UM0218
User manual

STR75x Software Library

Introduction

About This Manual

This document is the STR75x software library user manual. It describes the STR75x peripheral software library: a collection of routines, data structures and macros that cover the features of each peripheral.

This manual is structured as follows: some definitions, document conventions and software library rules are provided in Chapter 1. Chapter 2 provides a detailed description of the software library: The package content, the installation steps, the library structure and an example on how to use the library. Finally, Chapters 3 to 20 describe the software library, peripheral configuration structure and function descriptions for each peripheral in detail.

About STR75x Library

The STR75x software library is a software package consisting of device drivers for all standard STR75x peripherals. You can use any STR75x device in applications without in-depth study of each peripheral specification. As a result, using this library can save the you a lot of the time that you would otherwise spend in coding and also the cost of developing and integrating your application.

Each device driver consists of a set of functions covering the functionality of the peripheral. Since all the STR75x peripherals and their corresponding registers are memory-mapped, a peripheral can be easily controlled using ‘C’ code. The source code, developed in ‘C’, is fully documented. A basic knowledge of ‘C’ programming is required.

The library contains a complete software in ‘C’ that can be easily ported to any ARM compatible ‘C’ compiler.

Contents

1 Document and library rules

The user manual document and the software library use the conventions described in the sections below.

1.1 Abbreviations

The table below describes the different abbreviations used in this document

Acronym	Peripheral / Unit
ADC	Analog-to-Digital Converter
CAN	Controller area network
CFG	Configuration Registers
DMA	DMA Controller
EIC	Enhanced Interrupt Controller
EXTIT	External Interrupt Controller
GPIO	General Purpose I/O Ports
I2C	I2C Interface Module
MRCC	MCU Reset and Clock Control
PWM	Synchronizable-PWM Timer
RTC	Real Time Clock
SMI	Serial memory interface
SSP	Synchronous Serial Peripheral
TB	Timebase Timer
TIM	Synchronizable Standard Timer
UART	Universal Asynchronous Receiver Transmitter
WDG	Watchdog Timer

1.2 Naming conventions

The Software library uses the following naming conventions:

- ***PPP*** is used to reference to any peripheral acronym, e.g. ***ADC***. See the section above for more information on peripheral acronyms.
- System and source/header file names are preceded by ‘75x_’, e.g. 75x_conf.h.
- Constants used in one file are defined within this file. A constant used in more than one file is defined in a header file. All constants use upper case characters.
- Registers are considered as constants. Their names are in upper case letters and have in most case the same acronyms as in the STR75x reference manual document.
- Peripheral function names are preceded with the corresponding peripheral acronym in upper case followed by an underscore. The first letter in each word is upper case, e.g.

SSP_SendData. Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.

- Functions for initializing PPP peripheral according to the specified parameters in the ***PPP_InitTypeDef*** are named ***PPP_Init***, e.g. ***TIM_Init***.
- Functions for deinitializing PPP peripheral registers to their default reset values are named ***PPP_DeInit***, e.g. ***TIM_DeInit***.
- Functions for filling the ***PPP_InitTypeDef*** structure with the reset value of each member are named ***PPP_StructInit***, e.g. ***UART_StructInit***.
- Functions for enabling or disabling the specified PPP peripheral are named ***PPP_Cmd***, e.g. ***SSP_Cmd***.
- Functions for enabling or disabling an interrupt source of the specified PPP peripheral are named ***PPP_ITConfig***, e.g. ***MRCC_ITConfig***.
- Functions for enabling or disabling the DMA interface of the specified PPP peripheral are named ***PPP_DMAConfig***, e.g. ***PWM_DMAConfig***.
- Functions used to configure a peripheral function end with Config, e.g. ***GPIO_PinRemapConfig***.
- Functions for checking whether the specified PPP flag is set or not are named ***PPP_GetFlagStatus***, e.g. ***I2C_GetFlagStatus***.
- Functions for clearing a PPP flag are named ***PPP_ClearFlag***, e.g. ***I2C_ClearFlag***.
- Functions for checking whether the specified PPP interrupt has occurred or not are named ***PPP_GetITStatus***, e.g. ***SMI_GetITStatus***.
- Functions for clearing a PPP interrupt pending bit are named ***PPP_ClearITPendingBit***, e.g. ***SMI_ClearITPendingBit***.

1.3 Coding Rules

The following rules are used in the Software Library.

- 15 specific types are defined for variables whose type and size are fixed. These types are defined in the file 75x_type.h:

```
typedef signed long      s32;
typedef signed short     s16;
typedef signed char      s8;

typedef volatile signed long      vs32;
typedef volatile signed short     vs16;
typedef volatile signed char      vs8;

typedef unsigned long       u32;
typedef unsigned short      u16;
typedef unsigned char       u8;

typedef volatile unsigned long      vu32;
typedef volatile unsigned short     vu16;
typedef volatile unsigned char      vu8;

typedef volatile unsigned long  const    vuc32;  /* Read Only */
typedef volatile unsigned short const    vuc16;  /* Read Only */
typedef volatile unsigned char  const    vuc8;   /* Read Only */
```

- ***bool*** type is defined in the file ***75x_type.h*** as:

```
typedef enum
{
  FALSE = 0,
  TRUE  = !FALSE
} bool;
```

- ***FlagStatus*** type is defined in the file ***75x_type.h***. Two values can be assigned to this variable: ***SET*** or ***RESET***.

```
typedef enum
{
  RESET = 0,
  SET   = !RESET
} FlagStatus;
```

- ***FunctionalState*** type is defined in the file ***75x_type.h***. Two values can be assigned to this variable: ***ENABLE*** or ***DISABLE***.

```
typedef enum
{
  DISABLE = 0,
  ENABLE  = !DISABLE
} FunctionalState;
```

- ***ErrorStatus*** type is defined in the file ***75x_type.h***. Two values can be assigned to this variable: ***SUCCESS*** or ***ERROR***.

```
typedef enum
{
  ERROR   = 0,
  SUCCESS = !ERROR
} ErrorStatus;
```

- Pointers to peripherals are used to access the peripheral control registers. Peripheral pointers point to data structures that represent the mapping of the peripheral control registers. A structure is defined for each peripheral in the file ***75x_map.h***. The example below illustrates the ***SSP*** register structure declaration:

```
/*----------------------- Synchronous Serial Peripheral ----------------------*/
typedef struct
{
  vu32 CR0;
  vu32 CR1;
  vu32 DR;
  vu32 SR;
  vu32 PR;
  vu32 IMSCR;
  vu32 RISR;
  vu32 MISR;
  vu32 ICR;
  vu32 DMACR;
} SSP_TypeDef;
```

Register names are the register acronyms written in upper case for each peripheral. EMPTYi (i is an integer that indexes the reserved field) replaces a reserved field.

Peripherals are declared in ***75x_map.h*** file. The following example shows the declaration of the *SSP* peripheral:

```
#ifndef EXT
 #Define EXT extern
#endif
...
#define PERIPH_BASE    0xFFFF0000
...
/* SSP0 Base Address definition*/
#define SSP0_BASE      (PERIPH_BASE + 0xB800)
...
/* SSP0 peripheral declaration*/
#ifndef DEBUG
...
#define SSP0         ((SSP_TypeDef *) SSP0_BASE)
...
#else
...
#ifdef _SSP0
  EXT SSP_TypeDef           *SSP0;
#endif /*_SSP0 */
...
#endif
```

To enter debug mode you have to define the label *DEBUG* in the file ***75x_conf.h***. Debug mode allows you to see the contents of peripheral registers but it uses more memory space. In both cases *SSP0* is a pointer to the first address of *SSP0* peripheral.

The *DEBUG* variable is defined in the file ***75x_conf.h*** as follows:

```
#define DEBUG
```

DEBUG mode is initialized as follows in the ***75x_lib.c*** file:

```
#ifdef DEBUG
void debug(void)
{
  ...
  #ifdef _SSP0
    SSP0 = (SSP_TypeDef *)  SSP0_BASE;
  #endif /*_SSP0 */
  ...
}
#endif /* DEBUG*/
```

To include the *SSP* peripheral library in your application, define the label *_SSP* and to access the *SSPn* peripheral registers, define the label *_SSPn*, i.e to access the registers of

SSP0 peripheral, *_SSP0* label must be defined in ***75x_conf.h*** file. *_SSP* and *_SSPn* labels are defined in the file ***75x_conf.h*** as follows:

```
#define _SSP
#define _SSP0
```

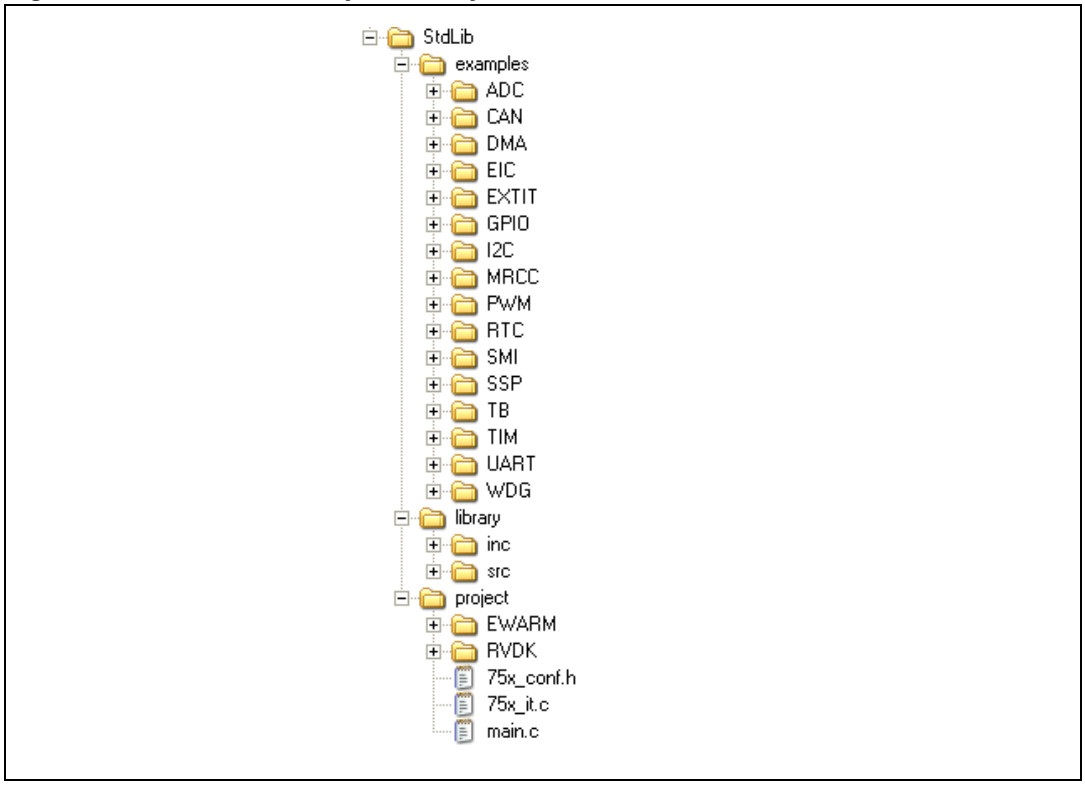
Each peripheral has several dedicated registers which contain different flags. Registers are defined within a dedicated structure for each peripheral. Flag definition is adapted to each peripheral case (defined in ***75x_ppp.h*** file). Flags are defined as acronyms written in upper case and prefixed by '***PPP_Flag_***' prefix.

2 Software library

2.1 Package description

The software library is supplied in one single zip package. The extraction of the zip file will give the one folder ***"STR75xStdLib\StdLib"*** containing the following sub-directories:

Figure 1. Software Library Directory Structure



2.1.1 Examples

This directory contains for each peripheral sub-directory, the minimum set of files needed to run a typical example on how to use a peripheral:

- ***Readme.txt***: a brief text file describing the example and how to make it work,
- ***75x_conf.h:*** the header file to configure used peripherals and miscellaneous defines,
- ***75x_it.c***: the source file containing the interrupt handlers (the function bodies may be empty if not used),
- ***main.c***: the example program,

Note: All examples are independent from any software tool chain.

2.1.2 Library

This directory contains all the subdirectories and files that form the core of the library:

- ***inc*** sub-directory contains the software library header files that do not need to be modified by the user:
  - ***75x_type.h***: Contains the common data types and enumeration used in all other files,
  - ***75x_map.h***: Contains the peripherals memory mapping and registers data structures,
  - ***75x_lib.h***: Main header file including all other headers,
  - ***75x_ppp.h*** (one header file per peripheral): contains the function prototypes, data structures and enumeration.
- ***src*** sub-directory contains the software library source files that do not need to be modified by the user:
  - ***75x_ppp.c*** (one source file per peripheral): contains the function bodies of each peripheral.

Note: All library files are coded in Strict ANSI-C and are independent from any software toolchain.

2.1.3 Project

This directory contains a standard template project program that compiles all library files and also all the user modifiable files needed to create a new project:

- ***75x_conf.h***: The configuration header file with all peripherals defined by default.
- ***75x_it.c***: The source file containing the interrupt handlers (the function bodies are empty in this template).
- ***main.c***: The main program body.

- ***EWARM***, ***RVDK***: For each toolchain usage, refer to Readme.txt file available in the same sub-directory.

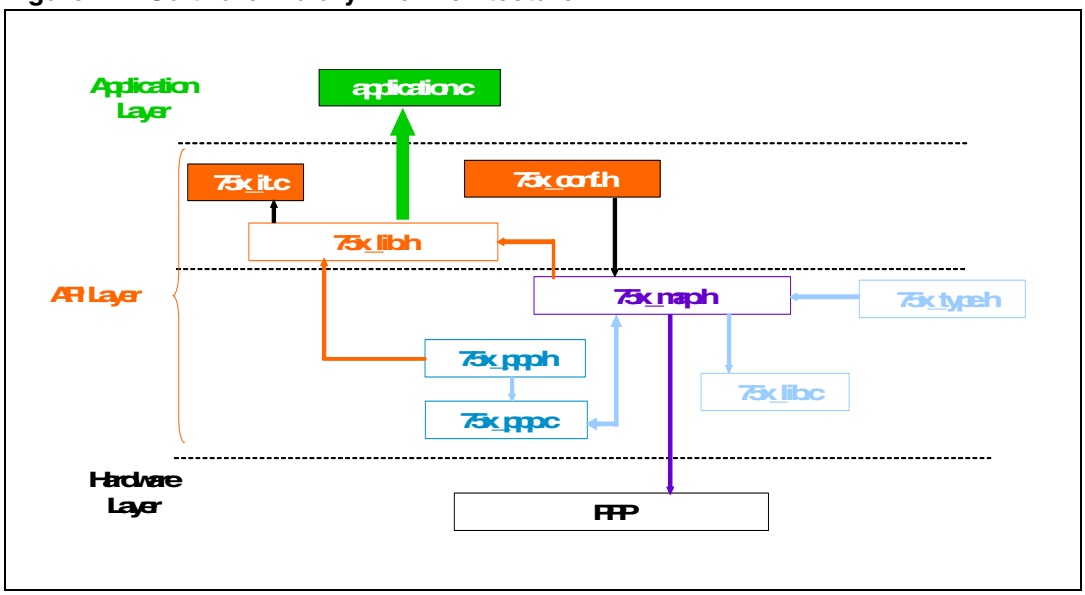
2.2 File Description

Several files are used in the software library. The following tables enumerate and describe the different files used in the software library.

File name	Description
75x_conf.h	Parameter configuration file. It should be modified by the user to specify several parameters to interface with the library before running any application. You can enable or disable peripherals if you use the template and you can also change the value of your external Quartz oscillator value.
main.c	The main example program body.
75x_it.c	Peripheral interrupt functions file. You can modify it by including the code of interrupt functions used in your application. In case of multiple interrupt requests mapped on the same interrupt vector, the function polls the interrupt flags of the peripheral to establish the exact source of the interrupt. The names of these functions are already provided in the software library.
75x_lib.h	Header file including all the peripheral header files. It is the only file to be included in the user application to interface with the library.
75x_lib.c	Debug mode initialization file. It includes the definition of variable pointers each one pointing to the first address of a specific peripheral and the definition of one function called when you choose to enter debug mode. This function initializes the defined pointers.
75x_map.h	This file implements memory mapping and physical registers address definition for both development and debug modes. This file is supplied with all peripherals.
75x_type.h	Common declarations file. It includes common types and constants used by all peripheral drivers.
75x_ppp.c	Driver source code file of PPP peripheral written in C language.
75x_ppp.h	Header file of PPP peripheral. It includes the definition of PPP peripheral functions and variables used within these functions.

The software library architecture and file inclusion relationship are shown in Figure 2.

Figure 2. Software Library File Architecture



Each peripheral has a source code file *75x_ppp.c* and a header file *75x_ppp.h*. The *75x_ppp.c* file contains all the software functions required to use the corresponding peripheral. A single memory mapping file *75x_map.h* is supplied for all peripherals. This file contains all the register declarations for both development and debug modes.

The header file *75x_lib.h* includes all the peripheral header files. This is the only file that needs to be included in the user application to interface with the library.

2.3 How to use the Library

This section describes step-by-step how to configure and initialize a PPP peripheral.

- In your main application file, declare a ***PPP_InitTypeDef*** structure, e.g:

```
PPP_InitTypeDef  PPP_InitStructure;
```

The `PPP_InitStructure` is a working variable located in data memory that allows you to initialize one or more PPP instances.

- Fill the `PPP_InitStructure` variable with the allowed values of the structure member.

There are two ways of doing this:

– Configuration of the whole structure: in this case you should proceed as follows:

```
PPP_InitStructure.member1 = val1;
PPP_InitStructure.member2 = val2;
PPP_InitStructure.memberN = valN; /* where N is the number of
                                     the structure members */
```

Note: The previous initialization could be merged in only one line like the following :

```
PPP_InitTypeDef  PPP_InitStructure = { val1, val2, …, valN}
```

This reduces and optimises code size.

– Configuration of a few members of a structure: in this case you should modify the `PPP_InitStructure` variable that has been already filled by a call to the ***PPP_StructInit(..)*** function. This ensures that the other members of the `PPP_InitStructure` variable have appropriate values (in most case their default values).

```
PPP_StructInit(&PPP_InitStructure);
PPP_InitStructure.memberX = valX;
PPP_InitStructure.memberY = valY; /* where X and Y are the only members
                                     that you want to configure */
```

- You have to initialize the PPP peripheral by calling the ***PPP_Init(..)*** function.

```
PPP_Init(PPP, &PPP_InitStructure);
```

- At this stage the PPP peripheral is initialized and can be enabled by making a call to ***PPP_Cmd(..)*** function.

```
PPP_Cmd(PPP, ENABLE);
```

To use the PPP peripheral, you can use a set of dedicated functions. These functions are specific to the peripheral and for more details refer to *Section 3 on page 24*.

Note: 1 Before configuring a peripheral, you have to enable its clock by calling the following function:

```
MRCC_PeripheralClockConfig (MRCC_Peripheral_PPPx , ENABLE);
```

*2 **PPP_DeInit(..)** function can be used to set all PPP peripheral registers to their reset values:*

```
PPP_DeInit(PPP);
```

3 *If after peripheral configuration, you want to modify one or more peripheral settings you should proceed as follows:*

```
PPP_InitStucture.memberX = valX;
PPP_InitStructure.memberY = valY; /* where X and Y are the only members
                                     that user wants to modify*/
PPP_Init(PPP, &PPP_InitStructure);
```

3 Peripheral software overview

This chapter describes each peripheral software library in detail. The related functions are fully documented. An example of use of the function is given and some important considerations are also provided.

Functions are described in the format below:

Function name	The name of the peripheral function
Function prototype	Prototype declaration
Behavior Description	Brief explanation of how the functions are executed
Input Parameter {x}	Description of the input parameters
Output parameter {x}	Description of the output parameters
Return Value	Value returned by the function
Required Preconditions	Requirements before to call the function
Called Functions	Other library functions called by the function

4 Configuration Registers (CFG)

The Configuration Register (CFG) is used to enable or disable FLASH burst mode and the USB filter. It reports the FLASH status and allows the remapping of the boot memory.

The first section describes the register structures used in the CFG software library. The second one presents the software library functions.

4.1 CFG register structure

The CFG register structure *CFG_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 GLCONF;
} CFG_TypeDef;
```

The following table presents the CFG registers:

Register	Description
GLCONF	Global Configuration Register

The CFG peripheral is declared in the same file:

```
...
#define CONFIG_BASE    0x60000000
...
#define CFG_BASE           (CONFIG_BASE + 0x0010)
...
#ifndef DEBUG
...
#define CFG           ((CFG_TypeDef *)           CFG_BASE)
...
#else
...
#ifdef _CFG
  EXT CFG_TypeDef           *CFG;
#endif /*_CFG */
...
#endif
```

When debug mode is used, CFG pointer is initialized in 75x_lib.c file:

```
#ifdef _CFG
  CFG = (CFG_TypeDef *)  CFG_BASE;
#endif /*_CFG */
```

_CFG variable must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _CFG
```

4.2 Software library functions

The following table lists the various functions of the CFG library.

Function Name	Description
CFG_BootSpaceConfig	Selects which memory space will be remapped at address 0x00.
CFG_FLASHBurstConfig	Enables or disables the FLASH Burst mode.
CFG_USBFilterConfig	Enables or disables the USB Filter.
CFG_GetFlagStatus	Checks whether the FLASH Busy flag is set or not.

4.2.1 CFG_BootSpaceConfig

Function Name	CFG_BootSpaceConfig
Function Prototype	`void CFG_BootSpaceConfig(u32 CFG_BootSpace)`
Behavior Description	Selects which memory space will be remapped at address 0x00.
Input Parameter	CFG_BootSpace: specifies the memory space to be remapped at address 0x00. Refer to section "*CFG_BootSpace*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

CFG_BootSpace

To select the memory space to remap at address 0x00, use one of the following values:

CFG_BootSpace	Meaning
CFG_BootSpace_FLASH	Embedded FLASH sector B0F0 mapped at 0h
CFG_BootSpace_SRAM	Embedded SRAM mapped at 0h
CFG_BootSpace_ExtSMI	SMI Bank 0 mapped at 0h

Example:

```
/* Remap embedded SRAM at address 0x00 */
  CFG_BootSpaceConfig(CFG_BootSpace_SRAM);
```

4.2.2 CFG_FLASHBurstConfig

Function Name	CFG_FLASHBurstConfig
Function Prototype	`void CFG_FLASHBurstConfig(u32 CFG_FLASHBurst)`
Behavior Description	Enables or disables the FLASH Burst mode.
Input Parameter	CFG_FLASHBurst: specifies the new state of the FLASH Burst mode. Refer to section "*CFG_FLASHBurst*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

CFG_FLASHBurst

To enable or disable the FLASH Burst mode, use one of the following values:

CFG_FLASHBurst	Meaning
CFG_FLASHBurst_Disable	FLASH Burst mode disabled
CFG_FLASHBurst_Enable	FLASH in Burst mode

Example:

```
/* Enable FLASH Burst mode */
  CFG_FLASHBurstConfig(CFG_FLASHBurst_Enable);
```

4.2.3 CFG_USBFilterConfig

Function Name	CFG_USBFilterConfig
Function Prototype	`void CFG_USBFilterConfig(u32 CFG_USBFilter)`
Behavior Description	Enables or disables the USB Filter.
Input Parameter	CFG_USBFilter: specifies the new state of the USB Filter. Refer to section "*CFG_USBFilter*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

CFG_USBFilter

To enable or disable the USB Filter, use one of the following values:

CFG_USBFilter	Meaning
CFG_USBFilter_Disable	USB Filter disabled
CFG_USBFilter_Enable	USB Filter enabled

Example:

```
/* Enable the USB Filter */
  CFG_USBFilterConfig(CFG_USBFilter_Enable);
```

4.2.4 CFG_GetFlagStatus

Function Name	CFG_GetFlagStatus
Function Prototype	`FlagStatus CFG_GetFlagStatus(void)`
Behavior Description	Checks whether the FLASH Busy flag is set or not.
Input Parameter	None
Output Parameter	None
Return Parameter	The new state of FLASH Busy flag (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get FLASH Busy flag status */
  FlagStatus Status;
  Status = CFG_GetFlagStatus();
```

5 MCU Reset and Clock Control (MRCC)

The MRCC may be used for a variety of purposes, including power management, low power mode selection and clock configuration.

The first section describes the register structure used in the MRCC software library. The second one presents the software library functions.

5.1 MRCC register structure

The MRCC registers structure *MRCC_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef  struct
{
  vu32 CLKCTL;
  vu32 RFSR;
  vu32 PWRCTRL;
  u32  EMPTY2;
  vu32 PCLKEN;
  vu32 PSWRES;
  u32  EMPTY3[2];
  vu32 BKP0;
  vu32 BKP1;
} MRCC_TypeDef;
```

The following table presents the MRCC registers:

Register	Description
CLKCTL	Clock Control Register
RFSR	Reset Flag Status Register
PWRCTRL	Power Control Register
PCLKEN	Peripheral Clock Enable Register
PSWRES	Peripheral Software Reset Register
BKP0	Backup Register 0
BKP1	Backup Register 1

The MRCC peripheral is declared in the same file:

```
...
#define CONFIG_BASE    0x60000000
...
#define MRCC_BASE          (CONFIG_BASE + 0x0020)
...

#ifndef DEBUG
...
#define MRCC          ((MRCC_TypeDef *) MRCC_BASE)
...
#else
...
#ifdef _MRCC
  EXT MRCC_TypeDef          *MRCC;
#endif /*_MRCC */
...
#endif
```

When debug mode is used, MRCC pointer is initialized in 75x_lib.c file:

```
#ifdef _MRCC
  MRCC = (MRCC_TypeDef *)  MRCC_BASE;
#endif /*_MRCC */
```

_MRCC variable must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _MRCC
```

5.2 Software library functions

The following table lists the various functions of the MRCC library.

Function Name	Description
MRCC_DeInit	Deinitializes the MRCC peripheral registers to their default reset values.
MRCC_XTDIV2Config	Enables or disables the oscillator divider by 2.
MRCC_CKSYSConfig	Configures the system clock (CK_SYS).
MRCC_HCLKConfig	Configures the AHB clock (HCLK).
MRCC_CKTIMConfig	Configures the TIM clock (CK_TIM).
MRCC_PCLKConfig	Configures the APB clock (PCLK).
MRCC_CKRTCConfig	Configures the RTC clock (CK_RTC).
MRCC_CKUSBConfig	Configures the USB clock (CK_USB).
MRCC_ITConfig	Enables or disables the specified MRCC interrupts.
MRCC_PeripheralClockConfig	Enables or disables the specified peripheral clock.
MRCC_PeripheralSWResetConfig	Enters or exit the specified peripheral to/from reset.
MRCC_GetClocksStatus	Returns the status and frequencies of different on chip clocks.
MRCC_LPMC_DBGConfig	Enables or disables Low Power Debug Mode.
MRCC_EnterWFIMode	Enters WFI Mode.
MRCC_EnterSTOPMode	Enters STOP mode.
MRCC_EnterSTANDBYMode	Enters STANDBY mode.
MRCC_GenerateSWReset	Generates a system software reset.
MRCC_WriteBackupRegister	Writes user data to the specified backup register.
MRCC_ReadBackupRegister	Reads data from the specified backup register.
MRCC_IOVoltageRangeConfig	Configures the I/O pins voltage range.
MRCC_MCOConfig	Selects the clock source to output on MCO pin (P0.1).
MRCC_OSC4MConfig	Configures the 4MHz main oscillator (OSC4M).
MRCC_OSC32KConfig	Configures the OSC32K oscillator.
MRCC_LPOSCConfig	Enables or disables the LPOSC oscillator.
MRCC_RTCMConfig	Enables or disables RTC clock measurement.
MRCC_SetBuilderCounter	Sets the builder counter value which defines the delay for the 4MHz main oscillator (OSC4M) clock to be stabilized.
MRCC_GetCKSYSCounter	Gets the result of the delay applied to CK_SYS before starting the CPU.
MRCC_GetFlagStatus	Checks whether the specified MRCC flag is set or not.
MRCC_ClearFlag	Clears the MRCC's pending flags.
MRCC_GetITStatus	Checks whether the specified MRCC interrupt has occurred or not.

Function Name	Description
MRCC_ClearITPendingBit	Clears the MRCC's interrupt pending bits.
MRCC_WaitForOSC4MStartUp	Waits for OSC4M start-up.

5.2.1 MRCC_DeInit

Function Name	MRCC_DeInit
Function Prototype	`void MRCC_DeInit(void)`
Behavior Description	Deinitializes the MRCC peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Note: 1 Depending on the system clock state, some bits in MRCC_CLKCTL register can't be reset.

2 The OSC32K, LPOSC and RTC clock selection configuration bits in MRCC_PWRCTRL register are not cleared by this function. To reset those bits, use the dedicated functions available within this driver.

3 The MRCC_RFSR, MRCC_BKP0 and MRCC_BKP1 registers are not reset by this function.

Example:

```
/* Deinitialize the MRCC peripheral */
  MRCC_DeInit();
```

5.2.2 MRCC_XTDIV2Config

Function Name	MRCC_XTDIV2Config
Function Prototype	`void MRCC_XTDIV2Config(u32 MRCC_XTDIV2)`
Behavior Description	Enables or disables the oscillator divider by 2.
Input Parameter	MRCC_XTDIV2: specifies the new state of the oscillator divider by 2. Refer to section "*MRCC_XTDIV2*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	This function must not be used when the PLL is enabled.
Called functions	None

MRCC_XTDIV2

To enable or disable oscillator divider by 2, use one of the following values:

MRCC_XTDIV2	Meaning
MRCC_XTDIV2_Disable	Oscillator divider by 2 disbaled
MRCC_XTDIV2_Enable	Oscillator divider by 2 enbaled

Example:

```
/* Enable the oscillator divider by 2 */
  MRCC_XTDIV2Config(MRCC_XTDIV2_Enable);
```

5.2.3 MRCC_CKSYSConfig

Function Name	MRCC_CKSYSConfig
Function Prototype	`ErrorStatus MRCC_CKSYSConfig(u32 MRCC_CKSYS, u32 MRCC_PLL)`
Behavior Description	Configures the system clock (CK_SYS).
Input Parameter1	MRCC_CKSYS: specifies the clock source used as system clock. Refer to section "*MRCC_CKSYS*" for more details on the allowed values of this parameter.
Input Parameter2	MRCC_PLL: specifies the PLL configuration. Refer to section "*MRCC_PLL*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: - SUCCESS: Clock configuration succeeded - ERROR: Clock configuration failed
Required preconditions	To use the RTC clock as system clock, the RTC clock source must be previously configured using *MRCC_CKRTCConfig()* function.
Called functions	None

MRCC_CKSYS

To select the system clock, use one of the following values:

MRCC_CKSYS	Meaning
MRCC_CKSYS_FREEOSC	Internal VCO of the PLL configured in free running mode used as system clock
MRCC_CKSYS_OSC4M	OSC4M used as system clock
MRCC_CKSYS_OSC4MPLL	OSC4M followed by PLL used as system clock
MRCC_CKSYS_RTC	RTC clock used as system clock

MRCC_PLL

To configure the PLL, use one of the following values:

MRCC_PLL	Meaning
MRCC_PLL_Disabled	PLL disabled
MRCC_PLL_NoChange	No change on PLL configuration
MRCC_PLL_Mul_12	Multiplication by 12
MRCC_PLL_Mul_14	Multiplication by 14
MRCC_PLL_Mul_15	Multiplication by 15
MRCC_PLL_Mul_16	Multiplication by 16

Note: 1 When fetching from internal Flash the max System Clock (CK_SYS) frequency is 60 MHz (64 MHz when fetching from SRAM).

The table below describes the allowed combination of ***MRCC_CKSYS*** and ***MRCC_PLL*** parameters:

		MRCC_CKSYS			
		MRCC_CKSYS_FREEOSC	MRCC_CKSYS_OSC4M	MRCC_CKSYS_OSC4MPLL	MRCC_CKSYS_RTC
MRCC_PLL	MRCC_PLL_Disabled	x	x	-	x
	MRCC_PLL_NoChange	-	x	-	x
	MRCC_PLL_Mul_12	-	-	x	-
	MRCC_PLL_Mul_14	-	-	x	-
	MRCC_PLL_Mul_15	-	-	x	-
	MRCC_PLL_Mul_16	-	-	x	-

"x:" combination allowed

"-": combination not allowed

Note: 1 When the OSC4M followed by PLL is selected as system clock source, using the parameter MRCC_CKSYS_OSC4MPLL, this function enables the PLL and waits for the Lock signal to be set to '1'

2 If an 8 MHz external Quartz oscillator is used as main oscillator (defined in 75x_conf.h file), this function sets to '1' the XTDIV2 bit in MRCC_CLKCTL register.

Example:

```
/* Set CK_SYS to 60 MHz */
  if(MRCC_CKSYSConfig(MRCC_CKSYS_OSC4MPLL, MRCC_PLL_Mul_15) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.4 MRCC_HCLKConfig

Function Name	MRCC_HCLKConfig
Function Prototype	`void MRCC_HCLKConfig(u32 MRCC_HCLK)`
Behavior Description	Configures the AHB clock (HCLK).
Input Parameter	MRCC_HCLK: defines the AHB clock. This clock is derived from the system clock(CK_SYS). Refer to section "*MRCC_HCLK*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_HCLK

To configure the AHB clock, use one of the following values:

MRCC_HCLK	Meaning
MRCC_CKSYS_Div1	AHB clock = CK_SYS
MRCC_CKSYS_Div2	AHB clock = CK_SYS/2
MRCC_CKSYS_Div4	AHB clock = CK_SYS/4
MRCC_CKSYS_Div8	AHB clock = CK_SYS/8

Example:

```
/* Configure HCLK such as HCLK = CK_SYS */
  MRCC_HCLKConfig(MRCC_CKSYS_Div1);
```

5.2.5 MRCC_CKTIMConfig

Function Name	MRCC_CKTIMConfig
Function Prototype	`void MRCC_CKTIMConfig(u32 MRCC_CKTIM)`
Behavior Description	Configures the TIM clock (CK_TIM).
Input Parameter	MRCC_CKTIM: defines the TIM clock. This clock is derived from the AHB clock(HCLK). Refer to section "*MRCC_CKTIM*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_CKTIM

To configure the TIM clock, use one of the following values:

MRCC_CKTIM	Meaning
MRCC_HCLK_Div1	TIM clock = HCLK
MRCC_HCLK_Div2	TIM clock = HCLK/2
MRCC_HCLK_Div4	TIM clock = HCLK/4
MRCC_HCLK_Div8	TIM clock = HCLK/8

Example:

```
/* Configure CKTIM such as CKTIM = HCLK/2 */
  MRCC_CKTIMConfig(MRCC_HCLK_Div2);
```

5.2.6 MRCC_PCLKConfig

Function Name	MRCC_PCLKConfig
Function Prototype	`void MRCC_PCLKConfig(u32 MRCC_PCLK)`
Behavior Description	Configures the APB clock (PCLK).
Input Parameter	MRCC_PCLK: defines the APB clock. This clock is derived from the TIM clock(CK_TIM). Refer to section "*MRCC_PCLK*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_PCLK

To configure the APB clock, use one of the following values:

MRCC_PCLK	Meaning
MRCC_CKTIM_Div1	APB clock = CKTIM
MRCC_CKTIM_Div2	APB clock = CKTIM/2

Example:

```
/* Configure PCLK such as PCLK = CKTIM */
  MRCC_PCLKConfig(MRCC_CKTIM_Div1);
```

5.2.7 MRCC_CKRTCConfig

Function Name	MRCC_CKRTCConfig
Function Prototype	`ErrorStatus MRCC_CKRTCConfig(u32 MRCC_CKRTC)`
Behavior Description	Configures the RTC clock (CK_RTC).
Input Parameter	MRCC_CKRTC: specifies the clock source to be used as RTC clock. Refer to section "*MRCC_CKRTC*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: - SUCCESS: Clock configuration succeeded - ERROR: Clock configuration failed
Required preconditions	None
Called functions	None

MRCC_CKRTC

To select the RTC clock source, use one of the following values:

MRCC_CKRTC	Meaning
MRCC_CKRTC_OSC4M_Div128	The RTC is clocked by the main oscillator divided by a fixed prescaler of 128
MRCC_CKRTC_OSC32K	The RTC is clocked by the 32kHz oscillator (OSC32K must be previously enabled using MRCC_OSC32KConfig() function)
MRCC_CKRTC_LPOSC	The RTC is clocked by the low power RC oscillator (LPOSC must be previously enabled using MRCC_LPOSCConfig() function)

Example:

```
/* Select OSC32K as CK_RTC clock source */
  if(MRCC_CKRTCConfig(MRCC_CKRTC_OSC32K) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.8 MRCC_CKUSBConfig

Function Name	MRCC_CKUSBConfig
Function Prototype	`ErrorStatus MRCC_CKUSBConfig(u32 MRCC_CKUSB)`
Behavior Description	Configures the USB clock (CK_USB).
Input Parameter	MRCC_CKUSB: specifies the clock source to be used as USB clock. Refer to section "*MRCC_CKUSB*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: - SUCCESS: Clock configuration succeeded - ERROR: Clock configuration failed
Required preconditions	The PLL must be enabled and locked.
Called functions	None

MRCC_CKUSB

To select the USB clock source, use one of the following values:

MRCC_CKUSB	Meaning
MRCC_CKUSB_Internal	Internal USB clock used (CK_PLL2 enabled).
MRCC_CKUSB_External	External USB clock used. In this case the CK_PLL2 is disabled.

Example:

```
/* Select internal USB clock source */
  if(MRCC_CKUSBConfig(MRCC_CKUSB_Internal) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.9 MRCC_ITConfig

Function Name	MRCC_ITConfig
Function Prototype	`void MRCC_ITConfig(u32 MRCC_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified MRCC interrupts.
Input Parameter1	MRCC_IT: specifies the MRCC interrupts sources to be enabled or disabled. Refer to section "*MRCC_IT*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified MRCC interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_IT

To enable or disable MRCC interrupts, use a combination of one or more of the following values:

MRCC_IT	Meaning
MRCC_IT_LOCK	PLL Lock interrupt mask
MRCC_IT_NCKD	No Clock Detected interrupt mask

Example:

```
/* Enbale No Clock Detected and PLL Lock interrupts */
  MRCC_ITConfig(MRCC_IT_NCKD | MRCC_IT_LOCK, ENABLE);
```

5.2.10 MRCC_PeripheralClockConfig

Function Name	MRCC_PeripheralClockConfig

Function Prototype	`void MRCC_PeripheralClockConfig(`
	`u32 MRCC_Peripheral,`
	`FunctionalState NewState)`

Behavior Description	Enables or disables the specified peripheral clock.
Input Parameter1	MRCC_Peripheral: specifies the peripheral to gates its clock. Refer to section "*MRCC_Peripheral*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified peripheral clock. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_Peripheral

To select the peripheral to gates its clock, use a combination of one or more of the following values:

MRCC_Peripheral	Meaning
MRCC_Peripheral_ALL	All peripheral clocks
MRCC_Peripheral_EXTIT	EXTIT clock
MRCC_Peripheral_RTC	RTC clock
MRCC_Peripheral_GPIO	GPIO clock
MRCC_Peripheral_UART2	UART2 clock
MRCC_Peripheral_UART1	UART1 clock
MRCC_Peripheral_UART0	UART0 clock
MRCC_Peripheral_I2C	I2C clock
MRCC_Peripheral_CAN	CAN clock
MRCC_Peripheral_SSP1	SSP1 clock
MRCC_Peripheral_SSP0	SSP0 clock
MRCC_Peripheral_USB	USB clock
MRCC_Peripheral_PWM	PWM clock
MRCC_Peripheral_TIM2	TIM2 clock
MRCC_Peripheral_TIM1	TIM1 clock
MRCC_Peripheral_TIM0	TIM0 clock
MRCC_Peripheral_TB	TB clock
MRCC_Peripheral_ADC	ADC clock

Example:

```
/* Enable GPIOs and TB clocks */
  MRCC_PeripheralClockConfig(MRCC_Peripheral_GPIO | MRCC_Peripheral_TB, ENABLE);
```

5.2.11 MRCC_PeripheralSWResetConfig

Function Name	MRCC_PeripheralSWResetConfig

Function Prototype	`void MRCC_PeripheralSWResetConfig(`
	`u32 MRCC_Peripheral,`
	`FunctionalState NewState)`

Behavior Description	Forces or releases a software reset of the specified peripheral.
Input Parameter1	MRCC_Peripheral: specifies the peripheral to reset. Refer to section "*MRCC_Peripheral on page 43*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified peripheral software reset. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enters the TB peripheral to reset */
  MRCC_PeripheralSWResetConfig(MRCC_Peripheral_TB, ENABLE);

/* Exits the TB peripheral from reset */
  MRCC_PeripheralSWResetConfig(MRCC_Peripheral_TB, DISABLE);
```

5.2.12 MRCC_GetClocksStatus

Function Name	MRCC_GetClocksStatus
Function Prototype	`void MRCC_GetClocksStatus(MRCC_ClocksTypeDef* MRCC_ClocksStatus)`
Behavior Description	Returns the status and frequencies of different on chip clocks.
Input Parameter	MRCC_ClocksStatus: pointer to a MRCC_ClocksTypeDef structure which will hold the clock information. Refer to section "*MRCC_ClocksTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_ClocksTypeDef

The MRCC_ClocksTypeDef structure is defined in the *75x_mrcc.h* file:

```
typedef struct
{
  CLKSourceTypeDef CKSYS_Source;
  CLKSourceTypeDef CKRTC_Source;
  CLKSourceTypeDef CKUSB_Source;
  CLKSourceTypeDef PLL_Status;
  CLKSourceTypeDef OSC4M_Status;
  CLKSourceTypeDef LPOSC_Status;
  CLKSourceTypeDef OSC32K_Status;
  u32 CKSYS_Frequency;
  u32 HCLK_Frequency;
  u32 CKTIM_Frequency;
  u32 PCLK_Frequency;
}MRCC_ClocksTypeDef;
```

CKSYS_Source

Returns CK_SYS clock source. This member can be one of the following values:

CKSYS_Source	Meaning
FREEOSC	CK_SYS clocked by the VCO of PLL.
OSC4MPLL	CK_SYS clocked by OSC4M followed by the PLL.
OSC4M	CK_SYS clocked by OSC4M.
CKRTC	CK_SYS clocked by CK_RTC.

CKRTC_Source

Returns CK_RTC clock source. This member can be one of the following values:

CKRTC_Source	Meaning
Disabled	CK_RTC disabled
OSC4M_Div128	CK_RTC clocked by the main oscillator divided by a fixed prescaler of 128
LPOSC	CK_RTC clocked by the low power RC oscillator
OSC32K	CK_RTC clocked by the 32kHz oscillator

CKUSB_Source

Returns CK_USB clock source. This member can be one of the following values:

CKUSB_Source	Meaning
Disabled	CK_USB disabled
Internal	Internal USB clock used
External	External USB clock used

PLL_Status

Returns the PLL status. This member can be one of the following values:

PLL_Status	Meaning
ON	PLL enabled
OFF	PLL disabled

OSC4M_Status

Returns the OSC4M status. This member can be one of the following values:

OSC4M_Status	Meaning
ON	OSC4M enabled
OFF	OSC4M disabled

LPOSC_Status

Returns the LPOSC status. This member can be one of the following values:

LPOSC_Status	Meaning
ON	LPOSC enabled
OFF	LPOSC disabled

OSC32K_Status

Returns the OSC32K status. This member can be one of the following values:

OSC32K_Status	Meaning
ON	OSC32K enabled
OFF	OSC32K disabled

CKSYS_Frequency

Returns CK_SYS clock frequency in Hz.

HCLK_Frequency

Returns HCLK clock frequency in Hz.

CKTIM_Frequency

Returns CKTIM clock frequency in Hz.

PCLK_Frequency

Returns PCLK clock frequency in Hz.

Note: Don't use this function when CK_SYS is clocked by an external clock source (OSC4M bypassed).

Example:

```
/* Gets the status and frequencies of different on chip clocks */
  MRCC_ClocksTypeDef  MRCC_Clocks;
  MRCC_GetClocksStatus(&MRCC_Clocks);
```

5.2.13 MRCC_LPMC_DBGConfig

Function Name	MRCC_LPMC_DBGConfig
Function Prototype	`void MRCC_LPMC_DBGConfig(u32 MRCC_LPDM)`
Behavior Description	Enables or disables Low Power Debug Mode.
Input Parameter	MRCC_LPDM: specifies the LPDM new state value. Refer to section "*MRCC_LPDM*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_LPDM

To specify if the Low Power Debug Mode is enabled or disabled, use one of the following values:

MRCC_LPDM	Meaning
MRCC_LPDM_Disable	Disable Low Power Debug Mode
MRCC_LPDM_Enable	Enable Low Power Debug Mode

Example:

```
/* Enable Low Power Debug Mode */
  MRCC_LPMC_DBGConfig(MRCC_LPDM_Enable);
```

5.2.14 MRCC_EnterWFIMode

Function Name	MRCC_EnterWFIMode
Function Prototype	`void MRCC_EnterWFIMode(u32 MRCC_WFIParam)`
Behavior Description	Enters WFI mode.
Input Parameter	MRCC_WFIParam: specifies the WFI mode control parameters. Refer to section "*MRCC_WFIParam*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	If the Flash is used in **Burst mode**, it must be kept enabled in WFI mode(use MRCC_WFIParam_FLASHOn as parameter)
Called functions	None

MRCC_WFIParam

To select the WFI mode control parameters, use one of the following values:

MRCC_WFIParam	Meaning
MRCC_WFIParam_FLASHPowerDown	WFI Mode with Flash in low power mode (DMA not allowed during WFI).
MRCC_WFIParam_FLASHOn	WFI Mode with Flash enabled (DMA allowed during WFI).
MRCC_WFIParam_FLASHOff	WFI Mode with Flash disabled(DMA not allowed during WFI).

Example:

```
/* Enter WFI mode with Flash enabled */
  MRCC_EnterWFIMode(MRCC_WFIParam_FLASHOn);
```

5.2.15 MRCC_EnterSTOPMode

Function Name	MRCC_EnterSTOPMode
Function Prototype	`void MRCC_EnterSTOPMode(u32 MRCC_STOPParam)`
Behavior Description	Enters STOP mode.
Input Parameter	MRCC_STOPParam: specifies the STOP mode control parameters. Refer to section "*MRCC_STOPParam*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_STOPParam

To select the STOP mode control parameters, use one of the following values:

MRCC_STOPParam	Meaning
MRCC_STOPParam_Default	OSC4M, FLASH and MVREG enabled in STOP mode
MRCC_STOPParam_OSC4MOff[1)]	Disable OSC4M and PLL in STOP mode
MRCC_STOPParam_FLASHOff[1)]	Disable FLASH in STOP mode
MRCC_STOPParam_MVREGOff[2)]	Disable main voltage regulator in STOP mode

1) These two parameters can be used together.

2) If this parameter is selcted, OSC4M and FLASH are also disabled.

Example:

```
/* Enter STOP mode with OSC4M and FLASH OFF */
  MRCC_EnterSTOPMode(MRCC_STOPParam_FLASHOff | MRCC_STOPParam_OSC4MOff);
```

5.2.16 MRCC_EnterSTANDBYMode

Function Name	MRCC_EnterSTANDBYMode
Function Prototype	`void MRCC_EnterSTANDBYMode(void)`
Behavior Description	Enters STANDBY mode.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	Make sure that WKPF flag is cleared before using this function.
Called functions	None

Example:

```
/* Clear Wake-Up flag */
  MRCC_ClearFlag(MRCC_FLAG_WKP);
/* Enter STANDBY mode */
  MRCC_EnterSTANDBYMode();
```

5.2.17 MRCC_GenerateSWReset

Function Name	MRCC_GenerateSWReset
Function Prototype	`void MRCC_GenerateSWReset(void)`
Behavior Description	Generates a system software reset.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate system software reset */
  MRCC_GenerateSWReset();
```

5.2.18 MRCC_WriteBackupRegister

Function Name	MRCC_WriteBackupRegister
Function Prototype	`void MRCC_WriteBackupRegister(MRCC_BKPReg MRCC_BKP, u32 Data)`
Behavior Description	Writes user data to the specified backup register.
Input Parameter1	MRCC_BKP: specifies the backup register. Refer to section "*MRCC_BKP*" for more details on the allowed values of this parameter.
Input Parameter2	Data: data to write.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_BKP

To select the backup register to write to, use one of the following values:

MRCC_BKP	Meaning
MRCC_BKP0	Backup register 0 selected.
MRCC_BKP1	Backup register 1 selected.

