MPLAB® XC8 USER'S GUIDE FOR EMBEDDED ENGINEERS FOR PIC MCUs

MPLAB® XC8 User's Guide for Embedded Engineers - PIC MCUs

This document presents five code examples for 8-bit PIC MCU devices and the MPLAB XC8 C compiler using the Common Code Interface (CCI). For more on CCI, see the "*MPLAB XC8 C Compiler User's Guide*" (DS50002053).

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

© 2015-2018 Microchip Technology Inc.

1. TURN LEDS ON OR OFF

This example will light alternate LEDs on the Explorer 8 board with a PIC16F1719 microcontroller (MCU). For more information, see **Section B. "Get Software and Hardware"**.

```
// PIC16F1719 Configuration Bit Settings

// For more on Configuration Bits, ← see Section 1.1
// consult your device data sheet

// CONFIG1
#pragma config FOSC = ECH      // External Clock, 4-20 MHz
#pragma config WDTE = OFF      // Watchdog Timer (WDT) disabled
#pragma config PWRTE = OFF     // Power-up Timer disabled
#pragma config MCLRE = ON      // MCLR/VPP pin function is MCLR
#pragma config CP = OFF        // Flash Memory Code Protection off
#pragma config BOREN = ON      // Brown-out Reset enabled
#pragma config CLKOUTEN = OFF // Clock Out disabled.
#pragma config IESO = ON       // Internal/External Switchover on
#pragma config FCMEN = ON      // Fail-Safe Clock Monitor enabled

// CONFIG2
#pragma config WRT = OFF       // Flash Memory Self-Write Protect off
#pragma config PPS1WAY = ON    // PPS one-way control enabled
#pragma config ZCDDIS = ON     // Zero-cross detect disabled
#pragma config PLLEN = OFF     // Phase Lock Loop disable
#pragma config STVREN = ON     // Stack Over/Underflow Reset enabled
#pragma config BORV = LO       // Brown-out Reset low trip point
#pragma config LPBOR = OFF     // Low-Power Brown Out Reset disabled
#pragma config LVP = OFF       // Low-Voltage Programming 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
#include <stdint.h>

void main(void) {

    uint8_t portValue = 0x05; ← see Section 1.3

    // Port B access ← see Section 1.4

    ANSELB = 0x0; // set to digital I/O (not analog)
    TRISB = 0x0;  // set all port bits to be output
    LATB = portValue; // write to port latch - RB[0:3] = LED[4:7]

    // Port D access
    ANSELD = 0x0; // set to digital I/O (not analog)
    TRISD = 0x0;  // set all port bits to be output
    LATD = portValue; // write to port latch - RD[0:3] = LED[0:3]

}
```


© 2015-2018 Microchip Technology Inc.

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.

WHICH CONFIGURATION BITS TO SET

In particular, be aware of the followings settings:

- **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 FOSC = ECH`
- **Watchdog timer**- It is recommended that you disable this timer until it is required. This prevents *unexpected Resets*. From the example code:
  `#pragma config WDTE = 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 CP = OFF`

Different configuration bits may need to be set up to use another 8-bit device (rather than the PIC16F1719 MCU used in this example). See your device data sheet for the name and function of corresponding configuration bits. Use the part number to search https://www.microchip.com for the appropriate data sheet.

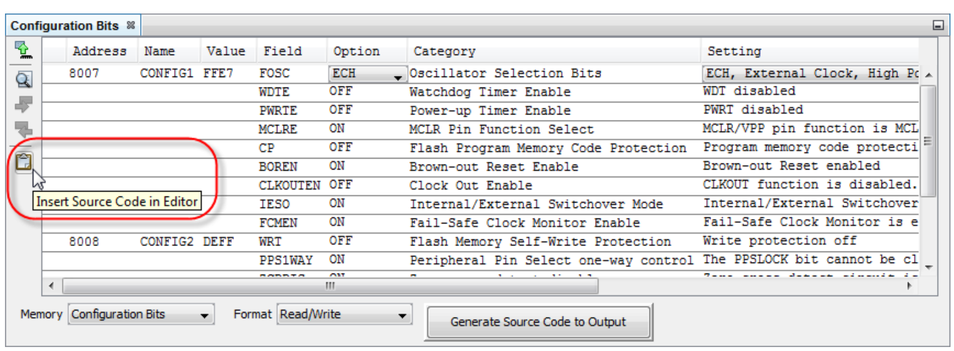
For more information about configuration bits that are available for each device, see the following file in the location where MPLAB XC8 was installed:

