MPLAB® XC16 User's Guide for Embedded Engineers

INTRODUCTION

This document presents five code examples for 16-bit devices and the MPLAB® XC16 C compiler. Some knowledge of microcontrollers and the C programming language is necessary.

1. TURN LEDS ON OR OFF

This example will light alternate LEDs on the Explorer 16 board with a PIC24FJ128GA010 Plug-In Module (PIM). For more information, see **Section B. "Get Software and Hardware"**.

```
#include <xc.h>        ← see Section 1.1

// PIC24FJ128GA010 Configuration Bit Settings

// For more on Configuration Bits,        ← see Section 1.2
// consult your device data sheet

// CONFIG2
#pragma config POSCMOD = XT   // XT Oscillator mode selected
#pragma config OSCIOFNC = ON  // OSC2/CLKO/RC15 as port I/O (RC15)
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor disabled
#pragma config FNOSC = PRI    // Primary Oscillator (XT, HS, EC)
#pragma config IESO = ON      // Int Ext Switch Over Mode enabled

// CONFIG1
#pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)
#pragma config FWPSA = PR128   // WDT Prescaler (1:128)
#pragma config WINDIS = ON     // Watchdog Timer Window Mode disabled
#pragma config FWDTEN = OFF    // Watchdog Timer disabled
#pragma config ICS = PGx2      // Emulator/debugger uses EMUC2/EMUD2
#pragma config GWRP = OFF      // Writes to program memory allowed
#pragma config GCP = OFF       // Code protection is disabled
#pragma config JTAGEN = OFF    // JTAG port is disabled

#define LEDS_ON_OFF 0x55

int main(void) {

    unsigned char portValue = LEDS_ON_OFF;

    // Port A access        ← see Section 1.3

    AD1PCFG = 0xFFFF;   // set to digital I/O (not analog)
    TRISA = 0x0000;     // set all port bits to be output
    LATA = portValue;   // write to port latch

    return 0;
}
```

1.1 Header File <xc.h>

This header file allows code in the source file to access compiler- or device-specific features. This and other header files may be found in the MPLAB XC16 installation directory in the `support` subdirectory.

Based on your selected device, the compiler will set macros that allow `xc.h` to vector to the correct device-specific header file. Do not include a device-specific header in your code or your code will not be portable.

1.2 Configuration Bits

Microchip devices have configuration registers with bits that enable and/or set up device features.

> **Note:** If you do not set Configuration bits correctly, your device will not operate at all or at least not as expected.

1.2.1 WHICH CONFIGURATION BITS TO SET

In particular, you need to look at:

- **Oscillator selection** - this must match your hardware's oscillator circuitry. If this is not correct, the *device clock may not run*. Typically, development boards use high-speed crystal oscillators. From the example code:
  ```
  #pragma config FNOSC = PRI
  #pragma config POSCMOD = XT
  ```
- **Watchdog timer**- it is recommended that you disable this timer until it is required. This prevents *unexpected resets*. From the example code:
  ```
  #pragma config FWDTEN = OFF
  ```
- **Code protection** - turn off code protection until it is required. This ensures that *device memory is fully accessible*. From the example code:
  ```
  #pragma config GCP = OFF
  ```

Different configuration bits may need to be set up to use another 16-bit device (rather than the MCU used in this example). See your device data sheet for the number and function of corresponding configuration bits. Use the part number to search http://www.microchip.com for the appropriate data sheet.

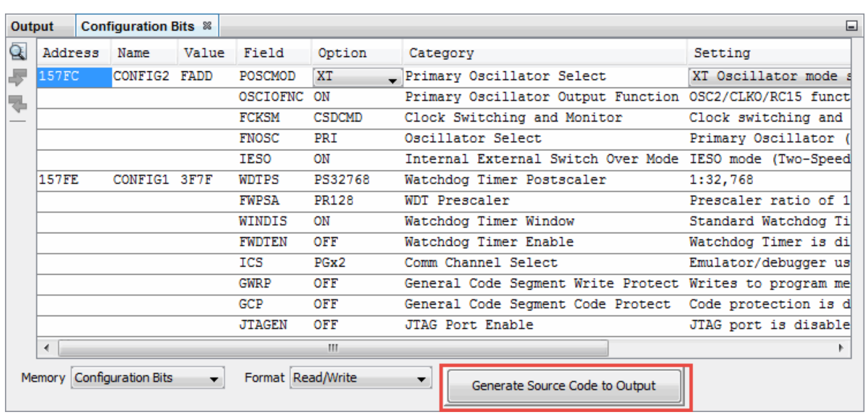
For more about configuration bits that are available for each device, see the following file in the location where MPLAB XC16 was installed:

MPLAB XC16 Installation Directory/docs/config_index.html

1.2.2 HOW TO SET CONFIGURATION BITS

In MPLAB X IDE, you can use the Configuration Bits window to view and set these bits. Select *Window>PIC Memory Views>Configuration Bits* to open this window.

FIGURE 1: CONFIGURATION WINDOW



Once you have the settings you want, click **Generate Source Code to Output** and then copy the `pragma` directives from the Output window into your code, as was done in the example code.

1.3 Port Access

Digital I/O device pins may be multiplexed with peripheral I/O pins. To ensure that you are using digital I/O only, disable the other peripheral(s). Do this by using the pre-defined C variables that represent the peripheral registers and bits. These variables are listed in the device-specific header file in the compiler `include` directory. To determine which peripherals share which pins, refer to your device data sheet.

For the example in this section, Port A pins are multiplexed with peripherals that are disabled by default. The only issue is that the pins default to analog; so, you will need to set them to digital I/O:

```
AD1PCFG = 0xFFFF;  // set to digital I/O (not analog)
```

A device pin is connected to either a digital I/O port (`PORT`) or latch (`LAT`) register in the device. For the example, `LATA` is used. The variable `portValue` is assigned a value that is then assigned to the latch:

```
LATA = portValue;   // write to port latch
```

In addition, there is a register for specifying the directionality of the pin – either input or output – called a TRIS register. For the example in this section, `TRISD` and `TRISB` are used. Setting a bit to 0 makes the pin an output, and setting a bit to 1 makes the pin an input. For this example:

```
TRISA = 0x0000;   // set all port bits to be output
```

2. FLASH LEDs USING `_delay()` FUNCTION

This example is a modification of the previous code. Instead of just turning on LEDs, this code will flash alternating LEDs.

```
#include <xc.h>
#include <libpic30.h>        ◄──── see Section 2.1

// PIC24FJ128GA010 Configuration Bit Settings
// For more on Configuration Bits, consult your device data sheet

// CONFIG2
#pragma config POSCMOD = XT   // XT Oscillator mode selected
#pragma config OSCIOFNC = ON  // OSC2/CLKO/RC15 as port I/O (RC15)
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor disabled
#pragma config FNOSC = PRI    // Primary Oscillator (XT, HS, EC)
#pragma config IESO = ON      // Int Ext Switch Over Mode enabled

// CONFIG1
#pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)
#pragma config FWPSA = PR128   // WDT Prescaler (1:128)
#pragma config WINDIS = ON     // Watchdog Timer Window Mode disabled
#pragma config FWDTEN = OFF    // Watchdog Timer disabled
#pragma config ICS = PGx2      // Emulator/debugger uses EMUC2/EMUD2
#pragma config GWRP = OFF      // Writes to program memory allowed
#pragma config GCP = OFF       // Code protection is disabled
#pragma config JTAGEN = OFF    // JTAG port is disabled

#define LEDS_ON_OFF 0x55
#define LEDS_OFF_ON 0xAA
#define IC_DELAY 1500000

int main(void) {

    unsigned char portValue;

     // Port A access
     AD1PCFG = 0xFFFF; // set to digital I/O (not analog)
     TRISA = 0x0000;   // set all port bits to be output

     while(1) {       ◄──── see Section 2.2

         portValue = LEDS_ON_OFF;
         LATA = portValue; // write to port latch

         // delay value change      ◄──── see Section 2.3

         __delay32(IC_DELAY);    // delay in instruction cycles

         portValue = LEDS_OFF_ON;
         LATA = portValue; // write to port latch

         __delay32(IC_DELAY);    // delay in instruction cycles

     }
     return -1;
}
```

2.1 Library Header File

In this example, the delay32 function from the libpic30 compiler library is used. To access this library, `libpic30.h` must be included.

2.2 The `while()` Loop and Variable Values

To make the LEDs on Port A change, a variable `portValue` is assigned a value in the first part of the loop, and a complementary value in the second part of the loop. To perform the loop, `while(1) { }` was used.

If the main function returns, it means there was an error, as the while loop should not normally end. There, a `-1` is returned.