Example:

```
/* Write data to Backup register 0 */
  MRCC_WriteBackupRegister(MRCC_BKP0, 0x4D524343);
```

5.2.19 MRCC_ReadBackupRegister

Function Name	MRCC_ReadBackupRegister
Function Prototype	`u32 MRCC_ReadBackupRegister(MRCC_BKPReg MRCC_BKP)`
Behavior Description	Reads data from the specified backup register.
Input Parameter	MRCC_BKP: specifies the backup register. Refer to section "*MRCC_BKP on page 52*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The content of the specified backup register.
Required preconditions	None
Called functions	None

Example:

```
/* Read Backup register 0 */
  u32 wData;
  wData = MRCC_ReadBackupRegister(MRCC_BKP0);
```

5.2.20 MRCC_IOVoltageRangeConfig

Function Name	MRCC_IOVoltageRangeConfig
Function Prototype	`void MRCC_IOVoltageRangeConfig(u32 MRCC_IOVoltageRange)`
Behavior Description	Configures the I/O pins voltage range.
Input Parameter	MRCC_IOVoltageRange: specifies the I/O pins voltage range. Refer to section "*MRCC_IOVoltageRange*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_IOVoltageRange

To select the I/O pin voltage range, use one of the following values:

MRCC_IOVoltageRange	Meaning
MRCC_IOVoltageRange_5V	I/O Pins are optimized for 5V operation, I/O can be used at 3.3V but with downgraded timing characteristics.
MRCC_IOVoltageRange_3V3	I/O Pins are optimized for 3.3V, and 5V operation is forbidden.

Example:

```
/* I/O Pins optimized for 3.3V operation */
  MRCC_IOVoltageRangeConfig(MRCC_IOVoltageRange_3V3);
```

5.2.21 MRCC_MCOConfig

Function Name	MRCC_MCOConfig
Function Prototype	`void MRCC_MCOConfig(u32 MRCC_MCO, u32 MCO_MCOPrescaler)`
Behavior Description	Selects the clock source to output on MCO pin (P0.1). To output the clock, the associated alternate function must be enabled in the I/O port controller.
Input Parameter1	MRCC_MCO: specifies the clock source to output. Refer to section "*MRCC_MCO*" for more details on the allowed values of this parameter.
Input Parameter2	MRCC_MCOPrescaler: specifies if prescaler, divide by 1 or 2, is applied to this clock before outputting it to MCO pin. Refer to section "*MRCC_MCOPrescaler*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	Care must be taken when changing MCO clock selection: to avoid any glitches on the MCO pin, the alternate function must be disabled during the change.
Called functions	None

MRCC_MCO

To select the clock source to output on MCO pin, use one of the following values:

MRCC_MCO	Meaning
MRCC_MCO_HCLK	AHB clock is output to the MCO pin
MRCC_MCO_PCLK	APB clock is output to the MCO pin
MRCC_MCO_OSC4M	4MHz main oscillator (OSC4M) clock is output to the MCO pin
MRCC_MCO_CKPLL2	USB clock is output to the MCO pin

MRCC_MCOPrescaler

To select the MCO prescaler, use one of the following values:

MRCC_MCOPrescaler	Meaning
MRCC_MCOPrescaler_1	MCO is directly applied on MCO pin
MRCC_MCOPrescaler_2	Additional divider by 2 is applied to MCO, reducing the output frequency on the MCO pin

Example:

```
/* Output HCLK clock on MCO pin */
  MRCC_MCOConfig(MRCC_MCO_HCLK, MRCC_MCOPrescaler_1);
```

5.2.22 MRCC_OSC4MConfig

Function Name	MRCC_OSC4MConfig
Function Prototype	`ErrorStatus MRCC_OSC4MConfig(u32 MRCC_OSC4M)`
Behavior Description	Configures the 4 MHz main oscillator (OSC4M). This function must be used when the CK_SYS is not clocked by the OSC4M and the PLL is not enabled.
Input Parameter	MRCC_OSC4M: specifies the new state of the OSC4M oscillator. Refer to section "*MRCC_OSC4M*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: - SUCCESS: Clock configuration succeeded - ERROR: Clock configuration failed
Required preconditions	None
Called functions	None

MRCC_OSC4M

To enable or disable the OSC4M oscillator, use one of the following values:

MRCC_OSC4M	Meaning
MRCC_OSC4M_Default	OSC4M enabled, bypass disabled
MRCC_OSC4M_Disable	OSC4M disabled
MRCC_OSC4M_Bypass	OSC4M bypassed

Example:

```
/* Disable OSC4M */
  if(MRCC_OSC4MConfig(MRCC_OSC4M_Disable) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.23 MRCC_OSC32KConfig

Function Name	MRCC_OSC32KConfig
Function Prototype	`ErrorStatus MRCC_OSC32KConfig(u32 MRCC_OSC32K, u32 MRCC_OSC32KBypass)`
Behavior Description	Configures the OSC32K oscillator. This function must be used when the CK_SYS is not clocked by the CK_RTC.
Input Parameter1	MRCC_OSC32K: specifies the new state of the OSC32K oscillator. Refer to section "*MRCC_OSC32K*" for more details on the allowed values of this parameter.
Input Parameter2	MRCC_OSC32KBypass: specifies if the OSC32K oscillator is bypassed or not. Refer to section "*MRCC_OSC32KBypass*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: - SUCCESS: Clock configuration succeeded - ERROR: Clock configuration failed
Required preconditions	None
Called functions	None

MRCC_OSC32K

To enable or disable the OSC32K oscillator, use one of the following values:

MRCC_OSC32K	Meaning
MRCC_OSC32K_Disable	OSC32K disabled.
MRCC_OSC32K_Enable	OSC32K enabled.

MRCC_OSC32KBypass

To select whether the OSC32K oscillator is bypassed or not, use one of the following values:

MRCC_OSC32KBypass	Meaning
MRCC_OSC32KBypass_Disable	OSC32K selected.
MRCC_OSC32KBypass_Enable	OSC32K bypassed.

Example:

```
/* Enable OSC32K */
  if(MRCC_OSC32KConfig(MRCC_OSC32K_Enable, MRCC_OSC32KBypass_Disable) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.24 MRCC_LPOSCConfig

Function Name	MRCC_LPOSCConfig
Function Prototype	`void MRCC_LPOSCConfig(u32 MRCC_LPOSC)`
Behavior Description	Enables or disables the LPOSC oscillator. This function must be used when the CK_SYS is not clocked by the CK_RTC.
Input Parameter	MRCC_LPOSC: specifies the new state of the LPOSC oscillator. Refer to section "*MRCC_LPOSC*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_LPOSC

To enable or disable the LPOSC oscillator, use one of the following values:

MRCC_OSC32K	Meaning
MRCC_LPOSC_Disable	LPOSC disabled
MRCC_LPOSC_Enable	LPOSC enabled

Example:

```
/* Enable LPOSC */
  if(MRCC_LPOSCConfig(MRCC_LPOSC_Enable) == SUCCESS)
  {
    /* Place your code here */
  }
  else
  {
     /* Add here some code to deal with this error */
  }
```

5.2.25 MRCC_RTCMConfig

Function Name	MRCC_RTCMConfig
Function Prototype	`void MRCC_RTCMConfig(u32 MRCC_RTCM)`
Behavior Description	Enables or disables RTC clock measurement.
Input Parameter	MRCC_RTCM: specifies whether CK_RTC is connected to TB timer IC1 or not. Refer to section "*MRCC_RTCM*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_RTCM

To enable or disable the RTC clock measurement, use one of the following values:

MRCC_RTCM	Meaning
MRCC_RTCM_Disable	CK_RTC not connected to TB timer IC1.
MRCC_RTCM_Enable	CK_RTC connected to TB timer IC1.

Example:

```
/* Enable RTC clock measurement */
  MRCC_RTCMConfig(MRCC_RTCM_Enable);
```

5.2.26 MRCC_SetBuilderCounter

Function Name	MRCC_SetBuilderCounter
Function Prototype	`void MRCC_SetBuilderCounter(u8 BuilderCounter)`
Behavior Description	Sets the builder counter value which defines the delay for the 4 MHz main oscillator (OSC4M) clock to be stabilized.
Input Parameter	BuilderCounter: defines the delay for the OSC4M oscillator clock to be stabilized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set Builder Counter value to 0x80 */
  MRCC_SetBuilderCounter(0x80);
```

5.2.27 MRCC_GetCKSYSCounter

Function Name	MRCC_GetCKSYSCounter
Function Prototype	`u16 MRCC_GetCKSYSCounter(void)`
Behavior Description	Gets the result of the delay applied to CK_SYS before starting the CPU.
Input Parameter	None
Output Parameter	None
Return Parameter	SCOUNT value.
Required preconditions	None
Called functions	None

Example:

```
/* Get CK_SYS Counter value */
  u16 hSCount;
  hSCount = MRCC_GetCKSYSCounter();
```

5.2.28 MRCC_GetFlagStatus

Function Name	MRCC_GetFlagStatus
Function Prototype	`FlagStatus MRCC_GetFlagStatus(u8 MRCC_FLAG)`
Behavior Description	Checks whether the specified MRCC flag is set or not.
Input Parameter	MRCC_FLAG: specifies the flag to check. Refer to section "*MRCC_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of MRCC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

MRCC_FLAG

The MRCC flags that can check for are listed in the following table:

MRCC_FLAG	Meaning
MRCC_FLAG_LOCK	PLL Locked flag
MRCC_FLAG_LOCKIF	PLL Lock Interrupt status flag
MRCC_FLAG_CKSEL	CK_SYS source staus flag
MRCC_FLAG_CKOSCSEL	CK_OSC clock source staus flag
MRCC_FLAG_NCKD	No Clock Detected flag
MRCC_FLAG_SWR	Software Reset flag
MRCC_FLAG_WDGR	Watchdog Reset flag
MRCC_FLAG_EXTR	External Reset flag
MRCC_FLAG_WKP	Wake-Up flag
MRCC_FLAG_STDB	STANDBY flag
MRCC_FLAG_BCOUNT	Builder Counter Flag
MRCC_FLAG_OSC32KRDY	Oscillator 32K Ready
MRCC_FLAG_CKRTCOK	CK_RTC OK
MRCC_FLAG_LPDONE	Low Power Bit Sequence has been performed
MRCC_FLAG_LP	Low Power Mode Entry

Example:

```
/* Get No Clock Detected flag status */
  FlagStatus Status;
  Status = MRCC_GetFlagStatus(MRCC_FLAG_NCKD);
```

5.2.29 MRCC_ClearFlag

Function Name	MRCC_ClearFlag
Function Prototype	`void MRCC_ClearFlag(u8 MRCC_FLAG)`
Behavior Description	Clears the MRCC's pending flags.
Input Parameter	MRCC_FLAG: specifies the flag to clear. Refer to section "*MRCC_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

MRCC_FLAG

To clear MRCC flags, use one of the following values:

MRCC_FLAG	Meaning
MRCC_FLAG_NCKD	No Clock Detected flag
MRCC_FLAG_SWR	Software Reset flag
MRCC_FLAG_WDGR	Watchdog Reset flag
MRCC_FLAG_EXTR	External Reset flag
MRCC_FLAG_WKP	Wake-Up Flag
MRCC_FLAG_STDB	STANDBY Flag
MRCC_FLAG_LPDONE	Low Power Bit Sequence has been performed

Example:

```
/* Clear No Clock Detected flag */
  MRCC_ClearFlag(MRCC_FLAG_NCKD);
```

5.2.30 MRCC_GetITStatus

Function Name	MRCC_GetITStatus
Function Prototype	`ITStatus MRCC_GetITStatus(u32 MRCC_IT)`
Behavior Description	Checks whether the specified MRCC interrupt has occurred or not.
Input Parameter	MRCC_IT: specifies the MRCC interrupt source to check. Refer to section "*MRCC_IT on page 41*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of MRCC_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the status of PLL lock interrupt */
  ITStatus Status;
  Status = MRCC_GetITStatus(MRCC_IT_LOCK);
```

5.2.31 MRCC_ClearITPendingBit

Function Name	MRCC_ClearITPendingBit
Function Prototype	`void MRCC_ClearITPendingBit(u32 MRCC_IT)`
Behavior Description	Clears the MRCC's interrupt pending bits.
Input Parameter	MRCC_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to section "*MRCC_IT on page 41*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear PLL lock interrupt pending bit*/
  MRCC_ClearITPendingBit(MRCC_IT_LOCK);
```

5.2.32 MRCC_WaitForOSC4MStartUp

Function Name	MRCC_WaitForOSC4MStartUp
Function Prototype	`ErrorStatus MRCC_WaitForOSC4MStartUp(void)`
Behavior Description	Waits for OSC4M start-up.
Input Parameter	None
Output Parameter	None
Return Parameter	An ErrorStatus enumuration value: - SUCCESS: OSC4M oscillator is stable and ready to use - ERROR: no clock is detected on OSC4M
Required preconditions	None
Called functions	None

Note: Before using OSC4M oscillator clock, user must call this function to wait for OSC4M clock stabilization. This can occurs:

- after RESET

- after STANDBY

- when software re-enables the OSC4M oscillator

- when waking-up from STOP mode with OSC4M OFF (MRCC_STOPParam_OSC4MOff)

- when the OSC4M clock recovers after an oscillator failure detection

Example:

```
ErrorStatus OSC4MStartUpStatus;

/* Wait for OSC4M start-up */
  OSC4MStartUpStatus = MRCC_WaitForOSC4MStartUp();

  if(OSC4MStartUpStatus == SUCCESS)
  {
    /* Add here code to configure the clocks */
  }
  else
  {
    /* Add here some code to deal with this error */
  }
```

6 General Purpose I/O Ports (GPIO)

The GPIO driver may be used for several purposes, including pin configuration, masking a port pin, reading a port pin and writing data into the port pin.

The first section describes the register structure used in the GPIO software library. The second one presents the software library functions.

6.1 GPIO register structure

The GPIO register structure *GPIO_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 PC0;
  vu32 PC1;
  vu32 PC2;
  vu32 PD;
  vu32 PM;
}GPIO_TypeDef;

typedef struct
{
  vu32 REMAP0R;
  vu32 REMAP1R;
}GPIO_REMAP_TypeDef;
```

The following table presents the GPIO registers:

Register	Description
PC0	Port configuration bits
PC1	Port configuration bits
PC2	Port configuration bits
PD	I/O Data bits
PM	I/O Port Mask bits
REMAP0R	I/O remapping register 0
REMAP1R	I/O remapping register 1

The 3 GPIO peripherals are declared in the same file:

```
...
#define PERIPH_BASE    0xFFFF0000
...
#define GPIO0_BASE          (PERIPH_BASE + 0xE400)
#define GPIOREMAP_BASE     (PERIPH_BASE + 0xE420)
#define GPIO1_BASE          (PERIPH_BASE + 0xE440)
#define GPIO2_BASE          (PERIPH_BASE + 0xE480)

#ifndef DEBUG
...
#define GPIO0 ((GPIO_TypeDef *) GPIO0_BASE)
#define GPIOREMAP ((GPIOREMAP_TypeDef*) GPIOREMAP_BASE)
#define GPIO1 ((GPIO_TypeDef *) GPIO1_BASE)
#define GPIO2 ((GPIO_TypeDef *) GPIO2_BASE)
```

```
...
#else
...
#ifdef _GPIO0
  EXT GPIO_TypeDef          *GPIO0;
#endif /*_GPIO0 */

#ifdef _GPIOREMAP
  EXT GPIOREMAP_TypeDef     *GPIOREMAP;
#endif /*_GPIOREMAP */

#ifdef _GPIO1
  EXT GPIO_TypeDef          *GPIO1;
#endif /*_GPIO1 */

#ifdef _GPIO2
  EXT GPIO_TypeDef          *GPIO2;
#endif /*_GPIO2 */
...
#endif
```

When debug mode is used, GPIO pointer is initialized in 75x_lib.c file:

```
#ifdef _GPIO0
GPIO0 = (GPIO_TypeDef *)GPIO0_BASE;
#endif /*_GPIO0*/

#ifdef _GPIOREMAP
GPIOREMAP = (GPIOREMAP_TypeDef *)GPIOREMAP_BASE;
#endif /*_GPIOREMAP*/

#ifdef _GPIO1
GPIO1 = (GPIO_TypeDef *)GPIO1_BASE;
#endif /*_GPIO1 */

#ifdef _GPIO2
GPIO2 = (GPIO_TypeDef *)GPIO2_BASE;
#endif /*_GPIO2 */
```

_GPIO, _GPIO0, _GPIO1, _GPIO2 and _GPIOREMAP must be defined, in the *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _GPIO
#define _GPIO0
#define _GPIO1
#define _GPIO2
#define _GPIOREMAP
...
```

6.2 Software library functions

The following table lists the various functions of the GPIO library.

Function Name	Description
GPIO_DeInit	Deinitializes the GPIOx peripheral registers to their default reset values.
GPIO_Init	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
GPIO_StructInit	Fills each GPIO_InitStruct member with its default value.
GPIO_Read	Reads the specified GPIO data port
GPIO_ReadBit	Reads the specified data port bit
GPIO_Write	Writes data to the specified GPIO data port
GPIO_WriteBit	Sets or clears the selected data port bit.
GPIO_PinMaskConfig	Enables or disables write protection to the selected bits in the I/O port registers (PxC2, PxC1, PxC0 and PxD).
GPIO_GetPortMask	Gets the GPIOx port mask value.
GPIO_PinRemapConfig	Changes the mapping of the specified pin.

6.2.1 GPIO_DeInit

Function Name	GPIO_DeInit
Function Prototype	`void GPIO_DeInit(GPIO_TypeDef* GPIOx)`
Behavior Description	Deinitializes the GPIOx peripheral registers to their default reset values.
Input Parameter	GPIOx: where x can be 0,1 or 2 to select the GPIO peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Note: The I/O remapping registers 0 and 1 are not reset by this function.

Example:

```
/* Deinitializes the GPIO0 peripheral registers to their default reset values */
  GPIO_DeInit(GPIO0);
```

6.2.2 GPIO_Init

Function Name	GPIO_Init
Function Prototype	`void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)`
Behavior Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct . This function will not change the configuration for a pin if the corresponding mask bit is set, except pins configured as input pull-up or pull-down. These pins are automatically masked after each configuration.
Input Parameter1	GPIOx: where x can be 0,1or 2 to select the GPIO peripheral.
Input Parameter2	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral. Refer to section "*GPIO_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

GPIO_InitTypeDef

The GPIO_InitTypeDef structure is defined in the *75x_gpio.h* file:

```
typedef struct
{
  u32 GPIO_Pin;
  GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
```

GPIO_Pin

Specifies GPIO pins to configure, (allows multiple pin configuration with the | operator). This member can be one of the following values:

GPIO_Pin	Meaning
GPIO_Pin_None	No pin Selected
GPIO_Pin_0	Pin 0 Selected
GPIO_Pin_1	Pin 1 Selected
GPIO_Pin_2	Pin 2 Selected
GPIO_Pin_3	Pin 3 Selected
GPIO_Pin_4	Pin 4 Selected
GPIO_Pin_5	Pin 5 Selected
GPIO_Pin_6	Pin 6 Selected
GPIO_Pin_7	Pin 7 Selected
GPIO_Pin_8	Pin 8 Selected
GPIO_Pin_9	Pin 9 Selected
GPIO_Pin_10	Pin 10 Selected
GPIO_Pin_11	Pin 11 Selected
GPIO_Pin_12	Pin 12 Selected
GPIO_Pin_13	Pin 13 Selected
GPIO_Pin_14	Pin 14 Selected
GPIO_Pin_15	Pin 15 Selected
GPIO_Pin_16	Pin 16 Selected
GPIO_Pin_17	Pin 17 Selected
GPIO_Pin_18	Pin 18 Selected
GPIO_Pin_19	Pin 19 Selected
GPIO_Pin_20	Pin 20 Selected
GPIO_Pin_21	Pin 21 Selected
GPIO_Pin_22	Pin 22 Selected
GPIO_Pin_23	Pin 23 Selected
GPIO_Pin_24	Pin 24 Selected
GPIO_Pin_25	Pin 25 Selected
GPIO_Pin_26	Pin 26 Selected
GPIO_Pin_27	Pin 27 Selected
GPIO_Pin_28	Pin 28 Selected
GPIO_Pin_29	Pin 29 Selected
GPIO_Pin_30	Pin 30 Selected

GPIO_Pin	Meaning
GPIO_Pin_31	Pin 31 Selected
GPIO_Pin_All	All Pins Selected

GPIO_Mode

Specifies the working mode for the selected pins. This member can be one of the following values:

GPIO_Mode	Meaning
GPIO_Mode_AIN	Analog Input
GPIO_Mode_IN_FLOATING	Input Floating
GPIO_Mode_IPD	Input Pull-Down
GPIO_Mode_IPU	Input Pull-up
GPIO_Mode_Out_OD	Open Drain Output
GPIO_Mode_Out_PP	Push-Pull Output
GPIO_Mode_AF_OD	Open Drain Output Alternate-Function
GPIO_Mode_AF_PP	Push-Pull Output Alternate-Function

Note: For pin configured in input mode, the selection between Pull-Up or Pull-Down is defined by the corresponding bit value in the Data Register (1:Pull-Up, 0:Pull-Down). In some cases, writing to PD register may change this pin configuration. For this when GPIO_Mode_IPD or GPIO_Mode_IPU mode is selected the corresponding pin is write protected (using port mask register).

Example:

```
/* Configure all GPIO0 pins in Push-Pull Output mode */
  GPIO_InitTypeDef  GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIO0, &GPIO_InitStructure);
```

6.2.3 GPIO_StructInit

Function Name	GPIO_StructInit
Function Prototype	`void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)`
Behavior Description	Fills each GPIO_InitStruct member with its default value, refer to the following table for moe details.
Input Parameter	GPIO_InitStruct: pointer to a GPIO_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The GPIO_InitStruct members have the following default values:

Member	Default value
GPIO_Pin	GPIO_Pin_All
GPIO_Mode	GPIO_Mode_IN_FLOATING

Example:

```
/* Initialize the GPIO Init Structure parameters */
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_StructInit(&GPIO_InitStructure);
```

6.2.4 GPIO_Read

Function Name	GPIO_Read
Function Prototype	`u32 GPIO_Read(GPIO_TypeDef* GPIOx)`
Behavior Description	Reads the specified GPIO data port
Input Parameter	GPIOx: where x can be 0,1, or 2 to select the GPIO peripheral.
Output Parameter	None
Return Parameter	GPIO data port word value.
Required preconditions	None
Called functions	None

Example:

```
/* Read the GPIO2 data port and store it in ReadValue variable */
  u32 ReadValue;
  ReadValue = GPIO_Read(GPIO2);
```

6.2.5 GPIO_ReadBit

Function Name	GPIO_ReadBit
Function Prototype	`u8 GPIO_ReadBit(GPIO_TypeDef* GPIOx, u32 GPIO_Pin)`
Behavior Description	Reads the specified data port bit.
Input Parameter1	GPIOx: where x can be 0,1or 2 to select the GPIO peripheral.
Input Parameter2	GPIO_Pin: specifies the port bit to read. Refer to section "*GPIO_Pin on page 68*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The port pin value.
Required preconditions	None
Called functions	None

Example:

```
/* Reads the seventh pin of the GPIO1 and store it in ReadValue variable */
  u8 ReadValue;
  ReadValue = GPIO_ReadBit(GPIO1, GPIO_Pin_7);
```

6.2.6 GPIO_Write

Function Name	GPIO_Write
Function Prototype	`void GPIO_Write(GPIO_TypeDef* GPIOx, u32 PortVal)`
Behavior Description	Writes data to the specified GPIO data port.
Input Parameter1	GPIOx: where x can be 0,1, or 2 to select the GPIO peripheral.
Input parameter2	PortVal: specifies the value to be written to the data port register.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Write data to GPIO0 data port */
  GPIO_Write(GPIO0, 0x11011266);
```

6.2.7 GPIO_WriteBit

Function Name	GPIO_WriteBit
Function Prototype	`void GPIO_WriteBit(GPIO_TypeDef* GPIOx,u32 GPIO_Pin, BitAction BitVal)`
Behavior Description	Sets or clears the selected data port bit.
Input parameter1	GPIOx: where x can be 0,1, or 2 to select the GPIO peripheral.
Input parameter2	GPIO_Pin: specifies the port bit to be written. Refer to section "*: GPIO_Pin on page 68*" for more details on the allowed values of this parameter.
Input Parameter3	BitVal: specifies the value to be written to the selected bit. BitVal must be one of the BitAction enum values: Bit_RESET: to clear the port pin. Bit_SET: to set the port pin.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the GPIO0 port pin 15 */
  GPIO_WriteBit(GPIO0, GPIO_Pin_15, Bit_SET);
```

6.2.8 GPIO_PinMaskConfig

Function Name	GPIO_PinMaskConfig
Function Prototype	`void GPIO_PinMaskConfig(GPIO_TypeDef* GPIOx, u32 GPIO_Pin, FunctionalState NewState)`
Behavior Description	Enables or disables write protection to the selected bits in the I/O port registers (PxC2, PxC1, PxC0 and PxD).
Input Parameter1	GPIOx: where x can be 0,1, or 2 to select the GPIO peripheral.
Input Parameter2	GPIO_Pin: specifies the port bit to be protected. Refer to section "*GPIO_Pin on page 68*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the port pin. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Protect the GPIO0 pin 5 and pin 14 from write operation */
  GPIO_PinMaskConfig(GPIO0, GPIO_Pin_5 | GPIO_Pin_14, ENABLE);
```

6.2.9 GPIO_GetPortMask

Function Name	GPIO_GetPortMask
Function Prototype	`u32 GPIO_GetPortMask(GPIO_TypeDef* GPIOx)`
Behavior Description	Gets the GPIOx port mask value.
Input Parameter1	GPIOx: where x can be 0,1, or 2 to select the GPIO peripheral.
Output Parameter	None
Return Parameter	GPIO port mask value.
Required preconditions	None
Called functions	None

Example:

```
/* Get GPIO0 port mask value */
  u32 PortMaskValue;
  PortMaskValue = GPIO_GetPortMask(GPIO0);
```

6.2.10 GPIO_PinRemapConfig

Function Name	GPIO_PinRemapConfig
Function Prototype	`void GPIO_PinRemapConfig(u16 GPIO_Remap, FunctionalState NewState)`
Behavior Description	Changes the mapping of the specified pin.
Input Parameter1	GPIO_Remap: selects the pin to remap. Refer to section "*GPIO_Remap*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the port pin. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

GPIO_Remap

To change the pin mapping, use one of the following values:

GPIO_Remap	**Meaning**
GPIO_Remap_SMI_CS3_EN	Enable SMI CS3
GPIO_Remap_SMI_CS2_EN	Enable SMI CS2
GPIO_Remap_SMI_CS1_EN	Enable SMI CS1
GPIO_Remap_SMI_EN	Enable SMI Alternate Functions: SMI_CS0, SMI_CK, SMI_DIN and SMI_DOUT
GPIO_Remap_DBGOFF	JTAG Disable

GPIO_Remap	Meaning
GPIO_Remap_UART1	UART1 Alternate Function mapping
GPIO_Remap_UART2	UART2 Alternate Function mapping
GPIO_Remap_SSP1	SSP1 Alternate Function mapping
GPIO_Remap_TIM2	TIM2 Alternate Function mapping
GPIO_Remap_TIM0	TIM0 Alternate Function mapping

Example:

```
/* TIM2_OC2 on P1.02, TIM2_TI2 on P1.03 */
  GPIO_PinRemapConfig(GPIO_Remap_TIM2, ENABLE);
```

7 Enhanced Interrupt Controller (EIC)

The driver may be used for several purposes, such as enabling and disabling interrupts (IRQ) and fast interrupts (FIQ), enabling and disabling individual IRQ channels, changing IRQ channel priorities, saving and restoring context and installing IRQ handlers.

The first section describes the register structure used in the EIC software library. The second one presents the software library functions.

7.1 EIC register structure

The EIC register structure *EIC_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 ICR;
  vuc32 CICR;
  vu32 CIPR;
  u32  EMPTY1;
  vu32 FIER;
  vu32 FIPR;
  vu32 IVR;
  vu32 FIR;
  vu32 IER;
  u32  EMPTY2[7];
  vu32 IPR;
  u32  EMPTY3[7];
  vu32 SIRn[32];
} EIC_TypeDef;
```

The following table presents the EIC registers:

Register	Description
ICR	Interrupt Control Register
CICR	Current Interrupt Channel Register
CIPR	Current Interrupt Priority Register
FIER	Fast Interrupt Enable Register
FIPR	Fast Interrupt Pending Register
IVR	Interrupt Vector Register
FIR	Fast Interrupt Register
IER	Interrupt Enable Register
IPR	Interrupt Pending Register
SIRn	Channel n Source Interrupt Register

The EIC peripheral is declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define EIC_BASE        (PERIPH_BASE + 0xF800)
...

#ifndef DEBUG
...
#define EIC   ((EIC_TypeDef *)  EIC_BASE)
...
#else
...
#ifdef _EIC
  EXT EIC_TypeDef           *EIC;
#endif /*_EIC */
...
#endif
```

When debug mode is used, EIC pointer is initialized in 75x_lib.c file:

```
#ifdef _EIC
  EIC = (EIC_TypeDef *)  EIC_BASE;
#endif /*_EIC */
```

_EIC variable must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _EIC
```

7.2 Software library functions

The following table lists the various functions of the EIC library.

Function Name	Description
EIC_DeInit	Deinitializes the EIC peripheral registers to their default reset values.
EIC_IRQInit	Configures the IRQ channels according to the specified parameters in the EIC_IRQInitStruct.
EIC_FIQInit	Configures the FIQ channels according to the specified parameters in the EIC_FIQInitStruct.
EIC_IRQStructInit	Fills each EIC_IRQInitStruct member with its default value.
EIC_FIQStructInit	Fills each EIC_FIQInitStruct member with its default value.
EIC_IRQCmd	Enables or disables EIC IRQ output request to CPU.
EIC_FIQCmd	Enables or disables EIC FIQ output request to CPU.
EIC_GetCurrentIRQChannel	Returns the current served IRQ channel identifier.
EIC_GetCurrentIRQChannelPriority	Returns the priority level of the current served IRQ channel.
EIC_CurrentIRQPriorityConfig	Changes the priority of the current served IRQ channel.
EIC_GetCurrentFIQChannel	Returns the current served FIQ channel identifier.
EIC_ClearFIQPendingBit	Clears the pending bit of the selected FIQ Channel.

7.2.1 EIC_DeInit

Function Name	EIC_DeInit
Function Prototype	`void EIC_DeInit(void)`
Behavior Description	Deinitializes the EIC peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Note: The IVR and SIRn registers are not reset by this function.

Example:

```
/* Deinitialize the EIC peripheral */
EIC_DeInit();
```

7.2.2 EIC_IRQInit

Function Name	EIC_IRQInit
Function Prototype	`void EIC_IRQInit(EIC_IRQInitTypeDef* EIC_IRQInitStruct)`
Behavior Description	Configures the IRQ channels according to the specified parameters in the EIC_IRQInitStruct.
Input Parameter	EIC_IRQInitStruct: pointer to a EIC_IRQInitTypeDef structure. Refer to section "*EIC_IRQInitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

EIC_IRQInitTypeDef

The EIC_IRQInitTypeDef structure is defined in the *75x_eic.h* file:

```
typedef struct
{
  u8 EIC_IRQChannel;
  u8 EIC_IRQChannelPriority;
  FunctionalState EIC_IRQChannelCmd;
}EIC_IRQInitTypeDef;
```

EIC_IRQChannel

Specifies the IRQ channel to be enabled or disabled. This member can be one of the following values:

EIC_IRQChannel	Meaning
WAKUP_IRQChannel	External WAKUP pin
TIM2_OC2_IRQChannel	TIM2 Timer Output Compare 2
TIM2_OC1_IRQChannel	TIM2 Timer Output Compare 1
TIM2_IC12_IRQChannel	TIM2 Timer Input Capture 1 & 2
TIM2_UP_IRQChannel	TIM2 Timer Update
TIM1_OC2_IRQChannel	TIM1 Timer Output Compare 2
TIM1_OC1_IRQChannel	TIM1 Timer Output Compare 1
TIM1_IC12_IRQChannel	TIM1 Timer Input Capture 1 & 2
TIM1_UP_IRQChannel	TIM1 Timer Update
TIM0_OC2_IRQChannel	TIM0 Timer Output Compare 2
TIM0_OC1_IRQChannel	TIM0 Timer Output Compare 1
TIM0_IC12_IRQChannel	TIM0 Timer Input Capture 1 & 2
TIM0_UP_IRQChannel	TIM0 Timer Update
PWM_OC123_IRQChannel	PWM Timer Output Compare 1, 2 & 3
PWM_EM_IRQChannel	PWM Timer Emergency
PWM_UP_IRQChannel	PWM Timer Update
I2C_IRQChannel	I2C Global interrupt
SSP1_IRQChannel	SSP1 Interrupt
SSP0_IRQChannel	SSP0 Interrupt
UART2_IRQChannel	UART2 Global interrupt
UART1_IRQChannel	UART1 Global interrupt
UART0_IRQChannel	UART0 Global interrupt
CAN_IRQChannel	CAN Global interrupt
USB_LP_IRQChannel	USB Low Priority Event Interrupt
USB_HP_IRQChannel	USB High Priority Event Interrupt
ADC_IRQChannel	ADC Global interrupt
DMA_IRQChannel	DMA Global Interrupt
EXTIT_IRQChannel	External Interrupt lines INT14 to INT1
MRCC_IRQChannel	MRCC Clock Controller Global interrupt
FLASHSMI_IRQChannel	FLASH and SMI Global interrupt
RTC_IRQChannel	RTC Global interrupt
TB_IRQChannel	TB Timer Global Interrupt

EIC_IRQChannelPriority

Specifies the IRQ channel, specified in the *EIC_IRQChannel* member, priority level. This member must be a number between 0 and 15.

EIC_IRQChannelCmd

Specifies whether the IRQ channel specified in the *EIC_IRQChannel* member will be enabled or disabled. This member can be: ENABLE or DISABLE.

Example:

```
/* Enable and configure the priority of the TB IRQ Channel*/
  EIC_IRQInitTypeDef  EIC_IRQInitStructure;
  EIC_IRQInitStructure.EIC_IRQChannel = TB_IRQChannel;
  EIC_IRQInitStructure.EIC_IRQChannelPriority = 1;
  EIC_IRQInitStructure.EIC_IRQChannelCmd = ENABLE;
  EIC_IRQInit(&EIC_IRQInitStructure);
```

7.2.3 EIC_FIQInit

Function Name	EIC_FIQInit
Function Prototype	`void EIC_FIQInit(EIC_FIQInitTypeDef* EIC_FIQInitStruct)`
Behavior Description	Configures the FIQ channels according to the specified parameters in the EIC_FIQInitStruct.
Input Parameter	EIC_FIQInitStruct: pointer to a EIC_FIQInitTypeDef structure. Refer to section "*EIC_FIQInitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

EIC_FIQInitTypeDef

The EIC_FIQInitTypeDef structure is defined in the *75x_eic.h* file:

```
typedef struct
{
  u8 EIC_FIQChannel;
  FunctionalState EIC_FIQChannelCmd;
}EIC_FIQInitTypeDef;
```

EIC_FIQChannel

Specifies the FIQ channel to be enabled or disabled. This member can be one of the following values:

EIC_FIQChannel	Meaning
EXTIT_Line0_FIQChannel	External interrupt line INT0
WATCHDOG_FIQChannel	Watchdog global interrupt

EIC_FIQChannelCmd

Specifies whether the IRQ channel specified in the *EIC_FIQChannel* member will be enabled or disabled. This member can be: ENABLE or DISABLE.

Example:

```
/* Enable WDG FIQ Channel */
  EIC_FIQInitTypeDef  EIC_FIQInitStructure;
  EIC_FIQInitStructure.EIC_FIQChannel = WATCHDOG_FIQChannel;
  EIC_FIQInitStructure.EIC_FIQChannelCmd = ENABLE;
  EIC_FIQInit(&EIC_FIQInitStructure);
```

7.2.4 EIC_IRQStructInit

Function Name	EIC_IRQStructInit
Function Prototype	`void EIC_IRQStructInit(EIC_IRQInitTypeDef* EIC_IRQInitStruct)`
Behavior Description	Fills each EIC_IRQInitStruct member with its default value, refer to the following table for more details.
Input Parameter	EIC_IRQInitStruct: pointer to a EIC_IRQInitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The EIC_IRQInitStruct members have the following default values:

Member	Default value
EIC_IRQChannel	0x1F
EIC_IRQChannelPriority	0
EIC_IRQChannelCmd	DISABLE

Example:

```
/* The following example illustrates how to initialize a EIC_IRQInitTypeDef structure
*/
EIC_IRQInitTypeDef  EIC_IRQInitStructure;
EIC_IRQStructInit(&EIC_IRQInitStructure);
```

7.2.5 EIC_FIQStructInit

Function Name	EIC_FIQStructInit
Function Prototype	`void EIC_FIQStructInit(EIC_FIQInitTypeDef* EIC_FIQInitStruct)`
Behavior Description	Fills each EIC_FIQInitStruct member with its default value, refer to the following table for more details.
Input Parameter	EIC_FIQInitStruct: pointer to a EIC_FIQInitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The EIC_FIQInitStruct members have the following default values:

Member	Default value
EIC_FIQChannel	0x03
EIC_FIQChannelCmd	DISABLE

Example:

```
/*  The following example illustrates how to initialize a EIC_FIQInitTypeDef
structure */
  EIC_FIQInitTypeDef  EIC_FIQInitStructure;
  EIC_FIQStructInit(&EIC_FIQInitStructure);
```

7.2.6 EIC_IRQCmd

Function Name	EIC_IRQCmd
Function Prototype	`void EIC_IRQCmd(FunctionalState NewState)`
Behavior Description	Enables or disables EIC IRQ output request to CPU.
Input Parameter	NewState: new state of the EIC IRQ output request to CPU. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the Interrupt controller to manage IRQ channel*/
  EIC_IRQCmd(ENABLE);
```

7.2.7 EIC_FIQCmd

Function Name	EIC_FIQCmd
Function Prototype	`void EIC_FIQCmd(FunctionalState NewState)`
Behavior Description	Enables or disables EIC FIQ output request to CPU.
Input Parameter	NewState: new state of the EIC FIQ output request to CPU. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the Interrupt controller to manage FIQ channel*/
  EIC_FIQCmd(ENABLE);
```

7.2.8 EIC_GetCurrentIRQChannel

Function Name	EIC_GetCurrentIRQChannel
Function Prototype	`u8 EIC_GetCurrentIRQChannel(void)`
Behavior Description	Returns the current served IRQ channel identifier.
Input Parameter	None
Output Parameter	None
Return Parameter	The current served IRQ channel.
Required preconditions	None
Called functions	None

Example:

```
/* Get the current served IRQ channel identifier */
  u8 CurrentIRQChannel;
  CurrentIRQChannel = EIC_GetCurrentIRQChannel();
```

7.2.9 EIC_GetCurrentIRQChannelPriority

Function Name	EIC_GetCurrentIRQChannelPriority
Function Prototype	`u8 EIC_GetCurrentIRQChannelPriority(void)`
Behavior Description	Returns the priority level of the current served IRQ channel.
Input Parameter	None
Output Parameter	None
Return Parameter	The priority level of the current served IRQ channel.
Required preconditions	None
Called functions	None

Example:

```
/* Get the priority level of the current served IRQ channel */
  u8 CurrentIRQChannelPriority;
  CurrentIRQChannelPriority = EIC_GetCurrentIRQChannelPriority();
```

7.2.10 EIC_CurrentIRQPriorityConfig

Function Name	EIC_CurrentIRQPriorityConfig
Function Prototype	`void EIC_CurrentIRQPriorityConfig(u8 NewPriority)`
Behavior Description	Changes the priority of the current served IRQ channel.
Input Parameter	NewPriority: new priority value of the IRQ interrupt routine currently serviced.
Output Parameter	None
Return Parameter	None
Required preconditions	The new priority value must be higher, or equal, than the priority value associated to the interrupt channel currently serviced.
Called functions	None

Example:

```
/* Set the priority of the current served IRQ channel to 10 */
  EIC_CurrentIRQPriorityConfig(10);
```

7.2.11 EIC_GetCurrentFIQChannel

Function Name	EIC_GetCurrentFIQChannel
Function Prototype	`u8 EIC_GetCurrentFIQChannel(void)`
Behavior Description	Returns the current served FIQ channel identifier.
Input Parameter	None
Output Parameter	None
Return Parameter	The current served FIQ channel.
Required preconditions	None
Called functions	None

Example:

```
/* Get the current served FIQ channel identifier */
  u8 CurrentFIQChannel;
  CurrentFIQChannel = EIC_GetCurrentFIQChannel();
```

7.2.12 EIC_ClearFIQPendingBit

Function Name	EIC_ClearFIQPendingBit
Function Prototype	`void EIC_ClearFIQPendingBit(u8 EIC_FIQChannel)`
Behavior Description	Clears the pending bit of the selected FIQ Channel.
Input Parameter	EIC_FIQChannel: specifies the FIQ channel to clear its pending bit. Refer to section *EIC_FIQChannel on page 81* for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear WDG FIQ pending bit */
  EIC_ClearFIQPendingBit(WATCHDOG_FIQChannel);
```

8 External Interrupt Controller (EXTIT)

The External interrupt controller consists of up to 16 edge detectors for generating interrupt requests. Each interrupt line can be independently configured to select the trigger event (rising or falling) and can be masked independently. A pending register maintains the status of the interrupt requests.