MPLAB XC8 Installation Directory/docs/chips

HOW TO SET CONFIGURATION BITS

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

FIGURE 1: CONFIGURATION BITS WINDOW



Once the settings are selected, click in code where you want this information placed and then click the **Insert Source Code in Editor** icon, as shown in the example code.

See MPLAB X IDE documentation for more information on this window.

1.2 Included Header Files

The `xc.h` header file allows code in the source file to access compiler-specific or device-specific features. Based on your selected device, the compiler sets 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.

The `stdint.h` header file defines fixed-size integer types. For example, `uint8_t` is an unsigned 8-bit integer.

These and other header files can be found in the MPLAB XC8 installation directory in the `pic/include` subdirectory.

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 D1, D3, D5, and D7 will be on and LEDs D2, D4, D6 and D8 will be off. See the *Explorer 8 Development Board User's Guide* (DS40001812) for the board schematic (**Section B.4 "Get and Set Up the Explorer 8 Board"**).

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 B and Port D pins are multiplexed with peripherals that are disabled by default. By default the port pins are analog, so you must set them to digital I/O. For Port B:

```
ANSELB = 0x0; // 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, `LATD` and `LATB` are used. The macro `LEDS_ON_OFF` is assigned to both latches. For Port D:

```
LATB = portValue; // write to port latch - RD[0:3] = LED[0:3]
```

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, `TRISB` and `TRISD` are used. Setting a bit to 0 makes the pin an output, and setting a bit to 1 makes the pin an input. For Port B:

```
TRISB = 0x0; // set all port bits to be output
```


© 2015-2018 Microchip Technology Inc.

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.

```
// PIC16F1719 Configuration Bit Settings

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

// CONFIG1
#pragma config FOSC = ECH     // External Clock, 4-20 MHz
#pragma config WDTE = OFF     // Watchdog Timer (WDT) disabled
#pragma config PWRTE = OFF    // Power-up Timer disabled
#pragma config MCLRE = ON     // MCLR/VPP pin function is MCLR
#pragma config CP = OFF       // Flash Memory Code Protection off
#pragma config BOREN = ON     // Brown-out Reset enabled
#pragma config CLKOUTEN = OFF // Clock Out disabled.
#pragma config IESO = ON      // Internal/External Switchover on
#pragma config FCMEN = ON     // Fail-Safe Clock Monitor enabled

// CONFIG2
#pragma config WRT = OFF      // Flash Memory Self-Write Protect off
#pragma config PPS1WAY = ON   // PPS one-way control enabled
#pragma config ZCDDIS = ON    // Zero-cross detect disabled
#pragma config PLLEN = OFF    // Phase Lock Loop disable
#pragma config STVREN = ON    // Stack Over/Underflow Reset enabled
#pragma config BORV = LO      // Brown-out Reset low trip point
#pragma config LPBOR = OFF    // Low-Power Brown Out Reset disabled
#pragma config LVP = OFF      // Low-Voltage Programming disabled

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

#include <xc.h>
#include <stdint.h>

void main(void) {

    uint8_t portValue;

    // Port B access
    ANSELB = 0x0; // set to digital I/O (not analog)
    TRISB = 0x0;  // set all port bits to be output

    // Port D access
    ANSELD = 0x0; // set to digital I/O (not analog)
    TRISD = 0x0;  // set all port bits to be output

    while(1) {   ◄──── see Section 2.1

        portValue = 0x05;
        LATB = portValue; // RB[0:3] = LED[4:7]
        LATD = portValue; // RD[0:3] = LED[0:3]

