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 using the Common Code Interface (CCI). For more on CCI, see the *MPLAB XC16 C Compiler User's Guide* (DS50002071).

Some knowledge of microcontrollers and the C programming language is necessary to use them.

1. TURN LEDS ON OR OFF

This example will light alternate LEDs on the Explorer 16/32 Development Board with a PIC24FJ128GA010 Plug-In Module (PIM). For more information, see **Section B. "Get Software and Hardware"**.

```
// PIC24FJ128GA010 Configuration Bit Settings

// For more on Configuration Bits, ← see Section 1.1
// 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

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>  ← see Section 1.2

int main(void) {

    unsigned char portValue = 0x55; ← see Section 1.3

    // Port A access ← see Section 1.4

    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 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.

1.1.1 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 might 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 https://www.microchip.com for the appropriate data sheet.

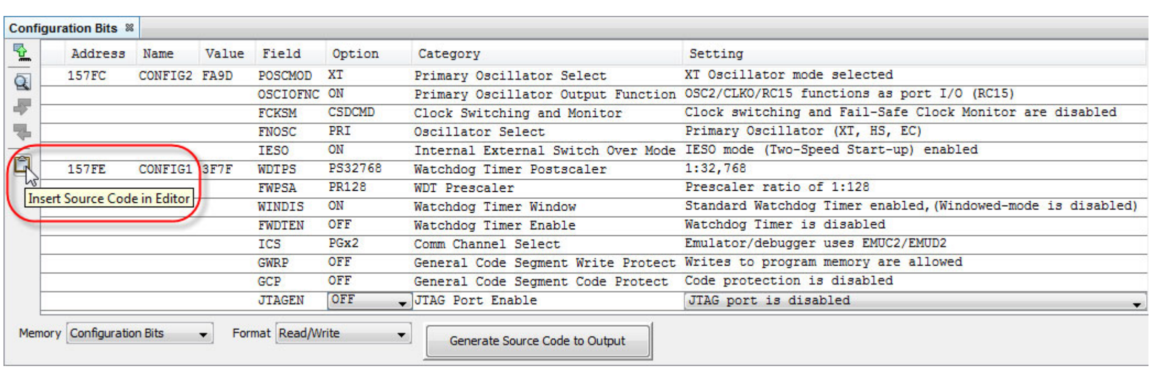
For more about configuration bits that are available for each device, see the following file in the location where MPLAB XC16 was installed:

MPLAB XC16 Installation Directory/docs/config_index.html

1.1.2 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 BITS WINDOW



Once you have the settings you want, click in code where you want this information placed and then click the “Insert Source Code in Editor” icon, as was done in the example code. See MPLAB X IDE documentation for more information on this window.

1.2 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 can be found in the MPLAB XC16 installation directory, in the `support` subdirectory.

Based on the 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.

1.3 Variable for LED Values

The value to be written to the LEDs, as explained in the next section, has been assigned to a variable (`portValue`), i.e., LEDs D3, D5, D7, and D9 will be on and LEDs D4, D6, D8, and D10 will be off. See **Section B. "Get Software and Hardware"** for the demo board schematic location.

1.4 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. By default, the port pins are 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 macro `LEDS_ON_OFF` is assigned to the latch:

```
LATA = LEDS_ON_OFF;    // 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 A DELAY FUNCTION

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

```
// 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

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#include <libpic30.h>  ← see Section 2.1

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 = 0x55;
        LATA = portValue; // write to port latch

        // delay value change  ← see Section 2.3

        __delay32(1500000);    // delay in instruction cycles

        portValue = 0xAA;
        LATA = portValue; // write to port latch

        __delay32(1500000);    // 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, the variable `portValue` is first assigned a value of `0x55` (LEDs 0, 2, 4, 6 are on) and then a complementary value of `0xAA` (LEDs 1,3,5,7 are on). 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. Therefore, a `-1` is returned to signify an error.

2.3 The `__delay32()` 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.

```
// 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

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>

// Interrupt function (CCI)  ← see Section 3.1

void __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

Functions may be made into interrupt functions by using the `__interrupt` specifier from the Common C Interface (CCI). 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 Port A LEDs as the previous example. However, in this example, values from a potentiometer (slider) 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) is used. The MCC is a plug-in available for installation under the MPLAB X IDE 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 the following URL:

https://www.microchip.com/mplab/mplab-code-configurator

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

FIGURE 2: ADC PROJECT RESOURCES - SYSTEM MODULE

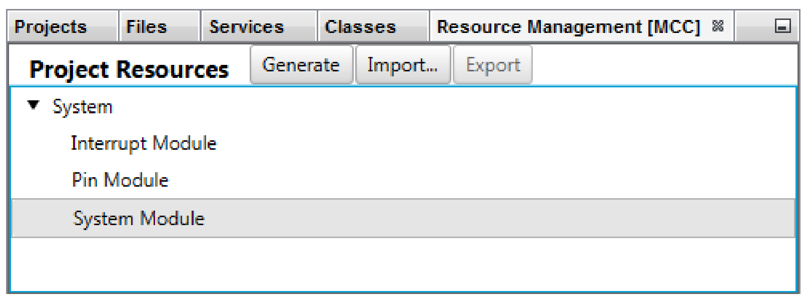


FIGURE 3: ADC PROJECT SYSTEM MODULE CONFIGURATION

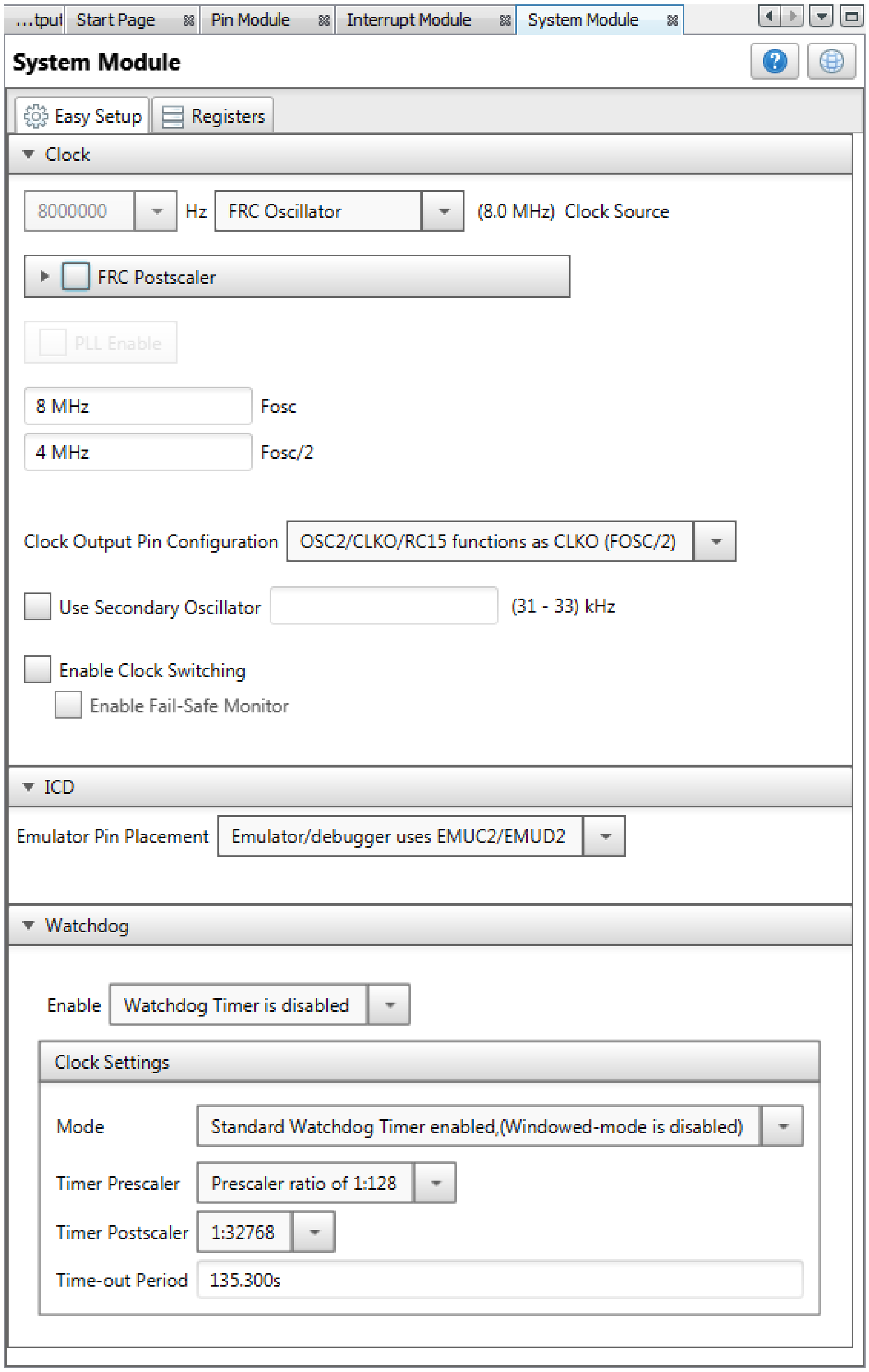
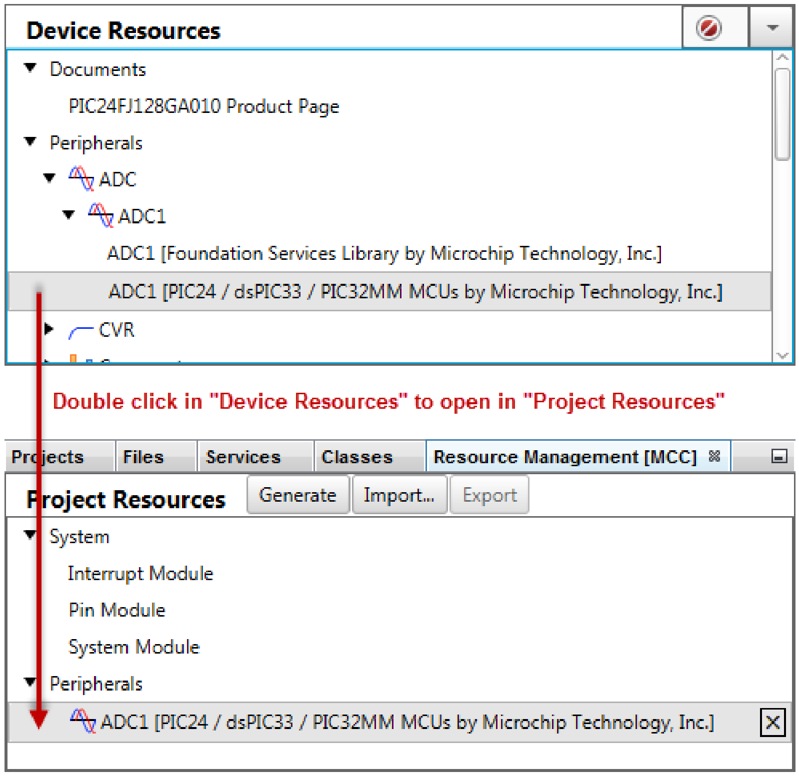