The first section describes the register structure used in the EXTIT software library. The second one presents the software library functions.

8.1 EXTIT register structure

The EXTIT register structure *EXTIT_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 MR;
  vu32 TSR;
  vu32 SWIR;
  vu32 PR;
} EXTIT_TypeDef;
```

The following table presents the EXTIT registers:

Register	Description
MR	Interrupt Mask Register
TSR	Trigger Selection Register
SWIR	Software Interrupt Register
PR	Pending Register

The EXTIT peripheral is declared in the same file:

```
...
#define PERIPH_BASE    0xFFFF0000
...
#define EXTIT_BASE         (PERIPH_BASE + 0xF400)

#ifndef DEBUG
...
#define EXTIT          ((EXTIT_TypeDef *)         EXTIT_BASE)
...
#else
...
#ifdef _EXTIT
  EXT EXTIT_TypeDef         *EXTIT;
#endif /*_EXTIT */
...
#endif
```

When debug mode is used, EXTIT pointer is initialized in 75x_lib.c file:

```
#ifdef _EXTIT
  EXTIT = (EXTIT_TypeDef *)EXTIT_BASE;
#endif /*_EXTIT */
```

_EXTIT must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _EXTIT
```

8.2 Software library functions

The following table lists the various functions of the EXTIT library.

Function Name	Description
EXTIT_DeInit	Deinitializes the EXTIT peripheral registers to their default reset values.
EXTIT_Init	Initializes the EXTIT peripheral according to the specified parameters in the EXTIT_InitStruct.
EXTIT_StructInit	Fills each EXTIT_InitStruct member with its default value.
EXTIT_GenerateSWInterrupt	Generates a Software interrupt.
EXTIT_GetFlagStatus	Checks whether the specified EXTIT line flag is set or not.
EXTIT_ClearFlag	Clears the EXTIT's line pending flags.
EXTIT_GetITStatus	Checks whether the specified EXTIT line is asserted or not.
EXTIT_ClearITPendingBit	Clears the EXTIT's line pending bits.

8.2.1 EXTIT_DeInit

Function Name	EXTIT_DeInit
Function Prototype	`void EXTIT_DeInit(void)`
Behavior Description	Deinitializes the EXTIT peripheral registers to their default reset values.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitializes the EXTIT */
EXTIT_DeInit();
```

8.2.2 EXTIT_Init

Function Name	EXTIT_Init
Function Prototype	`void EXTIT_Init(EXTIT_InitTypeDef* EXTIT_InitStruct)`
Behavior Description	Initializes the EXTIT peripheral according to the specified parameters in the EXTIT_InitStruct .
Input Parameter	EXTIT_InitStruct: pointer to a EXTIT_InitTypeDef structure that contains the configuration information for the specified EXTIT peripheral. Refer to section "*EXTIT_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

EXTIT_InitTypeDef

The EXTIT_InitTypeDef structure is defined in the *75x_extit.h* file:

```
typedef struct
{
  u32 EXTIT_ITLine;
  EXTITTrigger_TypeDef EXTIT_ITTrigger;
  FunctionalState EXTIT_ITLineCmd;
} EXTIT_InitTypeDef;
```

EXTIT_ITLine

Specifies the external lines to be enabled or disabled. This member can be one of the following values:

EXTIT_ITLine	Meaning
EXTIT_ITLineNone	No interrupt selected
EXTIT_ITLine0	External interrupt line 0
EXTIT_ITLine1	External interrupt line 1
EXTIT_ITLine2	External interrupt line 2
EXTIT_ITLine3	External interrupt line 3
EXTIT_ITLine4	External interrupt line 4
EXTIT_ITLine5	External interrupt line 5
EXTIT_ITLine6	External interrupt line 6
EXTIT_ITLine7	External interrupt line 7
EXTIT_ITLine8	External interrupt line 8
EXTIT_ITLine9	External interrupt line 9
EXTIT_ITLine10	External interrupt line 10
EXTIT_ITLine11	External interrupt line 11
EXTIT_ITLine12	External interrupt line 12
EXTIT_ITLine13	External interrupt line 13
EXTIT_ITLine14	External interrupt line 14
EXTIT_ITLine15	External interrupt line 15

EXTIT_ITTrigger

Specifies the trigger edge for the enabled lines. This member can be one of the following values:

EXTIT_ITTrigger	Meaning
EXTIT_ITTrigger_Falling	Interrupt request configured on falling edge of the input line
EXTIT_ITTrigger_Rising	Interrupt request configured on rising edge of the input line

EXTIT_ITLineCmd

Specifies the new state of the selected line. This member can be: ENABLE or DISABLE.

Example:

```
/* Enables external lines 12 and 14 interrupt generation on falling edge */
  EXTIT_InitTypeDef EXTIT_InitStructure;
  EXTIT_InitStructure.EXTIT_ITLine = EXTIT_ITLine12 | EXTIT_ITLine14;
  EXTIT_InitStructure.EXTIT_ITTrigger = EXTIT_ITTrigger_Falling;
  EXTIT_InitStructure.EXTIT_ITLineCmd = ENABLE;
  EXTIT_Init(&EXTIT_InitStructure);
```

8.2.3 EXTIT_StructInit

Function Name	EXTIT_StructInit
Function Prototype	`void EXTIT_StructInit(EXTIT_InitTypeDef* EXTIT_InitStruct)`
Behavior Description	Fills each EXTIT_InitStruct member with its default value, refer to the following table for moe details.
Input Parameter	EXTIT_InitStruct: pointer to a EXTIT_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The EXTIT_InitStruct members have the following default values:

Member	Default value
EXTIT_ITLine	EXTIT_ITLineNone
EXTIT_ITTrigger	EXTIT_ITTrigger_Falling
EXTIT_ITLineCmd	DISABLE

Example:

```
/* Initialize the EXTIT Init Structure parameters */
  EXTIT_InitTypeDef EXTIT_InitStructure;
  EXTIT_StructInit(&EXTIT_InitStructure);
```

8.2.4 EXTIT_GenerateSWInterrupt

Function Name	EXTIT_GenerateSWInterrupt
Function Prototype	`void EXTIT_GenerateSWInterrupt(u32 EXTIT_ITLine)`
Behavior Description	Generates a Software interrupt.
Input Parameter	EXTIT_ITLine: specifies the EXTIT lines to be enabled or disabled. Refer to section "*EXTIT_ITLine on page 91*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a software interrupt request */
  EXTIT_GenerateSWInterrupt(EXTIT_ITLine6);
```

8.2.5 EXTIT_GetFlagStatus

Function Name	EXTIT_GetFlagStatus
Function Prototype	`FlagStatus EXTIT_GetFlagStatus(u16 EXTIT_ITLine)`
Behavior Description	Checks whether the specified EXTIT line flag is set or not.
Input Parameter	EXTIT_ITLine: specifies the EXTIT lines flag to check.. Refer to section "*EXTIT_ITLine on page 91*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of EXTIT_ITLine (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the status of EXTIT line 8 */
  FlagStatus EXTITStatus;
  EXTITStatus = EXTIT_GetFlagStatus(EXTIT_ITLine8);
```

8.2.6 EXTIT_ClearFlag

Function Name	EXTIT_ClearFlag
Function Prototype	`void EXTIT_ClearFlag(u16 EXTIT_ITLine)`
Behavior Description	Clears the EXTIT's line pending flags.
Input Parameter	EXTIT_ITLine: specifies the EXTIT lines flags to clear. Refer to section "*: EXTIT_ITLine on page 91*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the EXTIT line 2 pending flag */
  EXTIT_ClearFlag(EXTIT_ITLine2);
```

8.2.7 EXTIT_GetITStatus

Function Name	EXTIT_GetITStatus
Function Prototype	`ITStatus EXTIT_GetITStatus(u16 EXTIT_ITLine)`
Behavior Description	Checks whether the specified EXTIT line is asserted or not.
Input Parameter	EXTIT_ITLine: specifies the EXTIT lines pending bits to check. Refer to section "*: EXTIT_ITLine on page 91*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of EXTIT_ITLine (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the status of EXTIT line 8 */
  ITStatus EXTITStatus;
  EXTITStatus = EXTIT_GetITStatus(EXTIT_ITLine8);
```

8.2.8 EXTIT_ClearITPendingBit

Function Name	EXTIT_ClearITPendingBit
Function Prototype	`void EXTIT_ClearITPendingBit(u16 EXTIT_ITLine)`
Behavior Description	Clears the EXTIT's line pending bits.
Input Parameter	EXTIT_ITLine: specifies the EXTIT lines pending bits to clear. Refer to section "*EXTIT_ITLine on page 91*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clears the EXTIT line 2 interrupt pending bit */
  EXTIT_ClearITpendingBit(EXTIT_ITLine2);
```

9 DMA Controller (DMA)

The DMA controller provides access to 4 data streams. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

The first section describes the register structure used in the DMA software library. The second one presents the software library functions.

9.1 DMA register structures

The DMA register structures *DMA_Stream_TypeDef and DMA_TypeDef* are defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16  SOURCEL;
  u16   EMPTY1;
  vu16  SOURCEH;
  u16   EMPTY2;
  vu16  DESTL;
  u16   EMPTY3;
  vu16  DESTH;
  u16   EMPTY4;
  vu16  MAX;
  u16   EMPTY5;
  vu16  CTRL;
  u16   EMPTY6;
  vuc16 SOCURRH;
  u16   EMPTY7;
  vuc16 SOCURRL;
  u16   EMPTY8;
  vuc16 DECURRH;
  u16   EMPTY9;
  vuc16 DECURRL;
  u16   EMPTY10;
  vuc16 TCNT;
  u16   EMPTY11;
  vu16  LUBUff;
  u16   EMPTY12;
} DMA_Stream_TypeDef;

typedef struct
{
  vu16 MASK;
  u16  EMPTY4;
  vu16 CLR;
  u16  EMPTY5;
  vuc16 STATUS;
  u16  EMPTY6;
  vu16 LAST;
  u16  EMPTY7;
} DMA_TypeDef;
```

The following table presents the DMA registers:

Register	Description
SOURCELx	*DMA Streamx* Source Base Address Low
SOURCEHx	*DMA Streamx* Source Base Address High
DESTLx	*DMA Streamx* Destination Base Address Low
DESTHx	*DMA Streamx* Destination Base Address High
MAXx	*DMA Streamx* Maximum Count Register
CTRLx	*DMA Streamx* Control Register
SOCURRHx	*DMA Streamx* Current Source Address High
SOCURRLx	*DMA Streamx* Current Source Address Low
DECURRHx	*DMA Streamx* Current Destination Address High
DECURRLx	*DMA Streamx* Current Destination Address Low
TCNTx	*DMA Streamx* Terminal Counter Register
LUBUFFx	*DMA Streamx* Last Used Buffer location
MASK	*DMA* Interrupt Mask Register
CLR	*DMA* Interrupt Clear Register
STATUS	*DMA* Interrupt Status Register
LAST	*DMA* Last Flag Register

The four DMA streams are declared in the same file :

```
...
#define PERIPH_BASE   0xFFFF0000
....
#define DMA_BASE            (PERIPH_BASE + 0xECF0)
#define DMA_Stream0_BASE    (PERIPH_BASE + 0xEC00)
#define DMA_Stream1_BASE    (PERIPH_BASE + 0xEC40)
#define DMA_Stream2_BASE    (PERIPH_BASE + 0xEC80)
#define DMA_Stream3_BASE    (PERIPH_BASE + 0xECC0)

#ifndef DEBUG
  ...
#define DMA ((DMA_TypeDef *)  DMA_BASE)
#define DMA_Stream0 ((DMA_Stream_TypeDef *) DMA_Stream0_BASE)
#define DMA_Stream1 ((DMA_Stream_TypeDef *) DMA_Stream1_BASE)
#define DMA_Stream2 ((DMA_Stream_TypeDef *) DMA_Stream2_BASE)
#define DMA_Stream3 ((DMA_Stream_TypeDef *) DMA_Stream3_BASE)
  ...
#else
  ...
#ifdef _DMA
  EXT DMA_TypeDef           *DMA;
#endif /*_DMA */

#ifdef _DMA_Stream0
  EXT DMA_Stream_TypeDef    *DMA_Stream0;
#endif /*_DMA_Stream0 */

#ifdef _DMA_Stream1
  EXT DMA_Stream_TypeDef    *DMA_Stream1;
#endif /*_DMA_Stream1 */
```

```
#ifdef _DMA_Stream2
  EXT DMA_Stream_TypeDef    *DMA_Stream2;
#endif /*_DMA_Stream2 */

#ifdef _DMA_Stream3
  EXT DMA_Stream_TypeDef    *DMA_Stream3;
#endif /*_DMA_Stream3 */
  ...
#endif
```

When debug mode is used, DMA pointer is initialized in 75x_lib.c file :

```
#ifdef _DMA
  DMA = (DMA_TypeDef *)  DMA_BASE;
#endif /*_DMA */

#ifdef _DMA_Stream0
  DMA_Stream0 = (DMA_Stream_TypeDef *)  DMA_Stream0_BASE;
#endif /*_DMA_Stream0 */

#ifdef _DMA_Stream1
  DMA_Stream1 = (DMA_Stream_TypeDef *)  DMA_Stream1_BASE;
#endif /*_DMA_Stream1 */

#ifdef _DMA_Stream2
  DMA_Stream2 = (DMA_Stream_TypeDef *)  DMA_Stream2_BASE;
#endif /*_DMA_Stream2 */

#ifdef _DMA_Stream3
  DMA_Stream3 = (DMA_Stream_TypeDef *)  DMA_Stream3_BASE;
#endif /*_DMA_Stream3 */
```

_DMA, _DMA_Stream0, _DMA_Stream1, _DMA_Stream2 and _DMA_Stream3 must be defined, in 75x_conf.h file, to access the peripheral registers as follows :

```
#define _DMA
#define _DMA_Stream0
#define _DMA_Stream1
#define _DMA_Stream2
#define _DMA_Stream3
...
```

9.2 Software library functions

The following table lists the various functions of the DMA library.

Function Name	Description
DMA_DeInit	Deinitializes the DMA stream registers to their default reset values.
DMA_Init	Initializes the DMA stream according to the specified parameters in the DMA_InitStruct .
DMA_StructInit	Fills each DMA_InitStruct member with its default value.
DMA_Cmd	Enables or disables the specified DMA stream.
DMA_ITConfig	Enables or disables the specified DMA interrupts.
DMA_GetCurrDSTAddr	Returns the current value of the destination address pointer related to the specified DMA stream.
DMA_GetCurrSRCAddr	Returns the current value of the source address pointer related to the specified DMA stream.
DMA_GetTerminalCounter	Returns the number of data units remaining in the current DMA stream transfer.
DMA_LastBufferSweepConfig	Activates or deactivates the last buffer sweep mode for the DMA stream configured in circular buffer mode.
DMA_LastBufferAddrConfig	Configures the circular buffer position where the last data to be used by the specified DMA stream is located.
DMA_GetFlagStatus	Checks whether the specified DMA flag is set or not.
DMA_ClearFlag	Clears the DMA's pending flags.
DMA_GetITStatus	Checks whether the specified DMA interrupt has occurred or not.
DMA_ClearITPendingBit	Clears the DMA's interrupt pending bits.

9.2.1 DMA_DeInit

Function Name	DMA_DeInit
Function Prototype	`void DMA_DeInit(DMA_Stream_TypeDef* DMA_Streamx)`
Behavior Description	Deinitializes the selected DMA stream registers to their default reset values.
Input Parameter	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Deinitialize the DMA stream2 */
DMA_DeInit(DMA_Stream2);
```

9.2.2 DMA_Init

Function Name	DMA_Init
Function Prototype	`void DMA_Init(DMA_Stream_TypeDef* DMA_Streamx, DMA_InitTypeDef* DMA_InitStruct)`
Behavior Description	Initializes the DMA stream according to the specified parameters in the DMA_InitStruct.
Input Parameter1	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Input Parameter2	DMA_InitStruct : pointer to a DMA_InitTypeDef structure that contains the configuration information for the specified DMA stream. Refer to the section "*DMA_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

DMA_InitTypeDef

The DMA_InitTypeDef structure is defined in the *75x_dma.h* file:

```
typedef struct
{
  u32 DMA_SRCBaseAddr;
  u32 DMA_DSTBaseAddr;
  u16 DMA_BufferSize;
  u16 DMA_SRC;
  u16 DMA_DST;
  u16 DMA_SRCSize;
  u16 DMA_SRCBurst;
  u16 DMA_DSTSize;
  u16 DMA_Mode;
  u16 DMA_M2M;
  u16 DMA_DIR;
} DMA_InitTypeDef;
```

DMA_SRCBaseAddr

Specifies the base address for streamx source DMA buffer.

DMA_DSTBaseAddr

Specifies the base address for streamx destination DMA buffer.

DMA_BufferSize

Specifies the buffer size, in data unit, of the specified stream. The data unit is equal to the configuration set in DMA_SRCSize member.

DMA_SRC

Specifies whether the current source register is incremented. This member can be one of the following values.

DMA_SRC	Meaning
DMA_SRC_INCR	Current source register incremented
DMA_SRC_NOT_INCR	Current source register unchanged

DMA_DST

Specifies whether the current destination register is incremented. This member can be one of the following values.

DMA_DST	Meaning
DMA_DST_INCR	Current destination register incremented
DMA_DST_NOT_INCR	Current destination register unchanged

DMA_SRCSize

Specifies the data width for source to DMA stream data transfer. This member can be one of the following values.

DMA_SRCSize	Meaning
DMA_SRCSize_Byte	Data width is 8 bits
DMA_SRCSize_HalfWord	Data width is 16 bits
DMA_SRCSize_Word	Data width is 32 bits

DMA_SRCBurst

Specifies the number of words in the peripheral burst. This member can be one of the following values.

DMA_SRCBurst	Meaning
DMA_SRCBurst_1Data	1 data transferred
DMA_SRCBurst_4Data	4 data transferred
DMA_SRCBurst_8Data	8 data transferred
DMA_SRCBurst_16Data	16 data transferred

Note: The DMA_SRCBurst_xData member specifies the number of transfered data where its width correspond to the DMA_SRCSize value which could be a byte, HalfWord or Word.

DMA_DSTSize

Specifies the data width for DMA stream to destination data transfer. This member can be one of the following values.

DMA_DSTSize	Meaning
DMA_DSTSize_Byte	Data width is 8 bits
DMA_DSTSize_HalfWord	Data width is 16 bits
DMA_DSTSize_Word	Data width is 32 bits

DMA_Mode

Specifies the operation mode of the DMAx stream. This member can be one of the following values.

DMA_Mode	Meaning
DMA_Mode_Circular	Circular buffer mode is used
DMA_Mode_Normal	Normal buffer mode is used

Note: The circular buffer mode cannot be used in the following situations:

- When buffer size (defined by the DMA_BufferSize member) is not a multiple of the configured burst size (defined by the DMA_SRCBurst member).

- When the memory to memory data transfer is configured on stream3.

DMA_M2M

Specifies whether stream3 memory- to-memory transfer is enabled. This member must be filled only if stream3 is used. This member can be one of the following values.

DMA_M2M	Meaning
DMA_M2M_Enable	Stream3 configured for memory-to-memory transfer
DMA_M2M_Disable	Stream3 not configured for memory-to-memory transfer

DMA_DIR

Specifies if the peripheral is the source or the destination. This member can be one of the following values.

DMA_DIR	Meaning
DMA_DIR_PeriphDST	Peripheral is the destination
DMA_DIR_PeriphSRC	Peripheral is the source

Example:

```
/* Initialize the DMA stream3 according to the DMA_InitStructure members */
  DMA_InitTypeDef  DMA_InitStructure;

  DMA_InitStructure.DMA_SRCBaseAddr = 0x80001000;
  DMA_InitStructure.DMA_DSTBaseAddr = 0xFFFF8400;
  DMA_InitStructure.DMA_BufferSize = 256;
  DMA_InitStructure.DMA_SRC = DMA_SRC_NOT_INCR;
  DMA_InitStructure.DMA_DST = DMA_DST_NOT_INCR;
  DMA_InitStructure.DMA_SRCSize = DMA_SRCSize_HalfWord;
  DMA_InitStructure.DMA_SRCBurst = DMA_SRCBurst_4Data;
  DMA_InitStructure.DMA_DSTSize = DMA_DSTSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeriphDST;
  DMA_Init(DMA_Stream3, &DMA_InitStructure);
```

9.2.3 DMA_StructInit

Function Name	DMA_StructInit
Function Prototype	`void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct)`
Behavior Description	Fills each DMA_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	DMA_InitStruct: pointer to a DMA_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

The DMA_InitStruct members default values are as follows:

Member	Default value
DMA_BufferSize	0
DMA_SRCBaseAddr	0
DMA_DSTBaseAddr	0
DMA_SRC	DMA_SRC_NOT_INCR
DMA_DST	DMA_DST_NOT_INCR
DMA_SRCSize	DMA_SRCSize_Byte
DMA_SRCBurst	DMA_SRCBurst_1Data
DMA_DSTSize	DMA_DSTSize_Byte
DMA_Mode	DMA_Mode_Normal
DMA_M2M	DMA_M2M_Disable
DMA_DIR	DMA_DIR_PeriphSRC

Example:

```
/* Initialize a DMA_InitTypeDef structure. */
DMA_InitTypeDef  DMA_InitStructure;
DMA_StructInit(&DMA_InitStructure);
```

9.2.4 DMA_Cmd

Function Name	DMA_Cmd
Function Prototype	`void DMA_Cmd(DMA_Stream_TypeDef* DMA_Streamx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified DMA stream.
Input Parameter1	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Input Parameter2	NewState : new state of the DMA streamx. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Enable DMA stream3*/
DMA_Cmd(DMA_Stream3, ENABLE);
```

9.2.5 DMA_ITConfig

Function Name	DMA_ITConfig
Function Prototype	`void DMA_ITConfig(u16 DMA_IT,FunctionalState NewState)`
Behavior Description	Enables or disables the specified DMA interrupts.
Input Parameter 1	DMA_IT: specifies the DMA interrupts sources to be enabled or disabled. You can select more than one interrupt, by ORing them. Refer to the section "*DMA_IT*" for more details on the allowed values of this parameter.
Input Parameter 2	NewState : new state of the specified DMA interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

DMA_IT

To enable or disable DMA interrupts, use a combination of one or more of the following values:

DMA_IT	Meaning
DMA_IT_SI0	Stream0 transfer end interrupt mask
DMA_IT_SI1	Stream1 transfer end interrupt mask
DMA_IT_SI2	Stream2 transfer end interrupt mask
DMA_IT_SI3	Stream3 transfer end interrupt mask
DMA_IT_SE0	Stream0 transfer error interrupt mask
DMA_IT_SE1	Stream1 transfer error interrupt mask
DMA_IT_SE2	Stream2 transfer error interrupt mask
DMA_IT_SE3	Stream3 transfer error interrupt mask
DMA_IT_ALL	All DMA interrupts enable/disable mask

Example:

```
/* Enable DMA stream3 transfer end interrupt. */
DMA_ITConfig(DMA_IT_SI3, ENABLE);
```

9.2.6 DMA_GetCurrDSTAddr

Function Name	DMA_GetCurrDSTAddr
Function Prototype	`u32 DMA_GetCurrDSTAddr(DMA_Stream_TypeDef* DMA_Streamx)`
Behavior Description	Returns the current value of the destination address pointer related to the specified DMA stream.
Input Parameter	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Output Parameter	None
Return Parameter	The current value of the destination address pointer related to the specified DMA stream.
Required preconditions	None
Called Functions	None

Example:

```
/* Get DMA stream3 current destination address value. */
u32 wCurrDSTAddr;
wCurrDSTAddr = DMA_GetCurrDSTAddr(DMA_Stream3);
```

9.2.7 DMA_GetCurrSRCAddr

Function Name	DMA_GetCurrSRCAddr
Function Prototype	`u32 DMA_GetCurrSRCAddr(DMA_Stream_TypeDef* DMA_Streamx)`
Behavior Description	Returns the current value of the source address pointer related to the specified DMA stream.
Input Parameter	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Output Parameter	None
Return Parameter	The current value of the source address pointer related to the specified DMA stream.
Required preconditions	None
Called Functions	None

Example:

```
/* Get DMA stream3 current source address value. */
u32 wCurrSRCAddr;
wCurrSRCAddr = DMA_GetCurrSRCAddr(DMA_Stream3);
```

9.2.8 DMA_GetTerminalCounter

Function Name	DMA_GetTerminalCounter
Function Prototype	`u16 DMA_GetTerminalCounter(DMA_Stream_TypeDef* DMA_Streamx)`
Behavior Description	Returns the number of data units remaining in the current DMA stream transfer.
Input Parameter	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Output Parameter	None
Return Parameter	The number of data units remaining in the current DMA stream transfer.
Required preconditions	None
Called Functions	None

Example:

```
/* Get  the number of data units remaining in the current DMA stream3 transfer. */
  u16 hTerminalCount;
  hTerminalCount = DMA_GetTerminalCounter(DMA_Stream3);
```

9.2.9 DMA_LastBufferSweepConfig

Function Name	DMA_LastBufferSweepConfig
Function Prototype	`void DMA_LastBufferSweepConfig( DMA_Stream_TypeDef* DMA_Streamx, FunctionalState NewState)`
Description	Activates or deactivates the last buffer sweep mode for the DMA stream configured in circular buffer mode.
Input Parameter1	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Input Parameter 2	NewState : new state of the Last buffer sweep DMA_Streamx. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameters	None
Called Functions	None

Example:

```
/* Activates the DMA stream3 last circular buffer sweep */
DMA_LastBufferSweepConfig(DMA_Stream3, ENABLE);
/* Deactivates the DMA stream1 last circular buffer sweep */
DMA_LastBufferSweepConfig(DMA_Stream1, DISABLE);
```

9.2.10 DMA_LastBufferAddrConfig

Function Name	DMA_LastBufferAddrConfig
Function Prototype	`void DMA_LastBufferAddrConfig(DMA_Stream_TypeDef* DMA_Streamx, u16 LastBufferAddr)`
Description	Configures the circular buffer position where the last data to be used by the specified DMA stream is located.
Input Parameter 1	DMA_Stream*x*: where x can be 0,1,2 or 3 to select the DMA stream.
Input Parameter 2	LastBufferAddr : specifies the circular buffer position where the last data to be used by the specified DMA stream is located. This member must be a number between 0 and the stream BufferSize-1.
Output Parameter	None
Return Parameters	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Configure the circular buffer position where the last data to be used by the DMA
stream3 is located. */
u16 hLastBufferAddress = 12;
DMA_LastBufferAddrConfig(DMA_Stream3, hLastBufferAddress);
```

9.2.11 DMA_GetFlagStatus

Function Name	DMA_GetFlagStatus
Function Prototype	`FlagStatus DMA_GetFlagStatus(u16 DMA_FLAG)`
Description	Checks whether the specified DMA flag is set or not.
Input Parameter	DMA_FLAG : specifies the flag to check. Refer to the section "*DMA_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameters	The new state of DMA_FLAG (SET or RESET).
Required Preconditions	None
Called Functions	None

DMA_FLAG

The DMA flags that can check for are listed in the following table:

DMA_FLAG	Meaning
DMA_FLAG_SI0	Stream0 transfer end interrupt flag
DMA_FLAG_SI1	Stream1 transfer end interrupt flag
DMA_FLAG_SI2	Stream2 transfer end interrupt flag
DMA_FLAG_SI3	Stream3 transfer end interrupt flag
DMA_FLAG_SE0	Stream0 transfer error interrupt flag
DMA_FLAG_SE1	Stream1 transfer error interrupt flag
DMA_FLAG_SE2	Stream2 transfer error interrupt flag
DMA_FLAG_SE3	Stream3 transfer error interrupt flag
DMA_FLAG_ACT0[1)]	Stream0 status
DMA_FLAG_ACT1[1)]	Stream1 status
DMA_FLAG_ACT2[1)]	Stream2 status
DMA_FLAG_ACT3[1)]	Stream3 status

1) These flags can't be cleared.

Example:

```
/* Test if the DMA stream3 transfer end interrupt flag is set or not. */
FlagStatus Status;
Status = DMA_GetFlagStatus(DMA_FLAG_SI3);
```

9.2.12 DMA_ClearFlag

Function Name	DMA_ClearFlag
Function Prototype	`void DMA_ClearFlag(u16 DMA_FLAG)`
Behavior Description	Clears the DMA's pending flags.
Input Parameter 1	DMA_FLAG: specifies the flag to clear. More than one flag can be cleared using the "|" operator. Refer to the section "*DMA_FLAG on page 110*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Clear the DMA stream3 transfer end interrupt pending bit */
DMA_ClearFlag(DMA_FLAG_SI3);
```

9.2.13 DMA_GetITStatus

Function Name	DMA_GetITStatus
Function Prototype	`ITStatus DMA_GetITStatus(u16 DMA_IT)`
Description	Checks whether the specified DMA interrupt has occurred or not.
Input Parameter	DMA_IT : specifies the DMA interrupt source to check. Refer to the section "*DMA_IT*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameters	The new state of DMA_IT (SET or RESET).
Required Preconditions	None
Called Functions	None

DMA_IT

The DMA interrupts that can be checked for are listed in the following table:

DMA_IT	Meaning
DMA_IT_SI0	Stream0 transfer end interrupt
DMA_IT_SI1	Stream1 transfer end interrupt
DMA_IT_SI2	Stream2 transfer end interrupt
DMA_IT_SI3	Stream3 transfer end interrupt
DMA_IT_SE0	Stream0 transfer error interrupt
DMA_IT_SE1	Stream1 transfer error interrupt
DMA_IT_SE2	Stream2 transfer error interrupt
DMA_IT_SE3	Stream3 transfer error interrupt

Example:

```
/* Test if the DMA stream3 transfer end interrupt has occurred or not. */
ITStatus Status;
Status = DMA_GetITStatus(DMA_IT_SI3);
```

9.2.14 DMA_ClearITPendingBit

Function Name	DMA_ClearITPendingBit
Function Prototype	`void DMA_ClearITPendingBIt(u16 DMA_IT)`
Behavior Description	Clears the DMA's interrupt pending bits.
Input Parameter 1	DMA_IT :specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to the section "*DMA_IT on page 106*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Clear the DMA stream3 transfer end interrupt pending bit */
DMA_ClearITPendingBit(DMA_IT_SI3);
```

10 Serial memory interface (SMI)

The Serial Memory Interface provides an AHB slave interface to external SPI memory devices. The CPU can use this memory as data or program memory.

The first section describes the register structure used in the SMI software library. The second one presents the software library functions.

10.1 SMI registers structure

The SMI registers structure *SMI_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 CR1;
  vu32 CR2;
  vu32 SR;
  vu32 TR;
  vuc32 RR;
} SMI_TypeDef;
```

The following table presents the SMI registers:

Register	Description
CR1	Control Register 1
CR2	Control Register 2
SR	Status Register
TR	Transmit Register
RR	Receive Register

The SMI peripheral is declared in the same file:

```
...
#define SMIR_BASE       0x90000000
...
#ifndef DEBUG
...
#define SMI   ((SMI_TypeDef *)  SMIR_BASE)
...
#else
...
#ifdef _SMI
  EXT SMI_TypeDef           *SMI;
#endif /*_SMI */
...
#endif
```

When debug mode is used, SMI pointer is initialized in 75x_lib.c file:

```
#ifdef _SMI
  SMI = (SMI_TypeDef *)  SMIR_BASE;
#endif /*_SMI */
```

_SMI, must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _SMI
```

10.2 Software library functions

The following table lists the various functions of the SMI library.

Function Name	Description
SMI_DeInit	Deinitializes the SMI peripheral registers to their default reset values.
SMI_Init	Initializes the SMI peripheral according to the specified parameters in the SMI_InitStruct.
SMI_StructInit	Fills each SMI_InitStruct member with its default value.
SMI_ModeConfig	Selects the SMI mode: hardware or software.
SMI_TxRxLengthConfig	Configures the number of bytes to be transmitted and received to/from external memory.
SMI_BankCmd	Enables or disables the specified memory Bank.
SMI_ITConfig	Enables or disables the specified SMI interrupts.
SMI_SelectBank	Selects the memory Bank to be accessed.
SMI_SendWENCmd	Sends a Write Enable command to the selected memory Bank.
SMI_SendRSRCmd	Sends a Read Status Register command to the selected memory Bank.
SMI_SendCmd	Sends command to the selected memory Bank.
SMI_FastReadConfig	Enables or disables the Fast Read Mode.
SMI_WriteBurstConfig	Enables or disables the Write Burst Mode.
SMI_WriteByte	Writes a Byte to the selected memory Bank
SMI_WriteHalfWord	Writes a Half Word to the selected memory Bank.
SMI_WriteWord	Writes a Word to the selected memory Bank.
SMI_ReadByte	Reads a Byte from the selected memory Bank.
SMI_ReadHalfWord	Reads a Half Word from the selected memory Bank.
SMI_ReadWord	Reads a Word from the selected memory Bank.
SMI_ReadMemoryStatusRegister	Reads the status register of the memory connected to the selected Bank.
SMI_GetFlagStatus	Checks whether the specified SMI flag is set or not.
SMI_ClearFlag	Clears the SMI's pending flags.
SMI_GetITStatus	Checks whether the specified SMI interrupt has occurred or not.
SMI_ClearITPendingBit	Clears the SMI's interrupt pending bits.

10.2.1 SMI_DeInit

Function Name	SMI_DeInit
Function Prototype	`void SMI_DeInit(void)`
Behavior Description	Deinitializes the SMI peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	This function must not be used when booting from the SMI external memory.
Called functions	None

Example:

```
/* Deinitialize the SMI peripheral */
  SMI_DeInit();
```

10.2.2 SMI_Init

Function Name	SMI_Init
Function Prototype	`void SMI_Init(SMI_InitTypeDef* SMI_InitStruct)`
Behavior Description	Initializes the SMI peripheral according to the specified parameters in the SMI_InitStruct.
Input Parameter	SMI_InitStruct: pointer to a SMI_InitTypeDef structure that contains the configuration information for the specified SMI peripheral. Refer to section "*SMI_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_InitTypeDef

The SMI_InitTypeDef structure is defined in the *75x_smi.h* file:

```
typedef struct
{
  u8 SMI_ClockHold;
  u8 SMI_Prescaler;
  u8 SMI_DeselectTime;
} SMI_InitTypeDef;
```

SMI_ClockHold

Specifies the SMI clock hold between each byte. This member must be a number between 0x00 and 0xFF.

SMI_Prescaler

Specifies the SMI clock prescaler. This member must be a number between 0x00 and 0x7F

SMI_DeselectTime

When Chip Select pin (CS) is deselected, it remains deselected for at least (*SMI_DeselectTime* + 1) SMI_CK periods. This member must be a number between 0x00 and 0xF.

Example:

```
/* SMI configuration */
  SMI_InitTypeDef SMI_InitStructure;

  SMI_InitStructure.SMI_ClockHold = 1;
  SMI_InitStructure.SMI_Prescaler = 2;
  SMI_InitStructure.SMI_DeselectTime = 1;
  SMI_Init(&SMI_InitStructure);
```

10.2.3 SMI_StructInit

Function Name	SMI_StructInit
Function Prototype	`void SMI_StructInit(SMI_InitTypeDef* SMI_InitStruct)`
Behavior Description	Fills each SMI_InitStruct member with its reset value, refer to the following table for more details.
Input Parameter	SMI_InitStruct: pointer to a SMI_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The SMI_InitStruct members have the following default values:

Member	Default value
SMI_ClockHold	0x00
SMI_Prescaler	0x02
SMI_DeselectTime	0x05

Example:

```
/* The following example illustrates how to initialize a SMI_InitTypeDef structure */
  SMI_InitTypeDef SMI_InitStructure;
  SMI_StructInit(&SMI_InitStructure);
```

10.2.4 SMI_ModeConfig

Function Name	SMI_ModeConfig
Function Prototype	`void SMI_ModeConfig(u32 SMI_Mode)`
Behavior Description	Selects the SMI mode: hardware or software.
Input Parameter	SMI_Mode: specifies the SMI mode. Refer to section "*SMI_ModeConfig*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_Mode

Specifies the SMI mode. This member can be one of the following values:

SMI_Mode	Meaning
SMI_Mode_HW	SMI used in Hardware mode
SMI_Mode_SW	SMI used in Software mode

Example:

```
/* SMI in hardware mode */
  SMI_ModeConfig(SMI_Mode_HW);
```

10.2.5 SMI_TxRxLengthConfig

Function Name	SMI_TxRxLengthConfig
Function Prototype	`void SMI_TxRxLengthConfig(u32 SMI_TxLength, u32 SMI_RxLength)`
Behavior Description	Configures the number of bytes to be transmitted and received to/from external memory. This function is used in Software mode only.
Input Parameter1	SMI_TxLength: specifies the number of bytes to be transmitted to external memory. Refer to section "*SMI_TxLength*" for more details on the allowed values of this parameter.
Input Parameter2	SMI_RxLength: specifies the number of bytes to be received from external memory. Refer to section "*SMI_RxLength*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_TxLength

Specifies the number of bytes to be transmitted to external memory . This member can be one of the following values:

SMI_TxLength	Meaning
SMI_TxLength_0Bytes	No bytes transmitted
SMI_TxLength_1Byte	1 byte transmitted
SMI_TxLength_2Bytes	2 bytes transmitted
SMI_TxLength_3Bytes	3 bytes transmitted
SMI_TxLength_4Bytes	4 bytes transmitted

SMI_RxLength

Specifies the number of bytes to be received from external memory. This member can be one of the following values:

SMI_RxLength	Meaning
SMI_RxLength_0Bytes	No bytes received
SMI_RxLength_1Byte	1 byte received
SMI_RxLength_2Bytes	2 bytes received
SMI_RxLength_3Bytes	3 bytes received
SMI_RxLength_4Bytes	4 bytes received

Example:

```
/* Transmit one byte and receive three bytes to/from external memory(in SW mode) */
  SMI_TxRxLengthConfig(SMI_TxLength_1Byte, SMI_RxLength_3Bytes);
```

10.2.6 SMI_BankCmd

Function Name	SMI_BankCmd
Function Prototype	`void SMI_BankCmd(u32 SMI_Bank, FunctionalState NewState)`
Behavior Description	Enables or disables the specified memory Bank.
Input Parameter1	SMI_Bank: specifies the memory Bank to be enabled or disabled. Refer to section "*SMI_Bank*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified memory Bank. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_Bank

To enable or disable the memory Banks, use a combination of one or more of the following values:

SMI_Bank	Meaning
SMI_Bank_0	Memory Bank0 will be enabled/disabled
SMI_Bank_1	Memory Bank1 will be enabled/disabled
SMI_Bank_2	Memory Bank2 will be enabled/disabled
SMI_Bank_3	Memory Bank3 will be enabled/disabled

Example:

```
/* Enable Bank0 and Bank1 */
  SMI_BankCmd(SMI_Bank_0 | SMI_Bank_1, ENABLE);
```

10.2.7 SMI_ITConfig

Function Name	SMI_ITConfig
Function Prototype	`void SMI_ITConfig(u32 SMI_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SMI interrupts.
Input Parameter1	SMI_IT: specifies the SMI interrupts sources to be enabled or disabled. Refer to section "*SMI_IT*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified SMI interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_IT

To enable or disable SMI interrupts, use a combination of one or more of the following values:

SMI_IT	Meaning
SMI_IT_WC	Write Complete Interrupt mask
SMI_IT_TF	Transfer Finished Interrupt mask

Example:

```
/* Enable Write Complete and Transfer Finished interrupt */
  SMI_ITConfig(SMI_IT_WC | SMI_IT_TF, ENABLE);
```

10.2.8 SMI_SelectBank

Function Name	SMI_SelectBank
Function Prototype	`void SMI_SelectBank(u32 SMI_Bank)`
Behavior Description	Selects the memory Bank to be accessed. Only one Bank can be selected at a time.
Input Parameter1	SMI_Bank: specifies the memory Bank to be selected. Refer to section "*SMI_Bank on page 121*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Select Bank0 */
  SMI_SelectBank(SMI_Bank_0);
```

10.2.9 SMI_SendWENCmd

Function Name	SMI_SendWENCmd
Function Prototype	`void SMI_SendWENCmd(void)`
Behavior Description	Sends a Write Enable command to the selected memory Bank. This function is used in Hardware mode only.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, select the memory Bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Note: After using this function you have to wait until the Transfer Finished Flag (TFF) is set then clear it.

Example:

```
/* Send Write Enable command to the selected memory */
  SMI_SendWENCmd();
```

10.2.10 SMI_SendRSRCmd

Function Name	SMI_SendRSRCmd
Function Prototype	`void SMI_SendRSRCmd(void)`
Behavior Description	Sends a Read Status Register command to the selected memory Bank.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, select the memory Bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Note: After using this function you have to wait until the Transfer Finished Flag (TFF) is set then clear it.

Example:

```
/* Send Read Status Register command to the selected memory */
  SMI_SendRSRCmd();
```

10.2.11 SMI_SendCmd

Function Name	SMI_SendCmd
Function Prototype	`void SMI_SendCmd(u32 Command)`
Behavior Description	Sends command to the selected memory Bank. This function is used in Software mode only.
Input Parameter	Command: specifies the command to send to the external memory.
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, select the memory Bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Note: After using this function you have to wait until the Transfer Finished Flag (TFF) is set then clear it.

Example:

```
/* Send Write Enable command to the selected memory */
  SMI_SendCmd(0x06);
```

10.2.12 SMI_FastReadConfig

Function Name	SMI_FastReadConfig
Function Prototype	`void SMI_FastReadConfig(u32 SMI_FastRead)`
Behavior Description	Enables or disables the Fast Read Mode.
Input Parameter	SMI_FastRead: specifies whether the Fast Read Mode is enabled or disabled. Refer to section "*SMI_FastRead*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_FastRead

To enable or disable the Fast Read Mode, use one of the following values:

SMI_FastRead	Meaning
SMI_FastRead_Disable	Disable Fast Read Mode.
SMI_FastRead_Enable	Enable Fast Read Mode.

Example:

```
/* Enable Fast Read mode */
  SMI_FastReadConfig(SMI_FastRead_Enable);
```

10.2.13 SMI_WriteBurstConfig

Function Name	SMI_WriteBurstConfig
Function Prototype	`void SMI_WriteBurstConfig(u32 SMI_WriteBurst)`
Behavior Description	Enables or disables the Write Burst Mode.
Input Parameter	SMI_WriteBurst: specifies whether the Write Burst Mode is enabled or disabled. Refer to section "*SMI_WriteBurst*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_WriteBurst

To enable or disable the Write Burst Mode, use one of the following values:

SMI_WriteBurst	Meaning
SMI_WriteBurst_Disable	Disable Write Burst Mode.
SMI_WriteBurst_Enable	Enable Write Burst Mode.

Example:

```
/* Enable Write Burst mode */
  SMI_WriteBurstConfig(SMI_WriteBurst_Enable);
```

10.2.14 SMI_WriteByte

Function Name	SMI_WriteByte
Function Prototype	`void SMI_WriteByte(u32 WriteAddr, u8 Data)`
Behavior Description	Writes a Byte to the selected memory Bank. This function is used in Hardware mode only.
Input Parameter1	WriteAddr: external memory address from which the data will be written.
Input Parameter2	Data: data to be written to the external memory.
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, send a Write Enable command to the selected memory Bank using *SMI_SendWENCmd()* function.
Called functions	None

Note: After using this function you have to wait until the Write Complete Flag (WCF) is set then clear it.

Example:

```
/* Write 0x5A value to memory connected to Bank0 at address 0x00 */
  SMI_WriteByte(0x80000000, 0x5A);
```

10.2.15 SMI_WriteHalfWord

Function Name	SMI_WriteHalfWord
Function Prototype	`void SMI_WriteHalfWord(u32 WriteAddr, u16 Data)`
Behavior Description	Writes a Half Word to the selected memory Bank. This function is used in Hardware mode only.
Input Parameter1	WriteAddr: external memory address from which the data will be written.
Input Parameter2	Data: data to be written to the external memory.
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, send a Write Enable command to the selected memory Bank using *SMI_SendWENCmd()* function.
Called functions	None

Note: After using this function you have to wait until the Write Complete Flag (WCF) is set and then clear it.

Example:

```
/* Write 0x6978 value to memory connected to Bank0 at address 0x02 */
  SMI_WriteHalfWord(0x80000002, 0x6978);
```

10.2.16 SMI_WriteWord

Function Name	SMI_WriteWord
Function Prototype	`void SMI_WriteWord(u32 WriteAddr, u32 Data)`
Behavior Description	Writes a Word to the selected memory Bank. This function is used in Hardware mode only.
Input Parameter1	WriteAddr: external memory address from which the data will be written.
Input Parameter2	Data: data to be written to the external memory.
Output Parameter	None
Return Parameter	None
Required preconditions	Before calling this function, send a Write Enable command to the selected memory Bank using *SMI_SendWENCmd()* function.
Called functions	None

Note: After using this function you have to wait until the Write Complete Flag (WCF) is set and then clear it.

Example:

```
/* Write 0x534D495F value to memory connected to Bank0 at address 0x04 */
  SMI_WriteWord(0x80000004, 0x534D495F);
```

10.2.17 SMI_ReadByte

Function Name	SMI_ReadByte
Function Prototype	`u8 SMI_ReadByte(u32 ReadAddr)`
Behavior Description	Reads a byte from the selected memory bank. This function is used in Hardware mode only.
Input Parameter	ReadAddr: external memory address to read from.
Output Parameter	None
Return Parameter	Data read from the external memory.
Required preconditions	Before calling this function, select the memory bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Example:

```
/* Read a 8 bits value starting from address 0x00 from memory connected to Bank0 */
  u8 bRxData;
  bRxData = SMI_ReadByte(0x80000000);
```

10.2.18 SMI_ReadHalfWord

Function Name	SMI_ReadHalfWord
Function Prototype	`u16 SMI_ReadHalfWord(u32 ReadAddr)`
Behavior Description	Reads a half word from the selected memory bank. This function is used in Hardware mode only.
Input Parameter	ReadAddr: external memory address to read from.
Output Parameter	None
Return Parameter	Data read from the external memory.
Required preconditions	Before calling this function, select the memory bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Example:

```
/* Read a 16 bits value starting from address 0x02 from memory connected to Bank0 */
  u16 hRxData;
  hRxData = SMI_ReadHalfWord(0x80000002);
```

10.2.19 SMI_ReadWord

Function Name	SMI_ReadWord
Function Prototype	`u32 SMI_ReadWord(u32 ReadAddr)`
Behavior Description	Reads a Word from the selected memory bank. This function is used in Hardware mode only.
Input Parameter	ReadAddr: external memory address to read from.
Output Parameter	None
Return Parameter	Data read from the external memory.
Required preconditions	Before calling this function, select the memory bank to be accessed using the *SMI_SelectBank()* function.
Called functions	None

Example:

```
/* Read a 32 bits value starting from address 0x04 from memory connected to Bank0 */
  u32 wRxData;
  wRxData = SMI_ReadWord(0x80000004);
```

10.2.20 SMI_ReadMemoryStatusRegister

Function Name	SMI_ReadMemoryStatusRegister
Function Prototype	`u8 SMI_ReadMemoryStatusRegister(void)`
Behavior Description	Reads the status register of the memory connected to the selected bank.
Input Parameter	None
Output Parameter	None
Return Parameter	External memory status register value.
Required preconditions	None
Called functions	None

Example:

```
/* Read the status register of the memory connected to the selected Bank */
  u8 bMemoryStatus;
  bMemoryStatus = SMI_ReadMemoryStatusRegister();
```

10.2.21 SMI_GetFlagStatus

Function Name	SMI_GetFlagStatus
Function Prototype	`FlagStatus SMI_GetFlagStatus(u32 SMI_FLAG)`
Behavior Description	Checks whether the specified SMI flag is set or not.
Input Parameter	SMI_FLAG: specifies the flag to check. Refer to section "*SMI_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SMI_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SMI_FLAG

The SMI flags that can check for are listed in the following table:

SMI_FLAG	Meaning
SMI_FLAG_Bank3_WM	Memory Bank3 Write Mode flag
SMI_FLAG_Bank2_WM	Memory Bank2 Write Mode flag
SMI_FLAG_Bank1_WM	Memory Bank1 Write Mode flag
SMI_FLAG_Bank0_WM	Memory Bank0 Write Mode flag
SMI_FLAG_ERF2	Error Flag 2: Forbidden Write Request
SMI_FLAG_ERF1	Error Flag 1: Forbidden Access
SMI_FLAG_WC	Write Complete flag
SMI_FLAG_TF	Transfer Finished flag

Example:

```
/* Get Write Complete flag status */
  FlagStatus Status;
  Status = SMI_GetFlagStatus(SMI_FLAG_WC);
```

10.2.22 SMI_ClearFlag

Function Name	SMI_ClearFlag
Function Prototype	`void SMI_ClearFlag(u32 SMI_FLAG)`
Behavior Description	Clears the SMI's pending flags.
Input Parameter	SMI_FLAG: specifies the flag to clear. Refer to section "*SMI_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SMI_FLAG

To clear SMI flags, use a combination of one or more of the following values:

SMI_FLAG	Meaning
SMI_FLAG_ERF2	Error Flag 2: Forbidden Write Request
SMI_FLAG_ERF1	Error Flag 1: Forbidden Access
SMI_FLAG_WC	Write Complete flag
SMI_FLAG_TF	Transfer Finished flag

Example:

```
/* Clear Write Complete flag */
  SMI_ClearFlag(SMI_FLAG_WC);
```

10.2.23 SMI_GetITStatus

Function Name	SMI_GetITStatus
Function Prototype	`ITStatus SMI_GetITStatus(u32 SMI_IT)`
Behavior Description	Checks whether the specified SMI interrupt has occurred or not.
Input Parameter	SMI_IT: specifies the SMI interrupt source to check. Refer to section "*SMI_IT on page 122*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SMI_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the status of Write Complete interrupt */
  ITStatus Status;
  Status = SMI_GetITStatus(SMI_IT_WC);
```

10.2.24 SMI_ClearITPendingBit

Function Name	SMI_ClearITPendingBit
Function Prototype	`void SMI_ClearITPendingBit(u32 SMI_IT)`
Behavior Description	Clears the SMI's interrupt pending bits.
Input Parameter	SMI_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to section "*SMI_IT on page 122*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear Write Complete interrupt pending bit */
  SMI_ClearITPendingBit(SMI_IT_WC);
```

11 Real Time Clock (RTC)

The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

The first section describes the register structure used in the RTC software library. The second one presents the software library functions.

11.1 RTC register structure

The RTC register structure *RTC_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 CRH;
  u16 EMPTY;
  vu16 CRL;
  u16 EMPTY1;
  vu16 PRLH;
  u16 EMPTY2;
  vu16 PRLL;
  u16 EMPTY3;
  vu16 DIVH;
  u16 EMPTY4;
  vu16 DIVL;
  u16 EMPTY5;
  vu16 CNTH;
  u16 EMPTY6;
  vu16 CNTL;
  u16 EMPTY7;
  vu16 ALRH;
  u16 EMPTY8;
  vu16 ALRL;
  u16 EMPTY9;
} RTC_TypeDef;
```

The following table presents the RTC registers:

Register	Description
CRH	Control Register High
CRL	Control Register Low
PRLH	Prescaler Load Register High
PRLL	Prescaler Load Register Low
DIVH	Divider Register High
DIVL	Divider Register Low
CNTH	Counter Register High
CNTL	Counter Register Low
ALRH	Alarm Register High
ALRL	Alarm Register Low

The RTC peripheral is declared in the same file:

```
...
#define PERIPH_BASE   0xFFFF 0000
...
#define RTC_BASE      (PERIPH_BASE + 0xF000)

#ifndef DEBUG
...
#define RTC           ((RTC_TypeDef *) RTC_BASE)
...
#else
...
#ifdef _RTC
  EXT RTC_TypeDef           *RTC;
#endif /*_RTC */
...
#endif
```

When debug mode is used, RTC pointer is initialized in 75x_lib.c file:

```
#ifdef _RTC
  RTC = (RTC_TypeDef *)RTC_BASE;
#endif /*_RTC */
```

_RTC, must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _RTC
```

11.2 Software library functions

The following table lists the various functions of the RTC library.

Function Name	Description
RTC_DeInit	Deinitializes the RTC peripheral registers to their default reset values.
RTC_ITConfig	Enables or disables the specified RTC interrupts.
RTC_EnterConfigMode	Enters the RTC configuration mode.
RTC_ExitConfigMode	Exits from the RTC configuration mode.
RTC_GetCounter	Gets the RTC counter value.
RTC_SetCounter	Sets the RTC counter value.
RTC_SetPrescaler	Sets the RTC prescaler value.
RTC_GetPrescaler	Gets the RTC Prescaler value.
RTC_SetAlarm	Sets the RTC Alarm value.
RTC_GetDivider	Gets the RTC Divider value.
RTC_WaitForLastTask	Waits until last write operation on RTC registers has finished.
RTC_WaitForSynchro	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock.
RTC_GetFlagStatus	Checks whether the specified RTC flag is set or not.
RTC_ClearFlag	Clears the RTC's pending flags.
RTC_GetITStatus	Checks whether the specified RTC interrupt has occurred or not.
RTC_ClearITPendingBit	Clears the RTC's interrupt pending bits.

11.2.1 RTC_DeInit

Function Name	RTC_DeInit
Function Prototype	`void RTC_DeInit(void)`
Behavior Description	Deinitializes the RTC peripheral registers to their default reset values.
Output Parameter	None
Return Parameter	None
Required preconditions	After using this function, you must call RTC_WaitForSynchro() function (wait until RSF flag is set).
Called functions	MRCC_PeripheralSWResetConfig()

Example:

```
/* Deinitializes RTC */
RTC_DeInit();

/* Wait until the RTC registers are synchronized with RTC APB clock */
RTC_WaitForSynchro();
```

11.2.2 RTC_ITConfig

Function Name	RTC_ITConfig
Function Prototype	`void RTC_ITConfig(u16 RTC_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified RTC interrupts.
Input Parameter1	RTC_IT: specifies the RTC interrupts sources to be enabled or disabled. Refer to section "*RTC_IT*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified RTC interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

RTC_IT

To enable or disable RTC interrupts, use a combination of one or more of the following values:

RTC_IT	Meaning
RTC_IT_Overflow	Overflow interrupt enabled
RTC_IT_Alarm	Alarm interrupt enabled
RTC_IT_Second	Second interrupt enabled

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Alarm interrupt enabled */
RTC_ITConfig(RTC_IT_Alarm, ENABLE);
```

11.2.3 RTC_EnterConfigMode

Function Name	RTC_EnterConfigMode
Function Prototype	`void RTC_EnterConfigMode(void)`
Behavior Description	Enters the RTC configuration mode.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the configuration mode */
RTC_EnterConfigMode();
```

11.2.4 RTC_ExitConfigMode

Function Name	RTC_ExitConfigMode
Function Prototype	`void RTC_ExitConfigMode(void)`
Behavior Description	Exits from the RTC configuration mode.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Exit the configuration mode */
RTC_ExitConfigMode();
```

11.2.5 RTC_GetCounter

Function Name	RTC_GetCounter
Function Prototype	`u32 RTC_GetCounter(void)`
Behavior Description	Gets the RTC counter value.
Output Parameter	None
Return Parameter	RTC counter value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the counter value */
u32 RTCCounterValue;
RTCCounterValue = RTC_GetCounter();
```

11.2.6 RTC_SetCounter

Function Name	RTC_SetCounter
Function Prototype	`void RTC_SetCounter(u32 CounterValue)`
Behavior Description	Sets the RTC counter value.
Input Parameter	CounterValue: RTC counter new value.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Counter value to 0xFFFF5555 */
RTC_SetCounter(0xFFFF5555);
```

11.2.7 RTC_SetPrescaler

Function Name	RTC_SetPrescaler
Function Prototype	`void RTC_SetPrescaler(u32 PrescalerValue)`
Behavior Description	Sets the RTC prescaler value.
Input Parameter	PrescalerValue: RTC prescaler new value.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Prescaler value to 0x7A12 */
RTC_SetPrescaler(0x7A12);
```

11.2.8 RTC_GetPrescaler

Function Name	RTC_GetPrescaler
Function Prototype	`u32 RTC_GetPrescaler(void)`
Behavior Description	Gets the RTC prescaler value.
Output Parameter	None
Return Parameter	RTC prescaler value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the current RTC Prescaler value */
u32 RTCPrescalerValue;
RTCPrescalerValue = RTC_GetPrescaler();
```

11.2.9 RTC_SetAlarm

Function Name	RTC_SetAlarm
Function Prototype	`void RTC_SetAlarm(u32 AlarmValue)`
Behavior Description	Sets the RTC alarm value.
Input Parameter	AlarmValue: RTC alarm new value.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call *RTC_WaitForLastTask()* function (wait until RTOFF flag is set).
Called functions	RTC_EnterConfigMode() RTC_ExitConfigMode()

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Alarm value to 0xFFFFFFFA */
RTC_SetAlarm(0xFFFFFFFA);
```

11.2.10 RTC_GetDivider

Function Name	RTC_GetDivider
Function Prototype	`u32 RTC_GetDivider(void)`
Behavior Description	Gets the RTC Divider value.
Output Parameter	None
Return Parameter	RTC divider value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the current RTC Divider value */
u32 RTCDividerValue;
RTCDividerValue = RTC_GetDivider();
```

11.2.11 RTC_WaitForLastTask

Function Name	RTC_WaitForLastTask
Function Prototype	`void RTC_WaitForLastTask(void)`
Behavior Description	Waits until last write operation on RTC registers has finished. This function must be called before any write to RTC registers.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Sets Alarm value to 0x10 */
RTC_SetAlarm(0x10);
```

11.2.12 RTC_WaitForSynchro

Function Name	RTC_WaitForSynchro
Function Prototype	`void RTC_WaitForSynchro(void)`
Behavior Description	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock. This function must be called before any read operation after an APB reset or an APB clock stop.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
u32 RTCPrescalerValue;

/* Deinitializes RTC */
RTC_DeInit();

/* Wait until the RTC registers are synchronized with RTC APB clock */
RTC_WaitForSynchro();

/* Gets the current RTC Prescaler value */
RTCPrescalerValue = RTC_GetPrescaler();
```

11.2.13 RTC_GetFlagStatus

Function Name	RTC_GetFlagStatus
Function Prototype	`FlagStatus RTC_GetFlagStatus(u16 RTC_FLAG)`
Behavior Description	Checks whether the specified RTC flag is set or not.
Input Parameter	RTC_FLAG: specifies the flag to check. Refer to section "*RTC_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of RTC_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

RTC_FLAG

The RTC flags that can check for are listed in the following table:

RTC_FLAG	Meaning
RTC_FLAG_Overflow	Overflow interrupt Flag
RTC_FLAG_Alarm	Alarm interrupt Flag
RTC_FLAG_Second	Second interrupt Flag

RTC_FLAG	Meaning
RTC_FLAG_RSF	Registers Synchronized Flag
RTC_FLAG_RTOFF	RTC operation OFF Flag

Example:

```
/* Gets the RTC overflow interrupt status */
FlagStatus OverrunFlagStatus;
OverrunFlagStatus = RTC_GetFlagStatus(RTC_Flag_Overflow);
```

11.2.14 RTC_ClearFlag

Function Name	RTC_ClearFlag
Function Prototype	`void RTC_ClearFlag(u16 RTC_FLAG)`
Behavior Description	Clears the RTC's pending flags.
Input Parameter	RTC_FLAG: specifies the flag to clear. Refer to section "*RTC_FLAG on page 141*" for more details on the allowed values of this parameter. The RTC_FLAG_RTOFF can not be cleared by software and the RTC_FLAG_RSF is cleared only after an APB reset or an APB clock stop.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Clears the RTC overflow flag */
RTC_ClearFlag(RTC_FLAG_Overflow);
```

11.2.15 RTC_GetITStatus

Function Name	RTC_GetITStatus
Function Prototype	`ITStatus RTC_GetITStatus(u16 RTC_IT)`
Behavior Description	Checks whether the specified RTC interrupt has occurred or not.
Input Parameter	RTC_IT: specifies the RTC interrupt source to check. Refer to section "*RTC_IT on page 136*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of the RTC_IT(SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Gets the RTC Second interrupt status */
ITStatus SecondITStatus;
SecondITStatus = RTC_GetITStatus(RTC_IT_Second);
```

11.2.16 RTC_ClearITPendingBit

Function Name	RTC_ClearITPendingBit
Function Prototype	`void RTC_ClearITPendingBit(u16 RTC_IT)`
Behavior Description	Clears the RTC's interrupt pending bits.
Input Parameter	RTC_IT: specifies the interrupt pending bit to clear. Refer to section "*RTC_IT on page 136*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	Before using this function, you must call RTC_WaitForLastTask() function (wait until RTOFF flag is set).
Called functions	None

Example:

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();

/* Clears the RTC Second interrupt */
RTC_ClearITPendingBit(RTC_IT_Second);
```

12 Watchdog Timer (WDG)

The Watchdog Timer peripheral can be used as free-running timer or as Watchdog to resolve processor malfunctions due to hardware or software failures.

The first section describes the register structure used in the WDG software library. The second one presents the software library functions.

12.1 WDG register structure

The WDG register structure *WDG_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
 vu16 CR;
 u16 EMPTY1;
 u16 PR;
 uv16 EMPTY2;
 vu16 VR;
 u16 EMPTY3;
 vu16 CNT;
 u16 EMPTY4;
 vu16 SR;
 u16 EMPTY5;
 vu16 MR;
 u16 EMPTY6;
 vu16 KR;
 u16 EMPTY7;
} WDG_TypeDef;
```

The following table presents the WDG registers:

Register	Description
CR	Control Register
PR	Prescaler Register
VR	Preload Value Register
CNT	Counter Register
SR	Status Register
MR	Mask Register
KR	Key Register

The WDG peripheral is declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define WDG_BASE        (PERIPH_BASE + 0xB000)

#ifndef DEBUG
...
#define WDG   ((WDG_TypeDef *)  WDG_BASE)
...
#else
...
#ifdef _WDG
  EXT WDG_TypeDef           *WDG;
#endif /*_WDG */
...
#endif
```

When debug mode is used, WDG pointer is initialized in 75x_lib.c file:

```
#ifdef _WDG
  WDG = (WDG_TypeDef *)  WDG_BASE;
#endif /*_WDG */
```

_WDG must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _WDG
```

12.2 Software library functions

The following table lists the various functions of the WDG library.

Function Name	Description
WDG_DeInit	Deinitializes the WDG peripheral registers to their default reset values.
WDG_Init	Initializes the WDG peripheral according to the specified parameters in the WDG_InitStruct.
WDG_StructInit	Fills each WDG_InitStruct member with its default value.
WDG_Cmd	Enables or disables the WDG peripheral.
WDG_ITConfig	Enables or disables the WDG End of Count(EC) interrupt.
WDG_GetCounter	Gets the WDG's current counter value.
WDG_GetFlagStatus	Checks whether the WDG End of Count(EC) flag is set or not.
WDG_ClearFlag	Clears the WDG's End of Count(EC) pending flag.
WDG_GetITStatus	Checks whether the WDG End of Count(EC) interrupt has occurred or not.
WDG_ClearITPendingBit	Clears the WDG's End of Count(EC) interrupt pending bit.

12.2.1 WDG_DeInit

Function Name	WDG_DeInit
Function Prototype	`void WDG_DeInit(void)`
Behavior Description	Deinitializes the WDG peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Deinitializes the WDG registers to their default reset value */
WDG_DeInit();
```

12.2.2 WDG_Init

Function Name	WDG_Init
Function Prototype	`void WDG_Init( WDG_InitTypeDef* WDG_InitStruct)`
Behavior Description	Initializes the WDG peripheral according to the specified parameters in the WDG_InitStruct .
Input Parameter	WDG_InitStruct: pointer to a WDG_InitTypeDef structure that contains the configuration information for the WDG peripheral. Refer to section "*WDG_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

WDG_InitTypeDef

The WDG_InitTypeDef structure is defined in the *75x_wdg.h* file:

```
typedef struct
{
  u16 WDG_Mode;
  u16 WDG_Preload;
  u8 WDG_Prescaler;
} WDG_InitTypeDef;
```

WDG_Mode

Specifies wether the WDG is used in Timer or Watchdog mode. This member can be one of the following values:

WDG_Mode	Meaning
WDG_Mode_WDG	Watchdog Timer is configured in Watchdog mode
WDG_Mode_Timer	Watchdog Timer is configured in Timer mode

WDG_Preload

This value is loaded in the WDG Counter when it starts counting. The time (µs) needed to reach the end of count is given by:

$$\frac{\text{Prescaler} \times \text{Preload} \times \text{Tclk}}{1000}$$

where Tclk is the Clock period measured in ns.

This member must be a number between 0x0000 and 0xFFFF.

WDG_Prescaler

Specifies the Prescaler value to divide the clock source. The clock of the Watchdog Timer Counter is divided by PR[7:0] + 1.

This member must be a number between 0x00 and 0xFF.

Example:

```
/* The following example illustrates how to configure the WDG */
  WDG_InitTypeDef WDG_InitStructure;

  WDG_InitStructure.WDG_Mode = WDG_Mode_Timer
  WDG_InitStructure.WDG_Prescaler = 0xF0;
  WDG_InitStructure.WDG_Preload = 0xFF00;
  WDG_Init(&WDG_InitStructure);
```

12.2.3 WDG_StructInit

Function Name	WDG_StructInit
Function Prototype	`void WDG_StructInit(WDG_InitTypeDef* WDG_InitStruct)`
Behavior Description	Fills each WDG_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	WDG_InitStruct: pointer to a WDG_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The WDG_InitStruct members have the following default values:

Member	Default value
WDG_Mode	WDG_Mode_Timer
WDG_Preload	0xFFFF
WDG_Prescaler	0xFF

Example:

```
/* The following example illustrates how to initialize a WDG_InitTypeDef structure */
  WDG_InitTypeDef WDG_InitStructure;
  WDG_StructInit(&WDG_InitStructure);
```

12.2.4 WDG_Cmd

Function Name	WDG_Cmd
Function Prototype	`void WDG_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the WDG peripheral.
Input Parameter	NewState: new state of the WDG peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the WDG */
  WDG_Cmd(ENABLE);
```

12.2.5 WDG_ITConfig

Function Name	WDG_ITConfig
Function Prototype	`void WDG_ITConfig(FunctionalState NewState)`
Behavior Description	Enables or disables the WDG End of Count(EC) interrupt.
Input Parameter	NewState: new state of the WDG End of Count(EC) interrupt. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the WDG end of count Interrupt */
  WDG_ITConfig(ENABLE);
```

12.2.6 WDG_GetCounter

Function Name	WDG_GetCounter
Function Prototype	`u16 WDG_GetCounter(void)`
Behavior Description	Gets the WDG's current counter value.
Output Parameter	None
Return Parameter	WDG current counter value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the WDG counter value */
  u16 CounterValue = WDG_GetCounter();
```

12.2.7 WDG_GetFlagStatus

Function Name	WDG_GetFlagStatus
Function Prototype	`FlagStatus WDG_GetFlagStatus(void)`
Behavior Description	Checks whether the WDG End of Count(EC) flag is set or not.
Output Parameter	None
Return Parameter	The new state of WDG End of Count(EC) flag (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Check if the WDG end of count flag is set or reset */
  if(WDG_GetFlagStatus() == SET)
  {
  }
```

12.2.8 WDG_ClearFlag

Function Name	WDG_ClearFlag
Function Prototype	`void WDG_ClearFlag(void)`
Behavior Description	Clears the WDG's End of Count(EC) pending flag.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the WDG end of count flag */
  WDG_ClearFlag();
```

12.2.9 WDG_GetITStatus

Function Name	WDG_GetITStatus
Function Prototype	`ITStatus WDG_GetITStatus(void)`
Behavior Description	Checks whether the WDG End of Count(EC) interrupt has occurred or not.
Output Parameter	None
Return Parameter	The new state of WDG End of Count(EC) interrupt (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Check if the WDG end of count interrupt has occurred or reset */
  if(WDG_GetITStatus() == SET)
  {
  }
```

12.2.10 WDG_ClearITPendingBit

Function Name	WDG_ClearITPendingBit
Function Prototype	`void WDG_ClearITPendingBit(void)`
Behavior Description	Clears the WDG's End of Count(EC) interrupt pending bit.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the WDG end of count interrupt pending bit */
  WDG_ClearITPendingBit();
```

13 Timebase Timer (TB)

The Timer Base peripheral can be used as free-running timer to generate a time base.

The first section describes the register structure used in the TB software library. The second one presents the software library functions.

13.1 TB register structure

The TB register structure *TB_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 CR;
  u16  EMPTY1;
  vu16 SCR;
  u16  EMPTY2;
  vu16 IMCR;
  u16  EMPTY3[7];
  vu16 RSR;
  u16  EMPTY4;
  vu16 RER;
  u16  EMPTY5;
  vu16 ISR;
  u16  EMPTY6;
  vu16 CNT;
  u16  EMPTY7;
  vu16 PSC;
  u16  EMPTY8[3];
  vu16 ARR;
  u16  EMPTY9[13];
  vu16 ICR1;
  u16  EMPTY10;
} TB_TypeDef;
```

The following table presents the TB registers:

Register	Description
CR	Control Register
SCR	Synchro Control Register
IMCR	Input Mode Control Register
RSR	Request Selection Register
RER	Request Enable Register
ISR	Interrupt Status Register
CNT	Counter Register
PSC	Prescaler Register
ARR	Auto-Reload Register
ICR1	Input Capture Register 1

The TB peripheral is declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define TB_BASE         (PERIPH_BASE + 0x8800)

#ifndef DEBUG
...
#define TB    ((TB_TypeDef *)   TB_BASE)
...
#else
...
#ifdef _TB
  EXT TB_TypeDef            *TB;
#endif /*_TB */
...
#endif
```

When debug mode is used, TB pointer is initialized in 75x_lib.c file:

```
#ifdef _TB
  TB = (TB_TypeDef *)TB_BASE;
#endif /*_TB */
```

_TB must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _TB
```

13.2 Software library functions

The following table lists the various functions of the TB library.

Function Name	Description
TB_DeInit	Deinitializes the TB peripheral registers to their default reset values.
TB_Init	Initializes the TB peripheral according to the specified parameters in the TB_InitStruct.
TB_StructInit	Fills each TB_InitStruct member with its default value.
TB_Cmd	Enables or disables the TB peripheral.
TB_ITConfig	Enables or disables the specified TB interrupts.
TB_SetPrescaler	Sets the TB Prescaler value.
TB_ResetCounter	Re-initializes the counter and generates an update of the registers.
TB_DebugCmd	Enables or disables the TB peripheral Debug control.
TB_CounterModeConfig	Configures the TB Counter Mode.
TB_SLaveModeConfig	Configures the TB slave Mode.
TB_GetCounter	Gets the TB counter value.
TB_GetICAP1	Gets the TB Input capture value.
TB_SetCounter	Sets the TB counter value.
TB_GetFlagStatus	Checks whether the specified TB flag is set or not.
TB_ClearFlag	Clears the TB's pending flags.
TB_GetITStatus	Checks whether the specified TB interrupt has occurred or not.
TB_ClearITPendingBit	Clears the TB's interrupt pending bits.

13.2.1 TB_DeInit

Function Name	TB_DeInit
Function Prototype	`void TB_DeInit()`
Behavior Description	Deinitializes the TB peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitializes the TB registers to their default reset value */
TB_DeInit();
```

13.2.2 TB_Init

Function Name	TB_Init
Function Prototype	`void TB_Init(TB_InitTypeDef* TB_InitStruct)`
Behavior Description	Initializes the TB peripheral according to the specified parameters in the TB_InitStruct.
Input Parameter	TB_InitStruct: pointer to a TB_InitTypeDef structure that contains the configuration information for the TB peripheral. Refer to section "*TB_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TB_InitTypeDef

The TB_InitTypeDef structure is defined in the *75x_tb.h* file:

```
typedef struct
{
  u16 TB_Mode;
  u16 TB_ClockSource;
  u16 TB_CounterMode;
  u16 TB_ICAPolarity;
  u16 TB_Prescaler;
  u16 TB_AutoReload;
} TB_InitTypeDef;
```

TB_Mode

The Timer Base peripheral can be used as free-running timer to generate a time base or in Input Capture mode to measure the RTC peripheral clock. This member can be one of the following values:

TB_CounterMode	Meaning
TB_Mode_Timing	TB free-running Mode.
TB_Mode_IC	TB Input Capture Mode.

TB_ClockSource

Specifies the TB counter clock source. This member can be one of the following values:

TB_ClockSource	Meaning
TB_ClockSource_CKTIM	CK_TIM used as clock source for TB counter.
TB_ClockSource_CKRTC	CK_RTC used as clock source for TB counter.

TB_CounterMode

Specifies the counter mode. This member can be one of the following values:

TB_CounterMode	Meaning
TB_CounterMode_Up	TB Up Counting Mode.
TB_CounterMode_Down	TB Down Counting Mode.
TB_CounterMode_CenterAligned	TB Center Aligned Mode.

TB_ICAPolarity

Specifies the Input Capture signal polarity. This member can be one of the following values:

TB_ICAPolarity	Meaning
TB_ICAPolarity_Falling	Input Capture falling edge
TB_ICAPolarity_Rising	Input Capture rising edge

TB_Prescaler

Specifies the Prescaler value used to divide the TB clock. The clock of the Timer Base Counter is divided by PSC[15:0] + 1.

This member must be a number between 0x0000 and 0xFFFF.

TB_AutoReload

Specifies the Preload value to be loaded in the active Auto-Reload Register at the next update event. It contains the TB_ARR value to be compared to the counter CNT Register.

This member must be a number between 0x0000 and 0xFFFF.

Example:

```
/* The following example illustrates how to configure the TB peripheral */
  TB_InitTypeDef TB_InitStructure;

  TB_InitStructure.TB_Mode = TB_Mode_Timing;
  TB_InitStructure.TB_ClockSource = TB_ClockSource_CKTIM;
  TB_InitStructure.TB_CounterMode = TB_CounterMode_UP;
  TB_InitStructure.TB_Prescaler = 0xF000;
  TB_InitStructure.TB_Autoreload = 0xFF00;
  TB_Init(&TB_InitStructure);
```

13.2.3 TB_StructInit

Function Name	TB_StructInit
Function Prototype	`void TB_StructInit(TB_InitTypeDef* TB_InitStruct)`
Behavior Description	Fills each TB_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	TB_InitStruct: pointer to a TB_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The TB_InitStruct members have the following default values:

Member	Default value
TB_Mode	TB_Mode_Timing
TB_ClockSource	TB_ClockSource_CKTIM
TB_CounterMode	TB_CounterMode_Up
TB_ICAPolarity	TB_ICAPolarity
TB_Prescaler	TB_Prescaler_Reset_Mask
TB_AutoReload	TB_Autoreload_Reset_Mask

Example:

```
/* The following example illustrates how to initialize a TB_InitTypeDef structure */
  TB_InitTypeDef TB_InitStructure;
  TB_StructInit(&TB_InitStructure);
```

13.2.4 TB_Cmd

Function Name	TB_Cmd
Function Prototype	`void TB_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the TB peripheral.
Input Parameter	NewState: new state of the TB peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TB peripheral */
  TB_Cmd(ENABLE);
```

13.2.5 TB_ITConfig

Function Name	TB_ITConfig
Function Prototype	`void TB_ITConfig(u16 TB_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified TB interrupts.
Input Parameter1	TB_IT: specifies the TB interrupts sources to be enabled or disabled. Refer to section "*TB_IT*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the specified TB interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TB_IT

To enable or disable TB interrupts, use a combination of one or more of the following values:

TB_IT	Meaning
TB_IT_Update[1)]	TB Update interrupt source.
TB_IT_GlobalUpdate[1)]	TB Global Update interrupt source.
TB_IT_IC	TB Input Capture interrupt source.

[1)] TB_IT_Update and TB_GlobalUpdate interrupt sources must not be configured simultaneously.

Example:

```
/* Enable the TB Update Interrupt */
  TB_ITConfig(TB_IT_Update, ENABLE);
```

13.2.6 TB_SetPrescaler

Function Name	TB_SetPrescaler
Function Prototype	`void TB_SetPrescaler(u16 Prescaler)`
Behavior Description	Sets the TB Prescaler value.
Input Parameter	Prescaler: specifies the TB Prescaler value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the TB new Prescaler value */
  TB_SetPrescaler(0xFF3F);
```

13.2.7 TB_ResetCounter

Function Name	TB_ResetCounter
Function Prototype	`void TB_ResetCounter(void)`
Behavior Description	Re-intializes the counter and generates an update of the registers. The Prescaler counter is cleared too, however the Prescaler Preload value is not affected. The counter is cleared if center-aligned mode or up-counting mode are selected, else it takes the auto-reload value.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Reset the TB counter */
  TB_ResetCounter();
```

13.2.8 TB_DebugCmd

Function Name	TB_DebugCmd
Function Prototype	`void TB_DebugCmd(FunctionalState NewState)`
Behavior Description	Enables or disables the TB peripheral Debug Control.
Input Parameter	NewState: new state of the TB Debug control. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TB Debug control */
  TB_DebugCmd(ENABLE);
```

13.2.9 TB_CounterModeConfig

Function Name	TB_CounterModeConfig
Function Prototype	`void TB_CounterModeConfig(u16 TB_CounterMode)`
Behavior Description	Configures the TB Counter Mode.
Input Parameter	TB_CounterMode: specifies the Counter Mode to be used. Refer to section "*TB_CounterMode on page 158*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Center Aligned counter mode for the TB */
  TB_CounterModeConfig(TB_CounterMode_CenterAligned);
```

13.2.10 TB_SLaveModeConfig

Function Name	TB_SLaveModeConfig
Function Prototype	`void TB_SLaveModeConfig(u16 TB_SMSMode)`
Behavior Description	Configures the TB slave Mode.
Input Parameter	TB_SMSMode: specifies the TB slave mode to be used. Refer to section "*TB_SMSMode*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TB_SMSMode

To select the TB slave mode, use one of the following values:

TB_SMSMode	Meaning
TB_SMSMode_Trigger	The counter starts at a rising edge of the trigger
TB_SMSMode_Gated	The counter clock is enabled when trigger signal is high
TB_SMSMode_External	The rising edge of selected trigger clocks the counter
TB_SMSMode_Reset	The rising edge of the selected trigger signal resets the counter

Example:

```
/* TB counter clock is enabled when trigger signal is high */
  TB_SLaveModeConfig(TB_SMSMode_Gated);
```

13.2.11 TB_GetCounter

Function Name	TB_GetCounter
Function Prototype	`u16 TB_GetCounter(void)`
Behavior Description	Gets the TB Counter value.
Input Parameter	None
Output Parameter	None
Return Parameter	TB counter register value.
Required preconditions	None
Called functions	None

Example:

```
/* Get the TB counter value */
  u16 TBCounter = 0x0000;
  TBCounter = TB_GetCounter();
```

13.2.12 TB_GetICAP1

Function Name	TB_GetICAP1
Function Prototype	`u16 TB_GetICAP1(void)`
Behavior Description	Gets the TB Input capture value.
Input Parameter	None
Output Parameter	None
Return Parameter	The TB ICR1 register value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the TB input capture value */
  u16 TBICAP = 0x0000;
  TBICAP = TB_GetICAP1();
```

13.2.13 TB_SetCounter

Function Name	TB_SetCounter
Function Prototype	`void TB_SetCounter(u16 Counter)`
Behavior Description	Sets the TB Counter value.
Input Parameter	Counter: specifies the TB Counter value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the TB new Counter value */
  TB_SetCounter(0xFF3F);
```

13.2.14 TB_GetFlagStatus

Function Name	TB_GetFlagStatus
Function Prototype	`FlagStatus TB_GetFlagStatus(u16 TB_FLAG)`
Behavior Description	Checks whether the specified TB flag is set or not.
Input Parameter1	TB_FLAG: specifies the flag to check. Refer to section "*TB_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of TB_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

TB_FLAG

The TB flags that can check for are listed in the following table:

TB_FLAG	Meaning
TB_FLAG_Update	TB Update interrupt flag.
TB_FLAG_IC	TB Input Capture interrupt flag.

Example:

```
/* Check if the TB Update flag is set or reset */
 if (TB_GetFlagStatus(TB_FLAG_Update) == SET)
 {
  }
```

13.2.15 TB_ClearFlag

Function Name	TB_ClearFlag
Function Prototype	`void TB_ClearFlag(u16 TB_FLAG)`
Behavior Description	Clears the TB's pending flags.
Input Parameter	TB_FLAG: specifies the flag to clear. Refer to section "*TB_FLAG on page 166*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the TB Input Capture pending flag */
  TB_ClearFlag (TB_FLAG_IC);
```

13.2.16 TB_GetITStatus

Function Name	TB_GetITStatus
Function Prototype	`ITStatus TB_GetITStatus(u16 TB_IT)`
Behavior Description	Checks whether the specified TB interrupt has occurred or not.
Input Parameter1	TB_IT: specifies the TB interrupt source to check. Refer to section "*TB_IT on page 160*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of TB_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Check if the TB Update interrupt has occurred or not */
  if (TB_GetITStatus(TB_IT_Update) == SET)
  {
  }
```

13.2.17 TB_ClearITPendingBit

Function Name	TB_ClearITPendingBit
Function Prototype	`void TB_ClearITPendingBit(u16 TB_FLAG)`
Behavior Description	Clears the TB's interrupt pending bits.
Input Parameter	TB_FLAG: specifies the interrupt pending bit to clear. Refer to section "*TB_IT on page 160*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the TB Input Capture interrupt pending bit */
 TB_ClearITPendingBit(TB_IT_IC);
```

14 Synchronizable Standard Timer (TIM)

The TIM driver may be used for a variety of purposes, including timing operations, external wave generation and measurement.

The first section describes the register structure used in the TIM software library. The second one presents the software library functions.

