SoftResetEnable	Enables soft reset.
	PLIB_USBHS_SuspendDisable	This is function PLIB_USBHS_SuspendDisable.
	PLIB_USBHS_SuspendEnable	This is function PLIB_USBHS_SuspendEnable.
	PLIB_USBHS_DMAOperationEnable	This is function PLIB_USBHS_DMAOperationEnable.
	PLIB_USBHS_TestModeEnter	This is function PLIB_USBHS_TestModeEnter.
	PLIB_USBHS_TestModeExit	This is function PLIB_USBHS_TestModeExit.
	PLIB_USBHS_PhyIDMonitoringDisable	This is function PLIB_USBHS_PhyIDMonitoringDisable.
	PLIB_USBHS_PhyIDMonitoringEnable	This is function PLIB_USBHS_PhyIDMonitoringEnable.
	PLIB_USBHS_USBIDOverrideDisable	This is function PLIB_USBHS_USBIDOverrideDisable.
	PLIB_USBHS_USBIDOverrideEnable	This is function PLIB_USBHS_USBIDOverrideEnable.
	PLIB_USBHS_USBIDOverrideValueSet	This is function PLIB_USBHS_USBIDOverrideValueSet.
	PLIB_USBHS_IsBDevice	This is function PLIB_USBHS_IsBDevice.

b) Endpoints Functions

	Name	Description
	PLIB_USBHS_Endpoint0FIFOFlush	This is function PLIB_USBHS_Endpoint0FIFOFlush.
	PLIB_USBHS_Endpoint0SetupPacketLoad	Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.
	PLIB_USBHS_Endpoint0SetupPacketUnload	This is function PLIB_USBHS_Endpoint0SetupPacketUnload.
	PLIB_USBHS_EndpointFIFOLoad	Loads the endpoint FIFO with provided data and then enables the endpoint transmit.
	PLIB_USBHS_EndpointFIFOUnload	Unloads the endpoint FIFO.
	PLIB_USBHS_EndpointRxFIFOFlush	This is function PLIB_USBHS_EndpointRxFIFOFlush.
	PLIB_USBHS_EndpointRxRequestClear	This is function PLIB_USBHS_EndpointRxRequestClear.
	PLIB_USBHS_EndpointRxRequestEnable	This is function PLIB_USBHS_EndpointRxRequestEnable.
	PLIB_USBHS_EndpointTxFIFOFlush	This is function PLIB_USBHS_EndpointTxFIFOFlush.
	PLIB_USBHS_EP0DataEndSet	This is function PLIB_USBHS_EP0DataEndSet.
	PLIB_USBHS_EP0INHandshakeClear	This is function PLIB_USBHS_EP0INHandshakeClear.
	PLIB_USBHS_EP0INHandshakeSend	This is function PLIB_USBHS_EP0INHandshakeSend.
	PLIB_USBHS_EP0INTokenSend	This is function PLIB_USBHS_EP0INTokenSend.
	PLIB_USBHS_EP0OUTHandshakeSend	This is function PLIB_USBHS_EP0OUTHandshakeSend.
	PLIB_USBHS_EP0RxPktRdyServiced	This is function PLIB_USBHS_EP0RxPktRdyServiced.
	PLIB_USBHS_EP0RxPktRdyServicedDataEnd	This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.
	PLIB_USBHS_EP0SentStallClear	This is function PLIB_USBHS_EP0SentStallClear.
	PLIB_USBHS_EP0SetupEndServiced	This is function PLIB_USBHS_EP0SetupEndServiced.
	PLIB_USBHS_EP0StallDisable	This is function PLIB_USBHS_EP0StallDisable.
	PLIB_USBHS_EP0StallEnable	This is function PLIB_USBHS_EP0StallEnable.
	PLIB_USBHS_EP0StatusClear	This is function PLIB_USBHS_EP0StatusClear.
	PLIB_USBHS_EP0StatusGet	This is function PLIB_USBHS_EP0StatusGet.
	PLIB_USBHS_EP0TxPktRdy	This is function PLIB_USBHS_EP0TxPktRdy.
	PLIB_USBHS_EP0TxPktRdyDataEnd	This is function PLIB_USBHS_EP0TxPktRdyDataEnd.
	PLIB_USBHS_HostRxEndpointConfigure	This is function PLIB_USBHS_HostRxEndpointConfigure.
	PLIB_USBHS_HostTxEndpointConfigure	This is function PLIB_USBHS_HostTxEndpointConfigure.
	PLIB_USBHS_RxEPINTokenSend	This is function PLIB_USBHS_RxEPINTokenSend.
	PLIB_USBHS_RxEPStatusClear	This is function PLIB_USBHS_RxEPStatusClear.
	PLIB_USBHS_RxEPStatusGet	This is function PLIB_USBHS_RxEPStatusGet.
	PLIB_USBHS_TxEPStatusClear	This is function PLIB_USBHS_TxEPStatusClear.
	PLIB_USBHS_TxEPStatusGet	This is function PLIB_USBHS_TxEPStatusGet.
	PLIB_USBHS_GetEP0CSRAddress	This is function PLIB_USBHS_GetEP0CSRAddress.
	PLIB_USBHS_GetEP0FIFOAddress	This is function PLIB_USBHS_GetEP0FIFOAddress.
	PLIB_USBHS_LoadEPInIndex	This is function PLIB_USBHS_LoadEPInIndex.

c) Interrupts Functions

	Name	Description
	PLIB_USBHS_TxInterruptDisable	Disables a TX endpoint interrupt source for the USB module.
	PLIB_USBHS_TxInterruptEnable	Enables a TX endpoint Interrupt for the USB module.
	PLIB_USBHS_GenInterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptFlagsGet	Gets general interrupt flags.
	PLIB_USBHS_InterruptEnableSet	Enables USB module event interrupts.
	PLIB_USBHS_DMAInterruptDisable	Disables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptEnable	Enables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptFlagsGet	Gets the DMA channel interrupt flags.
	PLIB_USBHS_TxInterruptFlagsGet	Gets the TX endpoint interrupt flags.
	PLIB_USBHS_RxInterruptDisable	Disables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptEnable	Enables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptFlagsGet	Gets RX endpoint interrupt flags.
	PLIB_USBHS_DMAInterruptGet	This is function PLIB_USBHS_DMAInterruptGet.

	PLIB_USBHS_GlobalInterruptDisable	This is function PLIB_USBHS_GlobalInterruptDisable.
	PLIB_USBHS_GlobalInterruptEnable	This is function PLIB_USBHS_GlobalInterruptEnable.

d) Device Functions

	Name	Description
	PLIB_USBHS_DeviceAddressGet	Returns the current USB device address.
	PLIB_USBHS_DeviceAddressSet	Sets the device address.
	PLIB_USBHS_DeviceAttach	This is function PLIB_USBHS_DeviceAttach.
	PLIB_USBHS_DeviceConnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceDetach	This is function PLIB_USBHS_DeviceDetach.
	PLIB_USBHS_DeviceDisconnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceEPFIFOLoad	This is function PLIB_USBHS_DeviceEPFIFOLoad.
	PLIB_USBHS_DeviceEPFIFOUnload	This is function PLIB_USBHS_DeviceEPFIFOUnload.
	PLIB_USBHS_DeviceRxEndpointConfigure	This is function PLIB_USBHS_DeviceRxEndpointConfigure.
	PLIB_USBHS_DeviceRxEndpointStallDisable	This is function PLIB_USBHS_DeviceRxEndpointStallDisable.
	PLIB_USBHS_DeviceRxEndpointStallEnable	This is function PLIB_USBHS_DeviceRxEndpointStallEnable.
	PLIB_USBHS_DeviceTxEndpointConfigure	This is function PLIB_USBHS_DeviceTxEndpointConfigure.
	PLIB_USBHS_DeviceTxEndpointPacketReady	This is function PLIB_USBHS_DeviceTxEndpointPacketReady.
	PLIB_USBHS_DeviceTxEndpointStallDisable	This is function PLIB_USBHS_DeviceTxEndpointStallDisable.
	PLIB_USBHS_DeviceTxEndpointStallEnable	This is function PLIB_USBHS_DeviceTxEndpointStallEnable.

e) Host Functions

	Name	Description
	PLIB_USBHS_HostRxEndpointDataToggleClear	This is function PLIB_USBHS_HostRxEndpointDataToggleClear.
	PLIB_USBHS_HostTxEndpointDataToggleClear	This is function PLIB_USBHS_HostTxEndpointDataToggleClear.

f) Status Functions

	Name	Description
	PLIB_USBHS_FullOrHighSpeedIsConnected	This is function PLIB_USBHS_FullOrHighSpeedIsConnected.
	PLIB_USBHS_HighSpeedIsConnected	This is function PLIB_USBHS_HighSpeedIsConnected.
	PLIB_USBHS_HostModeIsEnabled	This is function PLIB_USBHS_HostModeIsEnabled.
	PLIB_USBHS_ModuleSpeedGet	Returns the speed at which the module is operating.
	PLIB_USBHS_DMAErrorGet	This is function PLIB_USBHS_DMAErrorGet.
	PLIB_USBHS_GetReceiveDataCount	This is function PLIB_USBHS_GetReceiveDataCount.

g) VBUS Support Functions

	Name	Description
	PLIB_USBHS_VbusLevelGet	Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.
	PLIB_USBHS_VBUSLevelGet	This is function PLIB_USBHS_VBUSLevelGet.

h) Feature Existence Functions

	Name	Description
	PLIB_USBHS_ExistsEndpointFIFO	Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.
	PLIB_USBHS_ExistsEndpointOperations	This is function PLIB_USBHS_ExistsEndpointOperations.
	PLIB_USBHS_ExistsEP0Status	This is function PLIB_USBHS_ExistsEP0Status.
	PLIB_USBHS_ExistsHighSpeedSupport	This is function PLIB_USBHS_ExistsHighSpeedSupport.
	PLIB_USBHS_ExistsInterrupts	This is function PLIB_USBHS_ExistsInterrupts.
	PLIB_USBHS_ExistsModuleControl	This is function PLIB_USBHS_ExistsModuleControl.
	PLIB_USBHS_ExistsRxEPStatus	This is function PLIB_USBHS_ExistsRxEPStatus.
	PLIB_USBHS_ExistsSoftReset	This is function PLIB_USBHS_ExistsSoftReset.
	PLIB_USBHS_ExistsTxEPStatus	This is function PLIB_USBHS_ExistsTxEPStatus.
	PLIB_USBHS_ExistsClockResetControl	This is function PLIB_USBHS_ExistsClockResetControl.
	PLIB_USBHS_ExistsUSBIDControl	This is function PLIB_USBHS_ExistsUSBIDControl.