        // delay value change   ◄──── see Section 2.2

        _delay(25000); // delay in instruction cycles
```

```
        portValue = 0x0A;
        LATB = portValue; // RB[0:3] = LED[4:7]
        LATD = portValue; // RD[0:3] = LED[0:3]
        _delay(25000); // delay in instruction cycles

    }
}
```

2.1 The `while()` Loop and Variable Values

To make the LEDs on Port B and Port D change, the variable `portValue` is first assigned a value of `0x05` (LEDs 0, 2, 4, 6 are on) and then a complementary value of `0x0A` (LEDs 1,3,5,7 are on). To perform the loop, `while(1) { }` was used.

2.2 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. `_delay()` is a built-in function of the compiler.

For more details on the delay built-in, see the *MPLAB® XC8 C Compiler User's Guide* (DS50002053).

© 2015-2018 Microchip Technology Inc.

3. COUNT UP ON LEDs USING INTERRUPTS AS DELAY

This example is a modification of the previous code. Although the delay loop 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.

```
// PIC16F1719 Configuration Bit Settings

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

// CONFIG1
#pragma config FOSC = ECH     // External Clock, 4-20 MHz
#pragma config WDTE = OFF     // Watchdog Timer (WDT) disabled
#pragma config PWRTE = OFF    // Power-up Timer disabled
#pragma config MCLRE = ON     // MCLR/VPP pin function is MCLR
#pragma config CP = OFF       // Flash Memory Code Protection off
#pragma config BOREN = ON     // Brown-out Reset enabled
#pragma config CLKOUTEN = OFF // Clock Out disabled.
#pragma config IESO = ON      // Internal/External Switchover on
#pragma config FCMEN = ON     // Fail-Safe Clock Monitor enabled

// CONFIG2
#pragma config WRT = OFF      // Flash Memory Self-Write Protect off
#pragma config PPS1WAY = ON   // PPS one-way control enabled
#pragma config ZCDDIS = ON    // Zero-cross detect disabled
#pragma config PLLEN = OFF    // Phase Lock Loop disable
#pragma config STVREN = ON    // Stack Over/Underflow Reset enabled
#pragma config BORV = LO      // Brown-out Reset low trip point
#pragma config LPBOR = OFF    // Low-Power Brown Out Reset disabled
#pragma config LVP = OFF      // Low-Voltage Programming disabled

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

#include <xc.h>
#include <stdint.h>

// Interrupt function      <——— see Section 3.1

void __interrupt() tcInt(void){
   // only process Timer0-triggered interrupts
   if(INTCONbits.TMR0IE && INTCONbits.TMR0IF) {
            // static variable for permanent storage duration
            static uint8_t portValue;
            // write to port latches
            LATB = (portValue++ >> 4);  // RB[0:3] = LED[4:7]
            LATD = portValue++;  // RD[0:3] = LED[0:3]
            // clear this interrupt condition
            INTCONbits.TMR0IF = 0;
   }
}

void main(void){

    // Port B access
    ANSELB = 0x0; // set to digital I/O (not analog)
    TRISB = 0x0;  // set all port bits to be output

    // Port D access
    ANSELD = 0x0; // set to digital I/O (not analog)
    TRISD = 0x0;  // set all port bits to be output
```

```
    // Timer0 setup        ◄──────  see Section 3.2

    OPTION_REG = 0xD7; // timer 0 internal clock, prescaler 1:256
    INTCONbits.TMR0IE = 1; // enable interrupts for timer 0
    ei(); // enable all interrupts