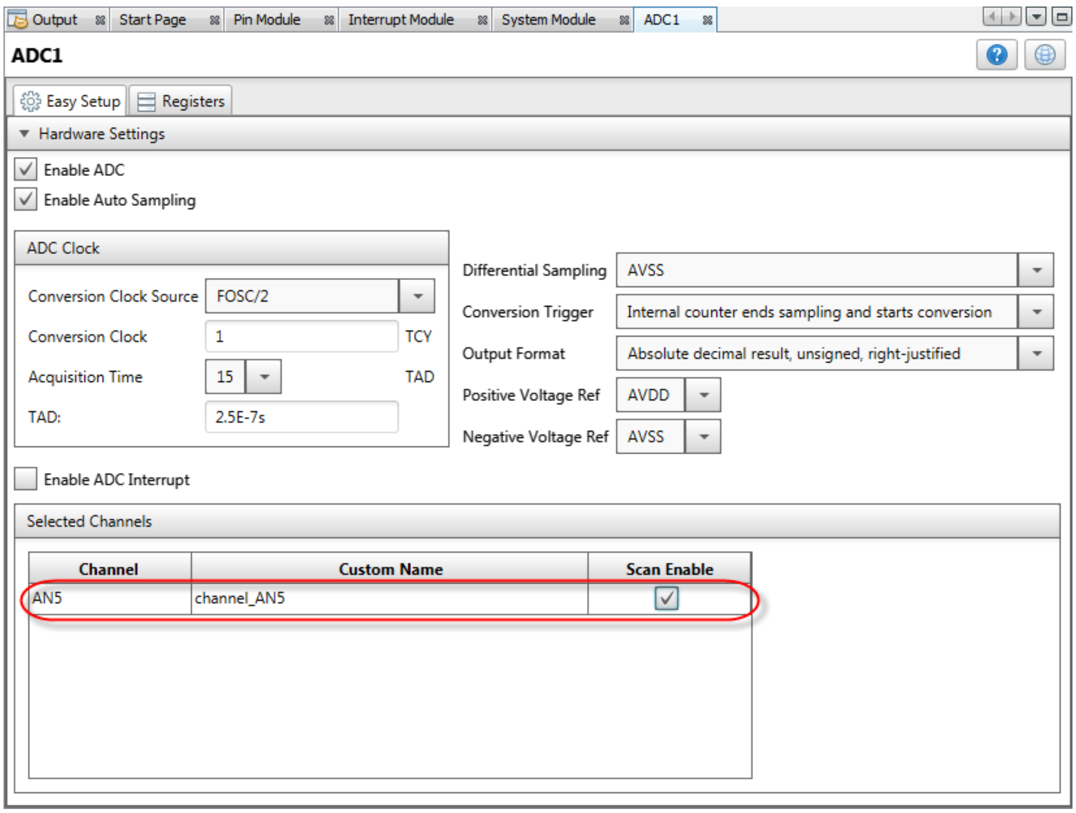
FIGURE 4: ADC PROJECT ADC1 SELECTION



The **PIC24/dsPIC33/PIC32MM MCUs** selection offers you the possibility to directly set up the peripheral in its finest details, via the Easy Setup view.

The **Foundation Services Library** is more abstract in nature, typically offering the basic functionality of the peripheral and easing the configuration process.

FIGURE 5: ADC PROJECT ADC1 CONFIGURATION



RB5 to AN5 map displays after selection is made in Figure 6.

FIGURE 6: ADC PROJECT ADC1 PIN RESOURCE

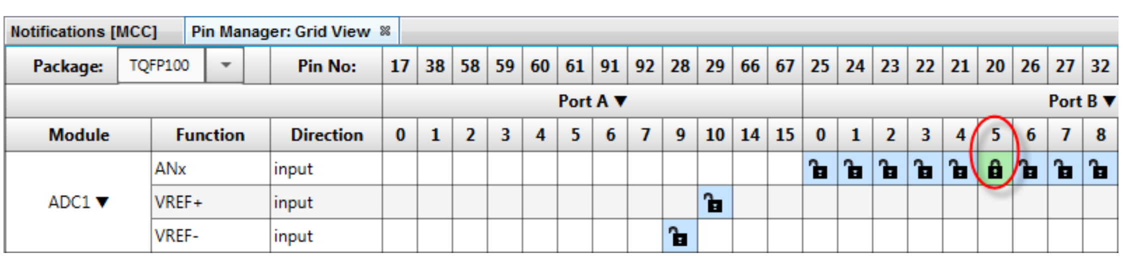


FIGURE 7: ADC PROJECT RESOURCES - PIN MODULE

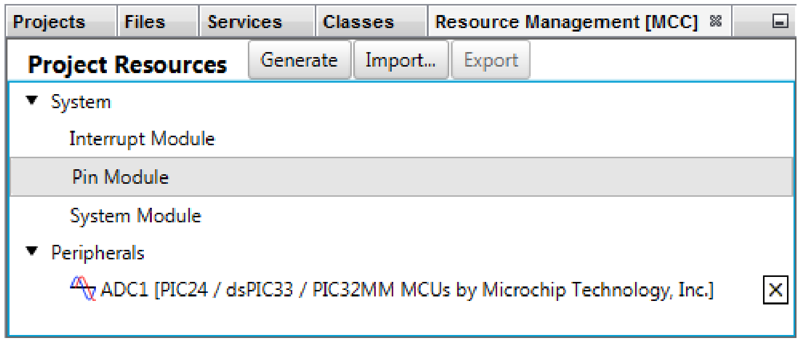
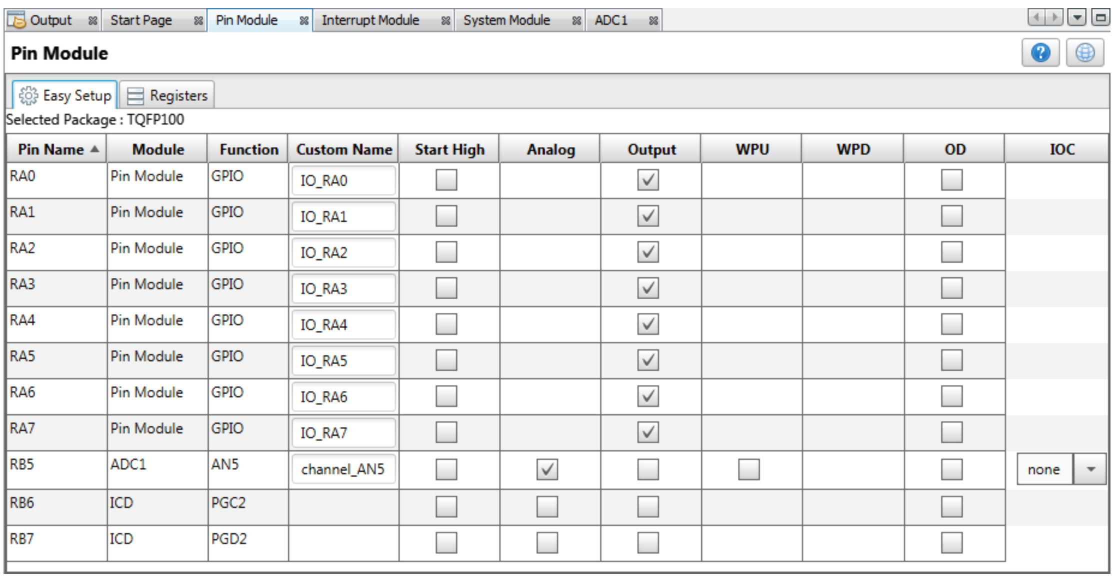


FIGURE 8: ADC PROJECT I/O PIN CONFIGURATION



Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	WPD	OD	IOC
RA0	Pin Module	GPIO	IO_RA0			✓				
RA1	Pin Module	GPIO	IO_RA1			✓				
RA2	Pin Module	GPIO	IO_RA2			✓				
RA3	Pin Module	GPIO	IO_RA3			✓				
RA4	Pin Module	GPIO	IO_RA4			✓				
RA5	Pin Module	GPIO	IO_RA5			✓				
RA6	Pin Module	GPIO	IO_RA6			✓				
RA7	Pin Module	GPIO	IO_RA7			✓				
RB5	ADC1	AN5	channel_AN5		✓					none
RB6	ICD	PGC2								
RB7	ICD	PGD2								

Pins RA0:7 will appear in the window above when they are selected in Figure 9.