i) Data Types and Constants

	Name	Description
	USBHS_CONFIGURATION	Provides the enumeration Configuration bits.
	USBHS_DATA01	Provides an enumeration data toggle for a packet.
	USBHS_DMA_ASSERT_TIMING	Provides enumeration DMA assertion timing (early versus late).
	USBHS_DMA_BURST_MODE	Provides enumeration of all DMA burst modes.
	USBHS_DMA_INTERRUPT	Provides enumeration of interrupts for DMA channels 0-7.
	USBHS_DMA_REQUEST_MODE	Used as an argument to set DMA request mode.
	USBHS_DMA_TRANSFER_MODE	Provides enumeration of all DMA transfer modes.
	USBHS_DYN_FIFO_PACKET_BUFFERING	Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.
	USBHS_DYN_FIFO_SIZE	Provides enumeration of dynamic FIFO sizes.
	USBHS_ENDPOINT_DIRECTION	Used as an argument to identify an endpoint direction.
	USBHS_LPM_INTERRUPT	Provides an enumeration of LPM interrupt sources.
	USBHS_LPM_LINK_STATE	Provides enumeration requested device state after accepting an LPM transaction.
	USBHS_LPM_MODE	Provides enumeration of Link Power Management (LPM) modes.
	USBHS_OPMODES	Provides enumeration of operating modes supported by USB.
	USBHS_PKTS_PER_MICROFRAME	Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.
	USBHS_SPEED	Provides enumeration Host endpoint speeds.
	USBHS_TEST_SPEED	Used as an argument for in setting module speeds in Test mode.
	USBHS_TRANSACTION_TYPE	Provides an enumeration of transaction types.
	USBHS_TXRX_FIFO_STATE	Provides enumeration of receive and transmit FIFO states, as reported by status bits.
	USBHS_MAX_DMA_CHANNEL_NUMBER	Number of available DMA Channels.
	USBHS_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

This section describes the Application Programming Interface (API) functions of the USBHS Peripheral Library.

Refer to each section for a detailed description.

a) USBHS Setup Functions

PLIB_USBHS_HighSpeedDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HighSpeedDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HighSpeedDisable.

PLIB_USBHS_HighSpeedEnable Function

Sets the operation speed of the USB Module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_HighSpeedEnable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function sets the operation speed of the USB module.

Remarks

None.

Preconditions

None.

Example

```
// Enable the USB module for high speed support.

PLIB_USBHS_HighSpeedEnable(USBHS_ID_0, USB_SPEED_HIGH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
speed	The operation speed of the USB module

Function

void PLIB_USBHS_ModuleSpeedSet(USBHS_MODULE_ID index, uint32_t speed)

PLIB_USBHS_ResetDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResetDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResetDisable.

PLIB_USBHS_ResetEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResetEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResetEnable.

PLIB_USBHS_ResumeDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResumeDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResumeDisable.

PLIB_USBHS_ResumeEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResumeEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResumeEnable.

PLIB_USBHS_SessionDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SessionDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SessionDisable.

PLIB_USBHS_SessionEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SessionEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SessionEnable.

PLIB_USBHS_SoftResetDisable Function

Disables soft reset.

File

plib_usbhs.h

C

```
void PLIB_USBHS_SoftResetDisable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function disables soft reset.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
PLIB_USBHS_SoftResetDisable(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_SoftResetDisable(USBHS_MODULE_ID index);

PLIB_USBHS_SoftResetEnable Function

Enables soft reset.

File

plib_usbhs.h

C

```
void PLIB_USBHS_SoftResetEnable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function enables soft reset.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
PLIB_USBHS_SoftResetEnable(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_SoftResetEnable(USBHS_MODULE_ID index);

PLIB_USBHS_SuspendDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SuspendDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SuspendDisable.

PLIB_USBHS_SuspendEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SuspendEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SuspendEnable.

PLIB_USBHS_DMAOperationEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAOperationEnable(USBHS_MODULE_ID index, uint8_t endpoint, uint8_t dmaChannel, void *
address, uint32_t count, bool direction);
```

Description

This is function PLIB_USBHS_DMAOperationEnable.

PLIB_USBHS_TestModeEnter Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_TestModeEnter(USBHS_MODULE_ID index, uint8_t testMode);
```

Description

This is function PLIB_USBHS_TestModeEnter.

PLIB_USBHS_TestModeExit Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_TestModeExit(USBHS_MODULE_ID index, uint8_t testMode);
```

Description

This is function PLIB_USBHS_TestModeExit.

PLIB_USBHS_PhyIDMonitoringDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_PhyIDMonitoringDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_PhyIDMonitoringDisable.

PLIB_USBHS_PhyIDMonitoringEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_PhyIDMonitoringEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_PhyIDMonitoringEnable.

PLIB_USBHS_USBIDOverrideDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_USBIDOverrideDisable.

PLIB_USBHS_USBIDOverrideEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_USBIDOverrideEnable.

PLIB_USBHS_USBIDOverrideValueSet Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideValueSet(USBHS_MODULE_ID index, USBHS_USBID_OVERRIDE_VALUE id);
```

Description

This is function PLIB_USBHS_USBIDOverrideValueSet.

PLIB_USBHS_IsBDevice Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_IsBDevice(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_IsBDevice.

b) Endpoints Functions

PLIB_USBHS_Endpoint0FIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0FIFOFlush(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_Endpoint0FIFOFlush.

PLIB_USBHS_Endpoint0SetupPacketLoad Function

Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_MODULE_ID index, void * setupPacket, uint8_t deviceAddress,
uint8_t hubAddress, uint8_t hubPortAddress, uint32_t speed);
```

Returns

None.

Description

This function loads the endpoint 0 FIFO with provided setup packet. This operation would typically be performed at the start of a endpoint 0 control transfer. Once the FIFO is loaded the function will load the target device address, hub and hub port address and then transmit the packet.

Remarks

Valid in Host mode only. Setup packet size should be 8 bytes. Packet will always be targeted to endpoint 0 of the target device.

Preconditions

None.

Example

```
// Send a setup packet to device 7 control endpoint
// connected directly to module.

PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_ID_0, setupPacket, 7, 0,
                                                    0, USB_SPEED_HIGH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
setupPacket	setup packet to be sent to the device
deviceAddress	USB address of the device
hubAddress	USB address of the hub if the device is connected to a hub. Should be zero otherwise.
hubPortAddress	Port number of the hub port to which the device is connected if it is connected to a hub. Should be zero otherwise.
speed	Speed of the device to which the packet should be sent.

Function

void PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_MODULE_ID index,

void * setupPacket, uint8_t deviceAddress,

uint8_t hubAddress, uint8_t hubPortAddress,

uint32_t speed)

PLIB_USBHS_Endpoint0SetupPacketUnload Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0SetupPacketUnload(USBHS_MODULE_ID index, void * dest);
```

Description

This is function PLIB_USBHS_Endpoint0SetupPacketUnload.

PLIB_USBHS_EndpointFIFOLoad Function

Loads the endpoint FIFO with provided data and then enables the endpoint transmit.

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint, void * source, size_t nBytes);
```

Returns

None.

Description

This function loads the endpoint FIFO with provided data. This operation would typically be performed for a endpoint transmit operation. Once the FIFO is loaded the function will enable the enable the endpoint for transmit.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
// Load endpoint 1 FIFO

PLIB_USBHS_EndpointFIFOLoad(USBHS_ID_0, 1, data, 10);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
endpoint	endpoint of which FIFO needs to be loaded
source	data that should be loaded
nBytes	number of bytes to load.

Function

void PLIB_USBHS_EndpointFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint,

void * source, size_t nBytes)

PLIB_USBHS_EndpointFIFOUnload Function

Unloads the endpoint FIFO.

File

plib_usbhs.h

C

```
int PLIB_USBHS_EndpointFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint, void * dest);
```

Returns

Returns the number of functions that were read.

Description

This function unloads the endpoint FIFO. This operation would typically be performed for a endpoint receive operation. The unloaded data is stored in provided buffer.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// Load endpoint 1 FIFO
int count;

count = PLIB_USBHS_EndpointFIFOUnload(USBHS_ID_0, 1, dest, 10);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
endpoint	endpoint of which FIFO to be unloaded.
dest	target address where unloaded data should be stored.

Function

int PLIB_USBHS_EndpointFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint,
void * destination)

PLIB_USBHS_EndpointRxFIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxFIFOFlush(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxFIFOFlush.

PLIB_USBHS_EndpointRxRequestClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxRequestClear(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxRequestClear.

PLIB_USBHS_EndpointRxRequestEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxRequestEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxRequestEnable.

PLIB_USBHS_EndpointTxFIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointTxFIFOFlush(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointTxFIFOFlush.

PLIB_USBHS_EP0DataEndSet Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0DataEndSet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0DataEndSet.

PLIB_USBHS_EP0INHandshakeClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INHandshakeClear(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INHandshakeClear.

PLIB_USBHS_EP0INHandshakeSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INHandshakeSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INHandshakeSend.

PLIB_USBHS_EP0INTokenSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INTokenSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INTokenSend.

PLIB_USBHS_EP0OUTHandshakeSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0OUTHandshakeSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0OUTHandshakeSend.

PLIB_USBHS_EP0RxPktRdyServiced Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0RxPktRdyServiced(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0RxPktRdyServiced.

PLIB_USBHS_EP0RxPktRdyServicedDataEnd Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0RxPktRdyServicedDataEnd(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.

PLIB_USBHS_EP0SentStallClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0SentStallClear(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0SentStallClear.

PLIB_USBHS_EP0SetupEndServiced Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0SetupEndServiced(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0SetupEndServiced.

PLIB_USBHS_EP0StallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StallDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StallDisable.

PLIB_USBHS_EP0StallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StallEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StallEnable.

PLIB_USBHS_EP0StatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StatusClear(USBHS_MODULE_ID index, USBHS_EP0_ERROR error);
```

Description

This is function PLIB_USBHS_EP0StatusClear.