14.1 TIM register structure

The TIM register structure *TIM_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 CR;
  u16  EMPTY1;
  vu16 SCR;
  u16  EMPTY2;
  vu16 IMCR;
  u16  EMPTY3;
  vu16 OMR1;
  u16  EMPTY4[5];
  vu16 RSR;
  u16  EMPTY5;
  vu16 RER;
  u16  EMPTY6;
  vu16 ISR;
  u16  EMPTY7;
  vu16 CNT;
  u16  EMPTY8;
  vu16 PSC;
  u16  EMPTY9[3];
  vu16 ARR;
  u16  EMPTY10;
  vu16 OCR1;
  u16  EMPTY11;
  vu16 OCR2;
  u16  EMPTY12[9];
  vu16 ICR1;
  u16  EMPTY13;
  vu16 ICR2;
  u16  EMPTY14[9];
  vu16 DMAB;
  u16  EMPTY15;
} TIM_TypeDef;
```

The following table presents the TIM registers:

Register	Description
CR	Control Register
SCR	Synchro Control Register
IMCR	Input Mode Control Register
OMR1	Output Mode Register
RSR	Request Selection Register
RER	Request Enable Register
ISR	Interrupt Status Register
CNT	Counter Register
PSC	Prescaler Register
ARR	Auto-Reload Register
OCR1	Output Compare Register1
OCR2	Output Compare Register2
ICR1	Input Capture Register1
ICR2	Input Capture Register2
DMAB	DMA Burst Address

The 3 TIM peripherals are declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define TIM0_BASE       (PERIPH_BASE + 0x8C00)
#define TIM1_BASE       (PERIPH_BASE + 0x9000)
#define TIM2_BASE       (PERIPH_BASE + 0x9400)

#ifndef DEBUG
...
#define TIM0  ((TIM_TypeDef *)  TIM0_BASE)
#define TIM1  ((TIM_TypeDef *)  TIM1_BASE)
#define TIM2  ((TIM_TypeDef *)  TIM2_BASE)
...
#else
...
#ifdef _TIM0
  EXT TIM_TypeDef           *TIM0;
#endif /*_TIM0 */

#ifdef _TIM1
  EXT TIM_TypeDef           *TIM1;
#endif /*_TIM1 */

#ifdef _TIM2
  EXT TIM_TypeDef           *TIM2;
#endif /*_TIM2 */
...
#endif
```

When debug mode is used, TIM pointer is initialized in 75x_lib.c file:

```
#ifdef _TIM0
  TIM0 = (TIM_TypeDef *)  TIM0_BASE;
#endif /*_TIM0 */

#ifdef _TIM1
  TIM1 = (TIM_TypeDef *)  TIM1_BASE;
#endif /*_TIM1 */

#ifdef _TIM2
  TIM2 = (TIM_TypeDef *)  TIM2_BASE;
#endif /*_TIM2 */
```

_TIM, _TIM0, _TIM1 and _TIM2 must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _TIM
#define _TIM0
#define _TIM1
#define _TIM2
```

14.2 Software library functions

The following table lists the various functions of the TIM library.

Function Name	Description
TIM_DeInit	Deinitializes the TIMx peripheral registers to their default reset values.
TIM_Init	Initializes the TIMx peripheral according to the specified parameters in the TIM_InitStruct.
TIM_StructInit	Fills each TIM_InitStruct member with its default value.
TIM_Cmd	Enables or disables the specified TIM peripheral.
TIM_ITConfig	Enables or disables the specified TIM interrupts.
TIM_PreloadConfig	Enables or disables TIM peripheral Preload register on OCRx.
TIM_DMAConfig	Configures the TIM0's DMA interface.
TIM_DMACmd	Enables or disables the TIM0's DMA interface.
TIM_ClockSourceConfig	Configures the TIM clock source.
TIM_SetPrescaler	Sets the TIM prescaler value.
TIM_SetPeriod	Sets the TIM period value.
TIM_SetPulse	Sets the TIM pulse value.
TIM_GetICAP1	Gets the Input Capture 1 value.
TIM_GetICAP2	Gets the Input Capture 2 value.
TIM_GetPWMIPulse	Gets the PWM input pulse value.
TIM_GetPWMIPeriod	Gets the PWM input period value.
TIM_DebugCmd	Enables or disables the specified TIM peripheral Debug control.
TIM_CounterModeConfig	Specifies the Counter Mode to be used.
TIM_ForcedOCConfig	Forces the TIM output waveform to active or inactive level.
TIM_ResetCounter	Re-initializes the counter and generates an update of the Registers.
TIM_SynchroConfig	Synchronizes two Timers in a specified mode.
TIM_GetFlagStatus	Checks whether the specified TIM flag is set or not.
TIM_ClearFlag	Clears the TIMx's pending flags.
TIM_GetITStatus	Checks whether the specified TIM interrupt has occurred or not.
TIM_ClearITPendingBit	Clears the TIMx's interrupt pending bits.

14.2.1 TIM_DeInit

Function Name	TIM_DeInit
Function Prototype	`void TIM_DeInit(TIM_TypeDef* TIMx)`
Behavior Description	Deinitializes the TIMx peripheral registers to their default reset values.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig()

Example:

```
/* Deinitializes the TIM1 registers to their default reset values */
  TIM_DeInit(TIM1);
```

14.2.2 TIM_Init

Function Name	TIM_Init
Function Prototype	`void TIM_Init(TIM_TypeDef* TIMx, TIM_InitTypeDef* TIM_InitStruct)`
Behavior Description	Initializes the TIMx peripheral according to the specified parameters in the TIM_InitStruct .
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_InitStruct: pointer to a TIM_InitTypeDef structure that contains the configuration information for the specified TIM peripheral. Refer to section "*TIM_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TIM_InitTypeDef

The TIM_InitTypeDef structure is defined in the *75x_tim.h* file:

```
typedef struct
{
  u16 TIM_Mode;
  u16 TIM_Prescaler;
  u16 TIM_ClockSource;
  u16 TIM_ExtCLKEdge;
  u16 TIM_CounterMode;
  u16 TIM_Period;
  u16 TIM_Channel;
  u16 TIM_Pulse1;
  u16 TIM_Pulse2;
  u16 TIM_RepetitivePulse;
  u16 TIM_Polarity1;
  u16 TIM_Polarity2;
  u16 TIM_IC1Selection;
  u16 TIM_IC2Selection;
  u16 TIM_IC1Polarity;
  u16 TIM_IC2Polarity;
  u16 TIM_PWMI_ICSelection;
  u16 TIM_PWMI_ICPolarity;
} TIM_InitTypeDef;
```

TIM_Mode

Specifies the Timer Mode. This member can be one of the following values:

TIM_Mode	Meaning
TIM_Mode_OCTiming	Output Compare Timing Mode.
TIM_Mode_OCActive	Output Compare Active Mode.
TIM_Mode_OCInactive	Output Compare Inactive Mode.
TIM_Mode_OCToggle	Output Compare Toggling Mode.
TIM_Mode_PWM	Pulse Width Modulation Mode.
TIM_Mode_PWMI	Pulse Width Modulation Input Mode.
TIM_Mode_IC	Input Capture Mode.
TIM_Mode_Encoder1	Encoder Mode.
TIM_Mode_Encoder2	Encoder Mode.
TIM_Mode_Encoder3	Encoder Mode.
TIM_Mode_OPM_PWM	One Pulse in PWM Mode
TIM_Mode_OPM_Toggle	One Pulse in Toggle Mode
TIM_Mode_OPM_Active	One Pulse in Active Mode

TIM_Prescaler

Specifies the Prescaler value used to divide the TIM clock. The clock of the Timer Counter is divided by TIM_Prescaler + 1.

This member must be a number between 0x0000 and 0xFFFF.

TIM_ClockSource

Specifies the Timer clock source. This member can be one of the following values:

TIM_ClockSource	Meaning
TIM_ClockSource_Internal	CK_TIM internal clock.
TIM_ClockSource_TI11	External input pin TI1 connected to IC1.
TIM_ClockSource_TI12	External input pin TI1 connected to IC2.
TIM_ClockSource_TI22	External input pin TI2 connected to IC2.
TIM_ClockSource_TI21	External input pin TI2 connected to IC1.

TIM_ExtCLKEdge

Specifies the external clock Polarity. This member can be one of the following values:

TIM_ExtCLKEdge	Meaning
TIM_ExtCLKEdge_Falling	External clock Falling Edge.
TIM_ExtCLKEdge_Rising	External clock Rising Edge.

TIM_CounterMode

Specifies the counter mode. This member can be one of the following values:

TIM_CounterMode	Meaning
TIM_CounterMode_Up	TIM Up Counting Mode.
TIM_CounterMode_Down	TIM Down Counting Mode.
TIM_CounterMode_CenterAligned1	TIM Center Aligned Mode1.
TIM_CounterMode_CenterAligned2	TIM Center Aligned Mode2.
TIM_CounterMode_CenterAligned3	TIM Center Aligned Mode3.

TIM_Period

Specifies the Period value to be loaded in the active Auto-Reload Register at the next update event. This member must be a number between 0x0000 and 0xFFFF.

TIM_Channel

Specifies the Timer Channel to be used. This member can be one of the following values:

TIM_Channel	Meaning
TIM_Channel_1	Timer Channel 1 is used.
TIM_Channel_2	Timer Channel 2 is used.
TIM_Channel_ALL	Timer Channel 1 and 2 are used.

TIM_Pulse1

Specifies the Pulse1 value to be loaded in the active Output Compare 1 Register. This member must be a number between 0x0000 and 0xFFFF.

TIM_Pulse2

Specifies the Pulse2 value to be loaded in the active Output Compare 2 Register. This member must be a number between 0x0000 and 0xFFFF.

TIM_RepetitivePulse

Specifies the one pulse mode. This member can be one of the following values:

TIM_RepetitivePulse	Meaning
TIM_RepetitivePulse_Disable	Disable Repetitive one pulse mode
TIM_RepetitivePulse_Enable	Enable Repetitive one pulse mode

TIM_Polarity1

Specifies the Output Compare 1 signal Polarity. This member can be one of the following values:

TIM_Polarity1	Meaning
TIM_Polarity1_High	Output Compare 1 Polarity active High
TIM_Polarity1_Low	Output Compare 1 Polarity active Low

TIM_Polarity2

Specifies the Output Compare 2 signal Polarity. This member can be one of the following values:

TIM_Polarity2	Meaning
TIM_Polarity2_High	Output Compare 2 Polarity active High
TIM_Polarity2_Low	Output Compare 2 Polarity active Low

TIM_IC1Selection

Determines if IC1 is connected to TI1 or TI2. This member can be one of the following values:

TIM_IC1Selection	Meaning
TIM_IC1Selection_TI1	Timer Input 1 is selected
TIM_IC1Selection_TI2	Timer Input 2 is selected

TIM_IC2Selection

Determines if IC2 is connected to TI1 or TI2. This member can be one of the following values:

TIM_IC2Selection	Meaning
TIM_IC2Selection_TI1	Timer Input 1 is selected
TIM_IC2Selection_TI2	Timer Input 2 is selected

TIM_IC1Polarity

Specifies the Input Capture 1 signal Polarity. This member can be one of the following values:

TIM_IC1Polarity	Meaning
TIM_IC1Polarity_Falling	Input Capture 1 Falling Edge
TIM_IC1Polarity_Rising	Input Capture 1 Rising Edge

TIM_IC2Polarity

Specifies the Input Capture 2 signal Polarity. This member can be one of the following values:

TIM_IC2Polarity	Meaning
TIM_IC2Polarity_Falling	Input Capture 2 Falling Edge
TIM_IC2Polarity_Rising	Input Capture 2 Rising Edge

TIM_PWMI_ICSelection

Determines if PWM Input signal is connected to TI1 or TI2. This member can be one of the following values:

TIM_PWMI_ICSelection	Meaning
TIM_PWMI_ICSelection_TI1	Timer Input 1 is selected
TIM_PWMI_ICSelection_TI2	Timer Input 2 is selected

TIM_PWMI_ICPolarity

Specifies the PWM Input signal Polarity. This member can be one of the following values:

TIM_PWMI_ICPolarity	Meaning
TIM_PWMI_ICPolarity_Falling	Input Capture Falling Edge
TIM_PWMI_ICPolarity_Rising	Input Capture Rising Edge

Used TIM_InitTypeDef structure members for each TIM mode:

	Output Compare				PWM	Input Capture			OPM
	Timing	Active	Inactive	Toggling		IC	PWMI	ENC	
TIM_Mode	x	x	x	x	x	x	x	x	x
TIM_Prescaler	x	x	x	x	x	x	x	x	x
TIM_ClockSource	x	x	x	x	x	x	x	x	x
TIM_CounterMode	x	x	x	x	x	x	x	x	x
TIM_Period	x	x	x	x	x	x	x	x	x
TIM_Channel	x	x	x	x	x	x	x	x	x
TIM_Pulse1	x	x	x	x	x				x
TIM_Pulse2	x	x	x	x	x				x
TIM_Polarity1		x		x	x				x
TIM_Polarity2		x		x	x				x
TIM_RepetitivePulse									x
TIM_IC1Selection						x		x	x
TIM_IC2Selection						x		x	x
TIM_IC1Polarity						x		x	x
TIM_IC2Polarity						x		x	x
TIM_PWMI_ICSelection							x		
TIM_PWMI_ICPolarity							x		

Note: Within the TIM_Init function you can configure only one channel at a time.

Example:

```
/* TIM Configuration in PWM Mode */
  TIM_InitTypeDef TIM_InitStructure;

  TIM_InitStructure.TIM_Mode = TIM_Mode_PWM;
  TIM_InitStructure.TIM_Prescaler = 0x1;
  TIM_InitStructure.TIM_ClockSource = TIM_ClockSource_Internal;
  TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_InitStructure.TIM_Period = 0xFFFF;
  TIM_InitStructure.TIM_Channel = TIM_Channel_2;
 TIM_InitStructure.TIM_Pulse2 = 0x3FFF;
  TIM_InitStructure.TIM_Polarity2 = TIM_Polarity2_Low;
  TIM_Init(TIM0, &TIM_InitStructure);
```

14.2.3 TIM_StructInit

Function Name	TIM_StructInit
Function Prototype	`void TIM_StructInit(TIM_InitTypeDef* TIM_InitStruct)`
Behavior Description	Fills each TIM_InitStruct member with its default value.
Input Parameter	TIM_InitStruct: pointer to a TIM_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The TIM_InitStruct members have the following default values:

Member	Default value
TIM_Mode	TIM_Mode_OCTiming
TIM_Prescaler	TIM_Prescaler_Reset_Mask
TIM_ClockSource	TIM_ClockSource_Internal
TIM_ExtCLKEdge	TIM_ExtCLKEdge_Rising
TIM_CounterMode	TIM_CounterMode_Up
TIM_Period	TIM_Period_Reset_Mask
TIM_Channel	TIM_Channel_ALL
TIM_Pulse1	TIM_Pulse1_Reset_Mask
TIM_Pulse2	TIM_Pulse2_Reset_Mask
TIM_RepetitivePulse	TIM_RepetitivePulse_Disable
TIM_Polarity1	TIM_Polarity1_Low
TIM_Polarity2	TIM_Polarity2_Low
TIM_IC1Selection	TIM_IC1Selection_TI1
TIM_IC2Selection	TIM_IC2Selection_TI1
TIM_IC1Polarity	TIM_IC1Polarity_Rising
TIM_IC2Polarity	TIM_IC2Polarity_Rising
TIM_PWMI_ICSelection	TIM_PWMI_ICSelection_TI1
TIM_PWMI_ICPolarity	TIM_PWMI_ICPolarity_Rising

Example:

```
/* The following example illustrates how to initialize a TIM_InitTypeDef structure*/
  TIM_InitTypeDef TIM_InitStructure;
  TIM_StructInit(&TIM_InitStructure);
```

14.2.4 TIM_Cmd

Function Name	TIM_Cmd
Function Prototype	`void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified TIM peripheral.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	NewState: new state of the TIMx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM1 */
  TIM_Cmd(TIM1, ENABLE);
```

14.2.5 TIM_ITConfig

Function Name	TIM_ITConfig
Function Prototype	`void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified TIM interrupts.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_IT: specifies the TIM interrupt sources to be enabled or disabled. Refer to section "*TIM_IT* "for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the specified TIMx interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TIM_IT

To enable or disable TIM interrupts, use a combination of one or more of the following values:

TIM_IT	Meaning
TIM_IT_IC1	Timer Input Capture 1 Interrupt source
TIM_IT_IC2	Timer Input Capture 2 Interrupt source
TIM_IT_OC1	Timer Output Compare 1 Interrupt source
TIM_IT_OC2	Timer Output Compare 2 Interrupt source
TIM_IT_Update[1)]	Timer Update Interrupt source
TIM_IT_GlobalUpdate[1)]	Timer Global Update Interrupt source

[1)] TIM_IT_Update and TIM_GlobalUpdate interrupt sources must not be configured simultaneously

Example:

```
/* Enables the TIM1 Input Capture 1 Interrupt */
  TIM_ITConfig(TIM1, TIM_IT_IC1, ENABLE);
```

14.2.6 TIM_PreloadConfig

Function Name	TIM_PreloadConfig
Function Prototype	`void TIM_PreloadConfig(TIM_TypeDef *TIMx, u16 TIM_Channel, FunctionalState Newstate)`
Behavior Description	Enables or disables TIM peripheral Preload register on OCRx.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_Channel: specifies the TIM channel to be used. Refer to"*TIM_Channel on page 176*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the TIMx peripheral Preload register This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enables the TIM1 Preload register on Output Compare 1 */
  TIM_PreloadConfig(TIM1, TIM_Channel_1, ENABLE);
```

14.2.7 TIM_DMAConfig

Function Name	TIM_DMAConfig
Function Prototype	`void TIM_DMAConfig(u16 TIM_DMASources, u16 TIM_OCRMState, u16 TIM_DMABase)`
Behavior Description	Configures the TIM0's DMA interface.
Input Parameter1	TIM_DMASource: specifies the DMA Request sources. Refer to section "*TIM_DMASource*" for more details on the allowed values of this parameter.
Input Parameter2	TIM_OCRMState: the state of output compare request mode. Refer to section "*TIM_OCRMState*" for more details on the allowed values of this parameter.
Input Parameter3	TIM_DMABase:DMA Base address. Refer to section "*TIM_DMABase*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TIM_DMASource

To select the TIM DMA Requests, use a combination of one or more of the following values:

TIM_DMASource	Meaning
TIM_DMASource_OC1	TIM Output Compare 1 DMA source
TIM_DMASource_OC2	TIM Output Compare 2 DMA source
TIM_DMASource_IC1	TIM Input Capture 1 DMA source
TIM_DMASource_IC2	TIM Input Capture 2 DMA source.
TIM_DMASource_Update	TIM Update DMA source.

TIM_OCRMState

To select the TIM Output compare Mode State, use one of the following values:

TIM_OCRMState	Meaning
TIM_OCRMState_Enable	OCx DMA requests are generated by the result of the comparison between the counter and the corresponding output compare register.
TIM_OCRMState_Disable	OCx DMA requests are generated by the Update Event.

TIM_DMABase

To select the DMA base, use one of the following values:

TIM_DMABase	Meaning
TIM_DMABase_CR	CR register used as TIM DMA Base.
TIM_DMABase_SCR	SCR register used as TIM DMA Base.
TIM_DMABase_IMCR	IMCR register used as TIM DMA Base.
TIM_DMABase_OMR1	OMR1 register used as TIM DMA Base.
TIM_DMABase_RSR	RSR register used as TIM DMA Base.
TIM_DMABase_RER	RER register used as TIM DMA Base.
TIM_DMABase_ISR	ISR register used as TIM DMA Base.
TIM_DMABase_CNT	CNT register used as TIM DMA Base.
TIM_DMABase_PSC	PSC register used as TIM DMA Base.
TIM_DMABase_ARR	ARR register used as TIM DMA Base.
TIM_DMABase_OCR1	OCR1 register used as TIM DMA Base.
TIM_DMABase_OCR2	OCR2 register used as TIM DMA Base.
TIM_DMABase_ICR1	ICR1 register used as TIM DMA Base.
TIM_DMABase_ICR2	ICR2 register used as TIM DMA Base.

Example:

```
/* TIM Output Compare 1 DMA request configuration */
  TIM_DMAConfig(TIM_DMASource_OC1, TIM_OCRMState_Enable, TIM_DMABase_OCR1);
```

14.2.8 TIM_DMACmd

Function Name	TIM_Cmd
Function Prototype	`void TIM_DMACmd(u16 TIM_DMASources, FunctionalState Newstate)`
Behavior Description	Enables or disables the TIM0's DMA interface.
Input Parameter1	TIM_DMASource: specifies the DMA Request sources. Refer to section "*TIM_DMASource on page 183*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM DMA Output compare 1 request */
TIM_DMACmd(TIM_DMASource_OC1, ENABLE);
```

14.2.9 TIM_ClockSourceConfig

Function Name	TIM_ClockSourceConfig
Function Prototype	`void TIM_ClockSourceConfig(TIM_TypeDef *TIMx, u16 TIM_ClockSource, u16 TIM_ExtCLKEdge)`
Behavior Description	Configures the TIM clock source.
Input Parameter 1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter 2	TIM_ClockSource: specifies the TIM source clock. Refer to"*TIM_ClockSource on page 175*" for more details on the allowed values of this parameter.
Input Parameter 3	TIM_ExtCLKEdge: specifies the External input signal edge. Refer to"*TIM_ExtCLKEdge on page 176*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Configure the TIM1 to be clocked by the internal CK_TIM clock */
  TIM_ClockSourceConfig(TIM1, TIM_ClockSource_Internal, TIM_ExtCLKEdge_Falling);
```

14.2.10 TIM_SetPrescaler

Function Name	TIM_SetPrescaler
Function Prototype	`void TIM_SetPrescaler(TIM_TypeDef* TIMx, u16 Prescaler)`
Behavior Description	Sets the TIM prescaler value.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	Prescaler: TIM prescaler new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 new Prescaler value */
  u16 TIMPrescaler = 0xFF00;
  TIM_SetPrescaler(TIM1, TIMPrescaler);
```

14.2.11 TIM_SetPeriod

Function Name	TIM_SetPeriod
Function Prototype	`void TIM_SetPeriod(TIM_TypeDef* TIMx, u16 Period)`
Behavior Description	Sets the TIM period value.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	Period: TIM period new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 new Period value */
  u16 TIMPeriod = 0xFFF0;
  TIM_SetPeriod(TIM1, TIMPeriod);
```

14.2.12 TIM_SetPulse

Function Name	TIM_SetPulse
Function Prototype	`void TIM_SetPulse(TIM_TypeDef* TIMx, u16 TIM_Channel, u16 Pulse)`
Behavior Description	Sets the TIM pulse value.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_Channel: specifies the TIM channel to be used. Refer to"*TIM_Channel on page 176*" for more details on the allowed values of this parameter.
Input Parameter3	Pulse: TIM pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the TIM1 new Channel 1 Pulse value */
  u16 TIMPulse = 0x00F0;
  TIM_SetPulse(TIM1, TIM_CHANNEL_1, TIMPulse);
```

14.2.13 TIM_GetICAP1

Function Name	TIM_GetICAP1
Function Prototype	`u16 TIM_GetICAP1(TIM_TypeDef* TIMx)`
Behavior Description	Gets the Input Capture 1 value.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	Input Capture 1 Register value.
Required preconditions	None
Called functions	None

Example:

```
/* Gets the input capture 1 value of the TIM1 */
  u16 ICAvalue = TIM_GetICAP1(TIM1);
```

14.2.14 TIM_GetICAP2

Function Name	TIM_GetICAP2
Function Prototype	`u16 TIM_GetICAP2(TIM_TypeDef* TIMx)`
Behavior Description	Gets the Input Capture 2 value.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	Input Capture 2 Register value
Required preconditions	None
Called functions	None

Example:

```
/* Gets TIM1 input capture 2 value */
  u16 IC2value = TIM_GetICAP2(TIM1);
```

14.2.15 TIM_GetPWMIPulse

Function Name	TIM_GetPWMIPulse
Function Prototype	`u16 TIM_GetPWMIPulse(TIM_TypeDef* TIMx)`
Behavior Description	Gets the PWM Input pulse value.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	Input Capture 2 Register value
Required preconditions	None
Called functions	None

Example:

```
/* Gets TIM1 Pulse value */
  u16 PWMPulse = TIM_GetPWMIPulse(TIM1);
```

14.2.16 TIM_GetPWMIPeriod

Function Name	TIM_GetPWMIPeriod
Function Prototype	`u16 TIM_GetPWMIPeriod(TIM_TypeDef* TIMx)`
Behavior Description	Gets the PWM Input period value.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	Input Capture 1 Register value
Required preconditions	None
Called functions	None

Example:

```
/* Gets the Period value of the TIM1 */
  u16 PWMIPeriod = TIM_GetPWMIPeriod(TIM1);
```

14.2.17 TIM_DebugCmd

Function Name	TIM_DebugCmd
Function Prototype	`void TIM_DebugCmd(TIM_TypeDef* TIMx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified TIM peripheral Debug control.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	NewState: new state of the TIMx Debug control. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the TIM1 Debug control */
  TIM_DebugCmd(TIM1, ENABLE);
```

14.2.18 TIM_CounterModeConfig

Function Name	TIM_CounterModeConfig
Function Prototype	`void TIM_CounterModeConfig(TIM_TypeDef* TIMx, u16 TIM_CounterMode)`
Behavior Description	Specifies the Counter Mode to be used.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_CounterMode: specifies the Counter Mode to be used. Refer to section "*TIM_CounterMode on page 176*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Center Aligned counter mode1 for the TIM1 */
  TIM_CounterModeConfig(TIM1, TIM_CounterMode_CenterAligned1);
```

14.2.19 TIM_ForcedOCConfig

Function Name	TIM_ForcedOCConfig
Function Prototype	`void TIM_ForcedOCConfig(TIM_TypeDef* TIMx, u16 TIM_Channel,u16 TIM_ForcedAction)`
Behavior Description	Forces the TIM output waveform to active or inactive level.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_Channel: specifies the TIM channel to be used. Refer to section "*TIM_Channel on page 176*" for more details on the allowed values of this parameter.
Input Parameter3	TIM_ForcedAction: specifies the forced Action to be set to the output waveform. Refer to "*TIM_ForcedAction*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

TIM_ForcedAction

The forced actions that can be used are listed in the following table:

TIM_ForcedAction	Meaning
TIM_ForcedAction_Active	Force active level on OCxREF.
TIM_ForcedAction_InActive	Force inactive level on OCxREF.

Example:

```
/* Forces the TIM1 Output Compare 1 signal to the active level */
  TIM_ForcedOCConfig(TIM1, TIM_Channel_1, ForcedAction_Active);
```

14.2.20 TIM_ResetCounter

Function Name	TIM_ResetCounter
Function Prototype	`void TIM_ResetCounter(TIM_TypeDef* TIMx)`
Behavior Description	Re-intializes the TIM counter and generates an update of the Registers. The Prescaler counter is cleared too, however the Prescaler Preload value is not affected. The counter is cleared if center-aligned mode or up-counting mode are selected, else it takes the auto-reload value.
Input Parameter	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Resets the TIM1 counter */
  TIM_ResetCounter(TIM1);
```

14.2.21 TIM_SynchroConfig

Function Name	TIM_SynchroConfig
Function Prototype	`void TIM_SynchroConfig( Master_TypeDef Master, Slave_TypeDef Slave, u16 TIM_SynchroAction, u16 TIM_SynchroMode);`
Behavior Description	Synchronizes two Timers in a specified mode.
Input Parameter 1	Master: specifies the peripheral master. Refer to "*Master_TypeDef*" for more details on the allowed values of this parameter.
Input Parameter 2	Slave: specifies the peripheral slave. Refer to "*Slave_TypeDef*" for more details on the allowed values of this parameter.
Input Parameter3	TIM_SynchroAction: specifies the synchronization Action to be used. Refer to "*TIM_SynchroAction*" for more details on the allowed values of this parameter.
Input Parameter4	TIM_SynchroMode: specifies the synchronization Mode to be used. Refer to "*TIM_SynchroMode*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Master_TypeDef

The peripherals that can used as master are listed in the following enumeration:

Master_TypeDef	Meaning
PWM_Master	PWM is used as Master.
TIM0_Master	TIM0 is used as Master.
TIM1_Master	TIM1 is used as Master.
TIM2_Master	TIM2 is used as Master.

Slave_TypeDef

The peripherals that can used as slave are listed in the following enumeration:

Slave_TypeDef	Meaning
PWM_Slave	PWM is used as Slave.
TIM0_Slave	TIM0 is used as Slave.
TIM1_Slave	TIM1 is used as Slave.
TIM2_Slave	TIM2 is used as Slave.

TIM_SynchroAction

The synchronization actions that can be used are listed in the following table:

TIM_SynchroAction	Meaning
TIM_SynchroAction_Enable	The CNT_EN bit is used as trigger output.
TIM_SynchroAction_Update	The Update event is used as trigger output.
TIM_SynchroAction_Reset	The CNT_RST bit is used as trigger output.
TIM_SynchroAction_OC	The OC1 signal is used as trigger output.

TIM_SynchroMode

The synchronization modes that can be used are listed in the following table:

TIM_SynchroMode	Meaning
TIM_SynchroMode_Gated	Counter clock is enabled when trigger signal is high. Both start and stop of the counter is controlled.
TIM_SynchroMode_Trigger	Counter starts at a rising edge of the trigger. Only the start of the counter is controlled.
TIM_SynchroMode_External	The rising edge of selected trigger clocks the counter.
TIM_SynchroMode_Reset	The rising edge of the selected trigger signal resets the counter and generates an update of the registers.

Example:

```
/* Synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0000. The
Synchronization gated mode is selected: Timer 2 is the slave and starts at the same
time and it is stoped when Timer 1 is disabled */
  TIM_SynchroConfig(TIM1_Master, TIM2_Slave, SynchroAction_Enable,
SynchroMode_Gated);
```

14.2.22 TIM_GetFlagStatus

Function Name	TIM_GetFlagStatus
Function Prototype	`FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, u16 TIM_FLAG)`
Behavior Description	Checks whether the specified TIM flag is set or not.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_FLAG: specifies the flag to check. Refer to section "*TIM_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of TIM_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

TIM_FLAG

The TIM flags that can be checked are listed in the following table:

TIM_FLAG	Meaning
TIM_FLAG_IC1	Timer Input Capture 1 Flag.
TIM_FLAG_IC2	Timer Input Capture 2 Flag.
TIM_FLAG_OC1	Timer Output Compare 1 Flag.
TIM_FLAG_OC2	Timer Output Compare 2 Flag.
TIM_FLAG_Update	Timer Update Flag.

Example:

```
/* Check if the TIM1 Update flag is set or reset */
 if (TIM_GetFlagStatus(TIM1, TIM_FLAG_Update) == SET)
  {
  }
```

14.2.23 TIM_ClearFlag

Function Name	TIM_ClearFlag
Function Prototype	`void TIM_ClearFlag(TIM_TypeDef* TIMx, u16 TIM_Flag)`
Behavior Description	Clears the TIMx's pending flags.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_FLAG: specifies the flag to clear. Refer to section "*TIM_FLAG on page 194*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the TIM1 Output Compare 1 flag */
  TIM_ClearFlag(TIM1, TIM_FLAG_OC1);
```

14.2.24 TIM_GetITStatus

Function Name	TIM_GetITStatus
Function Prototype	`ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, u16 TIM_IT)`
Behavior Description	Checks whether the specified TIM interrupt has occurred or not.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_IT: specifies the TIM interrupt source to check. Refer to section "*TIM_IT on page 182*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of TIM_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Check if the TIM1 Update interrupt has occurred or not */
  if (TIM_GetITStatus(TIM1, TIM_IT_Update) == SET)
  {
  }
```

14.2.25 TIM_ClearITPendingBit

Function Name	TIM_ClearITPending Bit
Function Prototype	`void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, u16 TIM_IT)`
Behavior Description	Clears the TIMx's interrupt pending bits.
Input Parameter1	TIMx: where x can be 0,1 or 2 to select the TIM peripheral.
Input Parameter2	TIM_IT: specifies the interrupt pending bit to clear. Refer to section "*TIM_IT on page 182*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the TIM1 Output Compare 1 interrupt pending bit */
  TIM_ClearITPendingBit (TIM1, TIM_IT_OC1);
```

15 Synchronizable-PWM Timer (PWM)

The PWM driver may be used for a variety of purposes, including timing operations and external wave generation.

The first section describes the register structure used in the PWM software library. The second one presents the software library functions.

15.1 PWM register structure

The PWM register structure *PWM_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 CR;
  u16  EMPTY1;
  vu16 SCR;
  u16  EMPTY2[3];
  vu16 OMR1;
  u16  EMPTY3;
  vu16 OMR2;
  u16  EMPTY4[3];
  vu16 RSR;
  u16  EMPTY5;
  vu16 RER;
  u16  EMPTY6;
  vu16 ISR;
  u16  EMPTY7;
  vu16 CNT;
  u16  EMPTY8;
  vu16 PSC;
  u16  EMPTY9;
  vu16 RCR;
  u16  EMPTY10;
  vu16 ARR;
  u16  EMPTY11;
  vu16 OCR1;
  u16  EMPTY12;
  vu16 OCR2;
  u16  EMPTY13;
  vu16 OCR3;
  u16  EMPTY14[15];
  vu16 DTR;
  u16  EMPTY15;
  vu16 DMAB;
  u16  EMPTY16;
} PWM_TypeDef;
```

The following table presents the PWM registers:

Register	Description
CR	Control Register
SCR	Synchro Control Register
OMR1	Output Mode Register1
OMR2	Output Mode Register2
RSR	Request Selection Register
RER	Request Enable Register
ISR	Interrupt Status Register
CNT	Counter Register
PSC	Prescaler Register
RCR	Repetition Counter Register
ARR	Auto-Reload Register
OCR1	Output Compare Register1
OCR2	Output Compare Register2
OCR3	Output Compare Register3
DTR	Dead Time Register
DMAB	DMA Burst Address

The PWM peripheral is declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define PWM_BASE        (PERIPH_BASE + 0x9800)

#ifndef DEBUG
...
#define PWM   ((PWM_TypeDef *)  PWM_BASE)
...
#else   /* DEBUG */
...
#ifdef _PWM
  EXT PWM_TypeDef           *PWM;
#endif /*_PWM */
...
#endif  /* DEBUG */
```

When debug mode is used, PWM pointer is initialized in 75x_lib.c file:

```
#ifdef _PWM
  PWM = (PWM_TypeDef *)  PWM_BASE;
#endif /*_PWM */
```

_PWM must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _PWM
...
```

15.2 Software library functions

The following table lists the various functions of the PWM library.

Function Name	Description
PWM_DeInit	Deinitializes the PWM peripheral registers to their default reset values.
PWM_Init	Initializes the PWM peripheral according to the specified parameters in the PWM_InitStruct.
PWM_StructInit	Fills each PWM_InitStruct member with its default value.
PWM_Cmd	Enables or disables the PWM peripheral.
PWM_CtrlPWMOutputs	Enables or disables PWM peripheral Main Outputs.
PWM_ITConfig	Enables or disables the PWM interrupts.
PWM_DMAConfig	Configures the PWM's DMA interface.
PWM_DMACmd	Enables or disables the PWM's DMA interface.
PWM_SetPrescaler	Sets the PWM prescaler value.
PWM_SetPeriod	Sets the PWM period value.
PWM_SetPulse	Sets the PWM pulse value.
PWM_SetPulse1	Sets the PWM Channel 1 pulse value.
PWM_SetPulse2	Sets the PWM Channel 2 pulse value.
PWM_SetPulse3	Sets the PWM Channel 3 pulse value.
PWM_DebugCmd	Enables or disables PWM peripheral Debug control.
PWM_CounterModeConfig	Specifies the Counter Mode to be used.
PWM_ForcedOCConfig	Forces the PWM output waveform to active or inactive level.
PWM_SetDeadTime	Inserts dead time between the OCx and OCNx.
PWM_ResetCounter	Re-intializes the counter and generates an update of the registers.
PWM_TRGOSelection	Sets the PWM Master Mode selection bits
PWM_GetFlagStatus	Checks whether the specified PWM flag is set or not.
PWM_ClearFlag	Clears the PWM's pending flags.
PWM_GetITStatus	Checks whether the specified PWM interrupt has occurred or not.
PWM_ClearITPendingBit	Clears the PWM's interrupt pending bits.

*Note: To synchronize the PWM with TIMx (x:0, 1 or 2) timers, you have to use the **TIM_SynchroConfig** function available within the TIM driver.*

15.2.1 PWM_DeInit

Function Name	PWM_DeInit
Function Prototype	`void PWM_DeInit(void)`
Behavior Description	Deinitializes the PWM peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitializes the PWM registers to their default reset value */
  PWM_DeInit();
```

15.2.2 PWM_Init

Function Name	PWM_Init
Function Prototype	`void PWM_Init(PWM_InitTypeDef* PWM_InitStruct)`
Behavior Description	Initializes the PWM peripheral according to the specified parameters in the PWM_InitStruct .
Input Parameter	PWM_InitStruct: pointer to a PWM_InitTypeDef structure that contains the configuration information for the PWM peripheral. Refer to section "*PWM_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PWM_InitTypeDef

The PWM_InitTypeDef structure is defined in the *75x_pwm.h* file:

```
typedef struct
{
  u16 PWM_Mode;
  u16 PWM_Prescaler;
  u16 PWM_CounterMode;
  u16 PWM_Period;
  u16 PWM_Complementary;
  u16 PWM_OCState;
  u16 PWM_OCNState;
  u16 PWM_Channel;
  u16 PWM_Pulse1;
  u16 PWM_Pulse2;
  u16 PWM_Pulse3;
  u16 PWM_Polarity1;
  u16 PWM_Polarity2;
  u16 PWM_Polarity3;
  u16 PWM_Polarity1N;
  u16 PWM_Polarity2N;
  u16 PWM_Polarity3N;
  u16 PWM_DTRAccess;
  u16 PWM_DeadTime;
  u16 PWM_Emergency;
  u16 PWM_LOCKLevel;
  u16 PWM_OSSIState;
  u8 PWM_RepetitionCounter;
} PWM_InitTypeDef;
```

PWM_Mode

Specifies the PWM mode. This member can be one of the following values:

PWM_Mode	Meaning
PWM_Mode_OCTiming	Output CompareTiming Mode
PWM_Mode_OCActive	Output Compare Active Mode
PWM_Mode_OCInactive	Output Compare Inactive Mode
PWM_Mode_OCToggle	Output Compare Toggling Mode
PWM_Mode_PWM	Pulse Width Modulation Mode

PWM_Prescaler

Specifies the Prescaler value to divide the PWM_ICK clock. The clock of the PWM Counter is divided by PWM_Prescaler + 1.

This member must be a number between 0x0000 and 0xFFFF.

PWM_CounterMode

Specifies the counter mode. This member can be one of the following values:

PWM_CounterMode	Meaning
PWM_CounterMode_Up	PWM Up Counting Mode.
PWM_CounterMode_Down	PWM Down Counting Mode.
PWM_CounterMode_CenterAligned1	PWM Center Aligned Mode1.
PWM_CounterMode_CenterAligned2	PWM Center Aligned Mode2.
PWM_CounterMode_CenterAligned3	PWM Center Aligned Mode3.

PWM_Period

Specifies the Period value to be loaded in the active Auto-Reload Register at the next update event. This member must be a number between 0x0000 and 0xFFFF.

PWM_Complementary

Enables or disables the complementary PWM feature. This member can be one of the following values:

PWM_Complementary	Meaning
PWM_Complementary_Enable	PWM Complementary Mode Enable.
PWM_Complementary_Disable	PWM Complementary Mode Disable.

PWM_OCState

Specifies the PWM Output Compare State to be used. This member can be one of the following values:

PWM_OCState	Meaning
PWM_OCState_Disable	PWM Output Compare State Disable.
PWM_OCState_Enable	PWM Output Compare State Enable.
PWM_OCState_OffState	PWM Output Compare State OffState.

PWM_OCNState

Specifies the PWM Output Compare State to be used. This member can be one of the following values:

PWM_OCNState	Meaning
PWM_OCNState_Disable	PWM Output Compare State Disable.
PWM_OCNState_Enable	PWM Output Compare State Enable.
PWM_OCNState_OffState	PWM Output Compare State OffState.

PWM_Channel

Specifies the PWM Channel to be used. This member can be one of the following values:

PWM_Channel	Meaning
PWM_Channel_1	PWM Channel 1 is used.
PWM_Channel_2	PWM Channel 2 is used.
PWM_Channel_3	PWM Channel 3 is used.
PWM_Channel_ALL	PWM Channel 1, 2 and 3 are used.

PWM_Pulse1

Specifies the Pulse 1 value to be loaded in the active Output Compare 1 Register.

The PWM_Pulse1 presents the DutyCycle value in PWM mode. This member must be a number between 0x0000 and 0xFFFF.

PWM_Pulse2

Specifies the Pulse 2 value to be loaded in the active Output Compare 2 Register.

The PWM_Pulse2 presents the DutyCycle value in PWM mode. This member must be a number between 0x0000 and 0xFFFF.

PWM_Pulse3

Specifies the Pulse 3 value to be loaded in the active Output Compare 3 Register.

The PWM_Pulse3 presents the DutyCycle value in PWM mode. This member must be a number between 0x0000 and 0xFFFF.

PWM_Polarity1

Specifies the Output Compare 1 signal Polarity. This member can be one of the following values:

PWM_Polarity1	Meaning
PWM_Polarity1_High	Output Compare 1 Polarity active High
PWM_Polarity1_Low	Output Compare 1 Polarity active Low

PWM_Polarity2

Specifies the Output Compare 2 signal Polarity. This member can be one of the following values:

PWM_Polarity2	Meaning
PWM_Polarity2_High	Output Compare 2 Polarity active High
PWM_Polarity2_Low	Output Compare 2 Polarity active Low

PWM_Polarity3

Specifies the Output Compare 3 signal Polarity. This member can be one of the following values:

PWM_Polarity3	Meaning
PWM_Polarity3_High	Output Compare 3 Polarity active High
PWM_Polarity3_Low	Output Compare 3 Polarity active Low

PWM_Polarity1N

Specifies the Output Compare 1N signal Polarity. This member can be one of the following values:

PWM_Polarity1N	Meaning
PWM_Polarity1N_High	Output Compare 1N Polarity active High
PWM_Polarity1N_Low	Output Compare 1N Polarity active Low

PWM_Polarity2N

Specifies the Output Compare 2N signal Polarity. This member can be one of the following values:

PWM_Polarity2N	Meaning
PWM_Polarity2N_High	Output Compare 2N Polarity active High
PWM_Polarity2N_Low	Output Compare 2N Polarity active Low

PWM_Polarity3N

Specifies the Output Compare 3N signal Polarity. This member can be one of the following values:

PWM_Polarity3N	Meaning
PWM_Polarity3N_High	Output Compare 3N Polarity active High
PWM_Polarity3N_Low	Output Compare 3N Polarity active Low

PWM_DTRAccess

Enable or disable the configuration of DTR register parameters: DeadTime, Emergency, LOCKLevel, OSSIState, OCState and OCNState.

This member can be one of the following values:

PWM_DTRAccess	Meaning
PWM_DTRAccess_Enable	DTR register Access is enabled
PWM_DTRAccess_Disable	DTR register Access is disabled

PWM_DeadTime

Specifies the dead time to manage the time between the switching-off and the switching-on instants of the outputs.

PWM_Emergency

Enable or disable the PWM Emergency Input pin. This member can be one of the following values:

PWM_Emergency	Meaning
PWM_Emergency_Enable	PWM Emergency Input pin is enabled
PWM_Emergency_Disable	PWM Emergency Input pin is disabled

PWM_LOCKLevel

Specifies the LOCK level Parameters to be used. This member can be one of the following values:

PWM_LOCKLevel	Meaning
PWM_LOCKLevel_OFF	No bits are locked.
PWM_LOCKLevel_1	LOCK level 1 is used.
PWM_LOCKLevel_2	LOCK level 2 is used.
PWM_LOCKLevel_3	LOCK level 3 is used.

PWM_OSSIState

Specifies the Off-State for Idle state. This member can be one of the following values:

PWM_OSSIState	Meaning
PWM_OSSIState_Enable	PWM OSSIState is enabled
PWM_OSSIState_Disable	PWM OSSIState is disabled

PWM_RepetitionCounter

Specifies the repetition counter value. Each time the RCR down-counter reaches zero, an update event is generated and it restarts counting from RCR value (N).

This means in PWM mode (N+1) corresponds to:

- The number of PWM periods in edge-aligned mode
- The number of half PWM period in center-aligned mode

This member must be a number between 0x00 and 0xFF.

Note: Within the PWM_Init function you can configure one channel at a time.