RB5 was previously selected in Figure 6.

RB6 and RB7 are preselected for debug communication.

Once visible in the window, pin configurations may be viewed or selected for each pin.

FIGURE 9: ADC PROJECT I/O PIN RESOURCES

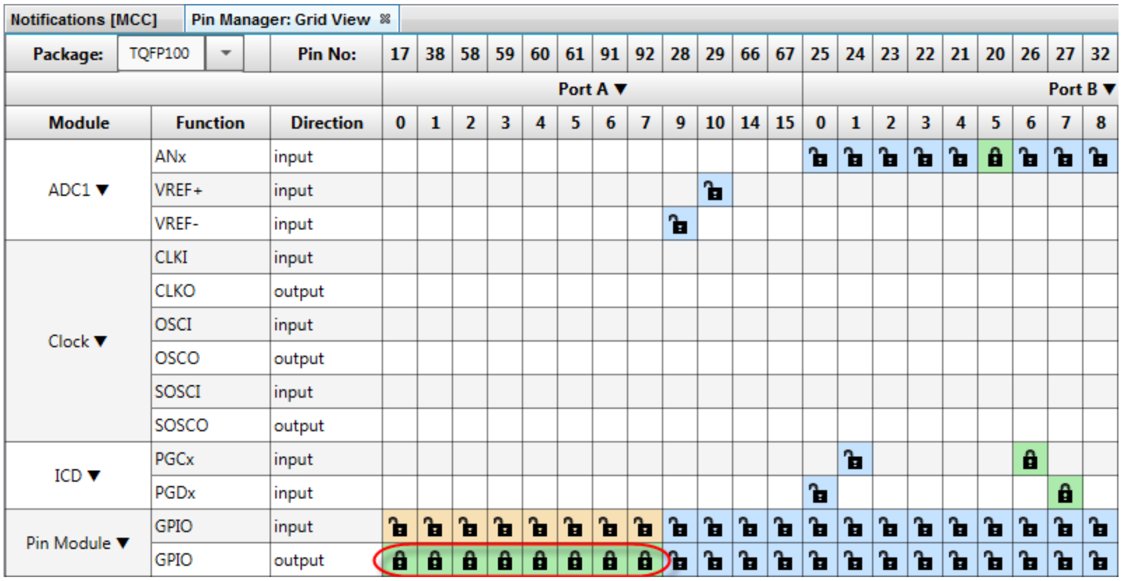
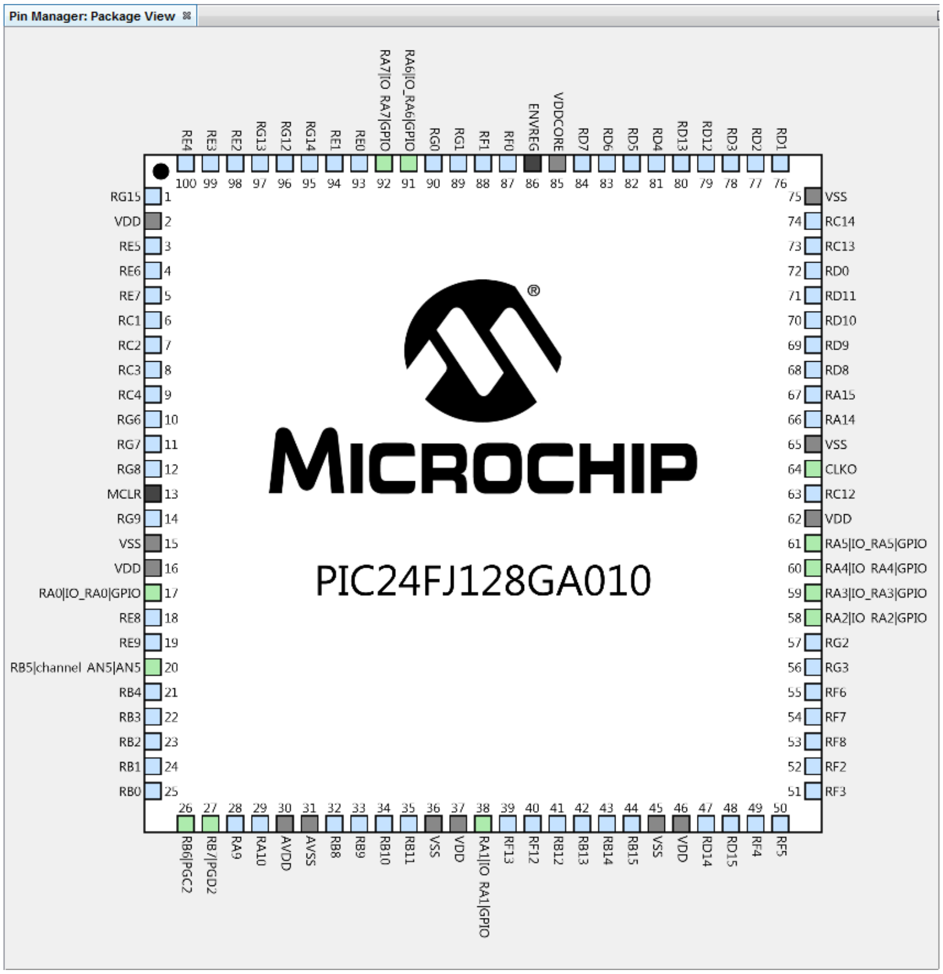


FIGURE 10: ADC PROJECT PIN PACKAGE



When the code is configured as shown in the previous figures, click the **Generate** button in the "Project Resources" window (Figure 7). 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.

Traps files are generated to catch potential errors. Although no interrupts will be used in this application, interrupt manager files are generated for future use.

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.