2.3 The `_delay()` Function

Because the speed of execution will, in most cases, cause the LEDs to flash faster than the eye can see, execution needs to be slowed. `__delay32()` is a library function that can be used by compiler.

For more details on the delay function, see the *16-Bit Language Tools Libraries Reference Manual* (DS50001456).

3. COUNT UP ON LEDs USING INTERRUPTS AS DELAY

This example is a modification of the previous code. Although the delay function in the previous example was useful in slowing down loop execution, it created dead time in the program. To avoid this, a timer interrupt can be used.

```
#include <xc.h>

// PIC24FJ128GA010 Configuration Bit Settings
// For more on Configuration Bits, consult your device data sheet

// CONFIG2
#pragma config POSCMOD = XT   // XT Oscillator mode selected
#pragma config OSCIOFNC = ON  // OSC2/CLKO/RC15 as port I/O (RC15)
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor disabled
#pragma config FNOSC = PRI    // Primary Oscillator (XT, HS, EC)
#pragma config IESO = ON      // Int Ext Switch Over Mode enabled

// CONFIG1
#pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)
#pragma config FWPSA = PR128   // WDT Prescaler (1:128)
#pragma config WINDIS = ON     // Watchdog Timer Window Mode disabled
#pragma config FWDTEN = OFF    // Watchdog Timer disabled
#pragma config ICS = PGx2      // Emulator/debugger uses EMUC2/EMUD2
#pragma config GWRP = OFF      // Writes to program memory allowed
#pragma config GCP = OFF       // Code protection is disabled
#pragma config JTAGEN = OFF    // JTAG port is disabled

// Interrupt function      <------ see Section 3.1

void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void){
    // static variable for permanent storage duration
    static unsigned char portValue = 0;
    // write to port latch
    LATA = portValue++;
    // clear this interrupt condition
    _T1IF = 0;
}

int main(void) {

    // Port A access
    AD1PCFG = 0xFFFF; // set to digital I/O (not analog)
    TRISA = 0x0000;   // set all port bits to be output

    // Timer1 setup      <------ see Section 3.2

    T1CON = 0x8010; // timer 1 on, prescaler 1:8, internal clock
    _T1IE = 1; // enable interrupts for timer 1
    _T1IP = 0x001; // set interrupt priority (lowest)

    while(1);

   return -1;
}
```

3.1 The Interrupt Function **isr()**

Functions are made into interrupt functions by using the interrupt attribute. Program Space Visibility (PSV) should be specified also, and for this simple example no PSV is used. For more on PSV, see the "*MPLAB XC16 C Compiler User's Guide*" (DS50002071).

The primary interrupt vector specific to Timer 1 is used, _T1Interrupt. Interrupt Vector Tables for each device are provided in the compiler install docs directory.

Within the interrupt function, the counter portValue is incremented when Timer1 generates an interrupt.

3.2 Timer1 Setup

Code also needs to be added to the main routine to turn on and set up the timer, enable timer interrupts, and change the latch assignment, now that the variable value changes are performed in the interrupt service routine.

4 DISPLAY POTENTIOMETER VALUES ON LEDS USING AN ADC

This example uses the same device and the Port A LEDs as the previous example. However, in this example, values from a potentiometer on the demo board provide Analog-to-Digital Converter (ADC) input through Port B that is converted and displayed on the LEDs.

Instead of generating code by hand, the MPLAB Code Configurator (MCC) v2.25 is used. The MCC is a plug-in available for installation under the MPLAB XIDE menu *Tools>Plugins*, **Available Plugins** tab. See MPLAB X IDE Help for more on how to install plugins.

For MCC installation information and the *MPLAB® Code Configurator User's Guide* (DS40001725), go to the MPLAB Code Configurator web page at:

http://www.microchip.com/mcc

For this example, the MCC was set up as shown in the following figures.