PLIB_USBHS_EP0StatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_EP0StatusGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StatusGet.

PLIB_USBHS_EP0TxPktRdy Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0TxPktRdy(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0TxPktRdy.

PLIB_USBHS_EP0TxPktRdyDataEnd Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0TxPktRdyDataEnd(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0TxPktRdyDataEnd.

PLIB_USBHS_HostRxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostRxEndpointConfigure(USBHS_MODULE_ID index, uint8_t hostEndpoint, uint32_t speed,
uint32_t pipeType, uint16_t endpointSize, uint16_t receiveFIFOAddress, uint16_t fifoSize, uint8_t
targetEndpoint, uint8_t targetDevice, uint8_t targetHub, uint8_t targetHubPort, uint8_t nakInterval);
```

Description

This is function PLIB_USBHS_HostRxEndpointConfigure.

PLIB_USBHS_HostTxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostTxEndpointConfigure(USBHS_MODULE_ID index, uint8_t hostEndpoint, uint32_t speed,
uint32_t pipeType, uint16_t endpointSize, uint16_t receiveFIFOAddress, uint16_t fifoSize, uint8_t
targetEndpoint, uint8_t targetDevice, uint8_t targetHub, uint8_t targetHubPort, uint8_t nakInterval);
```

Description

This is function PLIB_USBHS_HostTxEndpointConfigure.

PLIB_USBHS_RxEPINTokenSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxEPINTokenSend(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_RxEPINTokenSend.

PLIB_USBHS_RxEPStatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxEPStatusClear(USBHS_MODULE_ID index, uint8_t endpoint, USBHS_RXEP_ERROR error);
```

Description

This is function PLIB_USBHS_RxEPStatusClear.

PLIB_USBHS_RxEPStatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_RxEPStatusGet(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_RxEPStatusGet.

PLIB_USBHS_TxEPStatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxEPStatusClear(USBHS_MODULE_ID index, uint8_t endpoint, USBHS_TXEP_ERROR error);
```

Description

This is function PLIB_USBHS_TxEPStatusClear.

PLIB_USBHS_TxEPStatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_TxEPStatusGet(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_TxEPStatusGet.

PLIB_USBHS_GetEP0CSRAddress Function

File

plib_usbhs.h

C

```
uint8_t * PLIB_USBHS_GetEP0CSRAddress(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GetEP0CSRAddress.

PLIB_USBHS_GetEP0FIFOAddress Function

File

plib_usbhs.h

C

```
uint8_t * PLIB_USBHS_GetEP0FIFOAddress(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GetEP0FIFOAddress.

PLIB_USBHS_LoadEPInIndex Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_LoadEPInIndex(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_LoadEPInIndex.

c) Interrupts Functions

PLIB_USBHS_TxInterruptDisable Function

Disables a TX endpoint interrupt source for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxInterruptDisable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a TX endpoint interrupt source for the USB module.

Remarks

See also PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_TxInterruptDisable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_TxInterruptDisable(USBHS_MODULE_ID index,
USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_TxInterruptEnable Function

Enables a TX endpoint Interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxInterruptEnable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables the TX endpoint interrupt sources of the USB module that are used to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_TxInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_TxInterruptEnable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_TxInterruptEnable(USBHS_MODULE_ID index,
USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptDisable Function

Disables a general interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_GenInterruptDisable(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a general interrupt source for the USB module.

Remarks

See also PLIB_USBHS_GenInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_GenInterruptDisable( MY_USB_INSTANCE, USBHS_GENINT_VBUSERR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_GenInterruptDisable(USBHS_MODULE_ID index,
USBHS_GEN_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptEnable Function

Enables a general interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_GenInterruptEnable(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables the general interrupt sources of the USB module that are used to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_GenInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_GenInterruptEnable( MY_USB_INSTANCE, USBHS_GENINT_VBUSERR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_GenInterruptEnable(USBHS_MODULE_ID index,
USBHS_GEN_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptFlagsGet Function

Gets general interrupt flags.

File

plib_usbhs.h

C

```
USBHS_GEN_INTERRUPT PLIB_USBHS_GenInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the general interrupt flags.

Remarks

None.

Preconditions

None.

Example

```
USBHS_GEN_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_GenInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_GENINT_SUSPEND )
    {
       // Device has detected suspend signaling on the bus.
    }
    if ( interruptFlags & USBHS_GENINT_RESUME )
    {
        //
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_GEN_INTERRUPT PLIB_USBHS_GenInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_InterruptEnableSet Function

Enables USB module event interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_InterruptEnableSet(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT generalInterrupts,
USBHS_EPTXRX_INTERRUPT transmitInterrupts, USBHS_EPTXRX_INTERRUPT receiveInterrupts);
```

Returns

None.

Description

This function enables USB module event interrupts. Combines the functionality of the PLIB_USBHS_RxInterruptEnable, PLIB_USBHS_TxInterruptEnable, and PLIB_USBHS_GenInterruptEnable functions.

Remarks

None.

Preconditions

None.

Example

```
// Enable the reset interrupt, endpoint 1 transmit interrupt
// and endpoint 2 receive interrupts

PLIB_USBHS_InterruptEnableSet (USBHS_ID_0, USBHS_GENINT_RESET,
    USBHS_TXRXINT_EP1, USBHS_TXRXINT_EP2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
generalInterrupts	General Module interrupts.
transmitInterrupts	Transmit Endpoint Interrupts to be enabled.
receiveInterrupts	Receive Endpoint Interrupts to be enabled.

Function

void PLIB_USBHS_InterruptEnableSet(USBHS_MODULE_ID index,

USBHS_GEN_INTERRUPT generalInterrupts,

USBHS_EPTXRX_INTERRUPT transmitInterrupts,

USBHS_EPTXRX_INTERRUPT receiveInterrupts);

PLIB_USBHS_DMAInterruptDisable Function

Disables DMA channel interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAInterruptDisable(USBHS_MODULE_ID index, USBHS_DMA_INTERRUPT nChannelNumber);
```

Returns

None.

Description

This function disables DMA Channel interrupts.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USBHS_DMAInterruptDisable (USBHS_ID_0, USBHS_DMAINT_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
dmaChannelInterrupt	DMA channel Interrupt to disable

Function

void PLIB_USBHS_DMAInterruptDisable(USBHS_MODULE_ID index,
USBHS_DMA_INTERRUPT dmaChannelInterrupt)

PLIB_USBHS_DMAInterruptEnable Function

Enables DMA channel interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAInterruptEnable(USBHS_MODULE_ID index, USBHS_DMA_INTERRUPT nChannelNumber);
```

Returns

None.

Description

This function enables DMA Channel interrupts.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USBHS_DMAInterruptEnable (USBHS_ID_0, USBHS_DMAINT_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
dmaChannelInterrupt	DMA channel Interrupt to enable

Function

void PLIB_USBHS_DMAInterruptEnable(USBHS_MODULE_ID index,
USBHS_DMA_INTERRUPT dmaChannelInterrupt)

PLIB_USBHS_DMAInterruptFlagsGet Function

Gets the DMA channel interrupt flags.

File

plib_usbhs.h

C

```
USBHS_DMA_INTERRUPT PLIB_USBHS_DMAInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the DMA channel interrupt flags.

Remarks

None.

Preconditions

None.

Example

```
USBHS_DMA_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_DMAInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_DMAINT_1 )
    {
        // DMA Channel 1
    }
    if (interruptFlags & USBHS_DMAINT_2)
    {
        // DMA Channel 2
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_DMA_INTERRUPT PLIB_USBHS_DMAInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_TxInterruptFlagsGet Function

Gets the TX endpoint interrupt flags.

File

plib_usbhs.h

C

```
USBHS_EPTXRX_INTERRUPT PLIB_USBHS_TxInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the TX endpoint interrupt flags.

Remarks

TX interrupts must first be enabled by calling PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
USBHS_EPTXRX_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_TxInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_TXRXINT_EP0 )
    {
       // Endpoint Zero
    }
    if ( interruptFlags & USBHS_TXRXINT_EP1 )
    {
        // Endpoint One
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_EPTXRX_INTERRUPT PLIB_USBHS_TxInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_RxInterruptDisable Function

Disables a RX endpoint interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxInterruptDisable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a RX endpoint interrupt source for the USB module.

Remarks

See also PLIB_USBHS_RxInterruptEnable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero, use PLIB_USBHS_TxInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_RxInterruptDisable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_RxInterruptDisable(USBHS_MODULE_ID index,

USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_RxInterruptEnable Function

Enables a RX endpoint interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxInterruptEnable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables RX endpoint interrupt sources of the USB module to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_RxInterruptDisable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero use PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_RxInterruptEnable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_RxInterruptEnable(USBHS_MODULE_ID index,

USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_RxInterruptFlagsGet Function

Gets RX endpoint interrupt flags.

File

plib_usbhs.h

C

```
USBHS_EPTXRX_INTERRUPT PLIB_USBHS_RxInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the RX endpoint interrupt flags.

Remarks

RX interrupts must first be enabled by calling PLIB_USBHS_RxInterruptEnable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero use PLIB_USBHS_TxInterruptFlagsGet.

Preconditions

None.