FIGURE 11: ADC PROJECT TREE FOR CODE GENERATED BY MCC

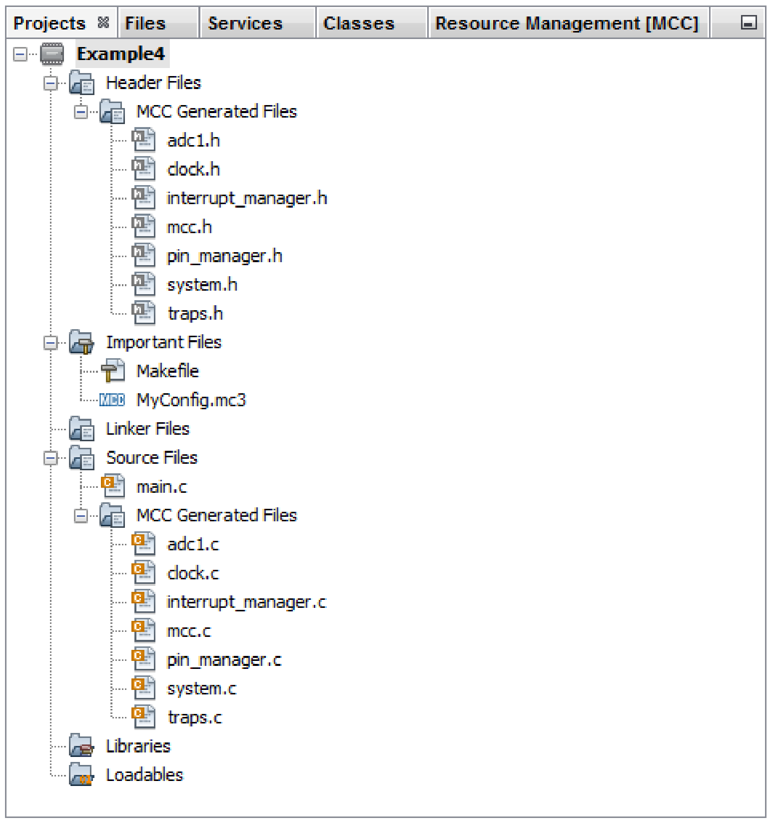
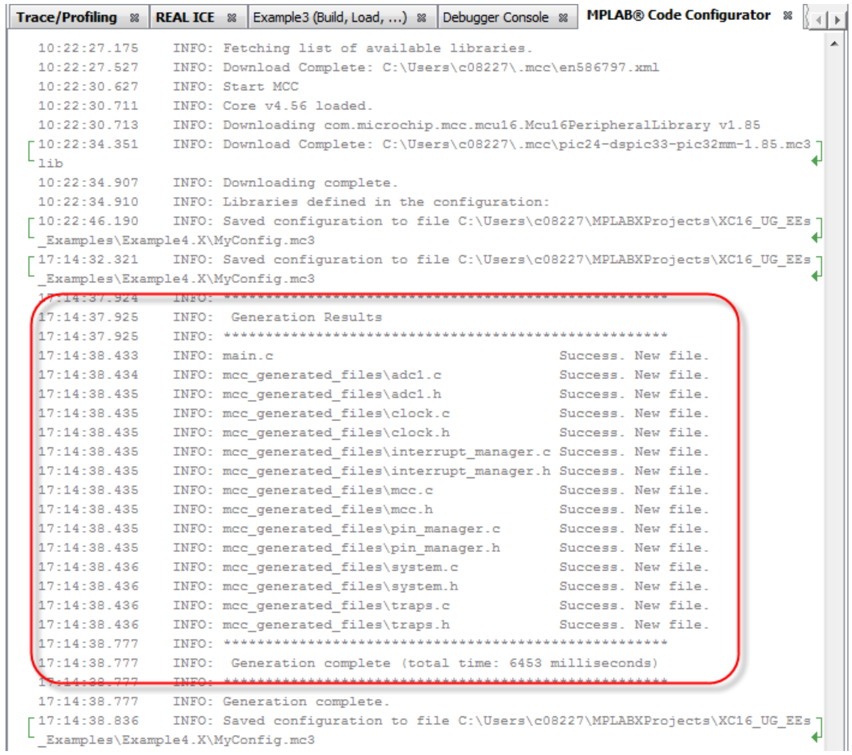