FIGURE 2: ADC PROJECT RESOURCES

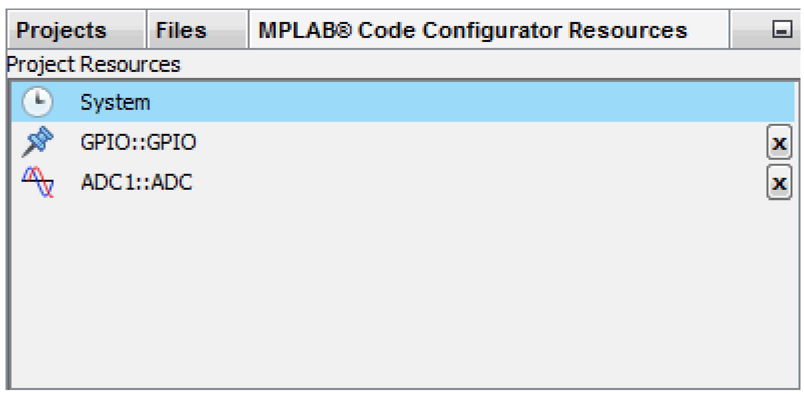


FIGURE 3: ADC SYSTEM PROJECT RESOURCE CONFIGURATION

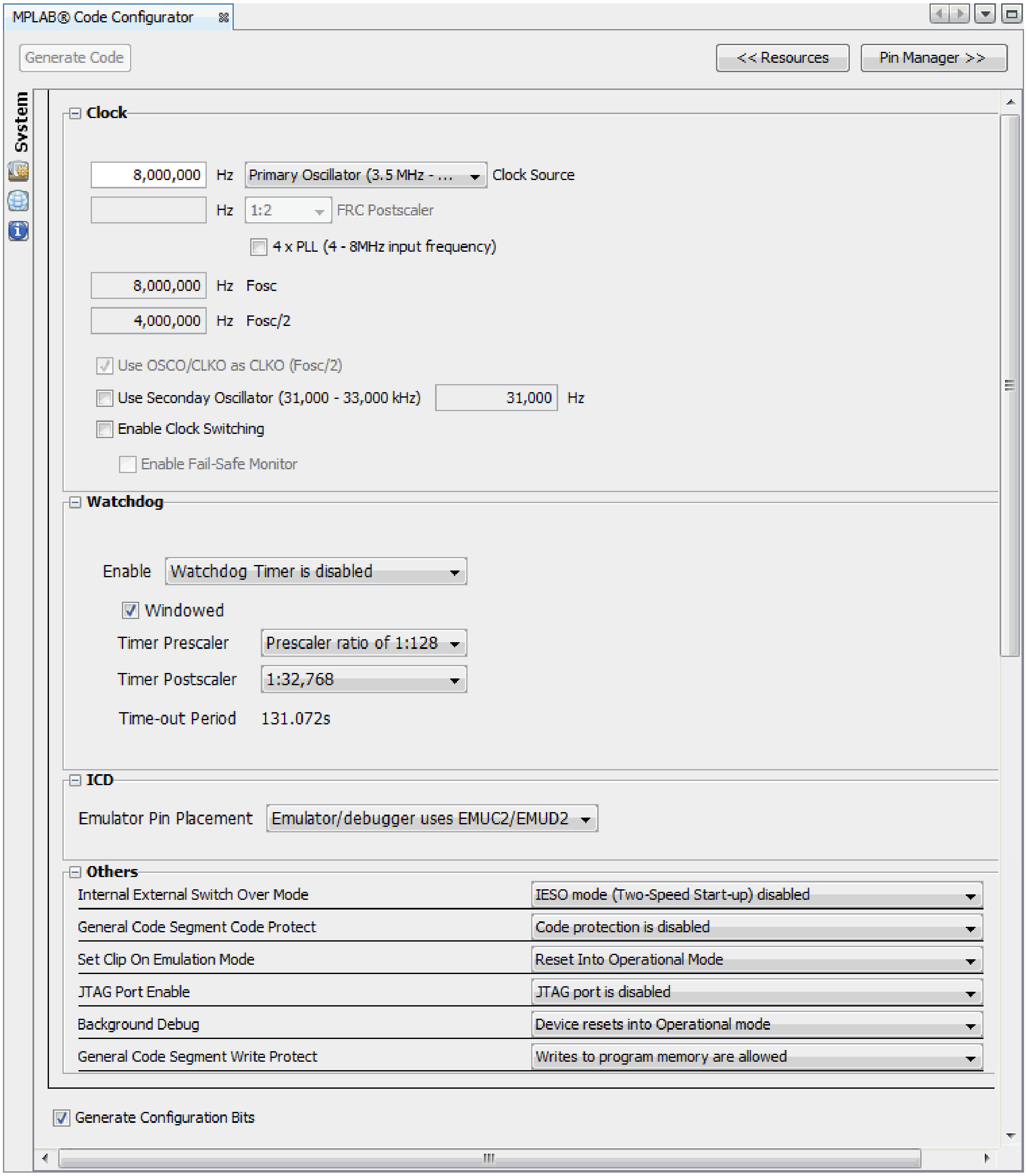


FIGURE 4: ADC PROJECT RESOURCE CONFIGURATION

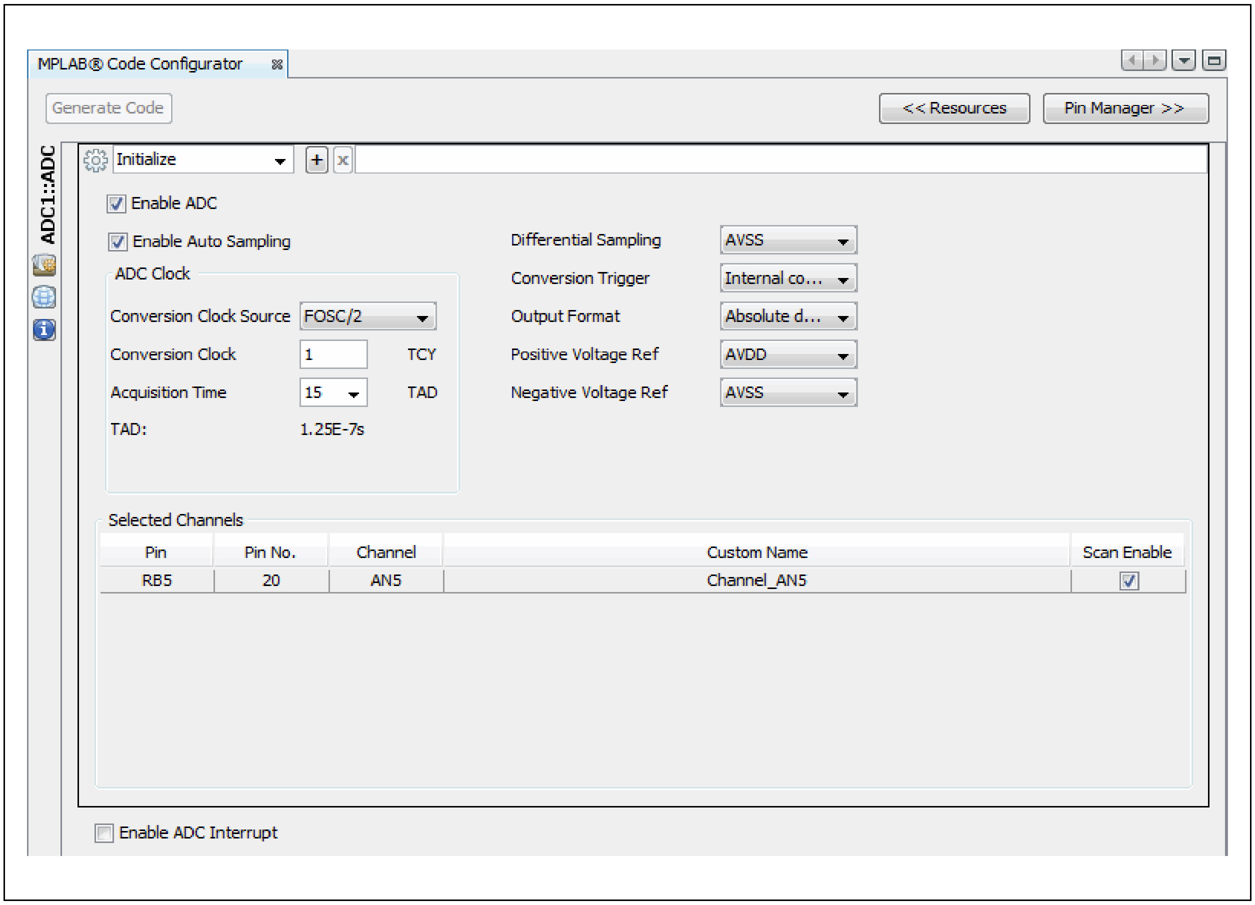


FIGURE 5: ADC PROJECT RESOURCE PIN TABLE

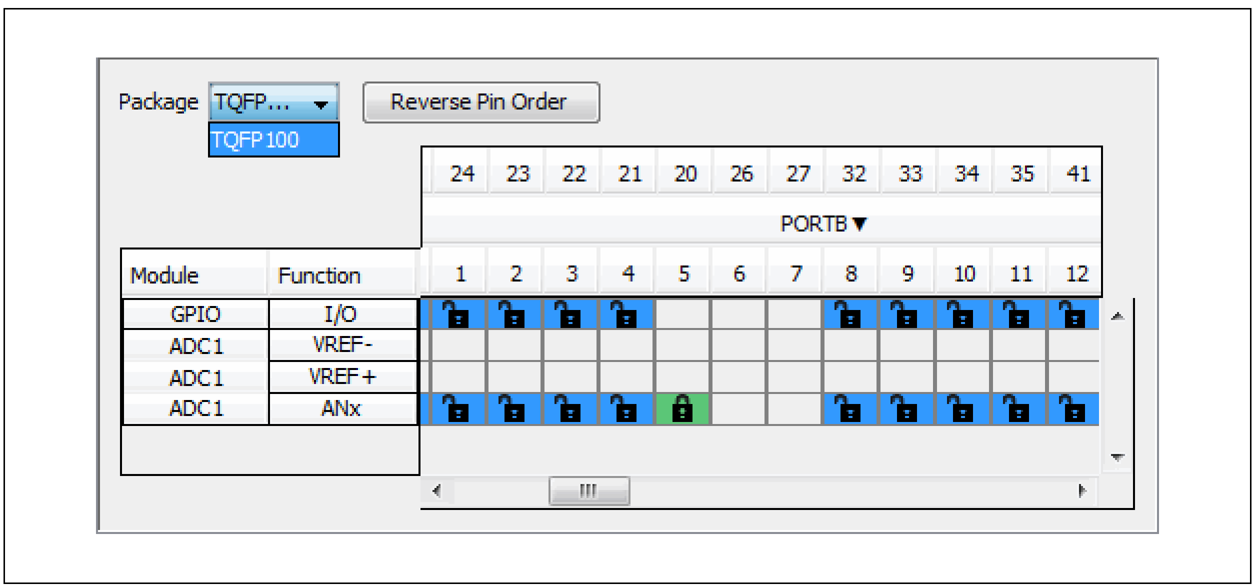


FIGURE 6: ADC GPIO PROJECT RESOURCE CONFIGURATION

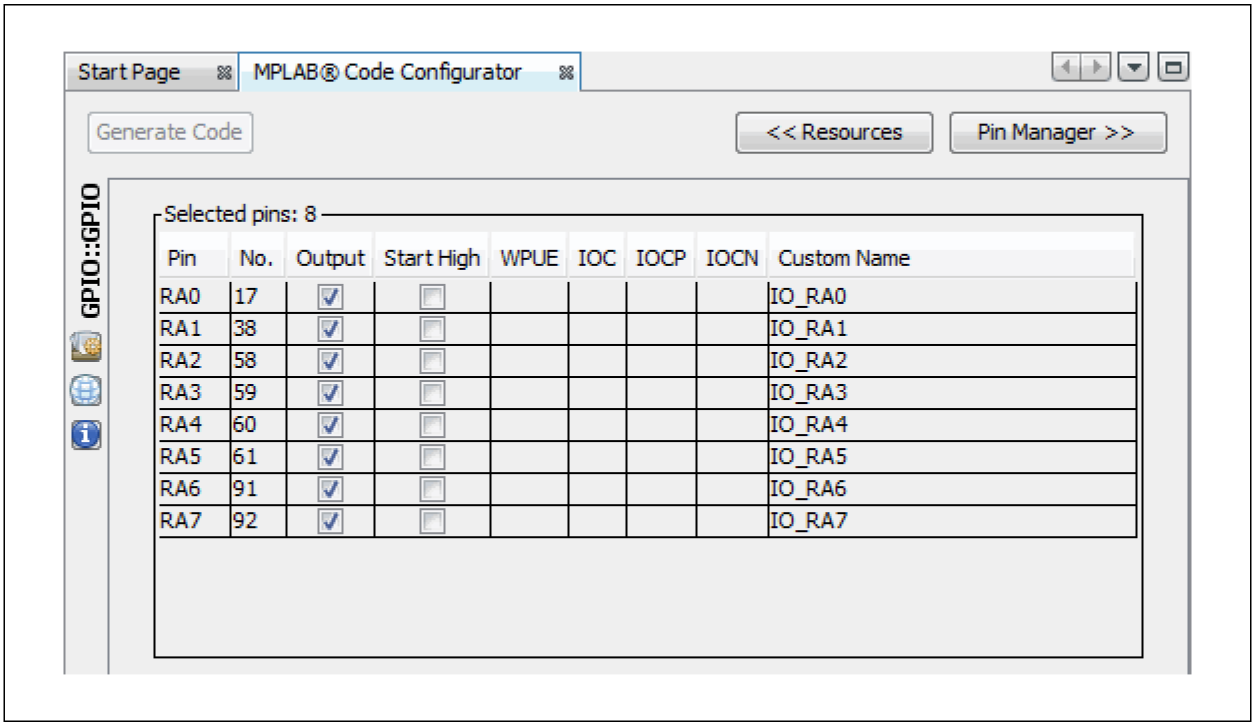


FIGURE 7: ADC GPIO PROJECT RESOURCE PIN TABLE