Example

```
USBHS_EPTXRX_INTERRUPT interruptFlags;
```

```
    interruptFlags = PLIB_USBHS_RxInterruptFlagsGet( MY_USB_INSTANCE );
    // All interrupt flags cleared on read.

    if ( interruptFlags > 0 )
        if ( interruptFlags & USBHS_TXRXINT_EP1 )
        {
           // Endpoint Zero
        }
        if ( interruptFlags & USBHS_TXRXINT_EP2 )
        {
            // Endpoint One
        }
        .
        .
        .
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_EPTXRX_INTERRUPT PLIB_USBHS_RxInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_DMAInterruptGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_DMAInterruptGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_DMAInterruptGet.

PLIB_USBHS_GlobalInterruptDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_GlobalInterruptDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GlobalInterruptDisable.

PLIB_USBHS_GlobalInterruptEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_GlobalInterruptEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GlobalInterruptEnable.

d) Device Functions

PLIB_USBHS_DeviceAddressGet Function

Returns the current USB device address.

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_DeviceAddressGet(USBHS_MODULE_ID index);
```

Returns

7-bit unsigned integer value indicating the current USB device address.

Description

This function returns the current USB device address.

Remarks

None.

Preconditions

The USB module should have been configured for Device mode operation.

Example

```
// This code example reads the current assigned USB device address.

uint8_t address;
address = PLIB_USBHS_DeviceAddressGet(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_USBHS_DeviceAddressGet (USBHS_MODULE_ID index)

PLIB_USBHS_DeviceAddressSet Function

Sets the device address.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceAddressSet(USBHS_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the device address. This function should be called with the address received from the host in the SET_ADDRESS request.

Remarks

None.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// This code example assigns a USB device address.

uint8_t address;
```

```
PLIB_USBHS_DeviceAddressSet(USBHS_ID_0, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	7-bit USB Device address that should be assigned to this device

Function

void PLIB_USBHS_DeviceAddressSet (USBHS_MODULE_ID index, uint8_t address)

PLIB_USBHS_DeviceAttach Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceAttach(USBHS_MODULE_ID index, uint32_t speed);
```

Description

This is function PLIB_USBHS_DeviceAttach.

PLIB_USBHS_DeviceConnect Function

Tri-states the USB D+ and D- lines.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceConnect(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB D+ and D- lines, which connects the device from the host.

Remarks

Only valid for Device mode.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// Disconnect the device from the host

PLIB_USBHS_DeviceConnect(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_DeviceConnect(USBHS_MODULE_ID index)

PLIB_USBHS_DeviceDetach Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceDetach(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_DeviceDetach.

PLIB_USBHS_DeviceDisconnect Function

Tri-states the USB D+ and D- lines.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceDisconnect(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function tri-states the USB D+ and D- lines, which disconnects the device from the host.

Remarks

Only valid for Device mode.

Preconditions

None.

Example

```
// Disconnect the device from the host

PLIB_USBHS_DeviceDisconnect(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_DeviceDisconnect(USBHS_MODULE_ID index)

PLIB_USBHS_DeviceEPFIFOLoad Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceEPFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint, void * source, size_t nBytes);
```

Description

This is function PLIB_USBHS_DeviceEPFIFOLoad.

PLIB_USBHS_DeviceEPFIFOUnload Function

File

plib_usbhs.h

C

```
int PLIB_USBHS_DeviceEPFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint, void * dest);
```

Description

This is function PLIB_USBHS_DeviceEPFIFOUnload.

PLIB_USBHS_DeviceRxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointConfigure(USBHS_MODULE_ID index, uint8_t endpoint, uint16_t endpointSize,
uint16_t fifoAddress, uint8_t fifoSize, uint32_t transferType);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointConfigure.

PLIB_USBHS_DeviceRxEndpointStallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointStallDisable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointStallDisable.

PLIB_USBHS_DeviceRxEndpointStallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointStallEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointStallEnable.

PLIB_USBHS_DeviceTxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointConfigure(USBHS_MODULE_ID index, uint8_t endpoint, uint16_t endpointSize,
uint16_t fifoAddress, uint8_t fifoSize, uint32_t transferType);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointConfigure.

PLIB_USBHS_DeviceTxEndpointPacketReady Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointPacketReady(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointPacketReady.

PLIB_USBHS_DeviceTxEndpointStallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointStallDisable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointStallDisable.

PLIB_USBHS_DeviceTxEndpointStallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointStallEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointStallEnable.

e) Host Functions

PLIB_USBHS_HostRxEndpointDataToggleClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostRxEndpointDataToggleClear(USBHS_MODULE_ID index, uint8_t hostEndpoint);
```

Description

This is function PLIB_USBHS_HostRxEndpointDataToggleClear.

PLIB_USBHS_HostTxEndpointDataToggleClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostTxEndpointDataToggleClear(USBHS_MODULE_ID index, uint8_t hostEndpoint);
```

Description

This is function PLIB_USBHS_HostTxEndpointDataToggleClear.

f) Status Functions

PLIB_USBHS_FullOrHighSpeedIsConnected Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_FullOrHighSpeedIsConnected(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_FullOrHighSpeedIsConnected.

PLIB_USBHS_HighSpeedIsConnected Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_HighSpeedIsConnected(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HighSpeedIsConnected.

PLIB_USBHS_HostModeIsEnabled Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_HostModeIsEnabled(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HostModeIsEnabled.

PLIB_USBHS_ModuleSpeedGet Function

Returns the speed at which the module is operating.

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ModuleSpeedGet(USBHS_MODULE_ID index);
```

Returns

Current module operation speed on the USB.

Description

This function returns the speed at which the module is operating. In case of device mode operation, this function returns the speed at which the device attached to the host.

Remarks

None.

Preconditions

None.

Example

```
USB_SPEED speed;
speed = PLIB_USBHS_ModuleSpeedGet(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_SPEED PLIB_USBHS_ModuleSpeedGet(USBHS_MODULE_ID index)

PLIB_USBHS_DMAErrorGet Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_DMAErrorGet(USBHS_MODULE_ID index, uint8_t dmaChannel);
```

Description

This is function PLIB_USBHS_DMAErrorGet.

PLIB_USBHS_GetReceiveDataCount Function

File

plib_usbhs.h

C

```
uint32_t PLIB_USBHS_GetReceiveDataCount(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_GetReceiveDataCount.

g) VBUS Support Functions

PLIB_USBHS_VbusLevelGet Function

Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.

File

plib_usbhs.h

C

```
USBHS_VBUS_LEVEL PLIB_USBHS_VbusLevelGet(USBHS_MODULE_ID index);
```

Returns

Detected VBUS level, see USBHS_VBUS_DETECT_LEVEL enumeration.

Description

Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_VBUS_DETECT_LEVEL PLIB_USBHS_VbusLevelGet(USBHS_MODULE_ID index)

PLIB_USBHS_VBUSLevelGet Function

File

plib_usbhs.h

C

```
USBHS_VBUS_LEVEL PLIB_USBHS_VBUSLevelGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_VBUSLevelGet.

h) Feature Existence Functions

PLIB_USBHS_ExistsEndpointFIFO Function

Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEndpointFIFO(USBHS_MODULE_ID index);
```

Returns

- true - The feature is supported
- false - The feature is not supported

Description

This interface identifies that the Endpoint FIFO feature is available on the Hi-Speed USB module. When this interface returns true, these functions are supported on the device:

- PLIB_USBHS_EndpointFIFOLoad
- PLIB_USBHS_EndpointFIFOUnload

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USBHS_ExistsEndpointFIFO(USBHS_MODULE_ID index)

PLIB_USBHS_ExistsEndpointOperations Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEndpointOperations(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsEndpointOperations.

PLIB_USBHS_ExistsEP0Status Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEP0Status(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsEP0Status.

PLIB_USBHS_ExistsHighSpeedSupport Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsHighSpeedSupport(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsHighSpeedSupport.

PLIB_USBHS_ExistsInterrupts Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsInterrupts(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsInterrupts.

PLIB_USBHS_ExistsModuleControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsModuleControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsModuleControl.

PLIB_USBHS_ExistsRxEPStatus Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsRxEPStatus(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsRxEPStatus.

PLIB_USBHS_ExistsSoftReset Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsSoftReset(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsSoftReset.

PLIB_USBHS_ExistsTxEPStatus Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsTxEPStatus(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsTxEPStatus.

PLIB_USBHS_ExistsClockResetControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsClockResetControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsClockResetControl.

PLIB_USBHS_ExistsUSBIDControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsUSBIDControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsUSBIDControl.

i) Data Types and Constants

USBHS_CONFIGURATION Enumeration

Provides the enumeration Configuration bits.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_CONFIG_SOFTCONN,
  USBHS_CONFIG_DYNFIFOSIZE,
```

```
  USBHS_CONFIG_HBWTXISO,
  USBHS_CONFIG_HBWRXISO,
  USBHS_CONFIG_BIGENDIAN,
  USBHS_CONFIG_AUTOSPLIT,
  USBHS_CONFIG_AUTOJOIN
} USBHS_CONFIGURATION;
```

Members

Members	Description
USBHS_CONFIG_SOFTCONN	Soft connect/disconnect supported
USBHS_CONFIG_DYNFIFOSIZE	Dynamic sizing of FIFO buffers supported
USBHS_CONFIG_HBWTXISO	High bandwidth TX isochronous transfers supported
USBHS_CONFIG_HBWRXISO	High bandwidth RX isochronous transfers supported
USBHS_CONFIG_BIGENDIAN	Big Endian byte ordering supported instead of Little Endian
USBHS_CONFIG_AUTOSPLIT	Automatic splitting of bulk packets supported
USBHS_CONFIG_AUTOJOIN	Automatic combining of bulk packets supported

Description

USBHS Hardware Configuration Bits Enumeration

This data type provides the enumeration configuration bits returned by PLIB_USBHS_ConfigurationBitsGet.

Remarks

See also PLIB_USBHS_ConfigurationBitsGet.

USBHS_DATA01 Enumeration

Provides an enumeration data toggle for a packet.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DATA0,
  USBHS_DATA1
} USBHS_DATA01;
```

Members

Members	Description
USBHS_DATA0	DATA0/1 = 0
USBHS_DATA1	DATA0/1 = 1

Description

USBHS Endpoint Data Toggle Enumeration

This data type provides an enumeration data toggle for a packet.

Remarks

None.

USBHS_DMA_ASSERT_TIMING Enumeration

Provides enumeration DMA assertion timing (early versus late).