FIGURE 12: ADC CODE GENERATED BY MCC



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 green.

```
/**
  Generated Main Source File

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

/*
(c) 2016 Microchip Technology Inc. and its subsidiaries. You may use
this software and any derivatives exclusively with Microchip products.

<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 PIM, with the Explorer 16/32 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) 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 13). However, at this time, there is no EEData device resource available for 16-bit devices in the MCC.

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

https://www.microchip.com/PIC24F32KA304

FIGURE 13: EEDATA PROJECT RESOURCES - SYSTEM MODULE

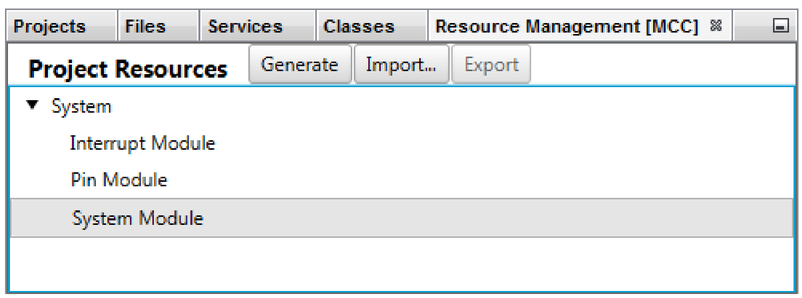


FIGURE 14: EEDATA PROJECT SYSTEM MODULE CONFIGURATION

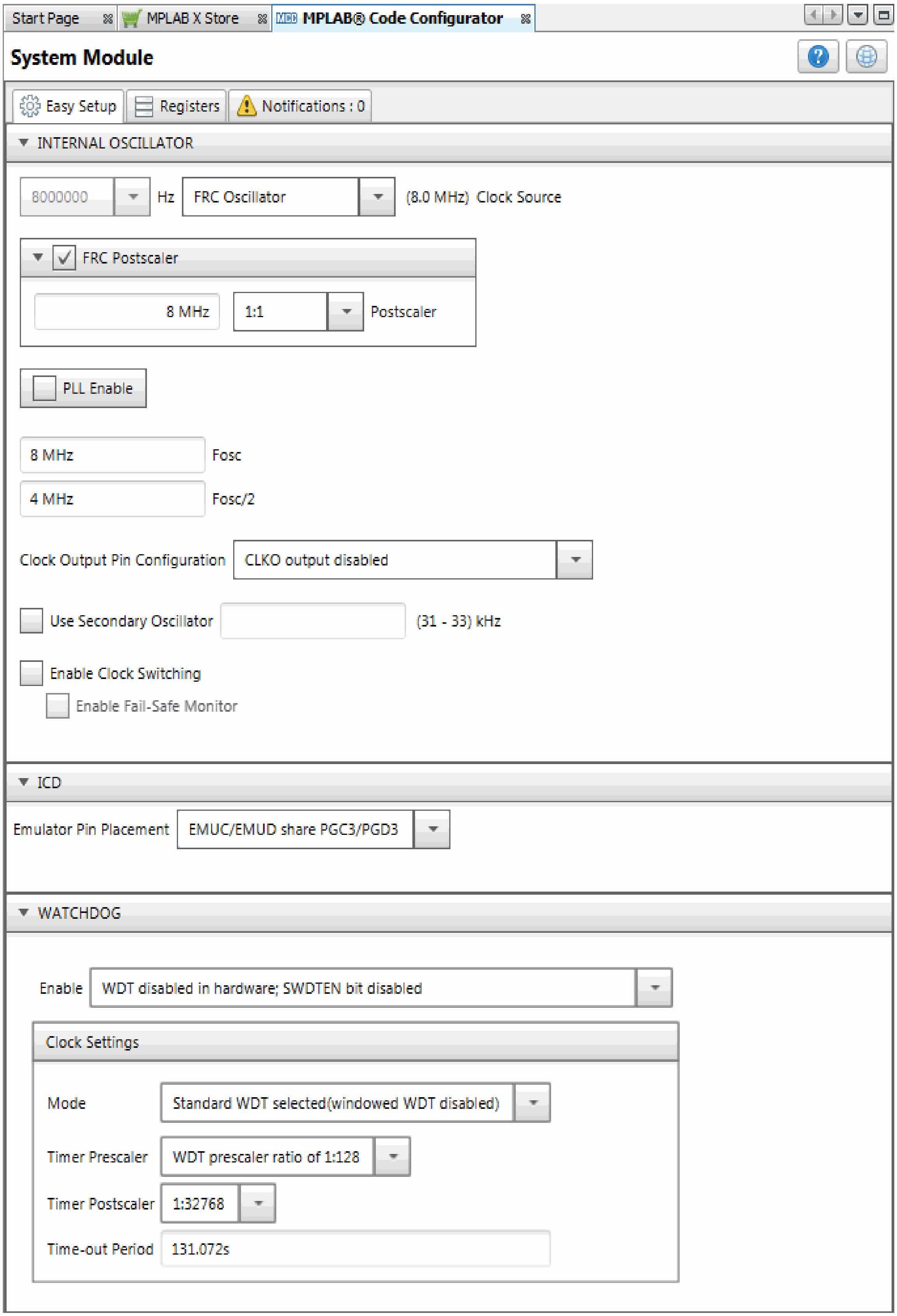


FIGURE 15: EEDATA PROJECT RESOURCES - PIN MODULE

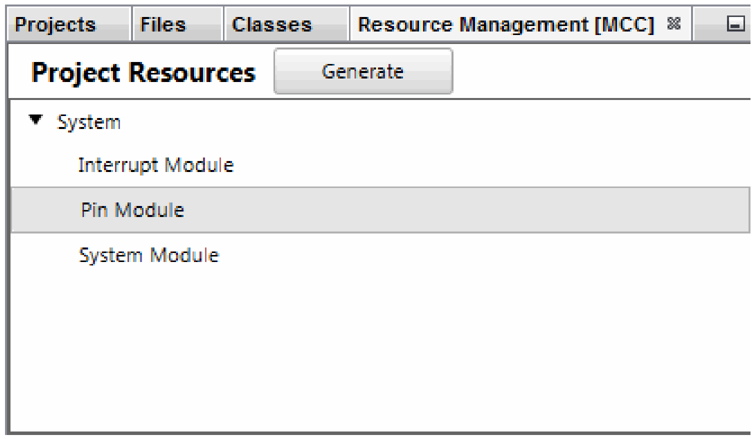
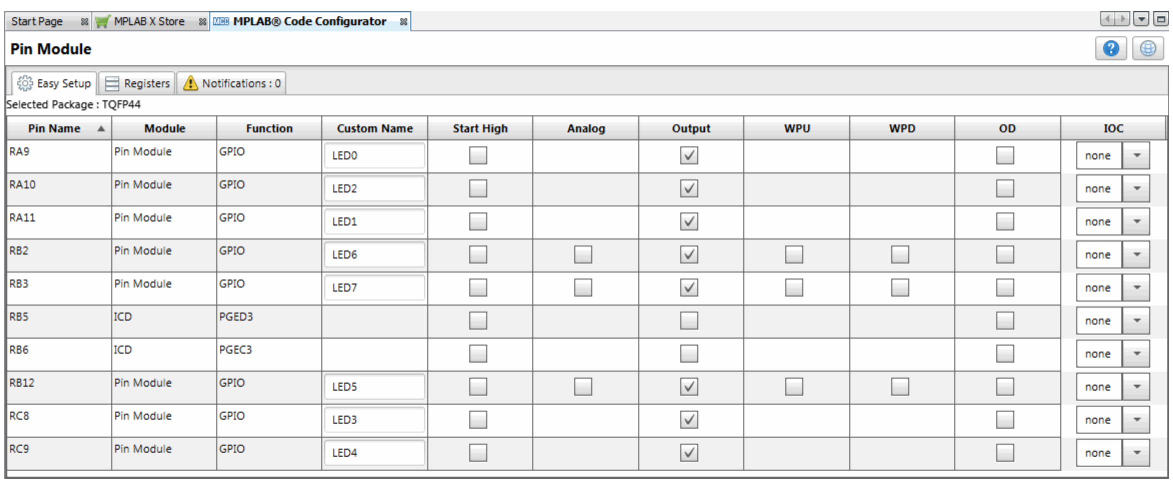


FIGURE 16: EEDATA PROJECT I/O PIN CONFIGURATION



Pins RA9:11, RB2:3, RB12 and RC8:9 will appear in the window above when they are selected in Figure 17.