    while(1);
}
```

3.1 The Interrupt Function

Functions are made into interrupt functions by using the `__interrupt()` specifier. As the `tcInt()` interrupt function may have to handle multiple interrupt sources, code is added to ensure the counter `portValue` is only incremented if Timer0 generated the interrupt.

3.2 Timer0 Setup

Code also needs to be added to the main routine to enable 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.

To enable all interrupts, `ei()` is used, defined in `xc.h`.

 © 2015-2018 Microchip Technology Inc.

4 DISPLAY POTENTIOMETER VALUES ON LEDS USING A/D

This example uses the same device and Port B and Port D LEDs as the previous example. However, in this example, values from a potentiometer on the demo board provide A/D input through Port A 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.

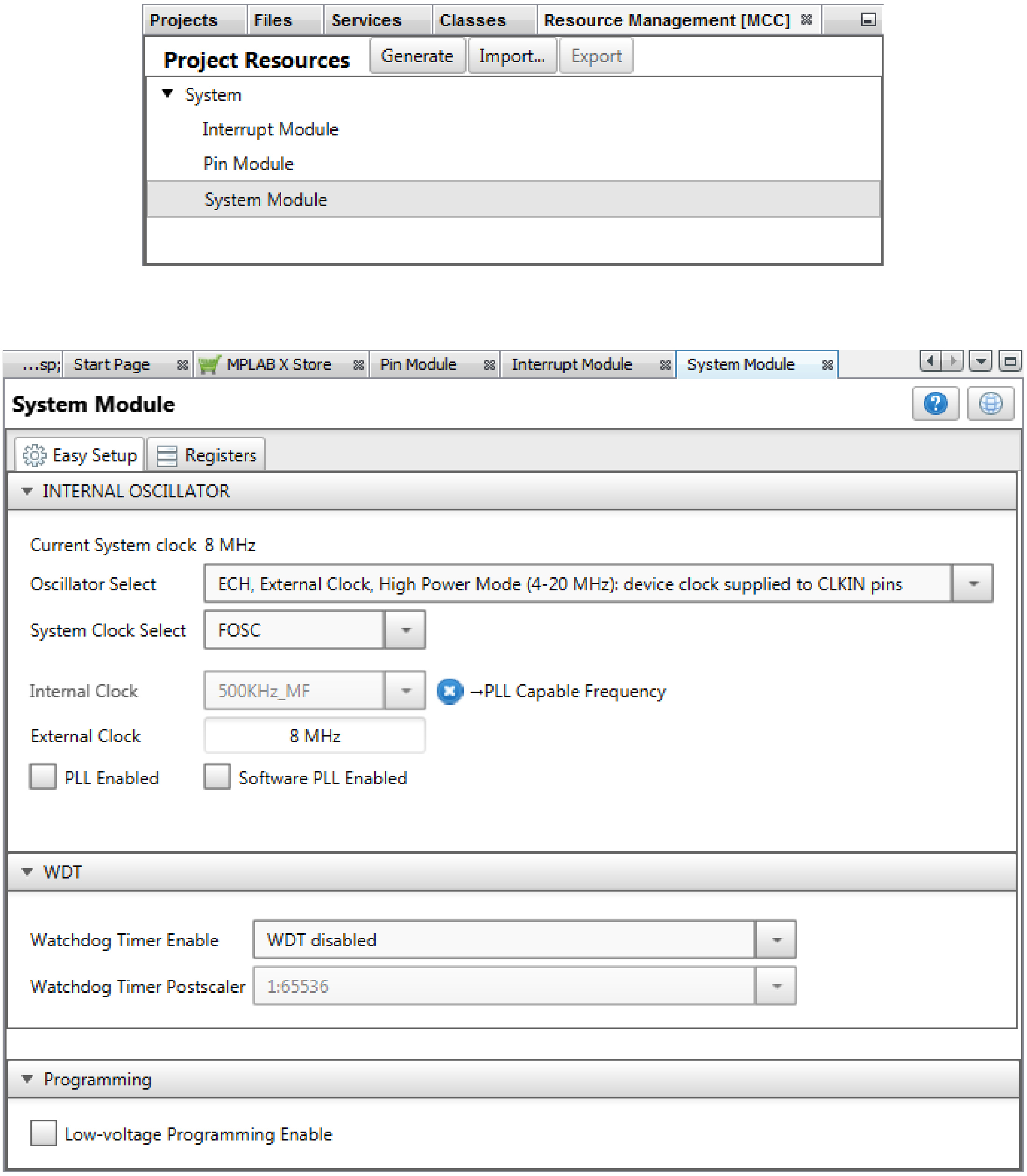
For information on the MCC, including the *MPLAB® Code Configurator User's Guide* (DS40001725), go to the MPLAB Code Configurator web page at:

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

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

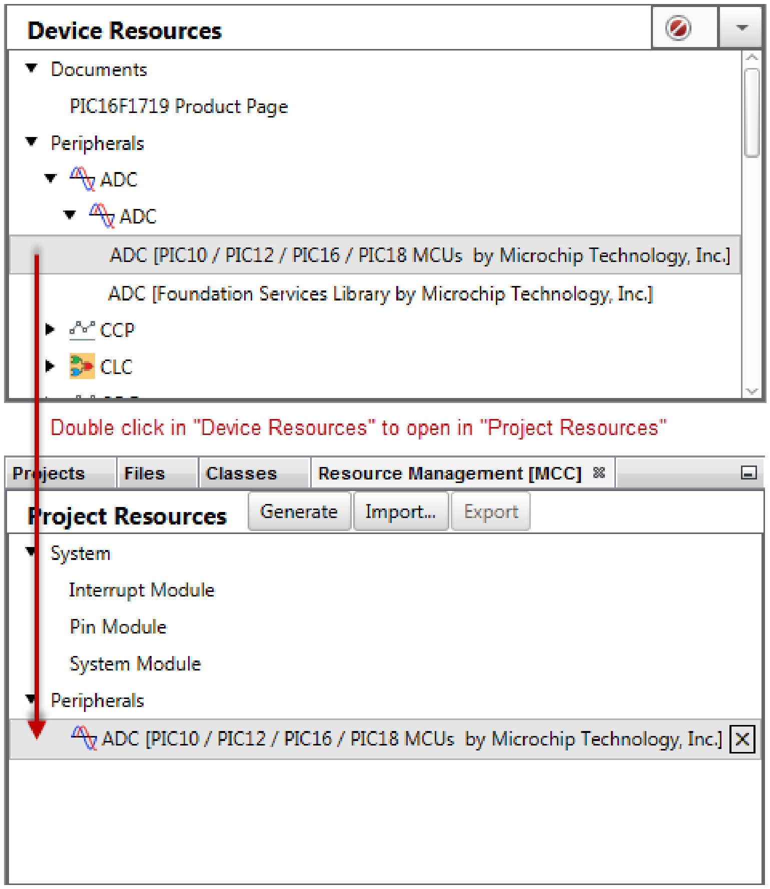
© 2015-2018 Microchip Technology Inc.

FIGURE 2: ADC PROJECT SYSTEM RESOURCE CONFIGURATION



© 2015-2018 Microchip Technology Inc.

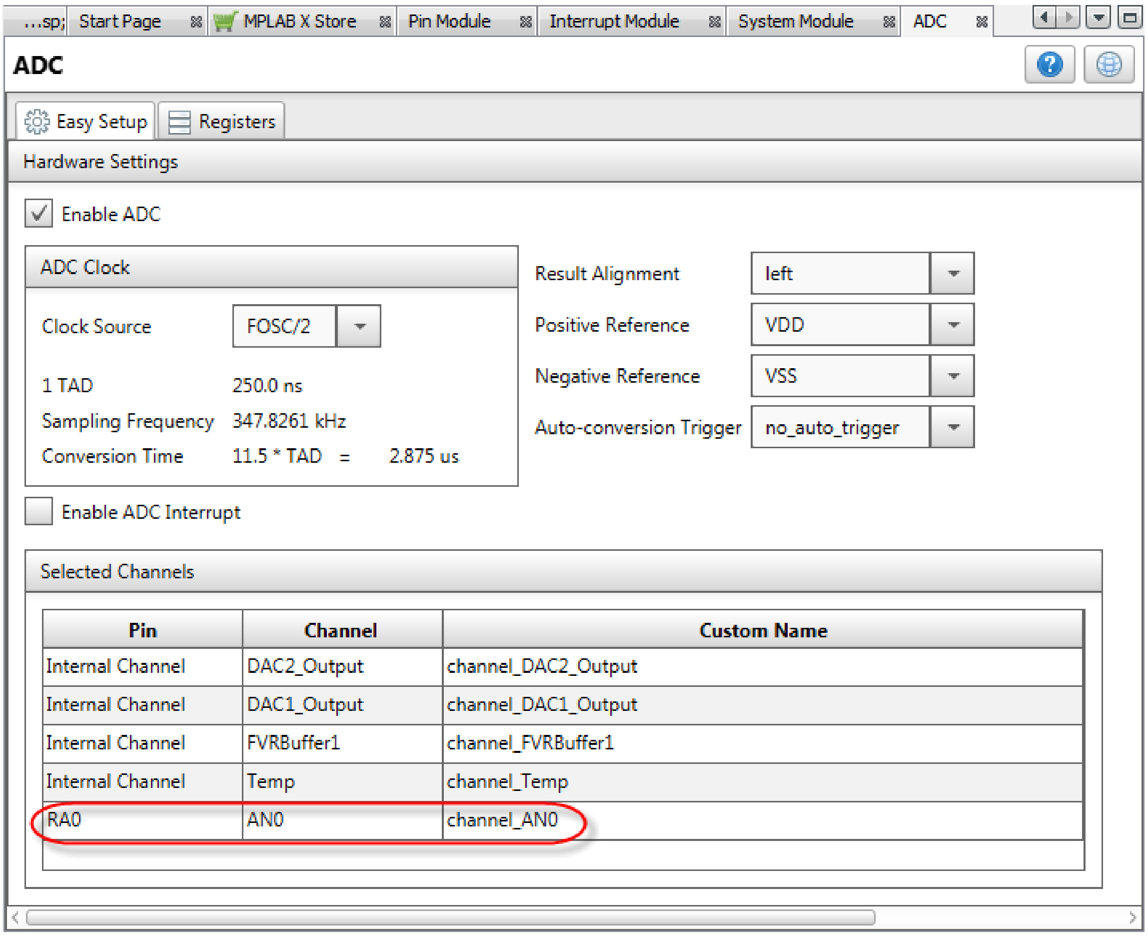
FIGURE 3: ADC PROJECT ADC RESOURCE SELECTION



The **PIC10/PIC12/PIC16/PIC18 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 4: ADC PROJECT ADC RESOURCE CONFIGURATION



RA0 to AN0 map displays after selection is made in Figure 5.