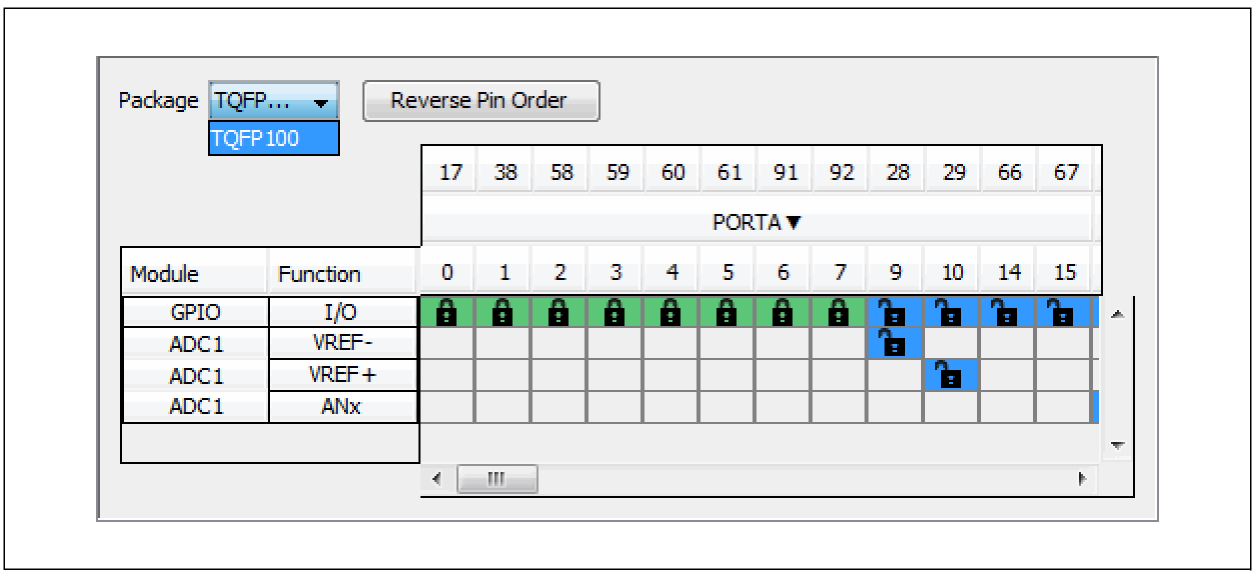
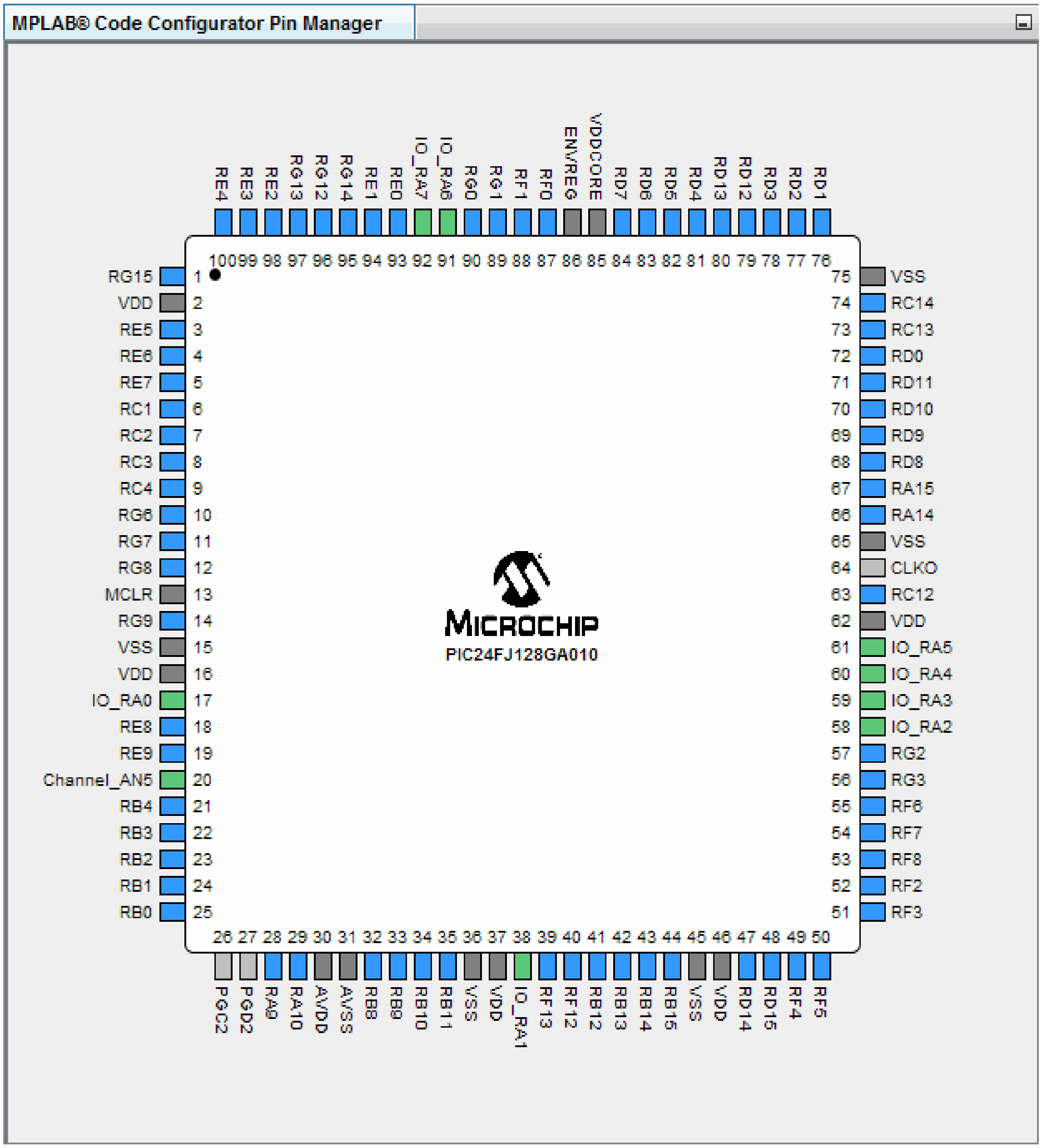


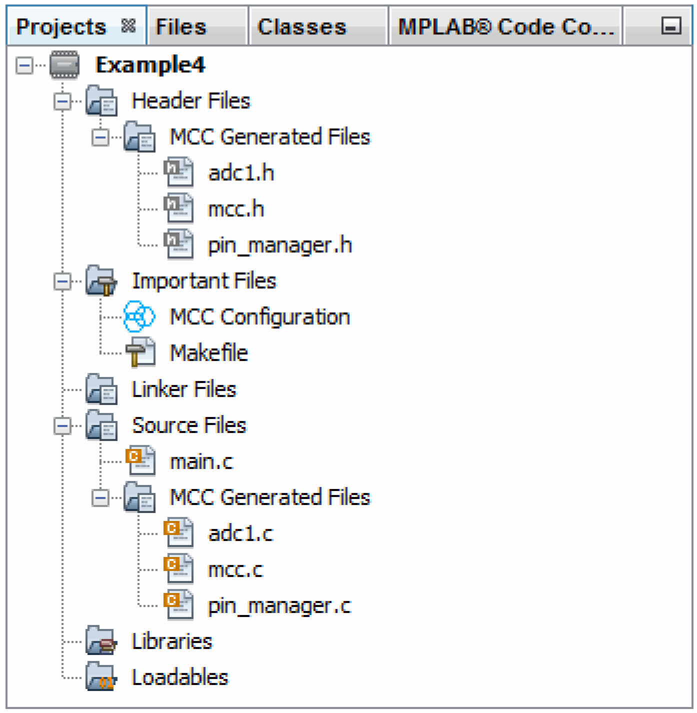
FIGURE 8: ADC GPIO PROJECT RESOURCE PIN SELECTION



When the code is configured as shown in the previous figures, click the **Generate Code** button on the MCC tab. Code generated by the MCC is modular. Therefore main, system and peripheral code are all in individual files. Also, each peripheral has its own header file.