RB6 and RB7 are preselected for debug communication.
Once visible in the window, pin configurations may be viewed or selected for each pin.

FIGURE 17: EEDATA PROJECT I/O PIN RESOURCES

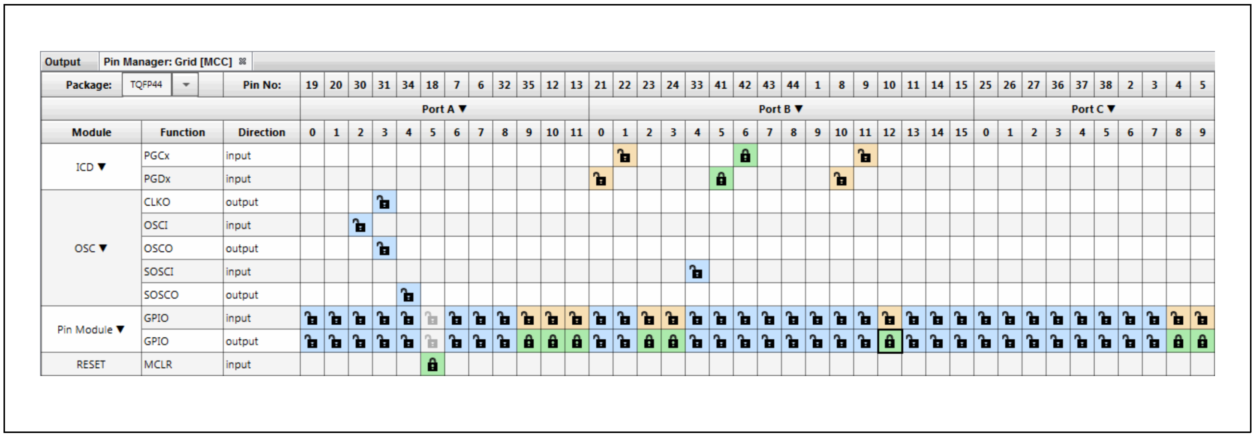
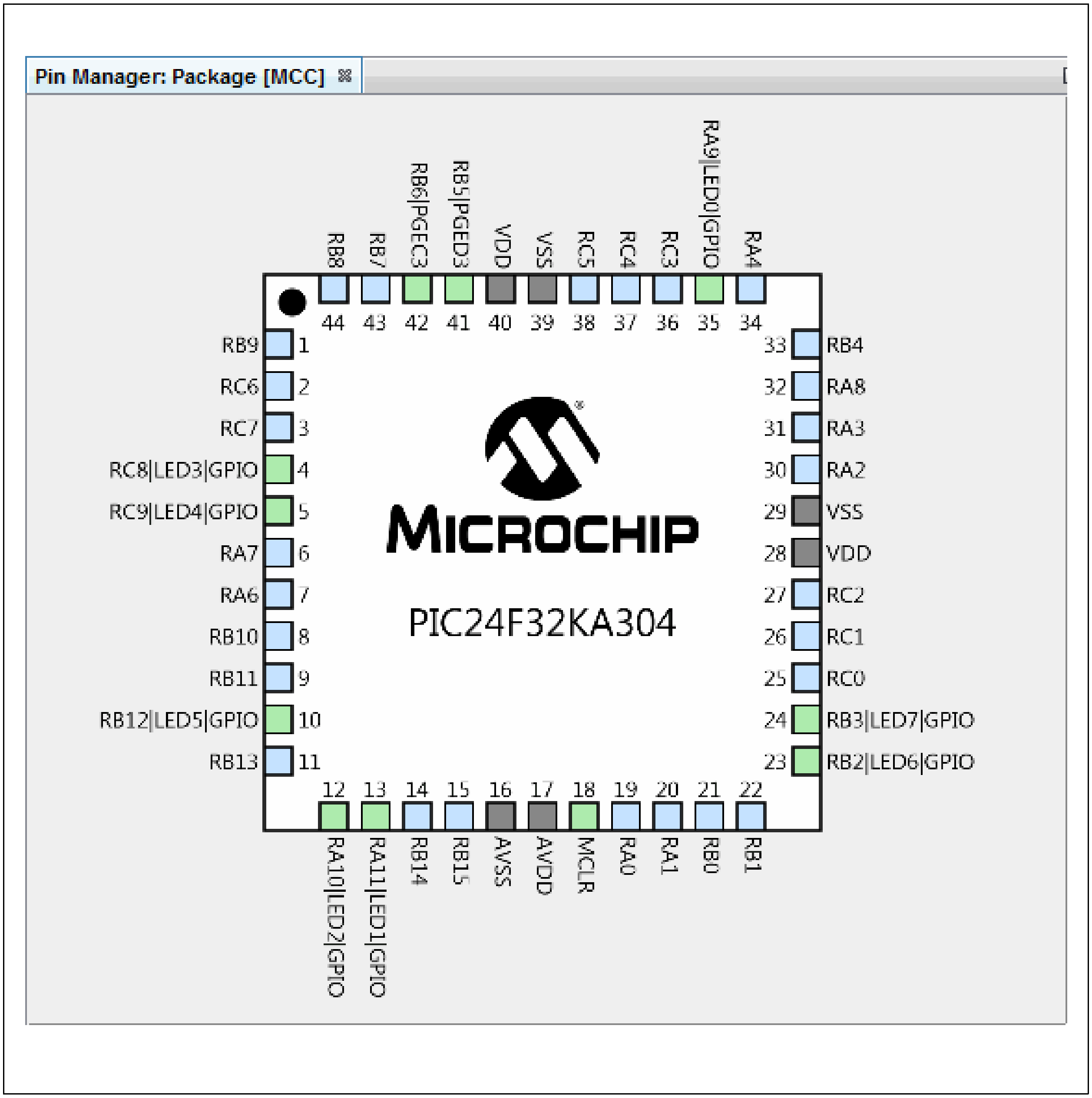


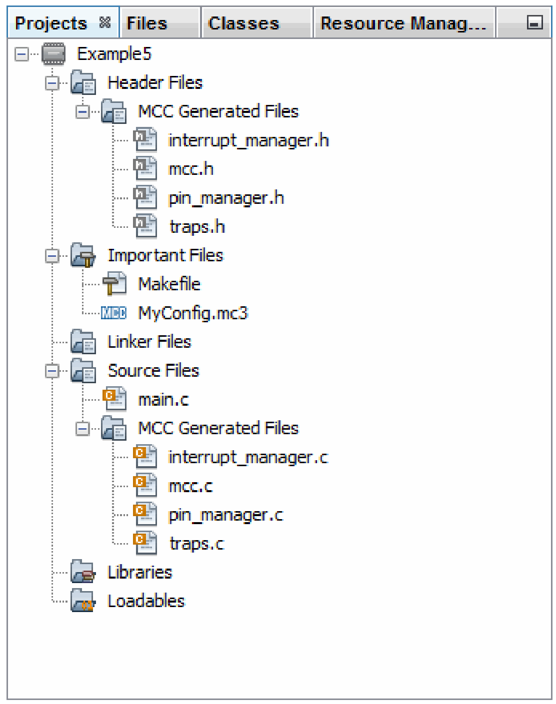
FIGURE 18: EEDATA PROJECT PIN PACKAGE



After your code is configured (as shown in the previous figures), click the **Generate** button on the "Project Resources" window. 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.

Traps files are generated to catch potential errors. Although no interrupts will be used in this application, interrupt manager files are generated for future use.

FIGURE 19: EEDATA PROJECT TREE FOR CODE GENERATED BY MCC



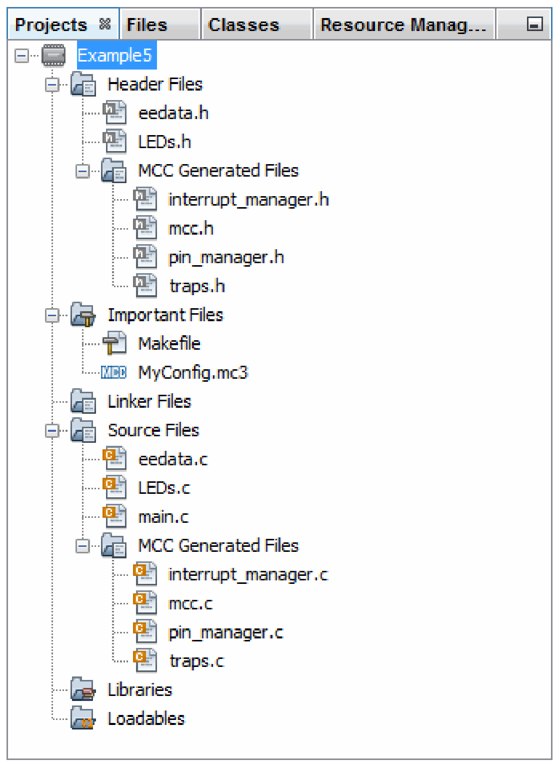
The GPIO-generated files default to analog input, so they must be changed to digital input in the `pin_manager.c` file (**Section 5.1**).

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 currently available in MCC for 16-bit devices, 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 20.

FIGURE 20: 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 that is changed is shown in green.