Example:

```
/* The following example illustrates how to configure the PWM complementary Mode */
  PWM_InitStructure.PWM_Mode = PWM_Mode_PWM;
  PWM_InitStructure.PWM_Prescaler = 0xFF;
  PWM_InitStructure.PWM_CounterMode = PWM_CounterMode_Up;
  PWM_InitStructure.PWM_Period = 0xFFFF;
  PWM_InitStructure.PWM_Complementary = PWM_Complementary_Enable;
  PWM_InitStructure.PWM_Channel = PWM_Channel_1;
  PWM_InitStructure.PWM_Pulse1 = 0x3FFF;
  PWM_InitStructure.PWM_DTRAccess = PWM_DTRAccess_Disable;
  PWM_InitStructure.PWM_Polarity1 = PWM_Polarity1_Low;
  PWM_InitStructure.PWM_Polarity1N = PWM_Polarity1N_Low;
  PWM_InitStructure.PWM_RepetitionCounter = 0x5;
  PWM_Init(&PWM_InitStructure);

  PWM_InitStructure.PWM_Channel = PWM_Channel_2;
  PWM_InitStructure.PWM_Pulse2 = 0x7FFF;
  PWM_InitStructure.PWM_Polarity2 = PWM_Polarity2_Low;
  PWM_InitStructure.PWM_Polarity2N = PWM_Polarity2N_Low;
  PWM_Init(&PWM_InitStructure);

  PWM_InitStructure.PWM_Channel = PWM_Channel_3;
  PWM_InitStructure.PWM_Pulse3 = 0xBFFF;
  PWM_InitStructure.PWM_Polarity3 = PWM_Polarity3_Low;
  PWM_InitStructure.PWM_Polarity3N = PWM_Polarity3N_Low;
  PWM_InitStructure.PWM_DTRAccess = PWM_DTRAccess_Enable;
  PWM_InitStructure.PWM_Emergency = PWM_Emergency_Enable;
  PWM_InitStructure.PWM_DeadTime = 0x25;
  PWM_InitStructure.PWM_LOCKLevel = PWM_LOCKLevel_3;
  PWM_InitStructure.PWM_OSSIState = PWM_OSSIState_Enable;
  PWM_Init(&PWM_InitStructure);
```

15.2.3 PWM_StructInit

Function Name	PWM_StructInit
Function Prototype	`void PWM_StructInit(PWM_InitTypeDef* PWM_InitStruct)`
Behavior Description	Fills each PWM_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	PWM_InitStruct: pointer to a PWM_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The PWM_InitStruct members have the following default values:

Member	Default value
PWM_Mode	PWM_Mode_OCTiming
PWM_Prescaler	PWM_Prescaler_Reset_Mask
PWM_CounterMode	PWM_CounterMode_Up
PWM_Period	PWM_Period_Reset_Mask
PWM_Complementary	PWM_Complementary_Disable
PWM_OCState	PWM_OCState_Disable
PWM_OCNState	PWM_OCNState_Disable
PWM_Channel	PWM_Channel_1
PWM_Pulse1	PWM_Pulse1_Reset_Mask
PWM_Pulse2	PWM_Pulse2_Reset_Mask
PWM_Pulse3	PWM_Pulse3_Reset_Mask
PWM_Polarity1	PWM_Polarity1_High
PWM_Polarity2	PWM_Polarity2_High
PWM_Polarity3	PWM_Polarity3_High
PWM_Polarity1N	PWM_Polarity1N_High
PWM_Polarity2N	PWM_Polarity2N_High
PWM_Polarity3N	PWM_Polarity3N_High
PWM_DTRAccess	PWM_DTRAccess_Disable
PWM_DeadTime	PWM_DeadTime_Reset_Mask
PWM_Emergency	PWM_Emergency_Disable
PWM_LOCKLevel	PWM_LOCKLevel_OFF
PWM_OSSIState	PWM_OSSIState_Disable
PWM_RepetitionCounter	PWM_RepetitionCounter_Reset_Mask

Example:

```
/* The following example illustrates how to initialize a PWM_InitTypeDef structure */
  PWM_InitTypeDef PWM_InitStructure;
  PWM_StructInit(&PWM_InitStructure);
```

15.2.4 PWM_Cmd

Function Name	PWM_Cmd
Function Prototype	`void PWM_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the PWM peripheral.
Input Parameter	NewState: new state of the PWM peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the PWM */
  PWM_Cmd(ENABLE);
```

15.2.5 PWM_CtrlPWMOutputs

Function Name	PWM_CtrlPWMOutputs
Function Prototype	`void PWM_CtrlPWMOutputs(FunctionalState Newstate)`
Behavior Description	Enables or disables PWM peripheral Main Outputs.
Input Parameter	NewState: new state of the PWM peripheral Main Outputs. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the PWM Main Outputs */
  PWM_CtrlPWMOutputs(ENABLE);
```

15.2.6 PWM_ITConfig

Function Name	PWM_ITConfig
Function Prototype	`void PWM_ITConfig(u16 PWM_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the PWM interrupts.
Input Parameter1	PWM_IT: specifies the PWM interrupts sources to be enabled or disabled. Refer to section "*PWM_IT*"for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the PWM interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PWM_IT

To enable or disable PWM interrupts, use a combination of one or more of the following values:

PWM_IT	Meaning
PWM_IT_OC1	PWM Output Compare 1 Interrupt source
PWM_IT_OC2	PWM Output Compare 2 Interrupt source
PWM_IT_OC3	PWM Output Compare 3 Interrupt source
PWM_IT_Update[1)]	PWM Update Interrupt source
PWM_IT_GlobalUpdate[1)]	PWM Global Update Interrupt source
PWM_IT_Emergency	PWM Emergency Interrupt source

[1)] PWM_IT_Update and PWM_GlobalUpdate interrupt sources must not be configured simultaneously

Example:

```
/* Enables the PWM Output Compare 1 Interrupt */
  PWM_ITConfig(PWM_IT_OC1, ENABLE);
```

15.2.7 PWM_DMAConfig

Function Name	PWM_DMAConfig
Function Prototype	`void PWM_DMAConfig(u16 PWM_DMASources, u16 PWM_OCRMState, u16 PWM_DMABase)`
Behavior Description	Configures the PWM's DMA interface.
Input Parameter1	PWM_DMASource: specifies the DMA Request sources. Refer to section "*PWM_DMASource*" for more details on the allowed values of this parameter.
Input Parameter2	PWM_OCRMState: the state of output compare request mode. Refer to section "*PWM_OCRMState*" for more details on the allowed values of this parameter.
Input Parameter3	PWM_DMABase:DMA Base address. Refer to section "*PWM_DMABase*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PWM_DMASource

To select the PWM DMA requests sources, use a combination of one or more of the following values:

PWM_DMASource	Meaning
PWM_DMASource_OC1	PWM Output Compare 1 DMA source
PWM_DMASource_OC2	PWM Output Compare 2 DMA source
PWM_DMASource_OC3	PWM Output Compare 3 DMA source
PWM_DMASource_Update	PWM Update DMA source.

PWM_OCRMState

To select the PWM Output compare Mode State, use one of the following values:

PWM_OCRMState	Meaning
PWM_OCRMState_Enable	OCx DMA requests are generated by the result of the comparaison between the counter and the corresponding output compare register.
PWM_OCRMState_Disable	OCx DMA requests are generated by the Update Event.

PWM_DMABase

To select the DMA base, use one of the following values:

PWM_DMABase	Meaning
PWM_DMABase_CR	CR register used as PWM DMA Base.
PWM_DMABase_SCR	SCR register used as PWM DMA Base.
PWM_DMABase_OMR1	OMR1 register used as PWM DMA Base.
PWM_DMABase_OMR2	OMR2 register used as PWM DMA Base.
PWM_DMABase_RSR	RSR register used as PWM DMA Base.
PWM_DMABase_RER	RER register used as PWM DMA Base.
PWM_DMABase_ISR	ISR register used as PWM DMA Base.
PWM_DMABase_CNT	CNT register used as PWM DMA Base.
PWM_DMABase_PSC	PSC register used as PWM DMA Base.
PWM_DMABase_RCR	RCR register used as PWM DMA Base.
PWM_DMABase_ARR	ARR register used as PWM DMA Base.
PWM_DMABase_OCR1	OCR1 register used as PWM DMA Base.
PWM_DMABase_OCR2	OCR2 register used as PWM DMA Base.
PWM_DMABase_OCR3	OCR3 register used as PWM DMA Base.
PWM_DMABase_DTR	DTR register used as PWM DMA Base.

Example:

```
/* PWM Output Compare 1 DMA request configuration */
  PWM_DMAConfig(PWM_DMASource_OC1, PWM_OCRMState_Enable, PWM_DMABase_OCR1);
```

15.2.8 PWM_DMACmd

Function Name	PWM_DMACmd
Function Prototype	`void PWM_DMACmd(u16 PWM_DMASources, FunctionalState Newstate)`
Behavior Description	Enables or disables the PWM's DMA interface.
Input Parameter1	PWM_DMASource: specifies the DMA Request sources. Refer to section "*PWM_DMASource on page 211*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the DMA Request sources. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the PWM DMA Output compare 1 request */
  PWM_DMACmd(PWM_DMASource_OC1, ENABLE);
```

15.2.9 PWM_SetPrescaler

Function Name	PWM_SetPrescaler
Function Prototype	`void PWM_SetPrescaler(u16 Prescaler)`
Behavior Description	Sets the PWM prescaler value.
Input Parameter	Prescaler: PWM prescaler new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM new Prescaler value */
  u16 PWMPrescaler = 0xFF00;
  PWM_SetPrescaler(PWMPrescaler);
```

15.2.10 PWM_SetPeriod

Function Name	PWM_SetPeriod
Function Prototype	`void PWM_SetPeriod(u16 Period)`
Behavior Description	Sets the PWM period value.
Input Parameter	Period: PWM period new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM new Period value */
  u16 PWMPeriod = 0xFFF0;
  PWM_SetPeriod(PWMPeriod);
```

15.2.11 PWM_SetPulse

Function Name	PWM_SetPulse
Function Prototype	`void PWM_SetPulse(u16 PWM_Channel, u16 Pulse)`
Behavior Description	Sets the PWM pulse value.
Input Parameter1	PWM_Channel: specifies the PWM channel to be used. Refer to *PWM_Channel on page 203* for more details on the allowed values of this parameter.
Input Parameter2	Pulse: PWM pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM new Channel 1 Pulse value */
  u16 PWMPulse = 0x00F0;
  PWM_SetPulse(PWM_Channel_1, PWMPulse);
```

15.2.12 PWM_SetPulse1

Function Name	PWM_SetPulse1
Function Prototype	`void PWM_SetPulse1(u16 Pulse)`
Behavior Description	Sets the PWM Channel 1 pulse value.
Input Parameter1	Pulse: PWM Channel 1 pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM Channel 1 Pulse value */

u16 PWMPulse = 0x3FF0;
PWM_SetPulse1(PWMPulse);
```

15.2.13 PWM_SetPulse2

Function Name	PWM_SetPulse1
Function Prototype	`void PWM_SetPulse1(u16 Pulse)`
Behavior Description	Sets the PWM Channel 1 pulse value.
Input Parameter1	Pulse: PWM Channel 1 pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Function Name	PWM_SetPulse2
Function Prototype	`void PWM_SetPulse2(u16 Pulse)`
Behavior Description	Sets the PWM Channel 2 pulse value.
Input Parameter1	Pulse: PWM Channel 2 pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM Channel 2 Pulse value */

u16 PWMPulse = 0x1000;
PWM_SetPulse2(PWMPulse);
```

15.2.14 PWM_SetPulse3

Function Name	PWM_SetPulse3
Function Prototype	`void PWM_SetPulse3(u16 Pulse)`
Behavior Description	Sets the PWM Channel 3 pulse value.
Input Parameter1	Pulse: PWM Channel 3 pulse new value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM Channel 3 Pulse value */

u16 PWMPulse = 0x0A00;
PWM_SetPulse3(PWMPulse);
```

15.2.15 PWM_DebugCmd

Function Name	PWM_DebugCmd
Function Prototype	`void PWM_DebugCmd(FunctionalState NewState)`
Behavior Description	Enables or disables PWM peripheral Debug control.
Input Parameter	NewState: new state of the PWM Debug control. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the PWM Debug control */
  PWM_DebugCmd(ENABLE);
```

15.2.16 PWM_CounterModeConfig

Function Name	PWM_CounterModeConfig
Function Prototype	`void PWM_CounterModeConfig(u16 PWM_CounterMode)`
Behavior Description	Specifies the Counter Mode to be used.
Input Parameter	PWM_CounterMode: specifies the Counter Mode to be used. Refer to section "*PWM_CounterMode on page 202*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Selects the Center Aligned counter mode1 for the PWM */
  PWM_CounterModeConfig(PWM_CounterMode_CenterAligned1);
```

15.2.17 PWM_ForcedOCConfig

Function Name	PWM_ForcedOCConfig
Function Prototype	`void PWM_ForcedOCConfig(PWM_Channel Channel,u16 PWM_ForcedAction)`
Behavior Description	Forces the PWM output waveform to active or inactive level.
Input Parameter1	PWM_Channel: specifies the PWM channel to be used. Refer to section "*PWM_Channel on page 203*" for more details on the allowed values of this parameter.
Input Parameter2	PWM_ForcedAction: specifies the forced Action to be set to the output waveform. Refer to "*PWM_ForcedAction*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PWM_ForcedAction

The forced Action that can used are listed in the following table:

PWM_ForcedAction	Meaning
PWM_ForcedAction_Active	Force active level on OCxREF.
PWM_ForcedAction_InActive	Force inactive level on OCxREF.

Example:

```
/* Forces the PWM Output Compare 1 signal to the active level */
  PWM_ForcedOCConfig(PWM_Channel_1, PWM_ForcedAction_Active);
```

15.2.18 PWM_SetDeadTime

Function Name	PWM_SetDeadTime
Function Prototype	`void PWM_SetDeadTime(u16 DeadTime)`
Behavior Description	Inserts dead time between the OCx and OCNx.
Input Parameter	DeadTime: PWM Dead Time value.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Sets the PWM new Dead Time value */
  u16 PWMDeadTime = 0x00F;
  PWM_SetDeadTime(PWMDeadTime);
```

15.2.19 PWM_ResetCounter

Function Name	PWM_ResetCounter
Function Prototype	`void PWM_ResetCounter(void)`
Behavior Description	Re-intializes the counter and generates an update of the Registers. The Prescaler counter is cleared too, however the Prescaler Preload value is not affected. The counter is cleared if center-aligned mode or up counting mode are selected, else it takes the auto-reload value.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Resets the PWM counter */
  PWM_ResetCounter();
```

15.2.20 PWM_TRGOSelection

Function Name	PWM_TRGOSelection
Function Prototype	`void PWM_TRGOSelection(u16 PWM_TRGOMode)`
Behavior Description	Sets the PWM Master Mode selection bits.
Input Parameter	PWM_TRGOMode: specifies the TRGO source. Refer to "*PWM_TRGOMode*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

PWM_TRGOMode

To select the TRGO source , use one of the following values:

PWM_TRGOMode	Meaning
PWM_TRGOMode_Enable	The CNT_EN bit is used as TRGO
PWM_TRGOMode_Update	The Update event is used as TRGO
PWM_TRGOMode_Reset	The CNT_RST bit is used as TRGO
PWM_TRGOMode_OC	The OC1 signal is used as TRGO

Example:

```
/* OC1 signal is used as TRGO */
  PWM_TRGOSelection(PWM_TRGOMode_OC);
```

15.2.21 PWM_GetFlagStatus

Function Name	PWM_GetFlagStatus
Function Prototype	`FlagStatus PWM_GetFlagStatus(u16 PWM_FLAG)`
Behavior Description	Checks whether the specified PWM flag is set or not.
Input Parameter	PWM_FLAG: specifies the flag to check. Refer to section "*PWM_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of PWM_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

PWM_FLAG

The PWM flags that can check for are listed in the following table:

PWM_FLAG	Meaning
PWM_FLAG_OC1	PWM Output Compare 1 Flag.
PWM_FLAG_OC2	PWM Output Compare 2 Flag.
PWM_FLAG_OC3	PWM Output Compare 3 Flag.
PWM_FLAG_Update	PWM Update Flag.
PWM_FLAG_Emergency	PWM Emergency Flag

Example:

```
/* Check if the PWM Update flag is set or reset: */
 if (PWM_GetFlagStatus(PWM_FLAG_Update) == SET)
  {
  }
```

15.2.22 PWM_ClearFlag

Function Name	PWM_ClearFlag
Function Prototype	`void PWM_ClearFlag(u16 PWM_FLAG)`
Behavior Description	Clears the PWM's pending flags.
Input Parameter	PWM_FLAG: specifies the flag to clear. Refer to section "*PWM_FLAG on page 221*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the PWM Output Compare 1 flag */
  PWM_ClearFlag(PWM_FLAG_OC1);
```

15.2.23 PWM_GetITStatus

Function Name	PWM_GetITStatus
Function Prototype	`ITStatus PWM_GetITStatus(u16 PWM_IT)`
Behavior Description	Checks whether the PWM interrupt has occurred or not.
Input Parameter	PWM_IT: specifies the PWM interrupt source to check. Refer to section "*PWM_IT on page 210*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of PWM_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Check if the PWM Output Compare 1 interrupt has occurred or not */
 if (PWM_GetITStatus(PWM_IT_OC1) == SET)
  {
  }
```

15.2.24 PWM_ClearITPendingBit

Function Name	PWM_ClearITPending Bit
Function Prototype	`void PWM_ClearITPendingBit(u16 PWM_IT)`
Behavior Description	Clears the PWM's interrupt pending bits.
Input Parameter	PWM_IT: specifies the interrupt pending bit to clear. Refer to section "*PWM_IT on page 210*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the PWM Output Compare 1 interrupt pending bit */
  PWM_ClearITPendingBit (PWM_IT_OC1);
```

16 Controller area network (CAN)

This peripheral performs communication according to the CAN protocol version 2.0 part A and B. The bitrate can be programmed to values up to 1Mbit/s. Another main feature of this cell is that 32 message objects are implemented which can be fully configured with 2 interfaces.

The first section describes the register structure used in the CAN software library. The second one presents the software library functions.

16.1 CAN register structures

The structures of the *CAN_TypeDef* and *CAN_MsgObj_TypeDef* registers are defined in the *75x_map.h* file as follows:

```
typedef volatile struct
{
  vu16 CRR;
  u16  EMPTY1;
  vu16 CMR;
  u16  EMPTY2;
  vu16 M1R;
  u16  EMPTY3;
  vu16 M2R;
  u16  EMPTY4;
  vu16 A1R;
  u16  EMPTY5;
  vu16 A2R;
  u16  EMPTY6;
  vu16 MCR;
  u16  EMPTY7;
  vu16 DA1R;
  u16  EMPTY8;
  vu16 DA2R;
  u16  EMPTY9;
  vu16 DB1R;
  u16  EMPTY10;
  vu16 DB2R;
  u16  EMPTY11[27];
} CAN_MsgObj_TypeDef;

typedef volatile struct
{
  vu16 CR;
  u16  EMPTY1;
  vu16 SR;
  u16  EMPTY2;
  vu16 ERR;
  u16  EMPTY3;
  vu16 BTR;
  u16  EMPTY4;
  vu16 IDR;
  u16  EMPTY5;
  vu16 TESTR;
  u16  EMPTY6;
  vu16 BRPR;
  u16  EMPTY7[3];
  CAN_MsgObj_TypeDef sMsgObj[2];
  u16  EMPTY8[16];
```

```
  vu16 TXR1R;
  u16  EMPTY9;
  vu16 TXR2R;
  u16  EMPTY10[13];
  vu16 ND1R;
  u16  EMPTY11;
  vu16 ND2R;
  u16  EMPTY12[13];
  vu16 IP1R;
  u16  EMPTY13;
  vu16 IP2R;
  u16  EMPTY14[13];
  vu16 MV1R;
  u16  EMPTY15;
  vu16 MV2R;
  u16  EMPTY16;
} CAN_TypeDef;
```

The following table presents the CAN registers:

Register	Description
CR	CAN Control Register
SR	CAN Status Register
ERR	CAN Error counter Register
BTR	CAN Bit Timing Register
IDR	CAN Interrupt Identifier Register
TESTR	CAN Test Register
BRPR	CAN BRP Extension Register
CRR	CAN IF1 Command Request Register
CMR	CAN IF1 Command Mask Register
M1R	CAN IF1 Message Mask 1 Register
M2R	CAN IF1 Message Mask 2 Register
A1R	CAN IF1 Message Arbitration 1 Register
A2R	CAN IF1 Message Arbitration 2 Register
MCR	CAN IF1 Message Control Register
DA1R	CAN IF1 DATA A 1 Register
DA2R	CAN IF1 DATA A 2 Register
DB1R	CAN IF1 DATA B 1 Register
DB2R	CAN IF1 DATA B 2 Register
CRR	CAN IF2 Command request Register
CMR	CAN IF2 Command Mask Register
M1R	CAN IF2 Message Mask 1 Register
M2R	CAN IF2 Message Mask 2 Register
A1R	CAN IF2 Message Arbitration 1 Register
A2R	CAN IF2 Message Arbitration 2 Register

Register	Description
MCR	CAN IF2 Message Control Register
DA1R	CAN IF2 DATA A 1 Register
DA2R	CAN IF2 DATA A 2 Register
DB1R	CAN IF2 DATA B 1 Register
DB2R	CAN IF2 DATA B 2 Register
TXR1R	CAN Transmission Request 1 Register
TXR2R	CAN Transmission Request 2 Register
ND1R	CAN New Data 1 Register
ND2R	CAN New Data 2 Register
IP1R	CAN Interrupt Pending 1 Register
IP2R	CAN Interrupt Pending 2 Register
MV1R	CAN Message Valid 1 Register
MV2R	CAN Message Valid 2 Register

The CAN peripheral is declared in the same file:

```
...
#define PERIPH_BASE     0xFFFF0000
...
#define CAN_BASE        (PERIPH_BASE + 0xC400)

#ifndef DEBUG
...
#define CAN ((CAN_TypeDef *) CAN_BASE)
...
#else
...
#ifdef _CAN
  EXT CAN_TypeDef           *CAN;
#endif /*_CAN */
...
#endif
```

When debug mode is used, CAN pointer is initialized in 75x_lib.c file:

```
#ifdef _CAN
  CAN = (CAN_TypeDef  *)CAN_BASE;
#endif /*CAN */
...
```

_CAN must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _CAN
...
```

16.2 Software library functions

The following table lists the various functions of the CAN library.

Function Name	Description
CAN_DeInit	Deinitializes the CAN peripheral registers to their default reset values.
CAN_Init	Initializes the CAN cell and sets the bitrate.
CAN_StructInit	Fills each CAN_InitStruct member with its default value.
CAN_EnterInitMode	Switches the CAN to initialization mode.
CAN_LeaveInitMode	Leaves initialization mode (switches to normal mode).
CAN_EnterTestMode	Switches the CAN to test mode.
CAN_LeaveTestMode	Leaves the current test mode (switches to normal mode).
CAN_SetBitrate	Sets up a standard CAN bitrate.
CAN_SetTiming	Sets up the CAN timing with specific parameters.
CAN_SetUnusedMsgObj	Configures the message object as unused.
CAN_SetTxMsgObj	Configures the message object as TX.
CAN_SetRxMsgObj	Configures the message object as RX.
CAN_InvalidateAllMsgObj	Configures all the message objects as unused.
CAN_ReleaseMessage	Releases the message object.
CAN_ReleaseTxMessage	Releases the transmit message object.
CAN_ReleaseRxMessage	Releases the receive message object.
CAN_SendMessage	Starts transmission of a message.
CAN_ReceiveMessage	Gets the message, if received.
CAN_WaitEndOfTx	Waits until current transmission is finished.
CAN_BasicSendMessage	Starts transmission of a message in BASIC mode.
CAN_BasicReceiveMessage	Gets the message in BASIC mode, if received.
CAN_IsMessageWaiting	Tests the waiting status of a received message.
CAN_IsTransmitRequested	Tests the request status of a transmitted message.
CAN_IsInterruptPending	Tests the interrupt status of a message object.
CAN_IsObjectValid	Tests the validity of a message object (ready to use).

16.2.1 CAN_DeInit

Function Name	CAN_DeInit
Function Prototype	`void CAN_DeInit(void)`
Behavior Description	Deinitializes the CAN peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	MRCC_PeripheralSWResetConfig()

Example:

This example illustrates how to initialize the *CAN* registers.

```
{
/* Initialize CAN registers*/
    CAN_DeInit ();
}
```

16.2.2 CAN_Init

Function Name	CAN_Init
Function Prototype	`void CAN_Init(CAN_InitTypeDef* CAN_InitStruct)`
Behavior Description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Input Parameter	CAN_InitStruct : pointer to a CAN_InitTypeDef structure that contains the configuration information for the CAN peripheral. Refer to section "*CAN_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	CAN_EnterInitMode() CAN_SetBitrate() CAN_LeaveInitMode() CAN_LeaveTestMode()

CAN_InitTypeDef

The CAN_InitTypeDef structure is defined in the *75x_can.h* file:

```
typedef struct
{
   u8  CAN_ConfigParameters;
   u32 CAN_Bitrate;
}CAN_InitTypeDef;
```

CAN_ConfigParameters

Specifies the CAN configuration parameters. This member can be any combination of the following values:

CAN_ConfigParameters	Meaning
CAN_CR_TEST	Test mode enable
CAN_CR_CCE	Configuration change enable
CAN_CR_DAR	Disable automatic retransmission
CAN_CR_EIE	Error interrupt enable
CAN_CR_SIE	Status change interrupt enable
CAN_CR_IE	Module interrupt enable
CAN_CR_INIT	Initialization

CAN_Bitrate

Specifies the CAN bit rate. This member can be one of the following values:

CAN_Bitrate	Meaning
CAN_BITRATE_100K	100kbit/s bit rate
CAN_BITRATE_125K	125kbit/s bit rate
CAN_BITRATE_250K	250kbit/s bit rate
CAN_BITRATE_500K	500kbit/s bit rate
CAN_BITRATE_1M	1Mbit/s bit rate

Example:

This example illustrates how to initialize the CAN at 100 kbit/s and enable the interrupts.

```
{
  /* Init Structure declarations for CAN*/
  CAN_InitTypeDef CAN_InitStructure;

  /*Configure CAN registers*/
  CAN_InitStructure.CAN_ConfigParameters = CAN_CR_IE;
  CAN_InitStructure.CAN_Bitrate = CAN_BITRATE_100K;
  CAN_Init(&CAN_InitStructure);
}
```

16.2.3 CAN_StructInit

Function Name	CAN_StructInit
Function Prototype	`void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)`
Behavior Description	Fills each CAN_InitStruct member with its default value.
Input Parameter	CAN_InitStruct: pointer to a CAN_InitTypeDef structure which will be initialized.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

This example illustrates how to initialize *CAN* init structure.

```
{
  /* Init Structures declarations for CAN */
  CAN_InitTypeDef CAN_InitStructure;

  /*Initialize CAN structure*/
  CAN_StructInit (&CAN_InitStructure);
}
```

16.2.4 CAN_EnterInitMode

Function Name	CAN_EnterInitMode
Prototype	`void CAN_EnterInitMode(u8 InitMask)`
Behavior Description	Switch the CAN into initialization mode. This function must be used in conjunction with CAN_LeaveInitMode().
Input Parameter	InitMask: specifies the CAN configuration in normal mode. Refer to section "*InitMask*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the INIT bit in the Control register, ORed with the mask and resets the Status register.

InitMask

Specifies the CAN configuration which will be set after the initialisation in normal mode. This member can be any combination of the following values:

InitMask	Meaning
CAN_CR_CCE	Configuration change enable
CAN_CR_DAR	Disable automatic retransmission
CAN_CR_EIE	Error interrupt enable
CAN_CR_SIE	Status change interrupt enable
CAN_CR_IE	Module interrupt enable

Example:

This example illustrates how to initialize the *CAN* enable interrupts.

```
{
  CAN_EnterInitMode(CAN_CR_IE);
}
```

16.2.5 CAN_LeaveInitMode

Function Name	CAN_LeaveInitMode
Prototype	`void CAN_LeaveInitMode()`
Behavior Description	Leaves initialization mode (switch into normal mode). This function must be used in conjunction with CAN_EnterInitMode().
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function clears the INIT and CCE bits in the Control register.

Example:

This example illustrates how to leave *CAN* init mode.

```
{
  CAN_LeaveInitMode();
}
```

16.2.6 CAN_EnterTestMode

Function Name	CAN_EnterTestMode
Prototype	`void CAN_EnterTestMode(u8 TestMask);`
Behavior Description	Switch the CAN into test mode. This function must be used in conjunction with CAN_LeaveTestMode().
Input Parameter	TestMask: specifies the configuration in test modes. Refer to section "*TestMask*"for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the TEST bit in the Control register to enable test mode and updates the Test register by ORing its value with the mask.

TestMask

Specifies the CAN configuration in test modes. This member can be any combination of the following values:

TestMask	Meaning
CAN_TESTR_LBACK	Loop Back mode enabled
CAN_TESTR_SILENT	Silent mode enabled
CAN_TESTR_BASIC	Basic mode enabled

Example:

This example illustrates how to switch the CAN into Loopback mode, i.e. RX is disconnected from the bus, and TX is internally linked to RX.

```
{
CAN_EnterTestMode(CAN_TESTR_LBACK);
}
```

16.2.7 CAN_LeaveTestMode

Function Name	CAN_LeaveTestMode
Prototype	`void CAN_LeaveTestMode()`
Behavior Description	Leaves the current test mode (switch into normal mode). This function must be used in conjunction with CAN_EnterTestMode().
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	None

Note: This function sets the TEST bit in the Control register to enable write access to the Test register, clears the LBACK, SILENT and BASIC bits in the Test register and clears the TEST bit in the Control register to disable write access to the Test register.

Example:

```
{
CAN_LeaveTestMode();
}
```

16.2.8 CAN_SetBitrate

Function Name	CAN_SetBitrate
Prototype	`void CAN_SetBitrate(u32 bitrate);`
Behavior Description	Setup a standard CAN bitrate.
Input Parameter	BitRate: specifies the bit rate. Refer to section "*BitRate*"for more details on the allowed values of this parameter.
Output Parameter	None
Return Value	None
Required preconditions	CAN_EnterInitMode() must have been called before. The APB clock must be 8 MHz
Called Functions	None

Note: This function writes the predefined timing value in the Bit Timing register and clears the BRPR register.

BitRate

Specifies the bit rate. This parameter can be one of the following values:

BitRate	Meaning
CAN_BITRATE_100K	100 kbit/s
CAN_BITRATE_125K	125 kbit/s
CAN_BITRATE_250K	250 kbit/s
CAN_BITRATE_500K	500 kbit/s
CAN_BITRATE_1M	1 Mbit/s

Example:

This example illustrates how to enable the configuration change bit, to be able to set the bitrate

```
{
CAN_EnterInitMode(CAN_CR_CCE);
CAN_SetBitrate(CAN_BITRATE_100K);
CAN_LeaveInitMode();
}
```

16.2.9 CAN_SetTiming

Function Name	CAN_SetTiming
Prototype	`void CAN_SetTiming(u32 tseg1, u32 tseg2, u32 sjw, u32 brp);`
Behavior Description	Setup the CAN timing with specific parameters.
Input Parameter 1	tseg1: Time Segment before the sample point position. It can take values from 1 to 16.
Input Parameter 2	tseg2: Time Segment after the sample point position. It can take values from 1 to 8.
Input Parameter 3	sjw: Synchronisation Jump Width. It can take values from 1 to 4.
Input Parameter 4	brp: Baud Rate Prescaler. It can take values from 1 to 1024.
Output Parameter	None
Return Value	None
Required preconditions	CAN_EnterInitMode() must have been called before.
Called Functions	None

Note: This function writes the timing value in the Bit Timing register, from the tseg1, tseg2, sjw parameters and bits 5..0 of brp parameter and writes the BRPR register with bits 9..0 of brp parameter; and all written values are the real values decremented by one unit.

Example:

This example illustrates how to enable the configuration change bit, to be able to set the specific timing parameters: TSEG1=11, TSEG2=4, SJW=4, BRP=5

```
{
CAN_EnterInitMode(CAN_CR_CCE);
CAN_SetTiming(11, 4, 4, 5);
CAN_LeaveInitMode();
}
```

16.2.10 CAN_SetUnusedMsgObj

Function Name	CAN_SetUnusedMsgObj
Prototype	`ErrorStatus CAN_SetUnusedMsgObj(u32 msgobj);`
Behavior Description	Configure the message object as unused.
Input Parameter	msgobj The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	An ErrorStatus enumeration value: - SUCCESS: Found Interface to treat the message - ERROR: No interface found to treat the message
Called Functions	CAN_GetFreeIF()

Note: This function searches for a free message interface from IF0 and IF1, sets the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register, clears the Mask1 and Mask2 registers, clears the Arb1 and Arb2 register, clears the Message Control register, clears the DataA1, DataA2, DataB1, DataB2 registers and writes the value 1+msgobj in the Command Request register.

Example:

This example illustrates how to invalidate the message objects from 16 to 31: these objects will not be used by the hardware

```
{
for (i=16; i<=31; i++) CAN_SetUnusedMsgObj( i);
}
```

16.2.11 CAN_SetTxMsgObj

Function Name	CAN_SetTxMsgObj
Prototype	`ErrorStatus CAN_SetTxMsgObj(u32 msgobj, u32 idType)`
Behavior Description	Configures the message object as TX.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	idType: The identifier type of the frames that will be transmitted using this message object. The value is one of the following : CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Interface to treat the message - ERROR: No interface to treat the message
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: 1 This function: Search for a free message interface from IF0 and IF1. Set the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. Clear the Mask1 and Arb1 registers. Set the MDir bit in the Mask2 register, also the MXtd bit if extended ID is used. Set the MsgVal and Dir bits in the Arb2 register, also the Xtd bit if extended ID is used. Set the TxIE and EoB bits in the Message Control register. Clear the DataA1, DataA2, DataB1, DataB2 registers. Write the value 1+msgobj to the Command Request register to copy the registers into the message RAM.

2 When defining which message object number to use for TX or RX, you must take into account the priority levels when processing the objects. The lower number (0) has the highest priority and the higher number (31) has the lowest priority, whatever their type. Also, for optimum performance, it is not recommended to have “holes” in the object list .

Example:

This example illustrates how to define transmit message object 0 with standard identifiers

```
{
CAN_SetTxMsgObj(0, CAN_STD_ID);
}
```

16.2.12 CAN_SetRxMsgObj

Function Name	CAN_SetRxMsgObj
Prototype	`ErrorStatus CAN_SetRxMsgObj(u32 msgobj, u32 idType, u32 idLow, u32 idHigh, bool singleOrFifoLast);`
Behavior Description	Configures the message object as RX.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	idType: The identifier type of the frames that will be transmitted using this message object. The value is one of the following : CAN_STD_ID (standard ID, 11-bit) CAN_EXT_ID (extended ID, 29-bit)
Input Parameter 3	idLow: The low part of the identifier range used for acceptance filtering. It can take values from 0 to 0x7FF for standard ID, and values from 0 to 0x1FFFFFFF for extended ID.
Input Parameter 4	idHigh: The high part of the identifier range used for acceptance filtering. It can take values from 0 to 0x7FF for standard ID, and values from 0 to 0x1FFFFFFF for extended ID. idHigh must be above idLow. For convenience, use one of the following values to set the maximum ID : CAN_LAST_STD_ID or CAN_LAST_EXT_ID
Input Parameter 5	singleOrFifoLast: End-of-buffer indicator, it can take the following values : -TRUE for a single receive object or a FIFO receive object that is the last one in the FIFO -FALSE for a FIFO receive object that is not the last one
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Interface to treat the message - ERROR: No interface to treat the message
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: 1 This function: Search for a free message interface from IF0 and IF1. Set the WR/RD, Mask, Arb, Control, DataA and DataB bits in the Command Mask register. Write the ID mask value formed from idLow and idHigh in the Mask1 and Mask2 registers, and also set the MXtd bit in the Mask2 register if extended ID is used. Write the ID arbitration value formed from idLow and idHigh in the Arb1 and Arb2 registers, set the MsgVal bit, and also Xtd bit in the Arb2 register if extended ID is used. Set the RxIE and UMask bits in the Message Control register, and also the EoB bit if the parameter singleOrFifoLast is TRUE. Clear the DataA1, DataA2, DataB1, DataB2 registers. Write the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

2 Care must be taken when defining an ID range : all combinations of idLow and idHigh will not always produce the expected result, because of the way identifiers are filtered by the hardware. The criteria applied to keep a received frame is as follows : (received ID) AND (ID mask) = (ID arbitration), where AND is a bitwise operator. Consequently, for idLow, it is

generally better to choose a value with some LSBs cleared, and for idHigh a value that "logically contains" idLow and with the same LSBs set. Example: the range 0x100-0x3FF will work, but the range 0x100-0x2FF will not because 0x100 is not logically contained in 0x2FF (i.e. 0x100 & 0x2FF = 0).

Example:

This example illustrates how to define FIFOs and acceptance filtering

```
{
/*Define a receive FIFO of depth 2 (objects 0 and 1) for standard identifiers, in
which IDs are filtered in the range 0x400-0x5FF*/
CAN_SetRxMsgObj( 0, CAN_STD_ID, 0x400, 0x5FF, FALSE);
CAN_SetRxMsgObj(1, CAN_STD_ID, 0x400, 0x5FF, TRUE);
/*Define a single receive object for extended identifiers, in which all IDs are
filtered in*/
CAN_SetRxMsgObj(2, CAN_EXT_ID, 0, CAN_LAST_EXT_ID, TRUE);
}
```

16.2.13 CAN_InvalidateAllMsgObj

Function Name	CAN_InvalidateAllMsgObj
Prototype	`void CAN_InvalidateAllMsgObj();`
Behavior Description	Configures all the message objects as unused.
Input Parameter	None
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	CAN_SetUnusedMsgObj()

Example:

```
{
CAN_InvalidateAllMsgObj();
}
```

16.2.14 CAN_ReleaseMessage

Function Name	CAN_ReleaseMessage
Prototype	`void CAN_ReleaseMessage(u32 msgobj);`
Behavior Description	Releases the message object.
Input Parameter	msgobj The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	None
Called Functions	CAN_GetFreeIF()

Note: This function:

Search for a free message interface from IF0 and IF1.

Set the bits ClrIntPnd and TxRqst/NewDat in the Command Mask register.

Write the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release the message object 0.

```
{
CAN_ReleaseMessage(0);
}
```

16.2.15 CAN_ReleaseTxMessage

Function Name	CAN_ReleaseTxMessage
Prototype	`void CAN_ReleaseTxMessage(u32 msgobj);`
Behavior Description	Releases the transmit message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	The message interface 0 must not be busy.
Called Functions	None

Note: This function:

Sets the ClrIntPnd and TxRqst/NewDat bits in the Command Mask register of message interface 0.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release transmit message object 0.

```
{
/*It is assumed that message interface 0 is always used for transmission*/
/*Release the transmit message object 0*/
CAN_ReleaseTxMessage(0);
}
```

16.2.16 CAN_ReleaseRxMessage

Function Name	CAN_ReleaseRxMessage
Prototype	`void CAN_ReleaseRxMessage(u32 msgobj);`
Behavior Description	Releases the receive message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	None
Required preconditions	The message interface 1 must not be busy.
Called Functions	None

Note: This function:

Sets the bits ClrIntPnd and TxRqst/NewDat in the Command Mask register of message interface 1.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM.

Example:

This example illustrates how to release the receive message object 0.

```
{
/*It is assumed that message interface 1 is always used for reception*/
/*Release the receive message object 0*/
CAN_ReleaseRxMessage(0);
}
```

16.2.17 CAN_SendMessage

Function Name	CAN_SendMessage
Prototype	`ErrorStatus CAN_SendMessage(u32 msgobj, canmsg* pCanMsg);`
Behavior Description	Starts transmission of a message.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	pCanMsg: Pointer to the canmsg structure that contains the data to transmit : ID type, ID value, data length, data values.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The message object must have been set up properly.
Called Functions	None

Note: This function:

Waits for message interface 0 to be free.

Read the Arbitration and Message Control registers.

Waits for message interface 0 to be free.

Updates the Arbitration, Message Control, DataA and DataB registers with the message contents.

Writes the value 1+msgobj to the Command Request register to copy the selected registers into the message RAM and to start the transmission.

Example:

This example illustrates how to send a standard ID data frame containing 4 data value.

```
{
canmsg CanMsg = { CAN_STD_ID, 0x111, 4, {0x10, 0x20, 0x40, 0x80} };
/*Send a standard ID data frame containing 4 data values*/
CAN_SendMessage(0, &CanMsg);
}
```

16.2.18 CAN_ReceiveMessage

Function Name	CAN_ReceiveMessage
Prototype	`ErrorStatus CAN_ReceiveMessage(u32 msgobj, bool release, canmsg* pCanMsg);`
Behavior Description	Gets the message, if received.
Input Parameter 1	msgobj: The message object number, from 0 to 31.
Input Parameter 2	Release: The message release indicator, it can take the following values : -TRUE : the message object is released at the same time as it is copied from message RAM, then it is free for next reception -FALSE : the message object is not released, it is to the caller to do it
Input Parameter 3	pCanMsg: Pointer to the canmsg structure where the received message is copied.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The message object must have been set up properly.
Called Functions	macro CAN_IsMessageWaiting()

Note: This function:

Test the bit corresponding to the message object number in the NewData registers.

Clear the bit RxOk in the Status register.

Copy the message contents from the message RAM to the registers and to the structure, and release the message object if asked.

Example:

This example illustrates how to receive a message.

```
{
  canmsg CanMsg;
/*Receive a message in the object 0 and ask for release*/
  if (CAN_ReceiveMessage(CAN0, 0, TRUE, &CanMsg))
  {
    /*Check or copy the message contents*/
  }
  else
  {
    /* Error handling*/
  }
}
```

16.2.19 CAN_WaitEndOfTx

Function Name	CAN_WaitEndOfTx
Prototype	`ErrorStatus CAN_WaitEndOfTx(void);`
Behavior	Tests the TxOk bit in the Status register, and loops until it is set. Clears this bit to prepare the next transmission.
Input Parameter	None
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission ended - ERROR: Transmission did not occur yet
Required preconditions	A message must have been sent before.
Called Functions	None

Example:

This example illustrates how to send frames.

```
{
/*Send consecutive data frames using message object 0*/
for (i = 0; i < 10; i++)
{
  CAN_SendMessage(0, CanMsgTable[i]);
  CAN_WaitEndOfTx();
}
}
```

16.2.20 CAN_BasicSendMessage

Function Name	CAN_BasicSendMessage
Prototype	`ErrorStatus CAN_BasicSendMessage(canmsg* pCanMsg);`
Behavior Description	Starts transmission of a message in BASIC mode. This mode does not use the message RAM.
Input Parameter	pCanMsg: Pointer to the canmsg structure that contains the data to transmit : ID type, ID value, data length, data values.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Transmission OK - ERROR: No transmission
Required preconditions	The CAN must have been switched into BASIC mode.
Called Functions	None

Note: This function:

Clears the bit NewDat in message interface 1.

Writes the Arbitration, Message Control, DataA and DataB registers of message interface 0, with the message contents.

Writes the value 1+msgobj to the Command Request register to start the transmission.

Example:

This example illustrates how to send frames.

```
{
/*Send consecutive data frames using message object 0*/
for (i = 0; i < 10; i++)
{
  CAN_SendMessage(0, CanMsgTable[i]);
  CAN_WaitEndOfTx();
}
}
```

16.2.21 CAN_BasicReceiveMessage

Function Name	CAN_BasicReceiveMessage
Prototype	`ErrorStatus CAN_BasicReceiveMessage(canmsg* pCanMsg);`
Behavior Description	Gets the message in BASIC mode, if received. This mode does not use the message RAM.
Input Parameter	pCanMsg: Pointer to the canmsg structure where the received message is copied.
Output Parameter	None
Return Value	An ErrorStatus enumeration value: - SUCCESS: Reception OK - ERROR: No message pending
Required preconditions	The CAN must have been switched into BASIC mode.
Called Functions	None

Note: This function:

Tests the bit NewDat in the Message Control register of message interface 1.

Clears the bit RxOk in the Status register.

Copies the message contents from the message interface 1 registers to the structure.

Example:

This example illustrates how to receive frame in basic mode.

```
{
 canmsg CanMsg;
/*Receive a message in BASIC mode*/
if (CAN_BasicReceiveMessage(&CanMsg))
{
  /* Check or copy the message contents*/
}
else
{
  /* Error handling*/
}
}
```

16.2.22 CAN_IsMessageWaiting

Function Name	CAN_IsMessageWaiting
Prototype	`u32 CAN_IsMessageWaiting(u32 msgobj);`
Behavior Description	Tests the waiting status of a received message.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message object has received a message waiting to be copied, else 0.
Required preconditions	The corresponding message object must have been set as RX.
Called Functions	None

Note: This function:

Tests the corresponding bit in the NewData 1 or 2 registers.

Example:

This example illustrates how to test the new data registers for the message oject 0.

```
{
/*Test if a message is pending in the receive message object 0*/
if (CAN_IsMessageWaiting(0))
{
  /* Receive the message from this message object (i.e. get its data from message
RAM)*/
}
}
```

16.2.23 CAN_IsTransmitRequested

Function Name	CAN_IsTransmitRequested
Prototype	`u32 CAN_IsTransmitRequested(u32 msgobj);`
Behavior Description	Tests the request status of a transmitted message.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message is requested to transmit, else 0.
Required preconditions	A message must have been sent before.
Called Functions	None

Note: This function:

Tests the corresponding bit in the Transmission Request 1 or 2 registers.

Example:

This example illustrates how to test the transmit request.