Note:	After you create a main project, you can either add a `main.c` template file before using MCC or you can let MCC add `main.c` for you.

FIGURE 9: ADC PROJECT TREE FOR CODE GENERATED BY MCC



Editing of `main.c` is always required to add functionality to your program. Review the generated files to find any functions or macros you may need in your code.

4.1 `main.c` Modified Code

The `main.c` template file has been edited as shown below. Some comments have been removed as described in < >. Code added to `main()` is in red.

```
/**
  Generated Main Source File

<See generated main.c file for file information.>
 */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.
All rights reserved.

<See generated main.c file for additional copyright information.>
 */

#include "mcc_generated_files/mcc.h"

unsigned int value = 0;

/*
                         Main application
 */
int main(void) {
    // initialize the device
    SYSTEM_Initialize();

    while (1) {

        // Wait for conversion           see Section 4.2
        // and then get result
        while(!ADC1_IsConversionComplete());
        value = ADC1_ConversionResultGet();

        // Shift for MSb
        value = value >> 2;

        // Write to Port Latch/LEDs           see Section 4.3
        LATA = value;

    }
    return -1;
}
/**
 End of File
 */
```

4.2 ADC Conversion and Result

MCC sets AD1CON1 bits to turn on the ADC, use automatic sample acquisition, and use an internal counter to end sampling and start conversion. Therefore `main()` code only needs to wait for the conversion to end and get the result.

From the `adc1.c` module, use the functions:

```
bool ADC1_IsConversionComplete(void)
uint16_t ADC1_ConversionResultGet(void)
```

For information on setting up other ADC features, see the *dsPIC33/PIC24 Family Reference Manual,* "Section 17. 10-bit Analog-to-Digital Converter (ADC)" (DS61104).

Since only 8 LEDs are available, and the ADC conversion result is 10-bit, the conversion result in the variable `value` is shifted to display the most significant bits. Some resolution will be lost.

4.3 Write to Port Latch and LEDs

The ADC conversion result `value` is displayed on the Port A LEDs.

5. DISPLAY EEPROM DATA VALUES ON LEDS

This example uses another Microchip device, the PIC24F32KA304 MCU, with the Explorer 16 board, to demonstrate how to write to and read from EEPROM Data (EEData). Read values are displayed on LEDs accessed from three ports.

MPLAB Code Configurator (MCC) v2.25 is used to generate some of the code. To find out how to install and get the user's guide for MCC, see:**Section 4 "Display Potentiometer Values on LEDs Using an ADC"**.

For this example, the MCC GUI was used to set up the System (oscillator speed, configuration bits, etc.) and the General Purpose I/O (GPIO) for Ports A, B, and C (Figure 10). However, at this time, there is no EEData device resource available for 16-bit devices.

Code for using the EEData module was found in the device data sheet and the dsPIC33/PIC24 Family Reference Manual, "Section 5. "Data EEPROM", both located on the device web page:

http://www.microchip.com/PIC24F32KA304

FIGURE 10: EEDATA PROJECT RESOURCES

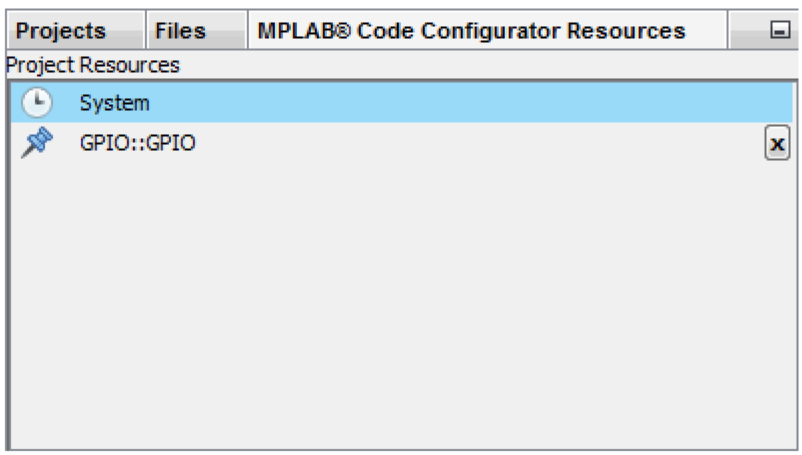


FIGURE 11: EEDATA SYSTEM PROJECT RESOURCE CONFIGURATION

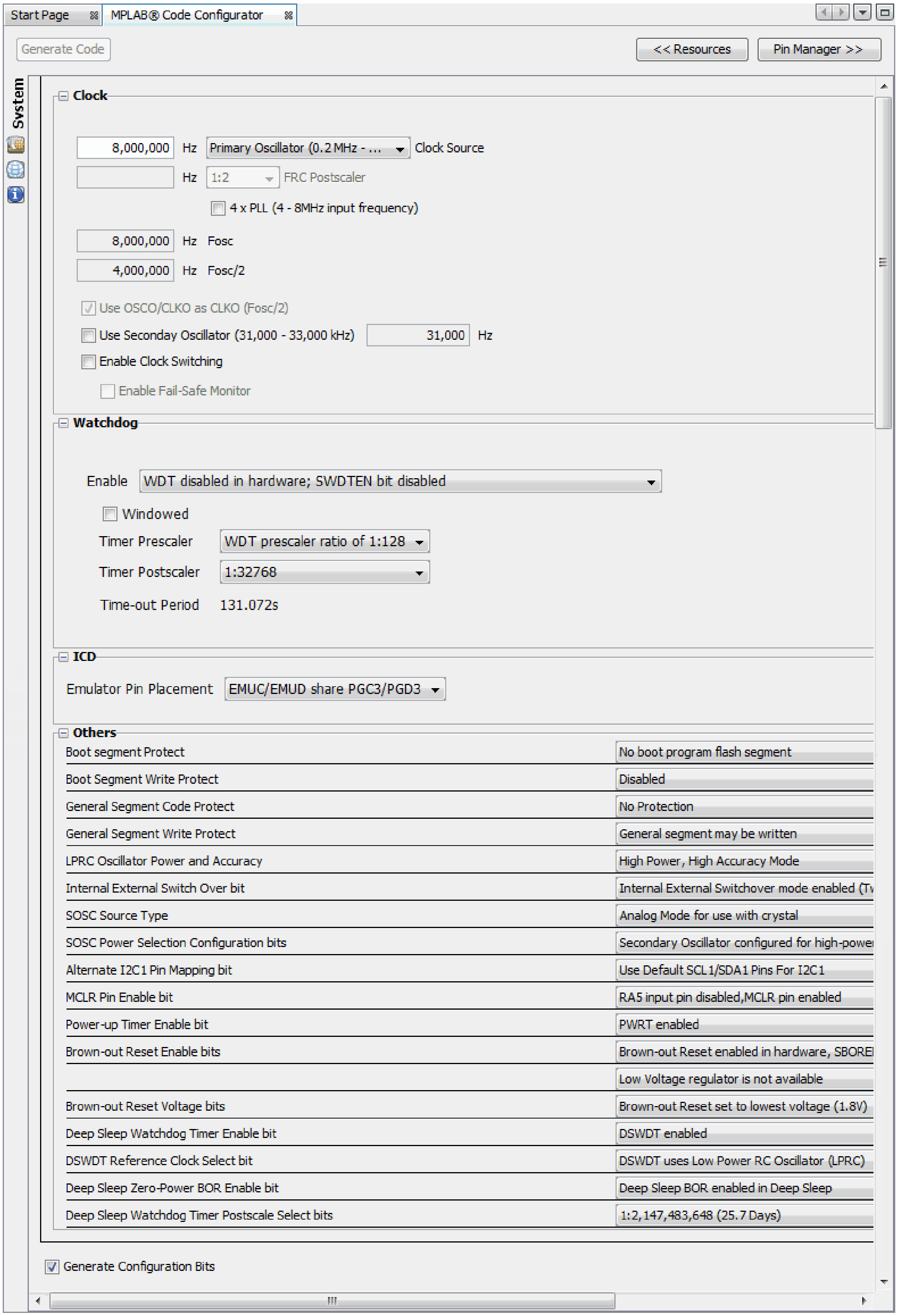


FIGURE 12: EEDATA GPIO PROJECT RESOURCE CONFIGURATION

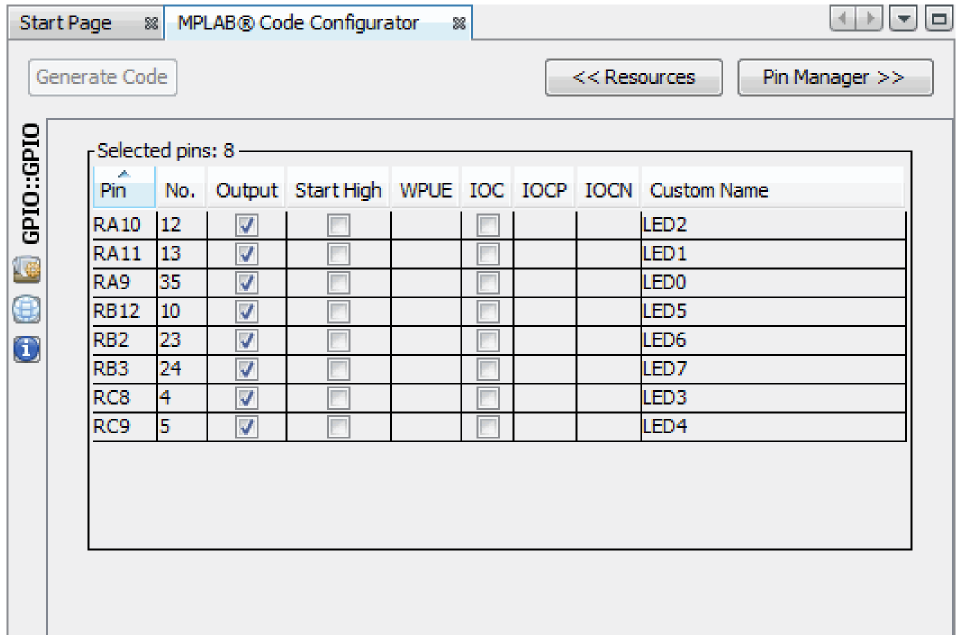


FIGURE 13: EEDATA GPIO PROJECT RESOURCE PIN SELECTION

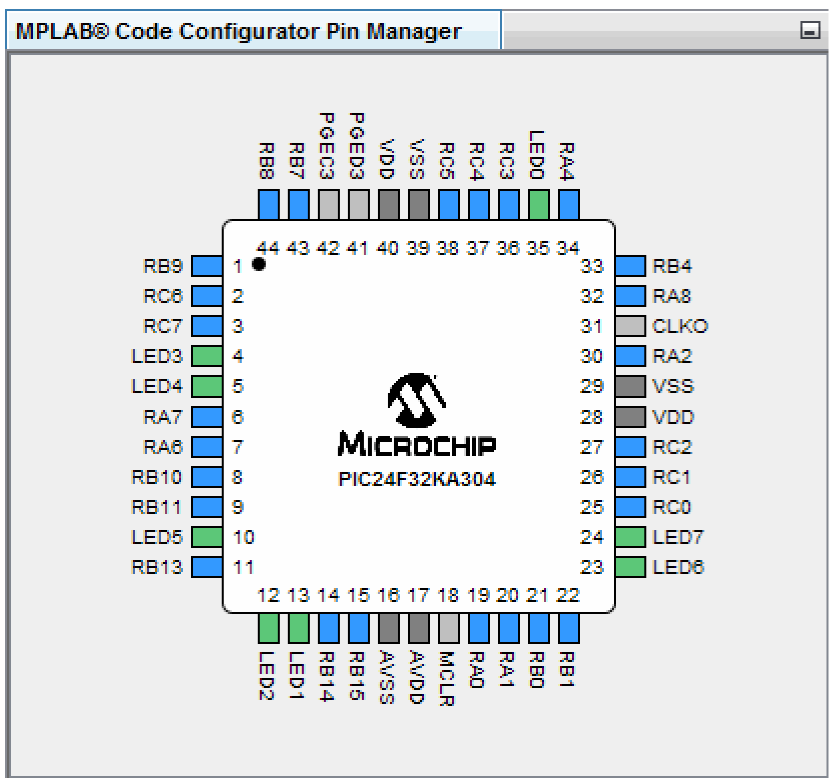
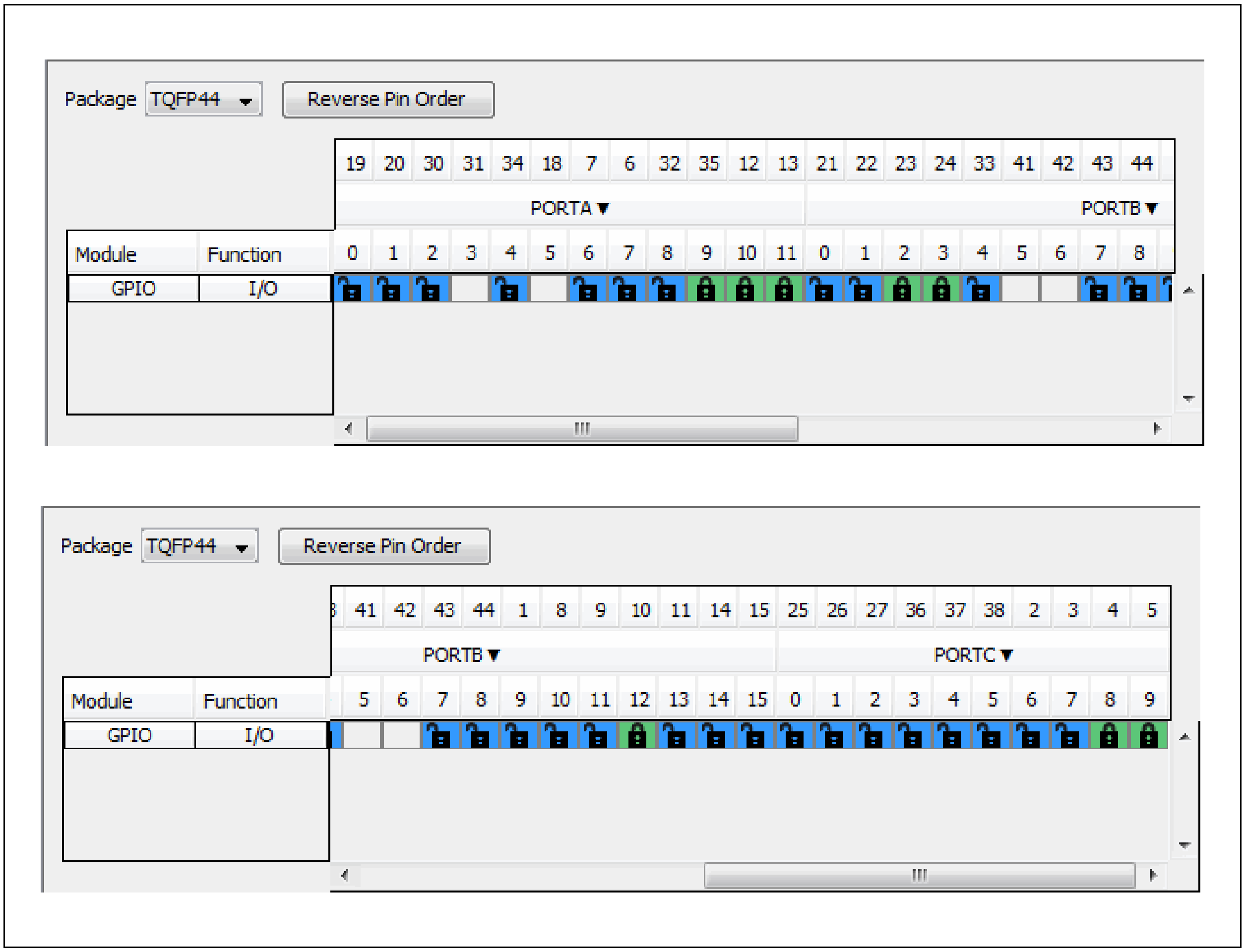


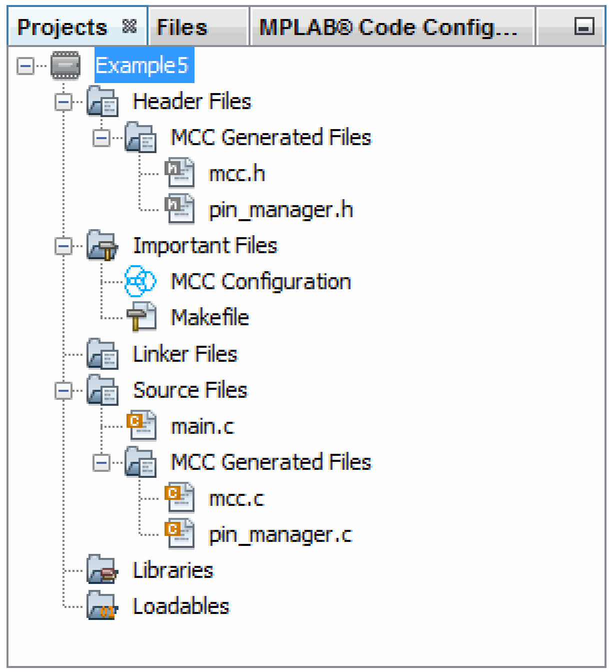
FIGURE 14: EEDATA GPIO PROJECT RESOURCE PIN TABLE



After your code is configured (as shown in the previous figures), click the **Generate Code** button on the MCC tab. Code generated by the MCC is modular. Therefore main, system, and peripheral code are all in individual files. Also, each peripheral has its own header file.