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_ASSERT_EARLY,
  USBHS_DMA_ASSERT_LATE
} USBHS_DMA_ASSERT_TIMING;
```

Members

Members	Description
USBHS_DMA_ASSERT_EARLY	Assert DMA 8 bytes before filling FIFO
USBHS_DMA_ASSERT_LATE	Assert DMA when FIFO is full

Description

USBHS DMA Assertion Timing Enumeration

This data type provides enumeration DMA assertion timing (early versus late).

Remarks

None.

USBHS_DMA_BURST_MODE Enumeration

Provides enumeration of all DMA burst modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_BURST_MODE0,
  USBHS_DMA_BURST_MODE1,
  USBHS_DMA_BURST_MODE2,
  USBHS_DMA_BURST_MODE3
} USBHS_DMA_BURST_MODE;
```

Members

Members	Description
USBHS_DMA_BURST_MODE0	Burst Mode 0: Bursts of unspecified length
USBHS_DMA_BURST_MODE1	Burst Mode 1: INCR4 or unspecified length
USBHS_DMA_BURST_MODE2	Burst Mode 2: INCR8, INCR4 or unspecified length
USBHS_DMA_BURST_MODE3	Burst Mode 3: INCR16, INCR8, INCR4 or unspecified length

Description

USBHS DMA Burst Modes Enumeration

This data type provides enumeration of all DMA burst modes.

Remarks

None.

USBHS_DMA_INTERRUPT Enumeration

Provides enumeration of interrupts for DMA channels 0-7.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMAINT_1,
  USBHS_DMAINT_2,
  USBHS_DMAINT_3,
  USBHS_DMAINT_4,
  USBHS_DMAINT_5,
  USBHS_DMAINT_6,
  USBHS_DMAINT_7,
  USBHS_DMAINT_8,
  USBHS_DMAINT_ANY,
  USBHS_DMAINT_ALL
} USBHS_DMA_INTERRUPT;
```

Description

USBHS DMA Interrupts Enumeration

This data type provides the enumeration of interrupts for DMA channels 0-7.

Remarks

None.

USBHS_DMA_REQUEST_MODE Enumeration

Used as an argument to set DMA request mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_MODE0,
  USBHS_DMA_MODE1
} USBHS_DMA_REQUEST_MODE;
```

Description

USBHS DMA Request Mode Enumeration

THis data type is used as an argument to set DMA request mode. Valid only for Endpoints 1-7.

Remarks

Used by PLIB_USBHS_EPnDMARequestModeSet.

USBHS_DMA_TRANSFER_MODE Enumeration

Provides enumeration of all DMA transfer modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_TRANSFER_MODE0,
  USBHS_DMA_TRANSFER_MODE1
} USBHS_DMA_TRANSFER_MODE;
```

Description

USBHS DMA Transfer Modes Enumeration

This data type provides enumeration of all DMA Transfer modes.

Remarks

None.

USBHS_DYN_FIFO_PACKET_BUFFERING Enumeration

Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_FIFO_SINGLEPKT,
  USBHS_FIFO_DOUBLEPKT
} USBHS_DYN_FIFO_PACKET_BUFFERING;
```

Description

USBHS Dynamic FIFO Double Packet versus Single Packet Buffering

This data type provides enumeration of dynamic FIFO double-packet versus single-packet buffering.

Remarks

None.

USBHS_DYN_FIFO_SIZE Enumeration

Provides enumeration of dynamic FIFO sizes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_FIFO_SIZE_8BYTES,
  USBHS_FIFO_SIZE_16BYTES,
  USBHS_FIFO_SIZE_32BYTES,
  USBHS_FIFO_SIZE_64BYTES,
  USBHS_FIFO_SIZE_128BYTES,
  USBHS_FIFO_SIZE_256BYTES,
  USBHS_FIFO_SIZE_512BYTES,
  USBHS_FIFO_SIZE_1024BYTES,
  USBHS_FIFO_SIZE_2048BYTES,
  USBHS_FIFO_SIZE_4096BYTES
} USBHS_DYN_FIFO_SIZE;
```

Description

USBHS Dynamic FIFO Size Enumeration

This data type provides enumeration of dynamic FIFO sizes.

Remarks

FIFO size = 2^(FIFO_Size<3:0> + 3)

USBHS_ENDPOINT_DIRECTION Enumeration

Used as an argument to identify an endpoint direction.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_ENDPOINT_RX,
  USBHS_ENDPOINT_TX
} USBHS_ENDPOINT_DIRECTION;
```

Members

Members	Description
USBHS_ENDPOINT_RX	RX endpoint
USBHS_ENDPOINT_TX	TX endpoint

Description

USBHS Endpoint Direction Enumeration

This data type is used as an argument to identify an endpoint direction. Valid only for Endpoints 1-7.

Remarks

None.

USBHS_LPM_INTERRUPT Enumeration

Provides an enumeration of LPM interrupt sources.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPMINT_STALL,
  USBHS_LPMINT_NYET,
  USBHS_LPMINT_ACK,
  USBHS_LPMINT_NOTCOMPLETE,
  USBHS_LPMINT_RESUMED,
  USBHS_LPMINT_ERROR
} USBHS_LPM_INTERRUPT;
```

Members

Members	Description
USBHS_LPMINT_STALL	Host: Device responded with STALL. Peripheral/Device: Sent a STALL.
USBHS_LPMINT_NYET	Host: Device responded with NYET. Peripheral/Device: Sent a NYET.
USBHS_LPMINT_ACK	Host: Device responded with ACK. Peripheral/Device: Sent an ACK.
USBHS_LPMINT_NOTCOMPLETE	Host: LPM transaction failed to complete. Peripheral/Device: NYET sent because data is pending in RX FIFOs.
USBHS_LPMINT_RESUMED	USB Module has resumed operation.
USBHS_LPMINT_ERROR	Host: Received Bit Stuff or PID error. Peripheral/Device: Unsupported LinkState field received.

Description

USBHS Link Power Management (LPM) Interrupts Enumeration

This data type provides an enumeration of LPM interrupt sources.

Remarks

None.

USBHS_LPM_LINK_STATE Enumeration

Provides enumeration requested device state after accepting an LPM transaction.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPM_RESERVED_0,
  USBHS_LPM_L1_STATE,
  USBHS_LPM_RESERVED_2,
  USBHS_LPM_RESERVED_3,
  USBHS_LPM_RESERVED_4,
  USBHS_LPM_RESERVED_5,
  USBHS_LPM_RESERVED_6,
  USBHS_LPM_RESERVED_7,
  USBHS_LPM_RESERVED_8,
  USBHS_LPM_RESERVED_9,
  USBHS_LPM_RESERVED_A,
  USBHS_LPM_RESERVED_B,
  USBHS_LPM_RESERVED_C,
  USBHS_LPM_RESERVED_D,
  USBHS_LPM_RESERVED_E,
  USBHS_LPM_RESERVED_F
} USBHS_LPM_LINK_STATE;
```

Description

USBHS Link Power Management Requested State Enumeration

This data type provides enumeration requested device state after accepting an LPM transaction.

Remarks

Used by PLIB_USBHS_LPMRequestedLinkStateGet and PLIB_USBHS_LPMRequestedLinkStateSet.

USBHS_LPM_MODE Enumeration

Provides enumeration of Link Power Management (LPM) modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPM_DISABLED,
  USBHS_LPM_EXTENDEDNOLPM,
  USBHS_LPM_DISABLED2,
  USBHS_LPM_LPMEXTENDED
} USBHS_LPM_MODE;
```

Members

Members	Description
USBHS_LPM_DISABLED	LPM and Extended transactions not supported and will time-out
USBHS_LPM_EXTENDEDNOLPM	Extended transactions supported but LPM transactions not supported and produce STALLs
USBHS_LPM_DISABLED2	LPM and Extended transactions not supported and will timeout
USBHS_LPM_LPMEXTENDED	LPM and Extended transactions are supported

Description

USBHS Link Power Management (LPM) Mode Enumeration

This data type provides enumeration of Link Power Management (LPM) modes.

Remarks

Used by PLIB_USBHS_LPMModeSet.

USBHS_OPMODES Enumeration

Provides enumeration of operating modes supported by USB.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_OPMODE_NONE,
  USBHS_OPMODE_DEVICE,
  USBHS_OPMODE_HOST,
  USBHS_OPMODE_OTG
} USBHS_OPMODES;
```

Members

Members	Description
USBHS_OPMODE_NONE	None
USBHS_OPMODE_DEVICE	Device
USBHS_OPMODE_HOST	Host
USBHS_OPMODE_OTG	OTG

Description

USB Operating Modes Enumeration

This data type provides enumeration of the operating modes supported by the USB module.

Remarks

None.

USBHS_PKTS_PER_MICROFRAME Enumeration

Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_ONE_PKTS,
  USBHS_TWO_PKTS,
  USBHS_THREE_PKTS,
  USBHS_RSVD_PKTS
} USBHS_PKTS_PER_MICROFRAME;
```

Description

USBHS Iso Packets Per Microframe Enumeration

This data type provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.

Remarks

Used by PLIB_USBHS_EPnPacketsInMicroFrameSet.

For Transmit Endpoints: PIC32MZ EC device SFR fields: USBIENCSR0.MULT<1:0> = USBIENCSR0<12:11>, with USBCSR3.ENDPOINT = 1,2,...7

For Receive Endpoints: MG register fields: RXMAXP.m-1 = RXMAXP<12:11>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR1.MULT<1:0> = USBIENCSR1<12:11>, with USBCSR3.ENDPOINT = 1,2,...7

USBHS_SPEED Enumeration

Provides enumeration Host endpoint speeds.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LOW_SPEED,
  USBHS_FULL_SPEED,
  USBHS_HIGH_SPEED
} USBHS_SPEED;
```

Description

USBHS Endpoint Speed Enumeration

This data type provides enumeration Host endpoint speeds.

Remarks

Used by PLIB_USBHS_EPnSpeedSet.