```
/**
  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(s)
 *********************************************************************/
    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 that has been added is shown in green.

```
/**
  Generated Main Source File

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

/*
(c) 2016 Microchip Technology Inc. and its subsidiaries. You may use
this software and any derivatives exclusively with Microchip products.

<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 a PIC24F32KA304 device, 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**).

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 can 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

A.1 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," then "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 can find the executable. 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.

A.2 Select the Common Compiler Interface (CCI)

After your project is created, right click on the project name, in the Projects window, and select Properties. In the dialog, click on the "xc16-gcc" category, select the "Preprocessing and messages" option category, and select the "Use CCI syntax" checkbox. Click the **OK** button.

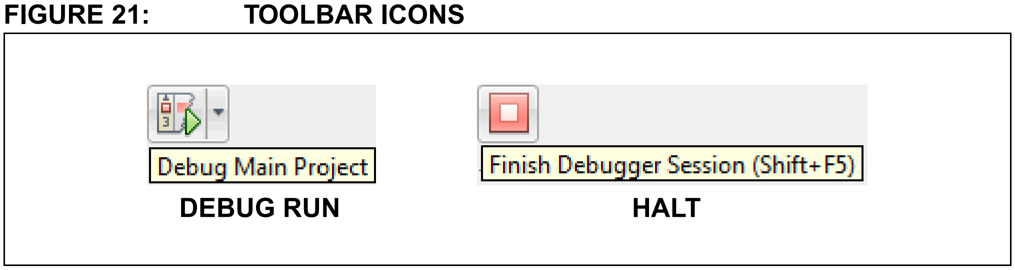
A.3 Debug the Examples

Do one of the following, based on the example you are using:

1. For examples 1, 2, and 3, create a file to hold the example code:
   a) Right click on the "Source Files" folder in the Projects window. Select *New>main.c*. The "New main.c" dialog will open.
   b) Under "File name," enter a name, e.g., example*n*, where *n* is the example number.
   c) Click **Finish**. The file will open in an editor window.
   d) Delete the template code in the file. Then cut and paste the example code from this user's guide into the empty editor window and select *File>Save*.
2. For examples 4 and 5, follow the instructions in each section to generate code using MCC and then edit `main.c` and other files with the code shown.

Finally, select Debug Run to build, download to a device, and execute the code. View the demo board LEDs for output. Click Halt to end execution.

FIGURE 21: TOOLBAR ICONS



B. GET SOFTWARE AND HARDWARE

For the MPLAB XC16 projects in this document, the Explorer 16/32 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 v5.10 and later can be found at:

https://www.microchip.com/mplab/mplab-x-ide

The MPLAB XC16 C Compiler v1.35 and later can be found at:

https://www.microchip.com/mplab/compilers

B.2 Get the MPLAB Code Configurator (MCC)

The MCC v3.66 and later can be found at:

https://www.microchip.com/mplab/mplab-code-configurator

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: https://www.microchip.com/MA240011

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

B.4 Get and Set Up the Explorer 16/32 Board

The Explorer 16/32 development board, schematic and documentation are available on the web site:

https://www.microchip.com/dm240001-2

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

TABLE 1-1: JUMPER/SWITCH SELECTS FOR PROJECTS

Jumper/Switch	Selection	Jumper/Switch	Selection
JP2	Closed	J37	Open
J19	Open	J38	Open
J22	Open	J39	Default
J23	Default	J41	Open
J25	Closed	J42	Open
J26	Closed	J43	Default
J27	Open	J44	Default
J28	Open	J45	Default
J29	Open	J50	Closed
J33	Open		

B.5 Get Microchip Debug Tools

Emulators and debuggers can be found on the Development Tools web page:

https://www.microchip.com/development-tools

NOTES:

ISBN: 978-1-5224-4089-5

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

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
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

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

China - Beijing
Tel: 86-10-8569-7000

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

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

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

China - Guangzhou
Tel: 86-20-8755-8029

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

China - Hong Kong SAR
Tel: 852-2943-5100

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

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

China - Shanghai
Tel: 86-21-3326-8000

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

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

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

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

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

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

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

India - Pune
Tel: 91-20-4121-0141

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

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

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

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

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

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

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

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

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

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

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

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

Finland - Espoo
Tel: 358-9-4520-820

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

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

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

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

Sweden - Gothenberg
Tel: 46-31-704-60-40

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

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