Note:	After you create a main project, you can either add a `main.c` template file before using MCC or you can let MCC add `main.c` for you.

FIGURE 15: EEDATA PROJECT TREE FOR CODE GENERATED BY MCC



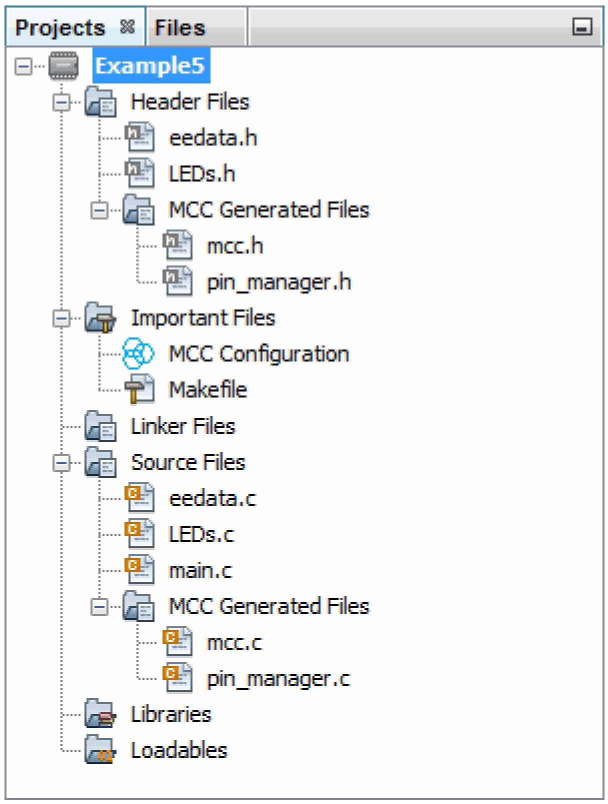
The GPIO generated files default to analog input, so they must be changed to digital input in the `pin_manager.c` file (**Section 5.1**).

In addition, because LED connections are not to one port but spread across three, an additional type definition and code to assign the port pins to the correct LED values are needed. A header file, `LEDs.h` (**Section 5.2**), and a C file, `LEDs.c` (**Section 5.3**), have been added to the project.

As previously mentioned, there is no EEData device resource available in MCC, so code needs to be added by hand. A header file `eedata.h` (**Section 5.4**) and a C file, `eedata.c` (**Section 5.5**), have been added to the project.

The final project tree will appear as shown in Figure 16.

FIGURE 16: EEDATA PROJECT TREE - FINAL



Editing of `main.c` is always required to add functionality to your program (**Section 5.6**). Review the generated files and additional files to find any functions or macros you may need in your code.

5.1 pin_manager.c Modified Code

The main.c template file has been edited as shown below. Some comments and generated content have been removed as described in < >. Code changed is in red.

```
/**
  System Interrupts Generated Driver File

<See generated pin_manager.c for file information.>

 */
/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc. All
rights reserved.

<See generated pin_manager.c for additional copyright information.>
 */

/**
    Section: Includes
 */
#include <xc.h>
#include "pin_manager.h"

/**
    void PIN_MANAGER_Initialize(void)
 */
void PIN_MANAGER_Initialize(void) {

<See generated pin_manager.c for port setup information.>

/*********************************************************************
     * Setting the Analog/Digital Configuration SFR

*********************************************************************/
    ANSA = 0x0;
    ANSB = 0x0;
    ANSC = 0x0;

}
```

5.2 LEDs.h Code

Some comments have been removed as described in < >.

```
/*---------------------------------------------------------------------
 * PICF32KA304 LEDs header
 *
 * (c) Copyright 1999-2015 Microchip Technology, All rights reserved
 *
<See generated header files for additional copyright information.>
 */

/*********************************************************************
 * Union of structures to hold value for display on LEDs
 * LAT_LEDx - bit fields of value
 * w - entire value
 ********************************************************************/
typedef union {
    struct {
      unsigned LAT_LED0:1;
      unsigned LAT_LED1:1;
      unsigned LAT_LED2:1;
      unsigned LAT_LED3:1;
      unsigned LAT_LED4:1;
      unsigned LAT_LED5:1;
      unsigned LAT_LED6:1;
      unsigned LAT_LED7:1;
    };
    struct {
      unsigned w:16;
    };
} LAT_LEDSBITS;
extern volatile LAT_LEDSBITS LAT_LEDSbits;

/* LAT_LEDSBITS */
#define _LED0 LAT_LEDSbits.LAT_LED0
#define _LED1 LAT_LEDSbits.LAT_LED1
#define _LED2 LAT_LEDSbits.LAT_LED2
#define _LED3 LAT_LEDSbits.LAT_LED3
#define _LED4 LAT_LEDSbits.LAT_LED4
#define _LED5 LAT_LEDSbits.LAT_LED5
#define _LED6 LAT_LEDSbits.LAT_LED6
#define _LED7 LAT_LEDSbits.LAT_LED7
#define _LEDS LAT_LEDSbits.w

/*********************************************************************
 * Function: DisplayValueOnLEDs
 * Precondition: None.
 * Overview: Display input value on Explorer 16 LEDs
 * Input: Value to display
 * Output: None.
 ********************************************************************/
void DisplayValueOnLEDs(unsigned int value);
/**
 End of File
 */
```


5.3 LEDs.c Code

Some comments have been removed as described in < >.

```
/**
  Display on LEDs Source File

<See LEDs.c for file description information.>

 */
/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc. All
rights reserved.

<See generated header files for additional copyright information.>
 */

#include "mcc_generated_files/mcc.h"
#include "LEDs.h"

volatile LAT_LEDSBITS LAT_LEDSbits;

/*********************************************************************
 * Function: DisplayValueOnLEDs
 * Precondition: None.
 * Overview: Display input value on Explorer 16 LEDs
 * Input: Value to display
 * Output: None.
 ********************************************************************/
void DisplayValueOnLEDs(unsigned int value) {

    _LEDS = value;

    _LATA9  = _LED0;
    _LATA10 = _LED1;
    _LATA11 = _LED2;
    _LATC8  = _LED3;
    _LATC9  = _LED4;
    _LATB12 = _LED5;
    _LATB2  = _LED6;
    _LATB3  = _LED7;

}

/**
 End of File
 */
```

5.4 eedata.h Code

Some comments have been removed as described in < >.

```
/*-------------------------------------------------------------------
 * PICF32KA304 Data EEPROM header
 *
 * (c) Copyright 1999-2015 Microchip Technology, All rights reserved
 *
<See generated header files for additional copyright information.>
 */

/********************************************************************
 * Function: EEData_WTL
 * Precondition: None.
 * Overview: Write one word of EEData
 * Input: Action to take: Erase or Write, Data to write
 * Output: None.
 *******************************************************************/
void EEData_WTL(unsigned int action, unsigned int data);

/********************************************************************
 * Function: EEData_Erase
 * Precondition: None.
 * Overview: Set up erase of one word of EEData
 * Input: None.
 * Output: None.
 *******************************************************************/
void EEData_Erase(void);

/********************************************************************
 * Function: EEData_Write
 * Precondition: None.
 * Overview: Set up write of one word of EEData
 * Input: Data to write
 * Output: None.
 *******************************************************************/
void EEData_Write(unsigned int data);

/********************************************************************
 * Function: EEData_Read
 * Precondition: None.
 * Overview: Read one word of EEData
 * Input: None.
 * Output: Value read from EEData
 *******************************************************************/
unsigned int EEData_Read(void);

/**
 End of File
 */
```