USBHS_TEST_SPEED Enumeration

Used as an argument for in setting module speeds in Test mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TEST_LOW_SPEED,
  USBHS_TEST_FULL_SPEED,
  USBHS_TEST_HIGH_SPEED
} USBHS_TEST_SPEED;
```

Description

USBHS Test Speeds Enumeration

This data type is used as an argument for setting module speeds in Test mode.

Remarks

For use with PLIB_USBHS_TestSpeedForce.

USBHS_TRANSACTION_TYPE Enumeration

Provides an enumeration of transaction types.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TRANSACTION_CONTROL,
  USBHS_TRANSACTION_ISO,
  USBHS_TRANSACTION_BULK,
  USBHS_TRANSACTION_INTERRUPT
} USBHS_TRANSACTION_TYPE;
```

Description

USBHS Endpoint Transaction Type Enumeration

This data type provides an enumeration of transaction types.

Remarks

Used by PLIB_USBHS_EPnTransactionTypeSet

USBHS_TXRX_FIFO_STATE Enumeration

Provides enumeration of receive and transmit FIFO states, as reported by status bits.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TX_FIFO_EMPTY,
  USBHS_TX_FIFO_NOT_EMPTY,
  USBHS_RX_FIFO_FULL,
  USBHS_RX_FIFO_NOTFULL
} USBHS_TXRX_FIFO_STATE;
```

Members

Members	Description
USBHS_TX_FIFO_EMPTY	TX FIFO Empty
USBHS_TX_FIFO_NOT_EMPTY	TX FIFO NOT Empty
USBHS_RX_FIFO_FULL	RX FIFO Full
USBHS_RX_FIFO_NOTFULL	RX FIFO NOT Full

Description

USBHS FIFO State Enumeration

This data type provides enumeration of receive and transmit FIFO states, as reported by status bits.

Remarks

None.

For Transmit Endpoints: MG register fields: TXCSRL.FIFONotEmpty = TXCSRL<1>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR0.FIFONE = USBIENCSR0<17>, with USBCSR3.ENDPOINT = 1,2,...7

For Receive Endpoints: MG register fields: RXCSRL.FIFOFull = RXCSRL<1>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR1.FIFOFULL = USBIENCSR1<17>, with USBCSR3.ENDPOINT = 1,2,...7

USBHS_MAX_DMA_CHANNEL_NUMBER Macro

Number of available DMA Channels.

File

plib_usbhs.h

C

```
#define USBHS_MAX_DMA_CHANNEL_NUMBER 8
```

Description

Number of DMA Channels

This constant defines the number of available DMA Channels.

USBHS_MAX_EP_NUMBER Macro

Maximum number of endpoints supported (not including EP0).

File

plib_usbhs.h

C