```
{
/*Send a message using object 0*/
CAN_SendMessage(0, &CanMsg);
/*Wait for the end of transmit request*/
while (CAN_IsTransmitRequested(0));
/*Now, the message is being processed by the priority handler of the CAN cell, and
ready to be emitted on the bus*/
}
```

16.2.24 CAN_IsInterruptPending

Function Name	CAN_IsInterruptPending
Prototype	`u32 CAN_IsInterruptPending(u32 msgobj);`
Behavior Description	Tests the interrupt status of a message object.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message has an interrupt pending, else 0.
Required preconditions	The interrupts must have been enabled.
Called Functions	None

Note: This function:

Tests the corresponding bit in the Interrupt Pending 1 or 2 registers.

Example:

This example illustrates how to test interrupt pending.

```
{
/*Send a message using object 0*/
CAN_SendMessage(0, &CanMsg)
/* Wait for the TX interrupt*/
while (!CAN_IsInterruptPending(0));
}
```

16.2.25 CAN_IsObjectValid

Function Name	CAN_IsObjectValid
Prototype	`u32 CAN_IsObjectValid(u32 msgobj);`
Behavior Description	Tests the validity of a message object (ready to use). A valid object means that it has been set up either as TX or as RX, and so is used by the hardware.
Input Parameter	msgobj: The message object number, from 0 to 31.
Output Parameter	None
Return Value	A non-zero value if the corresponding message object is valid, else 0.
Required preconditions	None
Called Functions	None

Note: This function:

Tests the corresponding bit in the Message Valid 1 or 2 registers.

Example:

This example illustrates how to test the validity of message object 10.

```
{
if (CAN_IsObjectValid(10))
{
  /* Do something with message object 10*/
}
}
```

17 I2C Interface Module (I2C)

The I2C Bus interface module serves as interface between the microcontroller and the serial I2C bus. It provides both multi-master and slave functions, and controls all I2C bus-specific sequencing, protocol, arbitration and timing.

The I2C driver may be used to manage the I2C functionality in transmission and reception and it reports the status of the action done.

The first section describes the register structure used in the I2C software library. The second one presents the software library functions.

17.1 I2C register structure

The I2C register structure *I2C_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu8 CR;
  u8  EMPTY1[3];
  vu8 SR1;
  u8  EMPTY2[3];
  vu8 SR2;
  u8  EMPTY3[3];
  vu8 CCR;
  u8  EMPTY4[3];
  vu8 OAR1;
  u8  EMPTY5[3];
  vu8 OAR2;
  u8  EMPTY6[3];
  vu8 DR;
  u8  EMPTY7[3];
  vu8 ECCR;
  u8  EMPTY8[3];
} I2C_TypeDef;
```

The following table presents the I2C registers:

Register	Description
CR	*I2C* Control Register
SR1	*I2C* Status Register1
SR2	*I2C* Status Register2
CCR	*I2C* Clock Control Register
ECCR	*I2C* Extended Clock Control Register
OAR1	*I2C* Own Address Register1
OAR2	*I2C* Own Address Register1
DR	*I2C* Data Register

The I2C peripheral is declared in the same file:

```
...
#define PERIPH_BASE  0xFFFF0000
...
#define I2C_BASE      (PERIPH_BASE + 0xCC00)

#ifndef DEBUG
...
#define I2C ((I2C_TypeDef *) I2C_BASE)
...
#else
...
#ifdef _I2C
  EXT I2C_TypeDef           *I2C;
#endif /*_I2C */
...
#endif
```

When debug mode is used, I2C pointer is initialized in 75x_lib.c file:

```
#ifdef _I2C
I2C = (I2C_TypeDef *)I2C_BASE;
#endif /*_I2C */
...
```

_I2C must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _I2C
...
```

17.2 Software library functions

The following table lists the various functions of the I2C library.

Function Name	Description
I2C_DeInit	Deinitializes the I2C peripheral registers to their default reset values.
I2C_Init	Initializes the I2C peripheral according to the specified parameters in the I2C_InitStruct.
I2C_StructInit	Fills each I2C_InitStruct member with its default value.
I2C_Cmd	Enables or disables the I2C peripheral.
I2C_GenerateSTART	Generates I2C communication START condition.
I2C_GenerateSTOP	Generates I2C communication STOP condition.
I2C_AcknowledgeConfig	Enables or disable I2C acknowledge feature.
I2C_ITConfig	Enables or disables the I2C interrupt.
I2C_GetLastEvent	Gets the last I2C event that has occurred.
I2C_CheckEvent	Checks whether the last I2C event is equal to the one passed as parameter.
I2C_SendData	Sends a data byte.
I2C_ReceiveData	Reads the received byte.
I2C_Send7bitAddress	Transmits the address byte to select the slave device.
I2C_ReadRegister	Reads the specified I2C register and returns its value.
I2C_GetFlagStatus	Checks whether the specified I2C flag is set or not.
I2C_ClearFlag	Clears the I2C's pending flags.

17.2.1 I2C_DeInit

Function Name	I2C_DeInit
Function Prototype	`void I2C_DeInit(void)`
Behavior Description	Deinitializes the I2C peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitialize the I2C peripheral */
  I2C_DeInit();
```

17.2.2 I2C_Init

Function Name	I2C_Init
Function Prototype	`void I2C_Init(I2C_InitTypeDef* I2C_InitStruct)`
Behavior Description	Initializes the I2C peripheral according to the specified parameters in the I2C_InitStruct .
Input Parameter	I2C_InitStruct: pointer to a I2C_InitTypeDef structure that contains the configuration information for the specified I2C peripheral. Refer to section "*I2C_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

I2C_InitTypeDef

The I2C_InitTypeDef structure is defined in the *75x_i2c.h* file:

```
typedef struct
{
  u32 I2C_CLKSpeed;
  u16 I2C_OwnAddress;
  u8 I2C_GeneralCall;
  u8 I2C_Ack;
} I2C_InitTypeDef;
```

I2C_CLKSpeed

Select the clok speed frequency. Allowed values are under 400000Hz.

I2C_OwnAddress

Select the device own address. It can be a 7-bit or 10-bit address.

I2C_GeneralCall

Enables/disables the General call feature. This member can be one of the following values:

I2C_GeneralCall	Meaning
I2C_GeneralCall_Enable	Enable the General call feature
I2C_GeneralCall_Disable	Disable the General call feature

I2C_Ack

Enables or disables the Acknowledgement. This member can be one of the following values:

I2C_Ack	Meaning
I2C_Ack_Enable	Enable the Acknowledgement
I2C_Ack_Disable	Disable the Acknowledgement

Example:

```
/* Initialize the I2C peripheral according to the I2C_InitStructure members */
  I2C_InitTypeDef I2C_InitStructure;

  I2C_InitStructure.I2C_OwnAddress = 0xA8;
  I2C_InitStructure.I2C_CLKSpeed = 100000;
  I2C_InitStructure.I2C_GeneralCall = I2C_GeneralCall_Disable ;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_Init(&I2C_InitStructure);
```

17.2.3 I2C_StructInit

Function Name	I2C_StructInit
Function Prototype	`void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)`
Behavior Description	Fills each I2C_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	I2C_InitStruct: pointer to a I2C_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The I2C_InitStruct members have the following default values:

Member	Default value
I2C_CLKSpeed	5000
I2C_OwnAddress	0
I2C_GeneralCall	I2C_GeneralCall_Disable
I2C_Ack	I2C_Ack_Disable

Example:

```
/* Initialize a I2C_InitTypeDef structure */
  I2C_InitTypeDef I2C_InitStructure;
  I2C_StructInit(&I2C_InitStructure);
```

17.2.4 I2C_Cmd

Function Name	I2C_Cmd
Function Prototype	`void I2C_Cmd(FunctionalState NewState)`
Behavior Description	Enables or disables the I2C peripheral.
Input Parameter	NewState: new state of the I2C peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable I2C peripheral */
  I2C_Cmd(ENABLE);
```

17.2.5 I2C_GenerateSTART

Function Name	I2C_GenerateSTART
Function Prototype	`void I2C_GenerateSTART(FunctionalState NewState)`
Behavior Description	Generates I2C communication START condition.
Input Parameter	NewState: new state of the I2C START condition generation. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a START condition */
  I2C_GenerateSTART(ENABLE);
```

17.2.6 I2C_GenerateSTOP

Function Name	I2C_GenerateSTOP
Function Prototype	`void I2C_GenerateSTOP(FunctionalState NewState)`
Behavior Description	Generates I2C communication STOP condition.
Input Parameter	NewState: new state of the I2C STOP condition generation. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Generate a STOP condition */
  I2C_GenerateSTOP(ENABLE);
```

17.2.7 I2C_AcknowledgeConfig

Function Name	I2C_AcknowledgeConfig
Function Prototype	`void I2C_AcknowledgeConfig(FunctionalState NewState)`
Behavior Description	Enables or disables I2C acknowledge feature.
Input Parameter	NewState: new state of the I2C Acknowledgement. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable Acknowledgement */
  I2C_AcknowledgeConfig(ENABLE);
```

17.2.8 I2C_ITConfig

Function Name	I2C_ITConfig
Function Prototype	`void I2C_ITConfig(FunctionalState NewState)`
Behavior Description	Enables or disables the I2C interrupt.
Input Parameter	NewState: new state of the I2C interrupt. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Disable I2C interrupt */
  I2C_ITConfig(DISABLE);
```

17.2.9 I2C_GetLastEvent

Function Name	I2C_GetLastEvent
Function Prototype	`u16 I2C_GetLastEvent(void)`
Behavior Description	Gets the last I2C event that has occurred.
Input Parameter	None
Output Parameter	None
Return Parameter	The last event that has occurred.
Required preconditions	None
Called functions	None

Example:

```
/* Get the I2C last happened event */
  u16 hLastEvent;
  hLastEvent = I2C_GetLastEvent();
```

17.2.10 I2C_CheckEvent

Function Name	I2C_CheckEvent
Function Prototype	`ErrorStatus I2C_CheckEvent(u16 I2C_EVENT)`
Behavior Description	Checks whether the last I2C event is equal to the one passed as parameter.
Input Parameter	I2C_EVENT: specifies the event to be checked. Refer to section "*I2C_EVENT*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	An ErrorStatus enumeration value: SUCCESS: Last event is equal to the I2C_Event ERROR: Last event is different from the I2C_Event
Required preconditions	None
Called functions	I2C_GetLastEvent().

I2C_EVENT

The I2C events that can check for are listed in the following table:

I2C_EVENT	Meaning
I2C_EVENT_SLAVE_ADDRESS_MATCHED	EV1.
I2C_EVENT_SLAVE_BYTE_RECEIVED	EV2.
I2C_EVENT_SLAVE_BYTE_TRANSMITTED	EV3.
I2C_EVENT_SLAVE_ACK_FAILURE	EV4.
I2C_EVENT_MASTER_MODE_SELECT	EV5.
I2C_EVENT_MASTER_MODE_SELECTED	EV6.
I2C_EVENT_MASTER_BYTE_RECEIVED	EV7.
I2C_EVENT_MASTER_BYTE_TRANSMITTED	EV8.
I2C_EVENT_MASTER_MODE_ADDRESS10	EV9.
I2C_EVENT_SLAVE_STOP_DETECTED	EV3-1.

Example:

```
/* Check if the event is equal to I2C_EVENT_MASTER_BYTE_RECEIVED */
  ErrorStatus  Status;
  Status = I2C_CheckEvent(I2C_EVENT_MSTER_BYTE_RECEIVED);
```

17.2.11 I2C_SendData

Function Name	I2C_SendData
Function Prototype	`void I2C_SendData(u8 Data)`
Behavior Description	Sends a data byte.
Input Parameter	Data: indicates the byte to be transmitted.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Transmit 0x5D byte */
  I2C_SendData(0x5D);
```

17.2.12 I2C_ReceiveData

Function Name	I2C_ReceiveData
Function Prototype	`u8 I2C_ReceiveData(void)`
Behavior Description	Reads the received byte.
Input Parameter	None
Output Parameter	None
Return Parameter	The received byte .
Required preconditions	None
Called functions	None

Example:

```
/* Read the received byte */
  u8 bReceivedData;
  bReceivedData = I2C_ReceiveData();
```

17.2.13 I2C_Send7bitAddress

Function Name	I2C_Send7bitAddress
Function Prototype	`void I2C_Send7bitAddress(u8 Address, u8 Direction)`
Behavior Description	Transmits the address byte to select the slave device.
Input Parameter1	Address: specifies the slave address which will be transmitted.
Input Parameter2	Direction: specifies whether the I2C device will be a Transmitter or a Receiver. This parameter could be: I2C_MODE_TRANSMITTER: Transmitter mode. I2C_MODE_RECEIVER: Receiver mode.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send, as transmitter, the Slave device address 0xA8 in 7-bit addressing mode */
  I2C_Send7bitAddress(0xA8, I2C_MODE_TRANSMITTER);
```

17.2.14 I2C_ReadRegister

Function Name	I2C_ReadRegister
Function Prototype	`u8 I2C_ReadRegister(u8 I2C_Register)`
Behavior Description	Reads the specified I2C register and returns its value.
Input Parameter	I2C_Register: specifies the register to read. Refer to section "*I2C_Register*" for more details on the allowed values of this parameter.
Output Parameter	Some flags could be cleared when the register is read.
Return Parameter	The value of the read register.
Required preconditions	None
Called functions	None

I2C_Register

The I2C registers that can be read are listed in the following table:

I2C_Register	Meaning
I2C_CR	I2C_CR selected for read.
I2C_SR1	I2C_SR1 selected for read.
I2C_SR2	I2C_SR2 selected for read.
I2C_CCR	I2C_CCR selected for read.
I2C_OAR1	I2C_OAR1 selected for read.
I2C_OAR2	I2C_OAR2 selected for read.
I2C_DR	I2C_DR selected for read.
I2C_ECCR	I2C_ECCR selected for read.

Example:

```
/* Return the I2C_CR register value */
  u8 bRegisterValue;
  bRegisterValue = I2C_ReadRegister (I2C_CR);
```

17.2.15 I2C_GetFlagStatus

Function Name	I2C_GetFlagStatus
Function Prototype	`FlagStatus I2C_GetFlagStatus(u16 I2C_FLAG)`
Behavior Description	Checks whether the specified I2C flag is set or not.
Input Parameter	I2C_FLAG: specifies the flag to check. Refer to section "*I2C_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	Some flags could be cleared when the register is read.
Return Parameter	The new state of I2C_FLAG (SET or RESET).
Required preconditions	None
Called functions	None.

I2C_FLAG

The I2C flags that can check for are listed in the following table:

I2C_FLAG	Meaning
I2C_FLAG_SB	Start bit flag (Master mode)
I2C_FLAG_M_SL	Master/Slave flag
I2C_FLAG_ADSL	Address matched flag (Slave mode)
I2C_FLAG_BTF	Byte transfer finished flag
I2C_FLAG_BUSY	Bus busy flag
I2C_FLAG_TRA	Transmitter/Receiver flag
I2C_FLAG_ADD10	10-bit addressing in master mode flag
I2C_FLAG_EVF	Event flag
I2C_FLAG_GCAL	General call flag (slave mode)
I2C_FLAG_BERR	Bus error flag
I2C_FLAG_ARLO	Arbitration lost flag
I2C_FLAG_STOPF	Stop detection flag (slave mode)
I2C_FLAG_AF	Acknowledge failure flag
I2C_FLAG_ENDAD	End of address transmission flag
I2C_FLAG_ACK	Acknowledge enabled flag

Example:

```
/* Return the I2C_FLAG_AF flag state */
  Flagstatus Status;
  Status = I2C_GetFlagStatus(I2C_FLAG_AF);
```

17.2.16 I2C_ClearFlag

Function Name	I2C_ClearFlag
Function Prototype	`void I2C_ClearFlag(u16 I2C_FLAG, ...)`
Behavior Description	Clears the I2C's pending flags.
Input Parameter1	I2C_FLAG: specifies the flag to clear. Refer to section "*I2C_FLAG on page 265*" for more details on the allowed values of this parameter.
Input Parameter2	Parameter needed in the case that the flag to be cleared need a write in one register
Output Parameter	Some flags could be cleared when the register is read.
Return Parameter	None
Required preconditions	None
Called functions	I2C_Cmd

Example:

```
/* Clear the Stop detection flag*/
  I2C_ClearFlag(I2C_FLAG_STOPF);
```

18 Synchronous Serial Peripheral (SSP)

The SSP is a master or slave interface for synchronous serial communication with peripheral devices that have either Motorola SPI or Texas Instruments SSI synchronous serial interfaces.

The first section describes the register structure used in the SSP software library. The second one presents the software library functions.

18.1 SSP register structure

The SSP register structure *SSP_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu32 CR0;
  vu32 CR1;
  vu32 DR;
  vu32 SR;
  vu32 PR;
  vu32 IMSCR;
  vu32 RISR;
  vu32 MISR;
  vu32 ICR;
  vu32 DMACR;
} SSP_TypeDef;
```

The following table presents the SSP registers:

Register	Description
CR0	SSP Control Logic register 0
CR1	SSP Control Logic register 1
DR	SSP Data register
SR	SSP Status register
PR	SSP Clock Prescaler register
IMSCR	SSP Interrupt Mask Set and Clear register
RISR	SSP Raw Interrupt Status register
MISR	SSP Masked Interrupt Status register
ICR	SSP Interrupt Status register
DMACR	SSP DMA Control register

The two SSP peripherals are declared in the same file:

```
...
#define PERIPH_BASE    0xFFFF0000
...
#define SSP0_BASE      (PERIPH_BASE + 0xB800)
#define SSP1_BASE      (PERIPH_BASE + 0xBC00)

#ifndef DEBUG
...
#define SSP0         ((SSP_TypeDef *) SSP0_BASE)
#define SSP1         ((SSP_TypeDef *) SSP1_BASE)
...
#else
...
#ifdef _SSP0
  EXT SSP_TypeDef           *SSP0;
#endif /*_SSP0 */

#ifdef _SSP1
  EXT SSP_TypeDef           *SSP1;
#endif /*_SSP1 */
...
#endif
```

When debug mode is used, the SSP pointer is initialized in the 75x_lib.c file:

```
#ifdef _SSP0
  SSP0 = (SSP_TypeDef *)SSP0_BASE;
#endif /*_SSP0 */

#ifdef _SSP1
  SSP1 = (SSP_TypeDef *)SSP1_BASE;
#endif /*_SSP1 */
```

_SSP, _SSP0 and _SSP1 must be defined, in the *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _SSP
#define _SSP0
#define _SSP1
```

18.2 Software library functions

The following table lists the various functions of the SSP library.

Function Name	Description
SSP_DeInit	Deinitializes the SSPx peripheral registers to their default reset values.
SSP_Init	Initializes the SSPx peripheral according to the specified parameters in the SSP_InitStruct.
SSP_StructInit	Fills each SSP_InitStruct member with its default value.
SSP_Cmd	Enables or disables the specified SSP peripheral.
SSP_ITConfig	Enables or disables the specified SSP interrupts.
SSP_DMACmd	Configures the SSP0 DMA interface.
SSP_DMATxConfig	Configures the SSP0 DMA transmit transfer.
SSP_DMARxConfig	Configures the SSP0 DMA receive transfer.
SSP_SendData	Transmits a Data through the SSP peripheral.
SSP_ReceiveData	Returns the most recent received data by the SSP peripheral.
SSP_LoopBackConfig	Enables or disables the Loop back mode for the selected SSP peripheral.
SSP_NSSInternalConfig	Configures by software the NSS pin.
SSP_GetFlagStatus	Checks whether the specified SSP flag is set or not.
SSP_ClearFlag	Clears the SSPx interrupt pending bits.
SSP_GetITStatus	Checks whether the specified SSP interrupt has occurred or not.
SSP_ClearITPendingBit	Clears the SSPx's interrupt pending bits.

18.2.1 SSP_DeInit

Function Name	SSP_DeInit
Function Prototype	`void SSP_DeInit(SSP_TypeDef* SSPx)`
Behavior Description	Deinitializes the SSPx peripheral registers to their default reset values.
Input Parameter	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitialize the SSP0 peripheral. */
SSP_DeInit(SSP0);
```

18.2.2 SSP_Init

Function Name	SSP_Init
Function Prototype	`void SSP_Init(SSP_TypeDef* SSPx, SSP_InitTypeDef* SSP_InitStruct)`
Behavior Description	Initializes the SSPx peripheral according to the specified parameters in the SSP_InitStruct .
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_InitStruct: pointer to a SSP_InitTypeDef structure that contains the configuration information for the specified SSP peripheral. Refer to section "*SSP_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP_InitTypeDef

The SSP_InitTypeDef structure is defined in the *75x_SSP.h* file:

```
typedef struct
{
u16 SSP_FrameFormat ;
u16 SSP_Mode ;
u16 SSP_CPOL ;
u16 SSP_CPHA ;
u16 SSP_DataSize ;
u16 SSP_NSS ;
u16 SSP_SlaveOutput ;
u8  SSP_ClockRate ;
u8  SSP_ClockPrescaler ;
} SSP_InitTypeDef;
```

SSP_FrameFormat

Specifies whether the frame format is Motorola SPI or TI synchronous serial. This member can be one of the following values:

SSP_FrameFormat	Meaning
SSP_FrameFormat_Motorola	Motorola frame format is selected
SSP_FrameFormat_TI	TI frame format is selected

SSP_Mode

Specifies the SSP operation mode . This member can be one of the following values:

SSP_Mode	Meaning
SSP_Mode_Master	SSP is configured as a master
SSP_Mode_Slave	SSP is configured as a slave

SSP_CPOL

Specifies the steady state value of the serial clock. This member can be one of the following values:

SSP_CPOL	Meaning
SSP_CPOL_Low	Clock is active low
SSP_CPOL_High	Clock is active high

SSP_CPHA

Specifies on which clock transition the bit capture is made. This member can be one of the following values:

SSP_CPHA	Meaning
SSP_CPHA_2Edge	Data is captured on the second edge
SSP_CPHA_1Edge	Data is captured on the first edge

SSP_DataSize

Specifies the word length operation of the receive and transmit FIFO. This member can be one of the following values:

SSP_DataSize	Meaning
SSP_DataSize_16b	Data Size is 16 bits
SSP_DataSize_15b	Data Size is 15 bits
SSP_DataSize_14b	Data Size is 14 bits
SSP_DataSize_13b	Data Size is 13 bits
SSP_DataSize_12b	Data Size is 12 bits
SSP_DataSize_11b	Data Size is 11 bits
SSP_DataSize_10b	Data Size is 10 bits
SSP_DataSize_9b	Data Size is 9 bits
SSP_DataSize_8b	Data Size is 8 bits
SSP_DataSize_7b	Data Size is 7 bits
SSP_DataSize_6b	Data Size is 6 bits
SSP_DataSize_5b	Data Size is 5 bits
SSP_DataSize_4b	Data Size is 4 bits

SSP_NSS

Specifies whether the NSS is managed by hardware (SS pin) or by software using the SSI bit. This member can be one of the following values:

SSP_NSS	Meaning
SSP_NSS_Hard	SS pin managed by external pin
SSP_NSS_Soft	Internal SS signal controlled by SSI bit

SSP_SlaveOutput

Specifies whether the slave output is enabled or disabled. This is used specially in cases where there are multiple slaves. This member can be one of the following values:

SSP_SlaveOutput	Meaning
SSP_SlaveOutput_Enable	Slave output enabled
SSP_SlaveOutput_Disable	Slave output disabled

SSP_ClockRate

Specifies the serial clock rate value which will be used to configure the transmit and receive bit rate of SCK. This member must be a value from 0 to 255.

SSP_ClockPrescaler

Specifies the division factor by which the input PCLK must be divided to be used by the SSP. This member must be an even number between 2 and 254. The least significant bit of the programmed number is hardcoded to zero. If an odd number is written the least significant bit is changed to zero by hardware.

Used SSP_InitTypeDef structure members for each SSP mode:

	MOTOROLA Mode		TI Mode	
Members	**Master**	**Slave**	**Master**	**Slave**
SSP_FrameFormat	x	x	x	x
SSP_Mode	x	x	x	x
SSP_CPOL	x	x		
SSP_CPHA	x	x		
SSP_DataSize	x	x	x	x
SSP_NSS	x	x	x	x
SSP_SlaveOutput		x		x
SSP_ClockRate	x	(1)	x	x
SSP_ClockPrescaler	x	(1)	x	x

(1): the communication clock is derived from the master so no need to set the slave clock

Example:

```
/* Initialize the SSP0 according to the SSP_InitStructure members. */
SSP_InitTypeDef  SSP_InitStructure;

SSP_InitStructure.SSP_FrameFormat = SSP_FrameFormat_TI;
SSP_InitStructure.SSP_Mode = SSP_Mode_Slave;
SSP_InitStructure.SSP_CPOL = SSP_CPOL_High;
SSP_InitStructure.SSP_CPHA = SSP_CPHA_1Edge;
SSP_InitStructure.SSP_DataSize = SSP_DataSize_8b;
SSP_InitStructure.SSP_NSS = SSP_NSS_Hard;
SSP_InitStructure.SSP_SlaveOutput = SSP_SlaveOutput_Enable;
SSP_InitStructure.SSP_ClockRate = 0xB;
SSP_InitStructure.SSP_ClockPrescaler = 12;
SSP_Init(SSP0, &SSP_InitStructure);
```

18.2.3 SSP_StructInit

Function Name	SSP_StructInit
Function Prototype	`void SSP_StructInit(SSP_InitTypeDef* SSP_InitStruct)`
Behavior Description	Fills each SSP_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	SSP_InitStruct: pointer to a SSP_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The SSP_InitStruct members have the following default values:

Member	Default value
SSP_FrameFormat	SSP_FrameFormat_Motorola
SSP_Mode	SSP_Mode_Master
SSP_CPOL	SSP_CPOL_Low
SSP_CPHA	SSP_CPHA_1Edge
SSP_DataSize	SSP_DataSize_8b
SSP_NSS	SSP_NSS_Hard
SSP_SlaveOutput	SSP_SlaveOutput_Enable
SSP_ClockRate	0
SSP_ClockPrescaler	0

Example:

```
/* Initialize a SSP_InitTypeDef structure. */
SSP_InitTypeDef  SSP_InitStructure;
SSP_StructInit(&SSP_InitStructure);
```

18.2.4 SSP_Cmd

Function Name	SSP_Cmd
Function Prototype	`void SSP_Cmd(SSP_TypeDef* SSPx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	NewState: new state of the SSPx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the SSP0 peripheral */
SSP_Cmd(SSP0, ENABLE);
```

18.2.5 SSP_ITConfig

Function Name	SSP_ITConfig
Function Prototype	`void SSP_ITConfig(SSP_TypeDef* SSPx, u16 SSP_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified SSP interrupts.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_IT: specifies the SSP interrupts sources to be enabled or disabled. Refer to section "*SSP_IT*" for more details on the allowed values of this parameter. The user can select more than one interrupt, by ORing them.
Input Parameter3	NewState: new state of the specified SSP interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP_IT

To enable or disable SSP interrupts, use a combination of one or more of the following values:

SSP_IT	Meaning
SSP_IT_TxFifo	Transmit FIFO half empty or less condition interrupt
SSP_IT_RxFifo	Receive FIFO half full or more condition interrupt
SSP_IT_RxTimeOut	Receive timeout interrupt
SSP_IT_RxOverrun	Receive overrun interrupt

Example:

```
/* Enable SSP0 Transmit FIFO and Receive FIFO interrupts. */
SSP_ITConfig(SSP0, SSP_IT_TxFifo | SSP_IT_RxFifo, ENABLE);
```

18.2.6 SSP_DMACmd

Function Name	SSP_DMACmd
Function Prototype	`void SSP_DMACmd(u16 SSP0_DMATransfer, FunctionalState NewState)`
Behavior Description	Configures the SSP0 DMA interface.
Input Parameter1	SSP0_DMATransfer: specifies the SSP0 DMA transfer to be enabled or disabled. Refer to section "*SSP0_DMATransfer*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of SSP0 DMA transfer. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP0_DMATransfer

To enable or disable SSP0 DMA transfer, use one of the following values:

SSP0_DMATransfer	Meaning
SSP0_DMA_Transmit	Transmit FIFO DMA transfer
SSP0_DMA_Receive	Receive FIFO DMA transfer

Example:

```
/* Enable SSP0 transmit FIFO DMA transfer. */
SSP_DMACmd(SSP0_DMA_Transmit, ENABLE);
```

18.2.7 SSP_DMATxConfig

Function Name	SSP_DMATxConfig
Function Prototype	`void SSP_DMATxConfig(u16 SSP0_DMATxReq)`
Behavior Description	Configures the SSP0 DMA transmit transfer.
Input Parameter1	SSP0_DMATxReq: specifies the SSP0 DMA request to be enabled. Refer to section "*SSP0_DMATxReq*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP0_DMATxReq

To select the SSP0 DMA transmit transfer request, use one of the following values:

SSP_DMATxReq	Meaning
SSP0_DMATxReq_Burst	Transmit FIFO DMA burst request enabled
SSP0_DMATxReq_Single	Transmit FIFO DMA single request enabled

Example:

```
/* Enable SSP0 transmit DMA single request. */
SSP_DMATxConfig(SSP0_DMATxReq_Single);
```

18.2.8 SSP_DMARxConfig

Function Name	SSP_DMARxConfig
Function Prototype	`void SSP_DMARxConfig(u16 SSP0_DMARxReq)`
Behavior Description	Configures the SSP0 DMA receive transfer.
Input Parameter1	SSP0_DMARxReq: specifies the SSP0 DMA request to be enabled. Refer to section "*SSP0_DMARxReq*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

SSP0_DMARxReq

To select the SSP0 DMA receive transfer request, use one of the following values:

SSP_DMARxReq	Meaning
SSP0_DMARxReq_Burst	Receive FIFO DMA burst request enabled
SSP0_DMARxReq_Single	Receive FIFO DMA single request enabled

Example:

```
/* Enable SSP0 receive DMA single request. */
SSP_DMATxConfig(SSP0_DMARxReq_Single);
```

18.2.9 SSP_SendData

Function Name	SSP_SendData
Function Prototype	`void SSP_SendData(SSP_TypeDef* SSPx, u16 Data)`
Behavior Description	Transmits a Data through the SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	Data : word to be transmitted.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Send 0xA5 through the SSP0 peripheral. */
SSP_SendData(SSP0, 0xA5);
```

18.2.10 SSP_ReceiveData

Function Name	SSP_ReceiveData
Function Prototype	`u16 SSP_ReceiveData(SSP_TypeDef* SSPx)`
Behavior Description	Returns the most recent received data by the SSP peripheral.
Input Parameter	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Output Parameter	None
Return Parameter	The value of the received data.
Required preconditions	None
Called Functions	None

Example:

```
/* Read the most recent data received by the SSP0 peripheral. */
u16 hReceivedData;
hReceivedData = SSP_ReceiveData(SSP0);
```

18.2.11 SSP_LoopBackConfig

Function Name	SSP_LoopBackConfig
Function Prototype	`SSP_LoopBackConfig(SSP_TypeDef* SSPx, FunctionalState NewState)`
Behavior Description	Enables or disables the Loop back mode for the selected SSP peripheral.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	NewState: new state of the Loop back mode. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Enable the Loop back mode for the SSP0 peripheral. */
SSP_LoopBackConfig(SSP0, ENABLE);
```

18.2.12 SSP_NSSInternalConfig

Function Name	SSP_NSSInternalConfig
Function Prototype	`SSP_NSSInternalConfig(SSP_TypeDef* SSPx, u16 NSSState)`
Behavior Description	Configures the NSS pin by software.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	NSSState: NSS internal state. This parameter can be: SSP_NSSInternal_Set: Set NSS pin internally SSP_NSSInternal_Reset: Reset NSS pin internally
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called Functions	None

Example:

```
/* Set internaly the NSS pin */
SSP_NSSInternalConfig(SSP0, SSP_NSSInternal_Set);

/* Reset internaly the NSS pin */
SSP_NSSInternalConfig(SSP0, SSP_NSSInternal_Reset);
```

18.2.13 SSP_GetFlagStatus

Function Name	SSP_GetFlagStatus
Function Prototype	`FlagStatus SSP_GetFlagStatus(SSP_TypeDef* SSPx, u16 SSP_FLAG)`
Behavior Description	Checks whether the specified SSP flag is set or not.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_FLAG: specifies the flag to check. Refer to section " *SSP_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SSP_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

SSP_FLAG

The SSP flags that can check for are listed in the following table:

SSP_FLAG	Meaning
SSP_FLAG_Busy	busy flag
SSP_FLAG_RxFifoFull	Receive FIFO full flag
SSP_FLAG_RxFifoNotEmpty	Receive FIFO not empty flag
SSP_FLAG_TxFifoNotFull	Transmit FIFO not full flag
SSP_FLAG_TxFifoEmpty	Transmit FIFO empty flag
SSP_FLAG_TxFifo	Transmit FIFO Masked interrupt flag
SSP_FLAG_RxFifo	Receive FIFO Masked interrupt flag
SSP_FLAG_RxTimeOut	Receive timeout Masked interrupt flag
SSP_FLAG_RxOverrun	Receive overrun Masked interrupt flag

Example:

```
/* Get the Receive FIFO full flag status */
FlagStatus Status;
Status = SSP_GetFlagStatus(SSP0, SSP_FLAG_RxFifoFull);
```

18.2.14 SSP_ClearFlag

Function Name	SSP_ClearFlag
Function Prototype	`void SSP_ClearFlag(SSP_TypeDef* SSPx, u16 SSP_FLAG)`
Behavior Description	Clears the SSPx's pending flags.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	SSP_FLAG: specifies the flag to clear. Refer to the section "*SSP_FLAG*" for more details on the allowed values of this parameter.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

SSP_FLAG

To clear SSP flags, use one of the following values:

SSP_FLAG	Meaning
SSP_FLAG_RxTimeOut	Receive timeout flag
SSP_FLAG_RxOverrun	Receive overrun flag

Example:

```
/* Clear the SSP0 Receive timeout flag */
SSP_ClearFlag(SSP0, SSP_FLAG_RxTimeOut);
```

18.2.15 SSP_GetITStatus

Function Name	SSP_GetITStatus
Function Prototype	`ITStatus SSP_GetITStatus(SSP_TypeDef* SSPx, u16 SSP_IT)`
Behavior Description	Checks whether the specified SSP interrupt has occurred or not.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter2	SSP_IT: specifies the interrupt source to check. Refer to section "*SSP_IT on page 275*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of SSP_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the SSP0 Receive FIFO interrupt status */
ITStatus Status;
Status = SSP_GetITStatus(SSP0, SSP_IT_RxFifo);
```

18.2.16 SSP_ClearITPendingBit

Function Name	SSP_ClearITPendingBit
Function Prototype	`void SSP_ClearITPendingBit(SSP_TypeDef* SSPx, u16 SSP_IT)`
Behavior Description	Clears the SSPx interrupt pending bits.
Input Parameter1	SSPx: where x can be 0 or 1 to select the SSP peripheral.
Input Parameter 2	SSP_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the “|” operator. Refer to the section “*SSP_IT*” for more details on the allowed values of this parameter.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

SSP_IT

To clear the SSP interrupts pending bits, use a combination of one or more of the following values:

SSP_IT	Meaning
SSP_IT_RxTimeOut	Receive timeout interrupt
SSP_IT_RxOverrun	Receive overrun interrupt

Example:

```
/* Clear the SSP0 Receive timeout interrupt pending bit */
SSP_ClearITPendingBit(SSP0, SSP_IT_RxTimeOut);
```

19 Universal Asynchronous Receiver Transmitter (UART)

The UART peripheral provides hardware management of the CTS and RTS signals and has LIN Master capability.

To optimize the data transfer between the processor and the peripheral, two FIFOs (receive/transmit) of 16 bytes each have been implemented. The UART can be served by the DMA controller.

The first section describes the register structure used in the UART software library. The second one presents the software library functions.

19.1 UART register structure

The UART register structure *UART_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 DR;
  u16  EMPTY;
  vu16 RSR;
  u16  EMPTY1[9];
  vu16 FR;
  u16  EMPTY2;
  vu16 BKR;
  u16  EMPTY3[3];
  vu16 IBRD;
  u16  EMPTY4;
  vu16 FBRD;
  u16  EMPTY5;
  vu16 LCR;
  u16  EMPTY6;
  vu16 CR;
  u16  EMPTY7;
  vu16 IFLS;
  u16  EMPTY8;
  vu16 IMSC;
  u16  EMPTY9;
  vu16 RIS;
  u16  EMPTY10;
  vu16 MIS;
  u16  EMPTY11;
  vu16 ICR;
  u16  EMPTY12;
  vu16 DMACR;
  u16  EMPTY13;
} UART_TypeDef;
```

The following table presents the UART registers:

Register	Description
DR	Data Register
RSR	Receive Status Register
FR	Flag Register
BKR	Break Register
IBRD	Integer Baud Rate Divider Register
FBRD	Fractional Baud Rate Divider Register
LCR	Line Control Register
CR	Control Register
IFLS	Interrupt FIFO Level Select
IMSC	Interrupt Mask Set/Clear Register
RIS	Raw Interrupt Status
MIS	Masked Interrupt Status
ICR	Interrupt Clear Register
DMACR	DMA Control Register

The 3 UART peripherals are declared in the same file:

```
...
#define PERIPH_BASE   0xFFFF 0000
...
#define UART0_BASE      (PERIPH_BASE + 0xD400)
#define UART1_BASE      (PERIPH_BASE + 0xD800)
#define UART2_BASE      (PERIPH_BASE + 0xDC00)

#ifndef DEBUG
...
#define UART0        ((UART_TypeDef *) UART0_BASE)
#define UART1        ((UART_TypeDef *) UART1_BASE)
#define UART2        ((UART_TypeDef *) UART2_BASE)
...
#else
...
#ifdef _UART0
  EXT UART_TypeDef          *UART0;
#endif /*_UART0 */

#ifdef _UART1
  EXT UART_TypeDef          *UART1;
#endif /*_UART1 */

#ifdef _UART2
  EXT UART_TypeDef          *UART2;
#endif /*_UART2 */
...
#endif
```

When debug mode is used, UART pointer is initialized in 75x_lib.c file:

```
#ifdef _UART0
  UART0 = (UART_TypeDef *)UART0_BASE;
#endif /*_UART0 */

#ifdef _UART1
  UART1 = (UART_TypeDef *)UART1_BASE;
#endif /*_UART1 */

#ifdef _UART2
  UART2 = (UART_TypeDef *)UART2_BASE;
#endif /*_UART2 */
```

_UART, _UART0, _UART1 and _UART2 must be defined, in *75x*_conf.h file, to access the peripheral registers as follows:

```
#define _UART
#define _UART0
#define _UART1
#define _UART2
...
```

19.2 Software library functions

The following table lists the various functions of the UART library.

Function Name	Description
UART_DeInit	Deinitializes the UARTx peripheral registers to their default reset values.
UART_Init	Initializes the UARTx peripheral according to the specified parameters in the UART_InitStruct.
UART_StructInit	Fills each UART_InitStruct member with its default value.
UART_Cmd	Enables or disables the specified UART peripheral.
UART_ITConfig	Enables or disables the specified UART interrupts.
UART_DMAConfig	Configures the UART0 DMA interface.
UART_DMACmd	Enables or disables the UART0 DMA interface.
UART_LoopBackConfig	Enables or disables LoopBack mode in UARTx.
UART_LINConfig	Sets the LIN break length.
UART_LINCmd	Enables or disables LIN master mode in UARTx.
UART_SendData	Transmits a single Byte of data through the UARTx peripheral.
UART_ReceiveData	Returns the most recent received Byte by the UARTx peripheral.
UART_SendBreak	Transmits break characters.
UART_RTSConfig	Sets or resets the RTS signal.
UART_GetFlagStatus	Checks whether the specified UART flag is set or not.
UART_ClearFlag	Clears the interrupt pending bits of UARTx.
UART_GetITStatus	Checks whether the specified UART interrupt has occurred or not.
UART_ClearITPendingBit	Clears the UARTx's interrupt pending bits.

19.2.1 UART_DeInit

Function Name	UART_DeInit
Function Prototype	`void UART_DeInit(UART_TypeDef* UARTx)`
Behavior Description	Deinitializes the UARTx peripheral registers to their default reset values.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	MRCC_PeripheralSWResetConfig()

Example:

```
/* Deinitializes the UART0 registers to their default reset value */
UART_DeInit(UART0);
```

19.2.2 UART_Init

Function Name	UART_Init
Function Prototype	`void UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)`
Behavior Description	Initializes the UARTx peripheral according to the specified parameters in the UART_InitStruct .
Input Parameter1	UARTx: where x can be 0,1or 2 to select the UART peripheral.
Input Parameter2	UART_InitStruct: pointer to a UART_InitTypeDef structure that contains the configuration information for the specified UART peripheral. Refer to section "*UART_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_InitTypeDef

The UART_InitTypeDef structure is defined in the *75x_uart.h* file:

```
typedef struct
{
  u16 UART_WordLength;
  u16 UART_StopBits;
  u16 UART_Parity;
  u32 UART_BaudRate;
  u16 UART_HardwareFlowControl;
  u16 UART_Mode;
  u16 UART_FIFO;
  UART_FIFOLevel UART_TxFIFOLevel;
  UART_FIFOLevel UART_RxFIFOLevel;
}UART_InitTypeDef;
```

UART_WordLength

Indicates the number of data bits transmitted or received in a frame. This member can be one of the following values:

UART_WordLength	Meaning
UART_WordLength_5D	5 data bits
UART_WordLength_6D	6 data bits
UART_WordLength_7D	7 data bits
UART_WordLength_8D	8 data bits

UART_StopBits

Specifies the number of transmitted stop bits. This member can be one of the following values:

UART_StopBits	Meaning
UART_StopBits_1	One stop bit is transmitted at the end of frame
UART_StopBits_2	Two stop bits are transmitted at the end of frame

UART_Parity

Specifies the parity mode. This member can be one of the following values:

UART_Parity	Meaning
UART_Parity_No	Parity Disable
UART_Parity_Even	Even Parity
UART_Parity_Odd	Odd Parity
UART_Parity_OddStick	1 is transmitted as bit parity
UART_Parity_EvenStick	0 is transmitted as bit parity

UART_BaudRate

Specifies the baudrate of the UART communication. The baudrate is computed with this formula :

IntegerDivider = ((APBClock) / (16 * (UART_InitStruct->UART_BaudRate)))

FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 64 + 0.5)

UART_HardFlowControl

Specifies whether hardware flow control mode is enabled or disabled. This member can be one of the following values:

UART_HardFlowControl	Meaning
UART_HardwareFlowControl_None	HFC Disable
UART_HardwareFlowControl_RTS	RTS Enable
UART_HardwareFlowControl_CTS	CTS Enable
UART_HardwareFlowControl_RTS_CTS	CTS and RTS Enable

UART_Mode

Specifies whether the receive mode is enabled or disabled. This member can be a combination of one or more of the following values:

UART_Transmit	Meaning
UART_Mode_Rx	Receive Enable
UART_Mode_Tx	Transmit Enable
UART_Mode_Tx_Rx	Transmit and Receive Enable

UART_FIFO

Specifies whether the FIFOs (Rx and Tx FIFOs) are enabled or disabled. This member can be one of the following values:

UART_Transmit	Meaning
UART_FIFO_Enable	FIFOs Enable
UART_FIFO_Disable	FIFOs Disable

UART_TxFIFOLevel

Specifies the level of the UART TxFIFO (Not used when FIFOs are disabled). This member can be one of the following values:

UART_TxFIFOLevel	Meaning
UART_FIFOLevel_1_8	FIFO becomes >= 1/8 full
UART_FIFOLevel_1_4	FIFO becomes >= 1/4 full
UART_FIFOLevel_1_2	FIFO becomes >= 1/2 full
UART_FIFOLevel_3_4	FIFO becomes >= 3/4 full
UART_FIFOLevel_7_8	FIFO becomes >= 7/8 full

UART_RxFIFOLevel

Specifies the level of the UART RxFIFO (Not used when FIFOs are disabled). This member can be one of the following values:

UART_RxFIFOLevel	Meaning
UART_FIFOLevel_1_8	FIFO becomes >= 1/8 full
UART_FIFOLevel_1_4	FIFO becomes >= 1/4 full
UART_FIFOLevel_1_2	FIFO becomes >= 1/2 full
UART_FIFOLevel_3_4	FIFO becomes >= 3/4 full
UART_FIFOLevel_7_8	FIFO becomes >= 7/8 full

Example:

```
/* The following example illustrates how to configure the UART0 */
  UART_InitTypeDef UART_InitStructure;

  UART_InitStructure.UART_WordLength = UART_WordLength_8D;
  UART_InitStructure.UART_StopBits = UART_StopBits_1;
  UART_InitStructure.UART_Parity = UART_Parity_Odd;
  UART_InitStructure.UART_BaudRate = 9600;
  UART_InitStructure.UART_HardwareFlowControl = UART_HardwareFlowControl_RTS_CTS;
  UART_InitStructure.UART_Mode = UART_Mode_Tx_Rx;
  UART_InitStructure.UART_FIFO = UART_FIFO_Enable;
  UART_InitStructure.UART_TxFIFOLevel = UART_FIFOLevel_1_2;
  UART_InitStructure.UART_RxFIFOLevel = UART_FIFOLevel_1_2;
  UART_Init(UART0, &UART_InitStructure);
```

19.2.3 UART_StructInit

Function Name	UART_StructInit
Function Prototype	`void UART_StructInit(UART_InitTypeDef* UART_InitStruct)`
Behavior Description	Fills each UART_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	UART_InitStruct: pointer to a UART_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

The UART_InitStruct members have the following default values:

Member	Default value
UART_WordLength	UART_WordLength_8D
UART_StopBits	UART_StopBits_1
UART_Parity	UART_Parity_Odd
UART_BaudRate	9600
UART_HardwareFlowControl	UART_HardwareFlowControl_None
UART_Mode	UART_Mode_Tx_Rx
UART_FIFO	UART_FIFO_Enable
UART_TxFIFOLevel	UART_FIFOLevel_1_2
UART_RxFIFOLevel	UART_FIFOLevel_1_2

Example:

```
/* The following example illustrates how to initialize a UART_InitTypeDef structure*/
  UART_InitTypeDef UART_InitStructure;
  UART_StructInit(&UART_InitStructure);
```

19.2.4 UART_Cmd

Function Name	UART_Cmd
Function Prototype	`void UART_Cmd(UART_TypeDef* UARTx, FunctionalState NewState)`
Behavior Description	Enables or disables the specified UART peripheral.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	NewState: new state of the UARTx peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the UART0 */
  UART_Cmd(UART0, ENABLE);
```

19.2.5 UART_ITConfig

Function Name	UART_ITConfig
Function Prototype	`void UART_ITConfig(UART_TypeDef* UARTx, u16 UART_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified UART interrupts.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the UART interrupts sources to be enabled or disabled. Refer to section "*UART_IT*" for more details on the allowed values of this parameter.
Input Parameter3	NewState: new state of the specified UARTx interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_IT

To enable or disable UART interrupts, use a combination of one or more of the following values:

UART_IT	Meaning
UART_IT_OverrunError	Overrun Error interrupt mask
UART_IT_BreakError	Break Error interrupt mask
UART_IT_ParityError	Parity Error interrupt mask
UART_IT_FrameError	Frame Error interrupt mask
UART_IT_ReceiveTimeOut	Receive Time Out interrupt mask
UART_IT_Transmit	Transmit interrupt mask
UART_IT_Receive	Receive interrupt mask
UART_IT_CTS	CTS interrupt mask

Example:

```
/* Enables the UART0 transmit and receive interrupts */
  UART_ITConfig(UART0, UART_IT_Transmit | UART_IT_Receive, ENABLE);
```

19.2.6 UART_DMAConfig

Function Name	UART_DMAConfig
Function Prototype	`void UART_DMAConfig(u16 UART0_DMATransfer, u16 UART0_DMAOnError)`
Behavior Description	Configures the UART0 DMA interface.
Input Parameter1	UART0_DMATransfer : specifies the configuration of the DMA request. Refer to section "*UART0_DMATransfer*" for more details on the allowed values of this parameter.
Input Parameter2	UART0_DMAOnError: specifies the DMA on error request. Refer to section "*UART0_DMAOnError*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART0_DMATransfer

To select DMA transfer, use one of the following values:

UART0_DMATransfer	**Meaning**
UART0_DMATransfer_Single	Single DMA transfer
UART0_DMATransfer_Burst	Burst DMA transfer

UART0_DMAOnError

To select wether the DMA is enabled/disabled when an error ocurrs, use one of the following values:

UART0_DMAOnError	**Meaning**
UART0_DMAOnError_Enable	DMA receive request enabled when the UART error interrupt is asserted.
UART0_DMAOnError_Disable	DMA receive request disabled when the UART error interrupt is asserted.