5.5 eedata.c Code

Some comments have been removed as described in < >.

```
/**
  Data EEPROM Write and Read

<See eedata.c for file description information.>

 */
/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc. All
rights reserved.

<See generated header files for additional copyright information.>
 */

#include <xc.h>
#include "eedata.h"

#define ERASE_EEWORD 0x4058
#define WRITE_EEWORD 0x4004

int __attribute__ ((space(eedata))) eeData = 0x0;
unsigned int offset = 0x0;

/*********************************************************************
 * Function: EEData_WTL
 * Precondition: None.
 * Overview: Write one word of EEData
 * Input: Action to take: Erase or Write, Data to write
 * Output: None.
 ********************************************************************/
void EEData_WTL(unsigned int action, unsigned int data) {

    // Set up NVMCON to write one word of data EEPROM
    NVMCON = action;

    // Set up a pointer to the EEPROM location to be written
    TBLPAG = __builtin_tblpage(&eeData);
    offset = __builtin_tbloffset(&eeData);
    __builtin_tblwtl(offset, data);

    // Issue Unlock Sequence & Start Write Cycle
    __builtin_write_NVM();

    // Wait for completion
    while(NVMCONbits.WR);
}

/*********************************************************************
 * Function: EEData_Erase
 * Precondition: None.
 * Overview: Set up erase of one word of EEData
 * Input: None.
 * Output: None.
 ********************************************************************/
void EEData_Erase(void) {

    EEData_WTL(ERASE_EEWORD, 0);
}
```

```
/*********************************************************************
 * Function: EEData_Write
 * Precondition: None.
 * Overview: Set up write of one word of EEData
 * Input: Data to write
 * Output: None.
 ********************************************************************/
void EEData_Write(unsigned int data) {

    EEData_WTL(WRITE_EEWORD, data);
}

/*********************************************************************
 * Function: EEData_Read
 * Precondition: None.
 * Overview: Read one word of EEData
 * Input: None.
 * Output: Value read from EEData
 ********************************************************************/
unsigned int EEData_Read(void) {

    // Set up a pointer to the EEPROM location to be read
    TBLPAG = __builtin_tblpage(&eeData);
    offset = __builtin_tbloffset(&eeData);

    // Read the EEPROM data
    return __builtin_tblrdl(offset);
}

/**
 End of File
 */
```

5.6 main.c Modified Code

The `main.c` template file has been edited as shown below. Some comments have been removed as described in < >. Code added is in red.

```
/**
  Generated Main Source File

<See generated main.c for file information.>
 */

/*
Copyright (c) 2013 - 2015 released Microchip Technology Inc.
All rights reserved.

<See generated main.c for additional copyright information.>
 */

#include "mcc_generated_files/mcc.h"
#include "eedata.h"
#include "LEDs.h"
#include "libpic30.h"

#define IC_DELAY 1000000

unsigned int data_write = 0x0;
unsigned int data_read = 0x0;

/*
                         Main application
 */
int main(void) {
    // initialize the device
    SYSTEM_Initialize();

    while (1) {

        data_write++;

        // Erase one word of data EEPROM        <-- see Section 5.7
        EEData_Erase();

        // Write one word of data EEPROM
        EEData_Write(data_write);

        // Read one word of data EEPROM         <-- see Section 5.8
        data_read = EEData_Read();

        // Display result on LEDs               <-- see Section 5.9
        DisplayValueOnLEDs(data_read);

        // Delay change on LEDs so visible
        __delay32(IC_DELAY); // delay in instruction cycles

    }

    return -1;
}
/**
 End of File
 */
```

5.7 Erase and Write to EEData

To write a single word in the EEData, the following sequence must be followed:

1. Erase one data EEPROM word.
2. Write the data word into the data EEPROM latch.
3. Program the data word into the EEPROM.

The code to erase and write one word to EEData is found in `eedata.c` (**Section 5.5**).

For the PIC24F32KA304 devices, a key sequence needs to be written to NVMKEY (in NVMCON) before EEData can be erased or written.

Built-in functions are used to simplify coding:

- `unsigned int __builtin_tblpage(const void *p);`
- `unsigned int __builtin_tbloffset(const void *p);`
- `void __builtin_tblwtl(unsigned int offset, unsigned int data);`
- `void __builtin_write_NVM(void);`

Details on these functions may be found in the *MPLAB XC16 C Compiler User's Guide* (DS50002071), Appendix G. "Built-in Functions".

5.8 Read from EEData

For this example, after EEData is written, the word of EEData is read.

The code to read one word to EEData is found in `eedata.c` (**Section 5.5**).

Again, built-in functions are used to simplify coding:

- `unsigned int __builtin_tblpage(const void *p);`
- `unsigned int __builtin_tbloffset(const void *p);`
- `unsigned int __builtin_tblrdl(unsigned int offset);`

Details on these functions may be found in the *MPLAB XC16 C Compiler User's Guide* (DS50002071), Appendix G. "Built-in Functions".

5.9 Display Data on LEDs and Delay

Displaying the data on the demo board LEDs is more involved for this device, as three ports provide connections to the LEDs. Therefore union and structure data types are used so that the whole data value can be assigned (`LAT_LEDSbits.w`) and then individual bits may be accessed so they can be assigned to the correct port pins for display (e.g., `LATAbits.LATA9 = LAT_LEDSbits.LAT_LED0`).

The code creating the union and structures is found in `LEDs.h` (**Section 5.2**).

The code assigning the port pins to LED values is found in `LEDs.c` (**Section 5.5**).

Because the speed of execution will, in most cases, cause the LEDs to flash faster than the eye can see, the `_delay()` function is used again (as in **Section 2.**) to slow execution.

A. RUN CODE IN MPLAB X IDE

First, create a project:

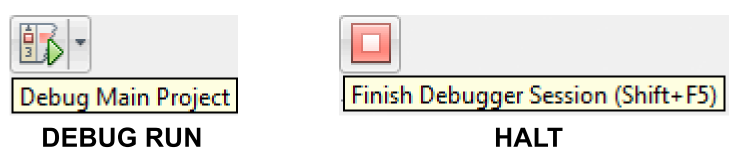
1. Launch MPLAB X IDE.
2. From the IDE, launch the New Project Wizard (*File>New Project*).
3. Follow the screens to create a new project:
   a) **Choose Project:** Select "Microchip Embedded", and then select "Standalone Project".
   b) **Select Device:** Select the example device.
   c) **Select Header:** None.
   d) **Select Tool:** Select your hardware debug tool by serial number (SN), SNxxxxxx. If you do not see an SN under your debug tool name, ensure that your debug tool is correctly installed. See your debug tool documentation for details.
   e) **Select Plugin Board:** None.
   f) **Select Compiler:** Select XC16 (*latest version number*) [`bin` *location*]. If you do not see a compiler under XC16, ensure the compiler is correctly installed and that MPLAB X IDE is aware of it. Select *Tools>Options*,, click the **Embedded** button on the **Build Tools** tab, and look for your compiler. See MPLAB XC16 and MPLAB X IDE documentation for details
   g) **Select Project Name and Folder:** Name the project.

Now, create a file to hold the example code (unless you have used MCC):

1. Right click on the project name in the Projects window. Select *New>Empty File*. The New Empty File dialog will open.
2. Under "File name", enter a name.
3. Click **Finish**.
4. Cut and paste the example code from this user's guide into the empty editor window and select *File>Save*.

Build, download to a device, and execute the code by selecting to Debug Run your code. You will see every other LED lit on the demo board. Click Halt to end execution.

FIGURE 17: TOOLBAR ICONS



B. GET SOFTWARE AND HARDWARE

For the MPLAB XC16 projects in this document, the Explorer 16 board with a PIC24F PIM is powered from a 9V external power supply and uses standard (ICSP™) communications. MPLAB X IDE was used for development.

B.1 Get MPLAB X IDE and MPLAB XC16 C Compiler

MPLAB X IDE v3.15 and later can be found at:

http://www.microchip.com/mplabx

The MPLAB XC16 C Compiler v1.25 and later can be found at:

http://www.microchip.com/mplabxc

B.2 Get the MPLAB Code Configurator (MCC)

The MCC v2.25 and later can be found at:

http://www.microchip.com/mcc

B.3 Get PIC® MCU Plug-in Module (PIM)

The PIC MCU PIMs used in the examples are available at the following locations on the Microchip Technology web site:

PIC24FJ128GA010: http://www.microchip.com/MA240011

PIC24F32KA304: http://www.microchip.com/MA240022

B.4 Get and Set Up the Explorer 16 Board

The Explorer 16 development board and documentation are available on the web site:

http://www.microchip.com/explorer16

Jumpers and switches were set up as shown in the following table.

TABLE 1-1: JUMPER/SWITCH SELECTS FOR PROJECTS

Jumper/ Switch	Selection	Description
J2	Closed	**Closed**: Enable LEDs on Port A <RA0:7> **Open**: Disable LEDs
J7	PIC24	**PIC24**: The debug tool communicates directly with PGCx/PGDx or EMUCx/EMUDx of the PIM or on-board device (determined by S2). **F4550**: The debug tool communicates with the on-board PIC18LF4550 USB device.
S2	PIM	**PIM** or **PIC** selection

B.5 Get Microchip Debug Tools

Emulators and Debuggers may be found on the Development Tools web page:

http://www.microchip.com/devtools

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

ISBN: 978-1-5224-0099-8

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15