```
#define USBHS_MAX_EP_NUMBER 7
```

Description

Maximum number of endpoints

This constant defines the maximum number of endpoints supported (not including EP0).

Files

Files

Name	Description
plib_usbhs.h	HS USB PLIB Interface Header for definitions common to the Hi-Speed USB module.

Description

This section lists the source and header files used by the library.

plib_usbhs.h

HS USB PLIB Interface Header for definitions common to the Hi-Speed USB module.

Enumerations

	Name	Description
	USBHS_CONFIGURATION	Provides the enumeration Configuration bits.
	USBHS_DATA01	Provides an enumeration data toggle for a packet.
	USBHS_DMA_ASSERT_TIMING	Provides enumeration DMA assertion timing (early versus late).
	USBHS_DMA_BURST_MODE	Provides enumeration of all DMA burst modes.
	USBHS_DMA_INTERRUPT	Provides enumeration of interrupts for DMA channels 0-7.
	USBHS_DMA_REQUEST_MODE	Used as an argument to set DMA request mode.
	USBHS_DMA_TRANSFER_MODE	Provides enumeration of all DMA transfer modes.
	USBHS_DYN_FIFO_PACKET_BUFFERING	Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.
	USBHS_DYN_FIFO_SIZE	Provides enumeration of dynamic FIFO sizes.
	USBHS_ENDPOINT_DIRECTION	Used as an argument to identify an endpoint direction.
	USBHS_LPM_INTERRUPT	Provides an enumeration of LPM interrupt sources.
	USBHS_LPM_LINK_STATE	Provides enumeration requested device state after accepting an LPM transaction.
	USBHS_LPM_MODE	Provides enumeration of Link Power Management (LPM) modes.
	USBHS_OPMODES	Provides enumeration of operating modes supported by USB.
	USBHS_PKTS_PER_MICROFRAME	Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.
	USBHS_SPEED	Provides enumeration Host endpoint speeds.
	USBHS_TEST_SPEED	Used as an argument for in setting module speeds in Test mode.
	USBHS_TRANSACTION_TYPE	Provides an enumeration of transaction types.
	USBHS_TXRX_FIFO_STATE	Provides enumeration of receive and transmit FIFO states, as reported by status bits.

Functions

	Name	Description
	PLIB_USBHS_DeviceAddressGet	Returns the current USB device address.
	PLIB_USBHS_DeviceAddressSet	Sets the device address.
	PLIB_USBHS_DeviceAttach	This is function PLIB_USBHS_DeviceAttach.
	PLIB_USBHS_DeviceConnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceDetach	This is function PLIB_USBHS_DeviceDetach.
	PLIB_USBHS_DeviceDisconnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceEPFIFOLoad	This is function PLIB_USBHS_DeviceEPFIFOLoad.
	PLIB_USBHS_DeviceEPFIFOUnload	This is function PLIB_USBHS_DeviceEPFIFOUnload.
	PLIB_USBHS_DeviceRxEndpointConfigure	This is function PLIB_USBHS_DeviceRxEndpointConfigure.
	PLIB_USBHS_DeviceRxEndpointStallDisable	This is function PLIB_USBHS_DeviceRxEndpointStallDisable.
	PLIB_USBHS_DeviceRxEndpointStallEnable	This is function PLIB_USBHS_DeviceRxEndpointStallEnable.
	PLIB_USBHS_DeviceTxEndpointConfigure	This is function PLIB_USBHS_DeviceTxEndpointConfigure.
	PLIB_USBHS_DeviceTxEndpointPacketReady	This is function PLIB_USBHS_DeviceTxEndpointPacketReady.
	PLIB_USBHS_DeviceTxEndpointStallDisable	This is function PLIB_USBHS_DeviceTxEndpointStallDisable.
	PLIB_USBHS_DeviceTxEndpointStallEnable	This is function PLIB_USBHS_DeviceTxEndpointStallEnable.
	PLIB_USBHS_DMAErrorGet	This is function PLIB_USBHS_DMAErrorGet.
	PLIB_USBHS_DMAInterruptDisable	Disables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptEnable	Enables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptFlagsGet	Gets the DMA channel interrupt flags.
	PLIB_USBHS_DMAInterruptGet	This is function PLIB_USBHS_DMAInterruptGet.
	PLIB_USBHS_DMAOperationEnable	This is function PLIB_USBHS_DMAOperationEnable.
	PLIB_USBHS_Endpoint0FIFOFlush	This is function PLIB_USBHS_Endpoint0FIFOFlush.
	PLIB_USBHS_Endpoint0SetupPacketLoad	Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.
	PLIB_USBHS_Endpoint0SetupPacketUnload	This is function PLIB_USBHS_Endpoint0SetupPacketUnload.
	PLIB_USBHS_EndpointFIFOLoad	Loads the endpoint FIFO with provided data and then enables the endpoint transmit.
	PLIB_USBHS_EndpointFIFOUnload	Unloads the endpoint FIFO.
	PLIB_USBHS_EndpointRxFIFOFlush	This is function PLIB_USBHS_EndpointRxFIFOFlush.
	PLIB_USBHS_EndpointRxRequestClear	This is function PLIB_USBHS_EndpointRxRequestClear.
	PLIB_USBHS_EndpointRxRequestEnable	This is function PLIB_USBHS_EndpointRxRequestEnable.
	PLIB_USBHS_EndpointTxFIFOFlush	This is function PLIB_USBHS_EndpointTxFIFOFlush.
	PLIB_USBHS_EP0DataEndSet	This is function PLIB_USBHS_EP0DataEndSet.
	PLIB_USBHS_EP0INHandshakeClear	This is function PLIB_USBHS_EP0INHandshakeClear.
	PLIB_USBHS_EP0INHandshakeSend	This is function PLIB_USBHS_EP0INHandshakeSend.
	PLIB_USBHS_EP0INTokenSend	This is function PLIB_USBHS_EP0INTokenSend.
	PLIB_USBHS_EP0OUTHandshakeSend	This is function PLIB_USBHS_EP0OUTHandshakeSend.
	PLIB_USBHS_EP0RxPktRdyServiced	This is function PLIB_USBHS_EP0RxPktRdyServiced.
	PLIB_USBHS_EP0RxPktRdyServicedDataEnd	This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.
	PLIB_USBHS_EP0SentStallClear	This is function PLIB_USBHS_EP0SentStallClear.
	PLIB_USBHS_EP0SetupEndServiced	This is function PLIB_USBHS_EP0SetupEndServiced.
	PLIB_USBHS_EP0StallDisable	This is function PLIB_USBHS_EP0StallDisable.
	PLIB_USBHS_EP0StallEnable	This is function PLIB_USBHS_EP0StallEnable.
	PLIB_USBHS_EP0StatusClear	This is function PLIB_USBHS_EP0StatusClear.
	PLIB_USBHS_EP0StatusGet	This is function PLIB_USBHS_EP0StatusGet.
	PLIB_USBHS_EP0TxPktRdy	This is function PLIB_USBHS_EP0TxPktRdy.
	PLIB_USBHS_EP0TxPktRdyDataEnd	This is function PLIB_USBHS_EP0TxPktRdyDataEnd.
	PLIB_USBHS_ExistsClockResetControl	This is function PLIB_USBHS_ExistsClockResetControl.
	PLIB_USBHS_ExistsEndpointFIFO	Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.
	PLIB_USBHS_ExistsEndpointOperations	This is function PLIB_USBHS_ExistsEndpointOperations.
	PLIB_USBHS_ExistsEP0Status	This is function PLIB_USBHS_ExistsEP0Status.
	PLIB_USBHS_ExistsHighSpeedSupport	This is function PLIB_USBHS_ExistsHighSpeedSupport.
	PLIB_USBHS_ExistsInterrupts	This is function PLIB_USBHS_ExistsInterrupts.

	PLIB_USBHS_ExistsModuleControl	This is function PLIB_USBHS_ExistsModuleControl.
	PLIB_USBHS_ExistsRxEPStatus	This is function PLIB_USBHS_ExistsRxEPStatus.
	PLIB_USBHS_ExistsSoftReset	This is function PLIB_USBHS_ExistsSoftReset.
	PLIB_USBHS_ExistsTxEPStatus	This is function PLIB_USBHS_ExistsTxEPStatus.
	PLIB_USBHS_ExistsUSBIDControl	This is function PLIB_USBHS_ExistsUSBIDControl.
	PLIB_USBHS_FullOrHighSpeedIsConnected	This is function PLIB_USBHS_FullOrHighSpeedIsConnected.
	PLIB_USBHS_GenInterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptFlagsGet	Gets general interrupt flags.
	PLIB_USBHS_GetEP0CSRAddress	This is function PLIB_USBHS_GetEP0CSRAddress.
	PLIB_USBHS_GetEP0FIFOAddress	This is function PLIB_USBHS_GetEP0FIFOAddress.
	PLIB_USBHS_GetReceiveDataCount	This is function PLIB_USBHS_GetReceiveDataCount.
	PLIB_USBHS_GlobalInterruptDisable	This is function PLIB_USBHS_GlobalInterruptDisable.
	PLIB_USBHS_GlobalInterruptEnable	This is function PLIB_USBHS_GlobalInterruptEnable.
	PLIB_USBHS_HighSpeedDisable	This is function PLIB_USBHS_HighSpeedDisable.
	PLIB_USBHS_HighSpeedEnable	Sets the operation speed of the USB Module.
	PLIB_USBHS_HighSpeedIsConnected	This is function PLIB_USBHS_HighSpeedIsConnected.
	PLIB_USBHS_HostModeIsEnabled	This is function PLIB_USBHS_HostModeIsEnabled.
	PLIB_USBHS_HostRxEndpointConfigure	This is function PLIB_USBHS_HostRxEndpointConfigure.
	PLIB_USBHS_HostRxEndpointDataToggleClear	This is function PLIB_USBHS_HostRxEndpointDataToggleClear.
	PLIB_USBHS_HostTxEndpointConfigure	This is function PLIB_USBHS_HostTxEndpointConfigure.
	PLIB_USBHS_HostTxEndpointDataToggleClear	This is function PLIB_USBHS_HostTxEndpointDataToggleClear.
	PLIB_USBHS_InterruptEnableSet	Enables USB module event interrupts.
	PLIB_USBHS_IsBDevice	This is function PLIB_USBHS_IsBDevice.
	PLIB_USBHS_LoadEPInIndex	This is function PLIB_USBHS_LoadEPInIndex.
	PLIB_USBHS_ModuleSpeedGet	Returns the speed at which the module is operating.
	PLIB_USBHS_PhyIDMonitoringDisable	This is function PLIB_USBHS_PhyIDMonitoringDisable.
	PLIB_USBHS_PhyIDMonitoringEnable	This is function PLIB_USBHS_PhyIDMonitoringEnable.
	PLIB_USBHS_ResetDisable	This is function PLIB_USBHS_ResetDisable.
	PLIB_USBHS_ResetEnable	This is function PLIB_USBHS_ResetEnable.
	PLIB_USBHS_ResumeDisable	This is function PLIB_USBHS_ResumeDisable.
	PLIB_USBHS_ResumeEnable	This is function PLIB_USBHS_ResumeEnable.
	PLIB_USBHS_RxEPINTokenSend	This is function PLIB_USBHS_RxEPINTokenSend.
	PLIB_USBHS_RxEPStatusClear	This is function PLIB_USBHS_RxEPStatusClear.
	PLIB_USBHS_RxEPStatusGet	This is function PLIB_USBHS_RxEPStatusGet.
	PLIB_USBHS_RxInterruptDisable	Disables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptEnable	Enables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptFlagsGet	Gets RX endpoint interrupt flags.
	PLIB_USBHS_SessionDisable	This is function PLIB_USBHS_SessionDisable.
	PLIB_USBHS_SessionEnable	This is function PLIB_USBHS_SessionEnable.
	PLIB_USBHS_SoftResetDisable	Disables soft reset.
	PLIB_USBHS_SoftResetEnable	Enables soft reset.
	PLIB_USBHS_SuspendDisable	This is function PLIB_USBHS_SuspendDisable.
	PLIB_USBHS_SuspendEnable	This is function PLIB_USBHS_SuspendEnable.
	PLIB_USBHS_TestModeEnter	This is function PLIB_USBHS_TestModeEnter.
	PLIB_USBHS_TestModeExit	This is function PLIB_USBHS_TestModeExit.
	PLIB_USBHS_TxEPStatusClear	This is function PLIB_USBHS_TxEPStatusClear.
	PLIB_USBHS_TxEPStatusGet	This is function PLIB_USBHS_TxEPStatusGet.
	PLIB_USBHS_TxInterruptDisable	Disables a TX endpoint interrupt source for the USB module.
	PLIB_USBHS_TxInterruptEnable	Enables a TX endpoint Interrupt for the USB module.
	PLIB_USBHS_TxInterruptFlagsGet	Gets the TX endpoint interrupt flags.
	PLIB_USBHS_USBIDOverrideDisable	This is function PLIB_USBHS_USBIDOverrideDisable.
	PLIB_USBHS_USBIDOverrideEnable	This is function PLIB_USBHS_USBIDOverrideEnable.
	PLIB_USBHS_USBIDOverrideValueSet	This is function PLIB_USBHS_USBIDOverrideValueSet.

	PLIB_USBHS_VbusLevelGet	Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.
	PLIB_USBHS_VBUSLevelGet	This is function PLIB_USBHS_VBUSLevelGet.

Macros

	Name	Description
	USBHS_MAX_DMA_CHANNEL_NUMBER	Number of available DMA Channels.
	USBHS_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

Hi-Speed USB (HS USB) Module Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB PLIB for families of Microchip microcontrollers that feature the Hi-Speed USB module.

Remarks

Required to access the High Speed USB Module registers.

File Name

plib_usbhs.h

Company

Microchip Technology Inc.

Watchdog Timer Peripheral Library

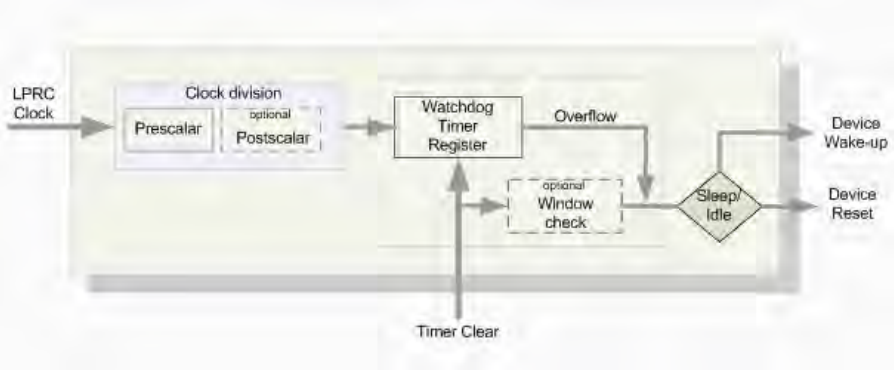
This section describes the Watchdog Timer (WDT) Peripheral Library.

Introduction

This library provides a low-level abstraction of the WDT Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Watchdog Timer (WDT) is to reset the microcontroller, in the event of a software malfunction, by resetting the device if it has not been cleared by software. To ensure that application does not hang, the application is supposed to reset the timer periodically. It can also be used to wake the device from Sleep or Idle mode. The WDT is a free-running timer, which uses the Low-Power RC (LPRC) Oscillator and requires no external components. Therefore, the WDT will continue to operate even if the system's primary clock source is stopped.



Using the Library

This topic describes the basic architecture of the WDT Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_wdt.h`

The interface to the WDT Peripheral Library is defined in the `plib_wdt.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the WDT Peripheral Library must include `peripheral.h`.

Library File:

The WDT Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

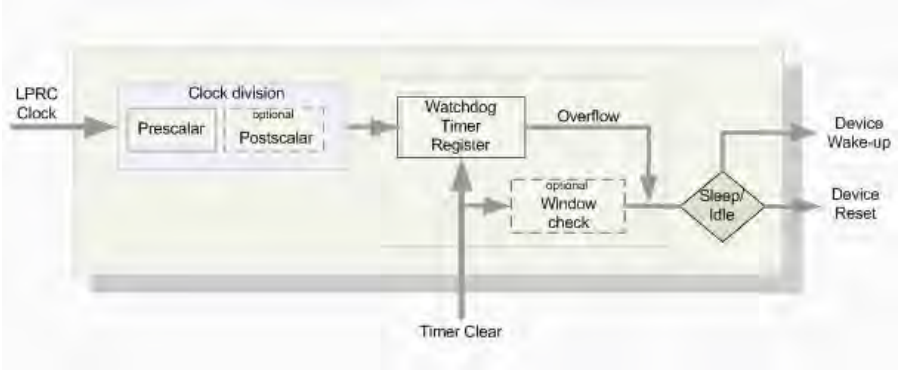
Hardware Abstraction Model

This library provides the low-level abstraction of the *Watchdog timer* module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available Watchdog Timer modules on the device. Please refer to the **"Watchdog Timer"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Watchdog Timer module on your device.

Description

Watchdog Timer Software Abstraction Block Diagram



The WDT uses the internal Low-Power RC (LPRC) Oscillator as the clock source. The clock is divided by the configured prescaler value. There may be one more postscaler divisors. The divided clock is then used to increment a counter. The WDT module uses the watchdog register as a timer. If there is no reset signal from the software and if the counter overflows, this results in a reset in normal mode. In the case of Sleep or Idle mode, the overflow will result in a device wake-up.

For Windowed mode, resetting the counter when the count is not in the specified window will also lead to a reset. If the device is in Sleep or Idle mode, the reset signal will be used to wake-up the device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the WDT module

Library Interface Section	Description
General Configuration Functions	This includes module enable, module disable, watchdog timer window enable, watchdog timer window disable and the timer reset routines.
General Status Functions	Status routines for the WDT.

How the Library Works

This section describes how the Watchdog Timer Peripheral Library works.

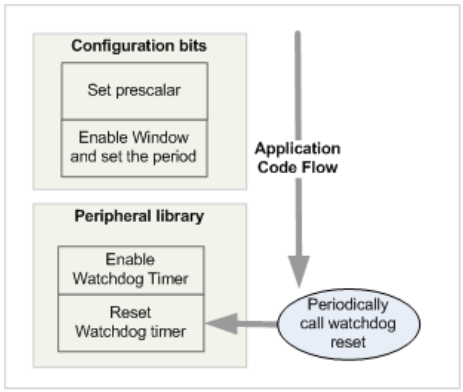
Description

Initializing the WDT module involves these processes.

- Selecting the postscaler
- Enabling the WDT using PLIB_WDT_Enable
- Selecting the window size if the application wants the WDT in Windowed mode using PLIB_WDT_WindowEnable

Note: Selecting the postscaler and window size should be done through the Configuration bits.

The application should periodically clear the timer once it is enabled; otherwise, a time-out will lead to a device Reset. Use PLIB_WDT_TimerClear to clear the WDT. A timer clear before a time-out may also lead to a reset in Windowed mode. The user application can clear the window only in the allowed window. Refer to the specific device data sheet to determine the allowed window period for your device.



Example