Example:

```
/* DMA configuration */
  UART_DMAConfig(UART0_DMATransfer_Single, UART0_DMAOnError_Enable);
```

19.2.7 UART_DMACmd

Function Name	UART_DMACmd
Function Prototype	`void UART_DMACmd(u16 UART0_DMAReq, FunctionalState NewState)`
Behavior Description	Enables or disables the UART0 DMA interface.
Input Parameter1	UART0_DMAReq: specifies the DMA request. Refer to section "*UART0_DMAReq*" for more details on the allowed values of this parameter.
Input Parameter2	NewState: new state of the UART0 DMA request. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART0_DMAReq

To select the DMA request to be enabled/disabled, use one of the following values:

UART0_DMAReq	**Meaning**
UART0_DMAReq_Tx	Transmit DMA request
UART0_DMAReq_Rx	Receive DMA request

Example:

```
/* Enable the DMA transfer on Rx action */
  UART_DMACmd(UART0_DMAReq_Rx, ENABLE);
```

19.2.8 UART_LoopBackConfig

Function Name	UART_LoopBackConfig
Function Prototype	`void UART_LoopBackConfig(UART_TypeDef* UARTx, FunctionalState NewState)`
Behavior Description	Enables or disables LoopBack mode in UARTx.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	NewState: new state of the UARTx's LoopBack mode. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* ENABLE the UART1 LoopBack mode */
  UART_LoopBackConfig(UART1, ENABLE);
```

19.2.9 UART_LINConfig

Function Name	UART_LINConfig
Function Prototype	`void UART_LINConfig(UART_TypeDef* UARTx, u16 UART_LINBreakLength)`
Behavior Description	Sets the LIN break length.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_LINBreakLength: Break length value. Refer to section "*UART_LINBreakLength*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

UART_LINBreakLength

To set the break length, use one of the following values:

UART_LINBreakLength	**Meaning**
UART_LINBreakLength_10	10 low bits
UART_LINBreakLength_11	11 low bits
UART_LINBreakLength_12	12 low bits
UART_LINBreakLength_13	13 low bits
UART_LINBreakLength_14	14 low bits
UART_LINBreakLength_15	15 low bits
UART_LINBreakLength_16	16 low bits
UART_LINBreakLength_17	17 low bits
UART_LINBreakLength_18	18 low bits
UART_LINBreakLength_19	19 low bits
UART_LINBreakLength_20	20 low bits

Example:

```
/* 13 bits are sent as break bits on UART1 */
  UART_LINConfig(UART1, UART_LINBreakLength_13);
```

19.2.10 UART_LINCmd

Function Name	UART_LINCmd
Function Prototype	`void UART_LINCmd(UART_TypeDef* UARTx, FunctionalState NewState)`
Behavior Description	Enables or disables LIN master mode in UARTx.
Input Parameter1	UARTx: where x can be 0,1, or 2 to select the UART peripheral.
Input Parameter2	NewState: new state of the UARTx LIN interface. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Enable the LIN interface */
  UART_LINCmd(UART2, ENABLE);
```

19.2.11 UART_SendData

Function Name	UART_SendData
Function Prototype	`void UART_SendData(UART_TypeDef* UARTx, u8 Data)`
Behavior Description	Transmits signle Byte of data through the UARTx peripheral.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	Data: the byte to transmit.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send one byte on UART1 */
  UART_SendData(UART1,0x22);
```

19.2.12 UART_ReceiveData

Function Name	UART_ReceiveData
Function Prototype	`vu8 UART_ReceiveData(UART_TypeDef* UARTx)`
Behavior Description	Returns the most recent received Byte by the UARTx peripheral.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	The received data.
Required preconditions	None
Called functions	None

Example:

```
/* Receive one byte on UART2 */
  u8 RxData;
  RxData = UART_ReceiveData(UART2);
```

19.2.13 UART_SendBreak

Function Name	UART_SendBreak
Function Prototype	`void UART_SendBreak(UART_TypeDef* UARTx)`
Behavior Description	Transmits break characters.
Input Parameter	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Send break character on UART1 */
  UART_SendBreak(UART1);
```

19.2.14 UART_RTSConfig

Function Name	UART_RTSConfig
Function Prototype	`void UART_RTSConfig(UART_TypeDef* UARTx, UART_RTSTypeDef RTSState)`
Behavior Description	Sets or resets the RTS signal.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	RTSState: new state of the RTS signal. This parameter can be: RTSSET or RTSRESET.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Set the UART1 RTS signal */
  UART_RTSConfig(UART1, RTSSET);
```

19.2.15 UART_GetFlagStatus

Function Name	UART_GetFlagStatus
Function Prototype	`FlagStatus UART_GetFlagStatus(UART_TypeDef* UARTx, u16 UART_FLAG)`
Behavior Description	Checks whether the specified UART flag is set or not.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_FLAG: specifies the flag to check. Refer to section "*UART_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of UART_FLAG (SET or RESET).
Required preconditions	None
Called functions	None

UART_FLAG

The UART flags that can check for are listed in the following table:

UART_FLAG	Meaning
UART_FLAG_OverrunError	Overrun error flag
UART_FLAG_Break	Break error flag
UART_FLAG_ParityError	Parity error flag
UART_FLAG_FrameError	Frame error flag
UART_FLAG_TxFIFOEmpty	Transmit FIFO Empty flag
UART_FLAG_RxFIFOFull	Receive FIFO Full flag
UART_FLAG_TxFIFOFull	Transmit FIFO Full flag
UART_FLAG_RxFIFOEmpty	Receive FIFO Empty flag
UART_FLAG_Busy	Busy flag
UART_FLAG_CTS	CTS flag
UART_RawIT_OverrunError	Overrun Error interrupt flag
UART_RawIT_BreakError	Break Error interrupt flag
UART_RawIT_ParityError	Parity Error interrupt flag
UART_RawIT_FrameError	Frame Error interrupt flag
UART_RawIT_ReceiveTimeOut	ReceiveTimeOut interrupt flag
UART_RawIT_Transmit	Transmit interrupt flag
UART_RawIT_Receive	Receive interrupt flag
UART_RawIT_CTS	CTS interrupt flag

Example:

```
/* Check if the transmit FIFO is full or not */
  FlagStatus Status;
  Status = UART_GetFlagStatus(UART0, UART_TxFIFOFull);
```

19.2.16 UART_ClearFlag

Function Name	UART_ClearFlag
Function Prototype	`void UART_ClearFlag(UART_TypeDef* UARTx, u16 UART_FLAG)`
Behavior Description	Clears the pending flags of UARTx.
Input Parameter1	UARTx: where x can be 0,1or 2 to select the UART peripheral.
Input Parameter2	UART_FLAG: specifies the flag to clear. Refer to section "*UART_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Note: Clearing one flag causes the other flags to be cleared.

UART_FLAG

This parameter can be one of the following parameters:

UART_FLAG	Meaning
UART_FLAG_OverrunError	Overrun error flag
UART_FLAG_Break	Break error flag
UART_FLAG_ParityError	Parity error flag
UART_FLAG_FrameError	Frame error flag

Example:

```
/* Clear Overrun error flag */
  UART_ClearFlag(UART0,UART_FLAG_OverrunError);
```

19.2.17 UART_GetITStatus

Function Name	UART_GetITStatus
Function Prototype	`ITStatus UART_GetITStatus(UART_TypeDef* UARTx, u16 UART_IT)`
Behavior Description	Checks whether the specified UART interrupt has occurred or not.
Input Parameter1	UARTx: where x can be 0,1 or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the interrupt source to check. Refer to section "*UART_IT on page 294*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of UART_IT (SET or RESET).
Required preconditions	None
Called functions	None

Example:

```
/* Get the UART0 Overrun Error interrupt status */
  ITStatus OverrunITStatus;
  OverrunITStatus = UART_GetITStatus(UART0, UART_IT_OverrunError);
```

19.2.18 UART_ClearITPendingBit

Function Name	UART_ClearITPendingBit
Function Prototype	`void UART_ClearITPendingBit(UART_TypeDef* UARTx, u16 UART_IT)`
Behavior Description	Clears the interrupt pending bits of UARTx.
Input Parameter1	UARTx: where x can be 0,1or 2 to select the UART peripheral.
Input Parameter2	UART_IT: specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to section "*UART_IT on page 294*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required preconditions	None
Called functions	None

Example:

```
/* Clear the Overrun error interrupt pending bit */
  UART_ClearITPendingBit(UART0,UART_IT_OverrunError);
```

20 Analog-to-Digital Converter (ADC)

The Analog to Digital Converter (ADC) comprises an input multiplexing channel selector feeding a successive approximation converter. The conversion resolution is 10 bits.

Section 20.1 describes the register structure used in the ADC software library. *Section 20.2* presents the software library functions.

20.1 ADC register structure

The ADC register structure *ADC_TypeDef* is defined in the *75x_map.h* file as follows:

```
typedef struct
{
  vu16 CLR0;
  u16  EMPTY1;
  vu16 CLR1;
  u16  EMPTY2;
  vu16 CLR2;
  u16  EMPTY3;
  vu16 CLR3;
  u16  EMPTY4;
  vu16 CLR4;
  u16  EMPTY5;
  vu16 TRA0;
  u16  EMPTY6;
  vu16 TRA1;
  u16  EMPTY7;
  vu16 TRA2;
  u16  EMPTY8;
  vu16 TRA3;
  u16  EMPTY9;
  vu16 TRB0;
  u16  EMPTY10;
  vu16 TRB1;
  u16  EMPTY11;
  vu16 TRB2;
  u16  EMPTY12;
  vu16 TRB3;
  u16  EMPTY13;
  vu16 DMAR;
  u16  EMPTY14[7];
  vu16 DMAE;
  u16  EMPTY15 ;
  vu16 PBR;
  u16  EMPTY16;
  vu16 IMR;
  u16  EMPTY17;
  vu16 D0;
  u16  EMPTY18;
  vu16 D1;
  u16  EMPTY19;
  vu16 D2;
  u16  EMPTY20;
  vu16 D3;
  u16  EMPTY21;
  vu16 D4;
  u16  EMPTY22;
  vu16 D5;
```

```
  u16  EMPTY23;
  vu16 D6;
  u16  EMPTY24;
  vu16 D7;
  u16  EMPTY25;
  vu16 D8;
  u16  EMPTY26;
  vu16 D9;
  u16  EMPTY27;
  vu16 D10;
  u16  EMPTY28;
  vu16 D11;
  u16  EMPTY29;
  vu16 D12;
  u16  EMPTY30;
  vu16 D13;
  u16  EMPTY31;
  vu16 D14;
  u16  EMPTY32;
  vu16 D15;
  u16  EMPTY33;
} ADC_TypeDef;
```

The following table presents the ADC registers:

Register	Description
CLR0	ADC Control Logic register 0
CLR1	ADC Control Logic register *1*
CLR2	ADC Control Logic register 2
CLR3	ADC Control Logic register 3
CLR4	ADC Control Logic register 4
TRA0	ADC Threshold Detection Channel0 register A
TRA1	ADC Threshold Detection Channel1 register A
TRA2	ADC Threshold Detection Channel2 register A
TRA3	ADC Threshold Detection Channel3 register A
TRB0	ADC Threshold Detection Channel0 register B
TRB1	ADC Threshold Detection Channel1 register B
TRB2	ADC Threshold Detection Channel2 register B
TRB3	ADC Threshold Detection Channel3 register B
DMAR	ADC DMA register
DMAE	ADC DMA Enable register
PBR	ADC Pending Bit register
IMR	ADC Interrupt Mask Register
D0	ADC DATA register for channel 0
D1	ADC DATA register for channel 1
D2	ADC DATA register for channel 2
D3	ADC DATA register for channel 3
D4	ADC DATA register for channel 4
D5	ADC DATA register for channel 5
D6	ADC DATA register for channel 6
D7	ADC DATA register for channel 7
D8	ADC DATA register for channel 8
D9	ADC DATA register for channel 9
D10	ADC DATA register for channel 10
D11	ADC DATA register for channel 11
D12	ADC DATA register for channel 12
D13	ADC DATA register for channel 13
D14	ADC DATA register for channel 14
D15	ADC DATA register for channel 15

The ADC peripheral is declared in the same file :

```
...
#define PERIPH_BASE   0xFFFF0000
....
#define ADC_BASE       (PERIPH_BASE + 0x8400)

#ifndef DEBUG
  ...
#define ADC ((ADC_TypeDef   *)  ADC_BASE)
  ...
#else
  ...
#ifdef _ADC
  EXT ADC_TypeDef           *ADC;
#endif /*_ADC */
  ...
#endif
```

When debug mode is used, the ADC pointer is initialized in the 75x_lib.c file :

```
#ifdef _ADC
  ADC = (ADC_TypeDef  *)ADC_BASE;
#endif /*_ADC */
```

_ADC, must be defined, in 75x_conf.h file, to access the peripheral registers as follows :

```
#define _ADC
...
```

20.2 Software library functions

The following table lists the various functions of the ADC library.

Function Name	Description
ADC_DeInit	Deinitializes the ADC peripheral registers to their default reset values.
ADC_Init	Initializes the ADC according to the specified parameters in the ADC_InitStruct.
ADC_StructInit	Fills each ADC_InitStruct member with its default value.
ADC_StartCalibration	Starts the calibration process. The user can enable and disable the calibration average
ADC_GetCalibrationStatus	Gets the ADC Calibration Status.
ADC_ConversionCmd	Starts or stops the ADC conversion in the selected mode.
ADC_GetSTARTBitStatus	Gets the ADC START/STOP bit Status.
ADC_Cmd	Enables the ADC peripheral or puts it in power down mode.
ADC_AutoClockConfig	Enables or disables the auto clock off feature.
ADC_AnalogWatchdogConfig	Configures the analog input channel to be used for the selected Analog Watchdog and defines its corresponding High and Low threshold values.
ADC_AnalogWatchdogCmd	Enables or disables the Analog watchdog detection feature for the selected channel
ADC_GetAnalogWatchdogResult	Returns the comparison result of the selected analog watchdog
ADC_InjectedConversionConfig	Configures the start trigger level for the injected channels and selects the injected analog input channels to be converted.
ADC_StartInjectedConversion	Starts by software the conversion of the injected analog channels
ADC_GetConversionValue	Reads the result of conversion from the appropriate data register.
ADC_ITConfig	Enables or disables the specified ADC interrupts.
ADC_DMAConfig	Configures the ADC's DMA interface.
ADC_DMACmd	Enables or disables DMA transfer for the ADC
ADC_GetDMAFirstEnabledChannel	Gets the first DMA enabled channel being converted when the DMA was enabled the last time
ADC_GetFlagStatus	Checks whether the specified ADC flag is set or not.
ADC_ClearFlag	Clears the ADC's pending flags.
ADC_GetITStatus	Checks whether the specified ADC interrupt has occurred or not.
ADC_ClearITPendingBit	Clears the ADC's interrupt pending bits.

20.2.1 ADC_DeInit

Function Name	ADC_DeInit
Function Prototype	`void ADC_DeInit(void)`
Behavior Description	Deinitializes the ADC peripheral registers to their default reset values.
Input Parameter	None
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	MRCC_PeripheralSWResetConfig().

Example:

```
/* Deinitialize the ADC peripheral */
  ADC_DeInit();
```

20.2.2 ADC_Init

Function Name	ADC_Init
Function Prototype	`void ADC_Init(ADC_InitTypeDef *ADC_InitStruct)`
Behavior Description	Initializes the ADC peripheral according to the specified parameters in the ADC_InitStruct.
Input Parameter	ADC_InitStruct : pointer to an ADC_InitTypeDef structure that contains the configuration information for the ADC peripheral. Refer to the section "*ADC_InitTypeDef*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_InitTypeDef

The ADC_InitTypeDef structure is defined in the *75x_adc.h* file:

```
typedef struct
{
  u16 ADC_ConversionMode;
  u16 ADC_ExtTrigger;
  u16 ADC_AutoClockOff;
  u8  ADC_SamplingPrescaler;
  u8  ADC_ConversionPrescaler;
  u8  ADC_FirstChannel;
  u8  ADC_ChannelNumber;
}ADC_InitTypeDef;
```

ADC_ConversionMode

Select the conversion mode. This member can be one of the following values:

ADC_ConversionMode	Meaning
ADC_ConversionMode_OneShot	One shot conversion mode Enabled
ADC_ConversionMode_Scan	Scan conversion mode Enabled

ADC_ExtTrigger

Selects the external start trigger. This member can be one of the following values.

ADC_ExtTrigger	Meaning
ADC_ExtTrigger_HighLevel	Sets the external start trigger to High Level of TIM0
ADC_ExtTrigger_LowLevel	Sets the external start trigger to Low Level of TIM0
ADC_ExtTrigger_RisingEdge	Sets the external start trigger to Rising edge of TIM0
ADC_ExtTrigger_FallingEdge	Sets the external start trigger to Falling edge of TIM0
ADC_ExtTrigger_Disable	Starts ADC conversion by soft

ADC_AutoClockOff

Specifies whether the auto clock off feature is enabled or disabled. This member can be one of the following values.

ADC_AutoClockOff	Meaning
ADC_AutoClockOff_Enable	AutoClockOff feature enabled
ADC_AutoClockOff_Disable	AutoClockOff feature disabled

ADC_SamplingPrescaler

Selects the sampling prescaler. Allowed values are from 0 to 7.

ADC_ConversionPrescaler

Selects the sampling prescaler. Allowed values are from 0 to 7.

ADC_FirstChannel

Selects the first channel to be converted. This member can be one of the following values:

ADC_FirstChannel	Meaning
ADC_CHANNEL0	Selects channel 0
ADC_CHANNEL1	Selects Channel 1
ADC_CHANNEL2	Selects channel 2
ADC_CHANNEL3	Selects channel 3
ADC_CHANNEL4	Selects channel 4
ADC_CHANNEL5	Selects channel 5
ADC_CHANNEL6	Selects channel 6
ADC_CHANNEL7	Selects channel 7

ADC_FirstChannel	Meaning
ADC_CHANNEL8	Selects channel 8
ADC_CHANNEL9	Selects channel 9
ADC_CHANNEL10	Selects channel 10
ADC_CHANNEL11	Selects channel 11
ADC_CHANNEL12	Selects channel 12
ADC_CHANNEL13	Selects channel 13
ADC_CHANNEL14	Selects channel 14
ADC_CHANNEL15	Selects channel 15

ADC_ChannelNumber

Selects the number of channels to be converted. The maximum number is 16 channels.

Example*:*

```
/* Initialize the ADC according to the ADC_InitStructure members */
ADC_InitTypeDef  ADC_InitStructure;

ADC_InitStructure.ADC_ConversionMode = ADC_ConversionMode_Scan;
ADC_InitStructure.ADC_ExtTrigger = ADC_ExtTrigger_Disable;
ADC_InitStructure.ADC_AutoClockOff = ADC_AutoClockOff_Disable;
ADC_InitStructure.ADC_SamplingPrescaler = 0x4;
ADC_InitStructure.ADC_ConversionPrescaler = 0x2;
ADC_InitStructure.ADC_FirstChannel = ADC_CHANNEL0;
ADC_InitStructure.ADC_ChannelNumber = 1;
ADC_Init(&ADC_InitStructure);
```

20.2.3 ADC_StructInit

Function Name	ADC_StructInit
Function Prototype	`void ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct)`
Behavior Description	Fills each ADC_InitStruct member with its default value, refer to the following table for more details.
Input Parameter	ADC_InitStruct : Pointer to an ADC_InitTypeDef structure which will be initialized.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

The ADC_InitStruct members have the following default values:

Member	Default value
ADC_ConversionMode	ADC_ConversionMode_OneShot
ADC_ExtTrigger	ADC_ExtTrigger_Disable
ADC_AutoClockOff	ADC_AutoClockOff_Disable
ADC_SamplingPrescaler	0
ADC_ConversionPrescaler	0
ADC_FirstChannel	ADC_CHANNEL0
ADC_ChannelNumber	1

Example:

```
/* Initialize an ADC_InitTypeDef structure. */
ADC_InitTypeDef ADC_InitStructure;
ADC_StructInit(&ADC_InitStructure);
```

20.2.4 ADC_StartCalibration

Function Name	ADC_StartCalibration
Function Prototype	`void ADC_StartCalibration(u16 ADC_CalibAverage)`
Behavior Description	Starts the calibration. Calibration average enabled/disabled .
Input Parameter	ADC_CalibAverage: Enable or disable the calibration average. Refer to the "*ADC_CalibAverage*" section for more details on the allowed values.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_CalibAverage

To enable or disable the calibration average, use one of the following values:

ADC_CalibAverage	Meaning
ADC_CalibAverage_Enable	Calibration average enabled
ADC_CalibAverage_Disable	Calibration average disabled

Example:

```
/* Start the ADC Calibration enabling the average. */
ADC_CalibrationStart(ADC_CalibAverage_Enable);
```

20.2.5 ADC_GetCalibrationStatus

Function Name	ADC_GetCalibrationStatus
Function Prototype	`FlagStatus ADC_GetCalibrationStatus(void)`
Behavior Description	Gets the ADC Calibration Status.
Input Parameter	None
Output Parameter	None
Return Parameter	The new state of ADC Calibration (SET or RESET).
Required Preconditions	None
Called Functions	None

Example:

```
/* Get the calibration status */
FlagStatus Status;
Status = ADC_GetCalibrationStatus();
```

20.2.6 ADC_ConversionCmd

Function Name	ADC_ConversionCmd
Function Prototype	`void ADC_ConversionCmd(u16 ADC_Conversion)`
Behavior Description	Starts or stops the ADC conversion .
Input Parameter	ADC_Conversion: specifies the ADC command to apply. Refer to the "*ADC_Conversion*" section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_Conversion

To Start or stop the ADC conversion, use one of the following values:

ADC_Conversion	Meaning
ADC_Conversion_Start	start the ADC conversion
ADC_Conversion_Stop	stop the ADC conversion

Example:

```
/* Start the ADC Conversion */
ADC_ConversionCmd(ADC_Conversion_Start);
```

20.2.7 ADC_GetSTARTBitStatus

Function Name	ADC_GetSTARTBitStatus
Function Prototype	`FlagStatus ADC_GetSTARTBitStatus(void)`
Behavior Description	Gets the ADC START/STOP bit Status
Input Parameter	None
Output Parameter	None
Return Parameter	The NewState of the ADC START/STOP bit (SET or RESET)
Required Preconditions	None
Called Functions	None

Example:

```
/* Get the calibration status */
FlagStatus Status;
Status = ADC_GetCalibrationStatus();
```

20.2.8 ADC_Cmd

Function Name	ADC_Cmd
Function Prototype	`void ADC_Cmd(FunctionalState NewState)`
Behavior Description	Enables the ADC peripheral or puts it in power down mode.
Input Parameter	NewState: new state of the ADC peripheral. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Enable the ADC peripheral */
ADC_Cmd(ENABLE);
```

20.2.9 ADC_AutoClockOffConfig

Function Name	ADC_AutoClockOffConfig
Function Prototype	`void ADC_AutoClockOffConfig(FunctionalState NewState)`
Behavior Description	Enables or disables the Auto clock off feature.
Input Parameter	NewState : new state of the AutoClockOff feature. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Enable the ADC AutoClockOff feature */
ADC_AutoClockOffConfig(ENABLE);
```

20.2.10 ADC_AnalogWatchdogConfig

Function Name	ADC_AnnalogWatchdogConfig
Function Prototype	`void ADC_AnalogWatchdogConfig(u16 ADC_AnalogWatchdog, u8 ADC_CHANNEL, u16 LowThreshold, u16 HighThreshold )`
Behavior Description	Configures the analog input channel to be used for the selected Analog Watchdog and defines its corresponding High and Low threshold values.

Function Name	ADC_AnnalogWatchdogConfig
Input Parameter 1	ADC_AnalogWatchdog: specifies the analog watchdog which will be affected to the desired converted channel Refer to the section '*ADC_AnalogWatchdog*' for more details on the allowed values of this parameter.
Input Parameter 2	ADC_CHANNEL: specifies the channel linked to the selected analog watchdog. Refer to the '*ADC_CHANNEL on page 321*' section for details on the allowed values.
Input Parameter 3	HighThreshold: Low Threshold for the selected Analog watchdog.
Input Parameter 4	LowThreshold: High Threshold for the selected Analog watchdog.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_AnalogWatchdog

To select the desired Analog Watchdogs use one of the following values:

ADC AnalogWatchdog	Meaning
ADC_AnalogWatchdog0	Analog Watchdog0 selected
ADC_AnalogWatchdog1	Analog Watchdog1 selected
ADC_AnalogWatchdog2	Analog Watchdog2 selected
ADC_AnalogWatchdog3	Analog Watchdog3 selected

Example:

```
/* Configure Analogwatchdog0 to channel4 with 0x55 as low threshold and 0x2FF as high
threshold */
ADC_AnalogWatchdogConfig(ADC_AnalogWatchdog0, ADC_CHANNEL4, 0x55, 0x2FF);
```

20.2.11 ADC_AnalogWatchdogCmd

Function Name	ADC_AnalogWatchdogCmd
Function Prototype	`void ADC_AnalogWatchdogCmd(u16 ADC_AnalogWatchdog, FunctionalState NewState)`
Behavior Description	Enables or disables the selected analog watchdog.
Input Parameter 1	ADC_AnalogWatchdog: specifies the analog watchdog to be enabled or disabled Refer to the section "*ADC_AnalogWatchdog on page 317*" for more details on the allowed values of this parameter.
Input Parameter 2	NewState : new state of the specified AnalogWatchdog. This parameter can be: ENABLE or DISABLE.
OutPut Parameter	None
Return Parameter	None

Function Name	ADC_AnalogWatchdogCmd
Required Preconditions	None
Called Functions	None

Example:

```
/* Enable AnalogWatchdog0 */
ADC_AnalogWatchdogCmd(ADC_AnalogWatchdog0, ENABLE);
```

20.2.12 ADC_GetAnalogWatchdogResult

Function Name	ADC_GetAnalogWatchdogResult
Function Prototype	`u16 ADC_GetAnalogWatchdogResult(u16 ADC_AnalogWatchdog)`
Behavior Description	Returns the comparison result of the selected analog watchdog.
Input Parameter	ADC_AnalogWatchdog: specifies the analog watchdog channel which its comparison result will be returned Refer to the section "*ADC_AnalogWatchdog on page 317*" for more details on the allowed values of this parameter.
OutPut Parameter	None
Return Parameter	The analog watchdog comparaison result value
Required Preconditions	None
Called Functions	None

Example:

```
/* Get the comparison result of the Analog Watchdog0 detection */
u16 hCompResult;
hCompResult = ADC_GetAnanlogWatchdogResult(ADC_AnanlogWatchdog0);
```

20.2.13 ADC_InjectedConversionConfig

Function Name	ADC_InjectedConversionConfig
Function Prototype	`void ADC_InjectedConversionConfig(u16 ADC_Injec_ExtTrigger, u8 FirstChannel,u8 ChannelNumber)`
Behavior Description	Configures the start trigger level for the injected channels and the injected analog input channels to be converted.
Input Parameter1	ADC_Injec_ExtTrigger: specifies the start trigger level. Refer to the section "*ADC_Injec_ExtTrigger*" for more details on the allowed values of this parameter.
Input Parameter2	FirstChannel: specifies the first injected channel to be converted. Refer to the "*ADC_CHANNEL on page 321*" section for details on the allowed values.
Input Parameter3	ChannelNumber: specifies the Number of the injected channels to be converted. The number must be between 1 and 16.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_Injec_ExtTrigger

The injected start trigger possibilities are listed in the following table:

ADC_Injec_ExtTrigger	Meaning
ADC_Injec_ExtTrigger_RisingEdge	Sets the injection start trigger to Rising edge of PWM Timer TRGO signal
ADC_Injec_ExtTrigger_FallingEdge	Sets the injection start trigger to Falling edge of PWM Timer TRGO signal
ADC_Injec_ExtTrigger_Disable	Starts the Injected conversion by software

Example:

```
/* Set the external start trigger as rising edge for ADC injected conversion and
select the ADC injected channels to be converted: Channel 0 to 2 */
ADC_InjectedConversionConfig(ADC_Injec_ExtTrigger_RisingEdge, ADC_CHANNEL0, 3);
```

20.2.14 ADC_StartInjectedConversion

Function Name	ADC_StartInjectedConversion
Function Prototype	`void ADC_StartInjectedConversion(void)`
Behavior Description	Starts by software the conversion of the injected input channels.
Input Parameter	None
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Start the ADC injected conversion */
ADC_StartInjectedConversion();
```

20.2.15 ADC_GetConversionValue

Function Name	ADC_GetConversionValue
Function Prototype	`u16 ADC_GetConversionValue(u8 ADC_CHANNEL)`
Behavior Description	Reads the conversion result from the appropriate data register.
Input Parameter	ADC_CHANNEL: specifies the ADC channel which its conversion value have to be returned. Refer to the "*ADC_CHANNEL*" section for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The returned value holds the conversion result of the selected channel.
Required preconditions	None
Called Functions	None

ADC_CHANNEL

To select the ADC channel to convert, use one of the following values:

ADC_CHANNEL	Meaning
ADC_CHANNEL0	Select channel 0
ADC_CHANNEL1	Select channel 1
ADC_CHANNEL2	Select channel 2
ADC_CHANNEL3	Select channel 3
ADC_CHANNEL4	Select channel 4
ADC_CHANNEL5	Select channel 5
ADC_CHANNEL6	Select channel 6
ADC_CHANNEL7	Select channel 7

ADC_CHANNEL	Meaning
ADC_CHANNEL8	Select channel 8
ADC_CHANNEL9	Select channel 9
ADC_CHANNEL10	Select channel 10
ADC_CHANNEL11	Select channel 11
ADC_CHANNEL12	Select channel 12
ADC_CHANNEL13	Select channel 13
ADC_CHANNEL14	Select channel 14
ADC_CHANNEL15	Select channel 15

Example:

```
/* Read the result of Channel0 conversion */
u16 hConversionValue;
hConversionValue = ADC_GetConversionValue(ADC_CHANNEL0);
```

20.2.16 ADC_ITConfig

Function Name	ADC_ITConfig
Function Prototype	`void ADC_ITConfig(u16 ADC_IT, FunctionalState NewState)`
Behavior Description	Enables or disables the specified ADC interrupts.
Input Parameter 1	ADC_IT: specifies the ADC interrupts to be enabled or disabled. Refer to the "*ADC_IT*" section for details on the allowed values. The user can slect more than one interrupt, by OR'ing them.
Input Parameter 2	NewState : new state of the ADC Interrupts. This parameter can be: ENABLE or DISABLE.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_IT

To enable or disable ADC interrupts, use a combination of one or more of the following values:

ADC_IT	Meaning
ADC_IT_ECH	End of chain interrupt
ADC_IT_EOC	End of conversion interrupt
ADC_IT_JECH	Injected end of chain conversion interrupt
ADC_IT_JEOC	Injected end of conversion interrupt
ADC_IT_AnalogWatchdog0_LowThreshold	Low threshold comparison for analog watchdog0 interrupt

ADC_IT	Meaning
ADC_IT_AnalogWatchdog0_HighThreshold	High threshold comparison for analog watchdog0 interrupt
ADC_IT_AnalogWatchdog1_LowThreshold	Low threshold comparison for analog watchdog1 interrupt
ADC_IT_AnalogWatchdog1_HighThreshold	High threshold comparison for analog watchdog1 interrupt
ADC_IT_AnalogWatchdog2_LowThreshold	Low threshold comparison for analog watchdog2 interrupt
ADC_IT_AnalogWatchdog2_HighThreshold	High threshold comparison for analog watchdog2 interrupt
ADC_IT_AnalogWatchdog3_LowThreshold	Low threshold comparison for analog watchdog3 interrupt
ADC_IT_AnalogWatchdog3_HighThreshold	High threshold comparison for analog watchdog3 interrupt
ADC_IT_ALL	All ADC interrupts

Example:

```
/* Enable the EOC interrupt source */
ADC_ITConfig(ADC_IT_EOC, ENABLE);
```

20.2.17 ADC_DMAConfig

Function Name	ADC_DMAConfig
Function Prototype	`void ADC_DMAConfig(u16 ADC_DMA_CHANNEL, FunctionalState NewState)`
Behavior Description	Configures the ADC's DMA interface.
Input Parameter 1	ADC_DMA_CHANNEL: specifies the channels to be enabled or disabled. Refer to the "*ADC_DMA_CHANNEL*" section for details on the allowed values.
Input Parameter 2	NewState : new state of the specified ADC DMA channels. This parameter can be: ENABLE or DISABLE.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_DMA_CHANNEL

To select the ADC DMA channels, use a combination of one or more of the following values:

ADC_DMA_CHANNEL	Meaning
ADC_DMA_CHANNEL0	Enables channel0 DMA capability
ADC_DMA_CHANNEL1	Enables channel1 DMA capability
ADC_DMA_CHANNEL2	Enables channel2 DMA capability
ADC_DMA_CHANNEL3	Enables channel3 DMA capability
ADC_DMA_CHANNEL4	Enables channel4 DMA capability
ADC_DMA_CHANNEL5	Enables channel5 DMA capability
ADC_DMA_CHANNEL6	Enables channel6 DMA capability
ADC_DMA_CHANNEL7	Enables channel7 DMA capability
ADC_DMA_CHANNEL8	Enables channel8 DMA capability
ADC_DMA_CHANNEL9	Enables channel9 DMA capability
ADC_DMA_CHANNEL10	Enables channel10 DMA capability
ADC_DMA_CHANNEL11	Enables channel11 DMA capability
ADC_DMA_CHANNEL12	Enables channel12 DMA capability
ADC_DMA_CHANNEL13	Enables channel13 DMA capability
ADC_DMA_CHANNEL14	Enables channel14 DMA capability
ADC_DMA_CHANNEL15	Enables channel15 DMA capability

Example:

```
/* Configure DMA capability for channel 0 and 1. */
ADC_DMAConfig(ADC_CHANNEL0|ADC_CHANNEL1, ENABLE);
```

20.2.18 ADC_DMACmd

Function Name	ADC_DMACmd
Function Prototype	`void ADC_DMACmd (u16 ADC_DMA)`
Behavior Description	Enables or disables the DMA transfer for the ADC
Input Parameter	ADC_DMA: specifies the DMA command. Refer to the "*ADC_DMA*" section for more details on the allowed values.
OutPut Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

ADC_DMA

To select the DMA Cmd, use one of the following values:

ADC_DMA	Meaning
ADC_DMA_Disable	Disables DMA capability
ADC_DMA_Enable	Enables DMA capability by setting global enable bit
ADC_DMA_ExtTrigger_HighLevel	DMA External Enable on high Level of TIM2 OC2 signal
ADC_DMA_ExtTrigger_LowLevel	DMA External Enable on Low Level of TIM2 OC2 signal

Example:

```
/* Enable by software the ADC DMA transfer */
ADC_DMACmd(ADC_DMA_Enable);
```

20.2.19 ADC_GetDMAFirstEnabledChannel

Function Name	ADC_GetDMAFirstEnabledChannel
Function Prototype	`u16 ADC_GetDMAFirstEnabledChannel(void)`
Behavior Description	Gets the first DMA-enabled channel configured at the time that DMA was last globally enabled.
Input Parameter	None
OutPut Parameter	None
Return Parameter	The first DMA enabled channel
Required Preconditions	None
Called Functions	None

Example:

```
/* Get the first DMA enabled channel */
u16 hFirstChannel;
hFirstChannel = ADC_GetDMAFirstEnabledChannel();
```

20.2.20 ADC_GetFlagStatus

Function Name	ADC_GetFlagStatus
Function Prototype	`FlagStatus ADC_GetFlagStatus(u16 ADC_FLAG)`
Behavior Description	Checks whether the specified ADC flag is set or not.
Input Parameter	ADC_FLAG: specifies the flag to check. Refer to the section "*ADC_FLAG*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of ADC_FLAG (SET or RESET).
Required Preconditions	None
Called Functions	None

ADC_FLAG

The ADC flags that can check for are listed in the following table:

ADC_FLAG	Meaning
ADC_FLAG_ECH	End of chain conversion flag
ADC_FLAG_EOC	End of conversion flag
ADC_FLAG_JECH	Injected end of chain conversion flag
ADC_FLAG_JEOC	Injected end of conversion flag
ADC_FLAG_AnalogWatchdog0_LowThreshold	Low threshold comparison for analog watchdog0 flag
ADC_FLAG_AnalogWatchdog0_HighThreshold	High threshold comparison for analog watchdog0 flag
ADC_FLAG_AnalogWatchdog1_LowThreshold	Low threshold comparison for analog watchdog1 flag
ADC_FLAG_AnalogWatchdog1_HighThreshold	High threshold comparison for analog watchdog1 flag
ADC_FLAG_AnalogWatchdog2_LowThreshold	Low threshold comparison for analog watchdog2 flag
ADC_FLAG_AnalogWatchdog2_HighThreshold	High threshold comparison for analog watchdog2 flag
ADC_FLAG_AnalogWatchdog3_LowThreshold	Low threshold comparison for analog watchdog3 flag
ADC_FLAG_AnalogWatchdog3_HighThreshold	High threshold comparison for analog watchdog3 flag

Example:

```
/* Get the EOC flag status */
FlagStatus Status;
Status = ADC_GetFlagStatus(ADC_FLAG_EOC);
```

20.2.21 ADC_ClearFlag

Function Name	ADC_ClearFlag
Function Prototype	`void ADC_ClearFlag(u16 ADC_FLAG)`
Behavior Description	Clears the ADC's pending flags.
Input Parameter	ADC_FLAG : specifies the flag to clear. More than one interrupt can be cleared using the "|" operator. Refer to the section "*ADC_FLAG on page 326*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Clear the ADC analog Watchdog0 low Threshold flag */
ADC_ClearFlag(ADC_FLAG_AnalogWatchdog0_LowThreshold);
```

20.2.22 ADC_GetITStatus

Function Name	ADC_GetITStatus
Function Prototype	`ITStatus ADC_GetITStatus(u16 ADC_IT)`
Behavior Description	Checks whether the specified ADC interrupt has occurred or not.
Input Parameter	ADC_IT: specifies the ADC interrupt source to check. Refer to the section "*ADC_IT on page 322*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	The new state of ADC_IT (SET or RESET).
Required Preconditions	None
Called Functions	None

Example:

```
/* Get the EOC interrupt flag status */
ITStatus Status;
Status = ADC_GetITStatus(ADC_IT_EOC);
```

20.2.23 ADC_ClearITPendingBit

Function Name	ADC_ClearITPendingBit
Function Prototype	`void ADC_ClearITPendingBit(u16 ADC_IT)`
Behavior Description	Clears the ADC's interrupt pending bits.
Input Parameter	ADC_IT :specifies the interrupt pending bit to clear. More than one interrupt can be cleared using the "|" operator. Refer to the section "*ADC_IT on page 322*" for more details on the allowed values of this parameter.
Output Parameter	None
Return Parameter	None
Required Preconditions	None
Called Functions	None

Example:

```
/* Clear the ADC analog Watchdog0 low Threshold interrupt pending bit */
ADC_ClearITPendingBit(ADC_IT_AnalogWatchdog0_LowThreshold);
```

21 Revision history

Date	Revision	Changes
20-Sep-2006	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com