```
/*****************************************************************
* This code fragment assumes the WDT is not enabled through    *
* device configuration bits. The Postscaler value must be set *
```

```
* through the device configuration and window size should be  *
* set if applicable                                           *
***************************************************/

/* Initializing the watchdog involves  */
PLIB_WDT_Enable(WDT_ID_0);

//Application loop
while(1)
{
    PLIB_WDT_TimerClear(WDT_ID_0);

    //user code
}
```

Note: Refer to the specific device data sheet for information on the allowed window periods for your device and other Configuration bit settings.

Configuring the Library

The library is configured for the supported Watchdog Timer module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_WDT_Disable	Disables the WDT module.
	PLIB_WDT_Enable	Enables the WDT module.
	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
	PLIB_WDT_WindowEnable	Enables the WDT Window mode.
	PLIB_WDT_TimerClear	Resets the WDT module.

b) General Status Functions

	Name	Description
	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
	PLIB_WDT_PostscalerValueGet	Returns the WDT postscaler value.
	PLIB_WDT_SleepModePostscalerValueGet	Returns the WDT Sleep Mode postscaler value.

c) Feature Existence Functions

	Name	Description
	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
	PLIB_WDT_ExistsPostscalerValue	Identifies whether the PostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.
	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.
	PLIB_WDT_ExistsSleepModePostscalerValue	Identifies whether the SleepModePostscalerValue feature exists on the WDT module.

d) Data Types and Constants

	Name	Description
	WDT_MODULE_ID	Identifies the supported Watchdog Timer modules.

Description

This section describes the Application Programming Interface (API) functions of the Watchdog Timer Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_WDT_Disable Function

Disables the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_Disable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function disables the WDT module if it is enabled in software.

Remarks

This function will not disable the WDT module if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The WDT module must be enabled through software.

Example

```
PLIB_WDT_Disable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_Disable (WDT_MODULE_ID index)

PLIB_WDT_Enable Function

Enables the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_Enable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function enables the WDT module. If it is already enabled through the Configuration bits, it will keep it enabled.

Remarks

Calling this function is not necessary if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_Enable (WDT_MODULE_ID index)

PLIB_WDT_WindowDisable Function

Disables the WDT Windowed mode.

File

plib_wdt.h

C

```
void PLIB_WDT_WindowDisable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function disables the WDT Windowed mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsWindowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_WindowDisable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_WindowDisable (WDT_MODULE_ID index)

PLIB_WDT_WindowEnable Function

Enables the WDT Window mode.

File

plib_wdt.h

C

```
void PLIB_WDT_WindowEnable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function enables the WDT Windowed mode.

Remarks

The window size must be set through the Configuration bits.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_WDT_ExistsWindowEnable in your application to determine whether this feature is available.

Preconditions

The window size must be set through the Configuration bits.

Example

```
PLIB_WDT_WindowEnable ( WDT_ID_0 );
PLIB_WDT_Enable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_WindowEnable (WDT_MODULE_ID index)

PLIB_WDT_TimerClear Function

Resets the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_TimerClear(WDT_MODULE_ID index);
```

Returns

None.

Description

This function resets the WDT module. The WDT module should be cleared periodically before the count crosses and forces the device to reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );

//Application loop
while(1)
{
    PLIB_WDT_TimerClear ( WDT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_TimerClear (WDT_MODULE_ID index)

b) General Status Functions

PLIB_WDT_IsEnabled Function

Returns the watchdog timer on/off(enable/disable) status.

File

plib_wdt.h

C

```
bool PLIB_WDT_IsEnabled(WDT_MODULE_ID index);
```

Returns

- true - If the watchdog timer is on
- false - If the watchdog timer is off

Description

Returns the 'true', if the watchdog timer is already ON. Otherwise returns 'false'.

Remarks

This function returns 'true' if the device is enabled either though the Configuration bits or in the software.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_WDT_IsEnabled ( WDT_ID_0 ) )
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

bool PLIB_WDT_IsEnabled (WDT_MODULE_ID index)

PLIB_WDT_PostscalerValueGet Function

Returns the WDT postscaler value.

File

plib_wdt.h

C

```
char PLIB_WDT_PostscalerValueGet(WDT_MODULE_ID index);
```

Returns

The postscaler value.

Description

This function returns the WDT postscaler value. The value will correspond to a division factor.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;

PLIB_WDT_Enable ( WDT_ID_0 );
value = PLIB_WDT_PostscalerValueGet ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

char PLIB_WDT_PostscalerValueGet (WDT_MODULE_ID index)

PLIB_WDT_SleepModePostscalerValueGet Function

Returns the WDT Sleep Mode postscaler value.

File

plib_wdt.h

C

```
char PLIB_WDT_SleepModePostscalerValueGet(WDT_MODULE_ID index);
```

Returns

The Sleep Mode postscaler value.

Description

This function returns the WDT postscaler value in Sleep Mode. The value will correspond to a division factor.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsSleepModePostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
value = PLIB_WDT_SleepModePostscalerValueGet ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

char PLIB_WDT_SleepModePostscalerValueGet (WDT_MODULE_ID index)

c) Feature Existence Functions

PLIB_WDT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsEnableControl(WDT_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- PLIB_WDT_Enable
- PLIB_WDT_Disable
- PLIB_WDT_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsEnableControl(WDT_MODULE_ID index)

PLIB_WDT_ExistsPostscalerValue Function

Identifies whether the PostscalerValue feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsPostscalerValue(WDT_MODULE_ID index);
```

Returns

- true - The PostscalerValue feature is supported on the device
- false - The PostscalerValue feature is not supported on the device

Description

This function identifies whether the PostscalerValue feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_PostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsPostscalerValue(WDT_MODULE_ID index)

PLIB_WDT_ExistsTimerClear Function

Identifies whether the TimerClear feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsTimerClear(WDT_MODULE_ID index);
```

Returns

- true - The TimerClear feature is supported on the device
- false - The TimerClear feature is not supported on the device

Description

This function identifies whether the TimerClear feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_TimerClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsTimerClear(WDT_MODULE_ID index)

PLIB_WDT_ExistsWindowEnable Function

Identifies whether the WindowEnable feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsWindowEnable(WDT_MODULE_ID index);
```

Returns

- true - The WindowEnable feature is supported on the device
- false - The WindowEnable feature is not supported on the device

Description

This function identifies whether the WindowEnable feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- PLIB_WDT_WindowEnable
- PLIB_WDT_WindowDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsWindowEnable(WDT_MODULE_ID index)

PLIB_WDT_ExistsSleepModePostscalerValue Function

Identifies whether the SleepModePostscalerValue feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsSleepModePostscalerValue(WDT_MODULE_ID index);
```

Returns

- true - The SleepModePostscalerValue feature is supported on the device
- false - The SleepModePostscalerValue feature is not supported on the device

Description

This function identifies whether the SleepModePostscalerValue feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_SleepModePostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsSleepModePostscalerValue(WDT_MODULE_ID index)

d) Data Types and Constants

WDT_MODULE_ID Enumeration

Identifies the supported Watchdog Timer modules.

File

plib_wdt_help.h

C

```
typedef enum {
  WDT_ID_0,
  WDT_NUMBER_OF_MODULES
} WDT_MODULE_ID;
```

Members

Members	Description
WDT_ID_0	WDT Module 1 ID
WDT_NUMBER_OF_MODULES	Number of available WDT modules.

Description

Watchdog Timer (WDT) Module ID

This enumeration identifies the available Watchdog Timer modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_wdt.h	Watchdog Timer (WDT) Peripheral Library interface header for Watchdog Timer common definitions.
plib_wdt_help.h	

Description

This section lists the source and header files used by the library.

plib_wdt.h

Watchdog Timer (WDT) Peripheral Library interface header for Watchdog Timer common definitions.

Functions

	Name	Description
	PLIB_WDT_Disable	Disables the WDT module.
	PLIB_WDT_Enable	Enables the WDT module.
	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
	PLIB_WDT_ExistsPostscalerValue	Identifies whether the PostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsSleepModePostscalerValue	Identifies whether the SleepModePostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.
	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.
	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
	PLIB_WDT_PostscalerValueGet	Returns the WDT postscaler value.
	PLIB_WDT_SleepModePostscalerValueGet	Returns the WDT Sleep Mode postscaler value.
	PLIB_WDT_TimerClear	Resets the WDT module.
	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
	PLIB_WDT_WindowEnable	Enables the WDT Window mode.

Description

Watchdog Timer (WDT) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Watchdog Timer Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Watchdog Timer peripheral.

File Name

plib_wdt.h

Company

Microchip Technology Inc.

plib_wdt_help.h

Enumerations

	Name	Description
	WDT_MODULE_ID	Identifies the supported Watchdog Timer modules.

Section

Data Types & Constants

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X