© 2015-2018 Microchip Technology Inc.

FIGURE 5: ADC PROJECT ADC PIN RESOURCE GRID

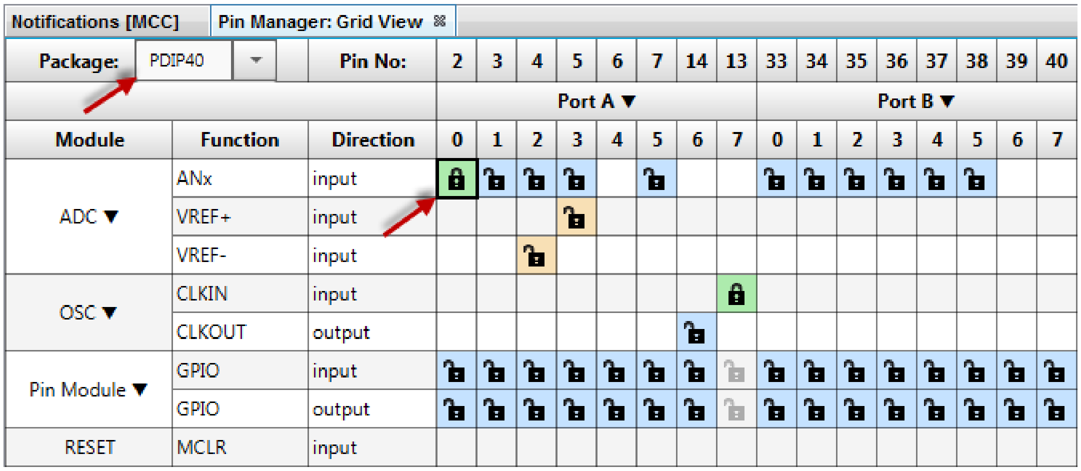
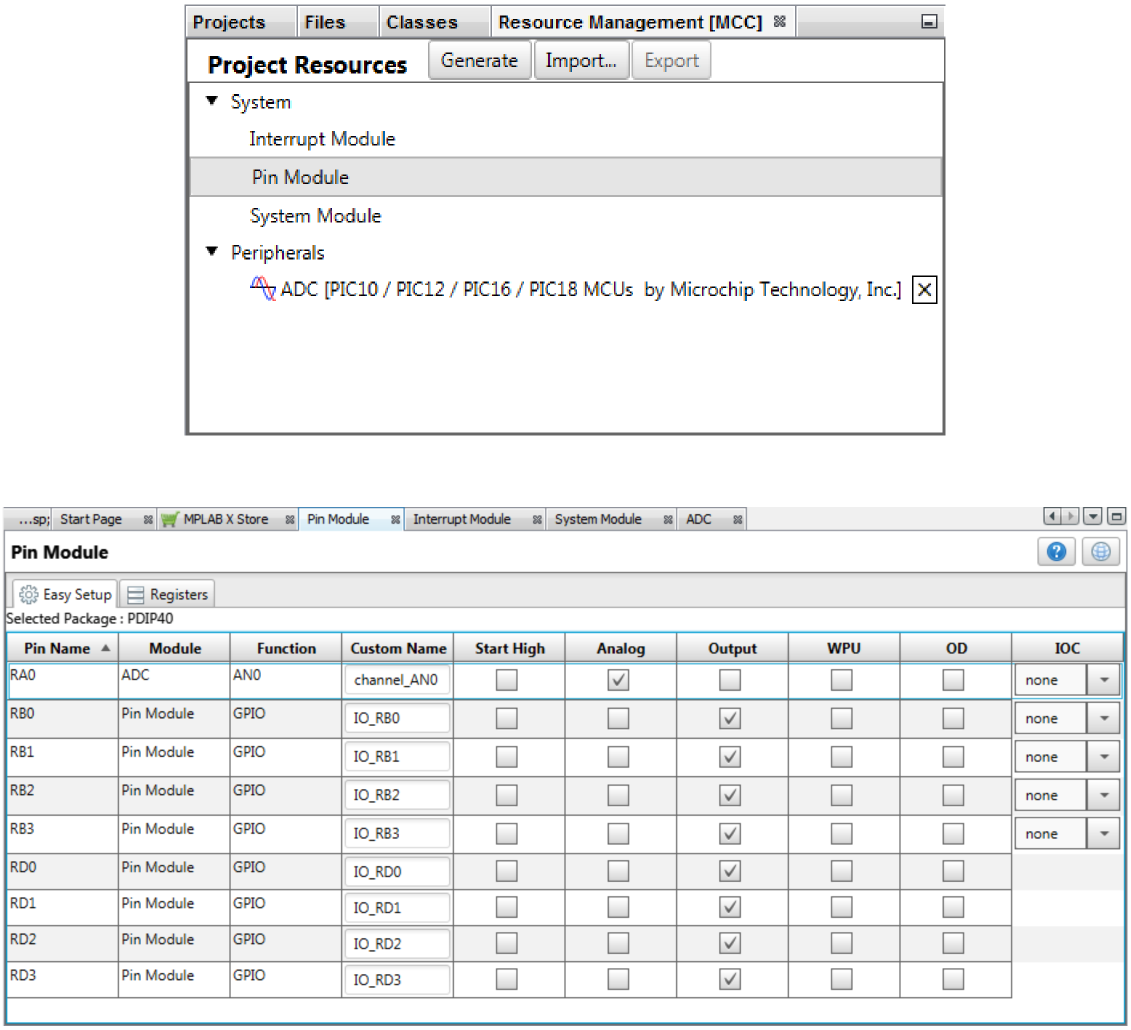


FIGURE 6: ADC PROJECT PIN RESOURCE CONFIGURATION



Pins RB0:3 and RD0:3 will appear in the window above when they are selected in Figure 7.
RA0 was previously selected in Figure 5.
Once visible in the window, pin configurations can be selected for each pin.

© 2015-2018 Microchip Technology Inc.

FIGURE 7: ADC PROJECT GPIO PIN RESOURCE- GRID

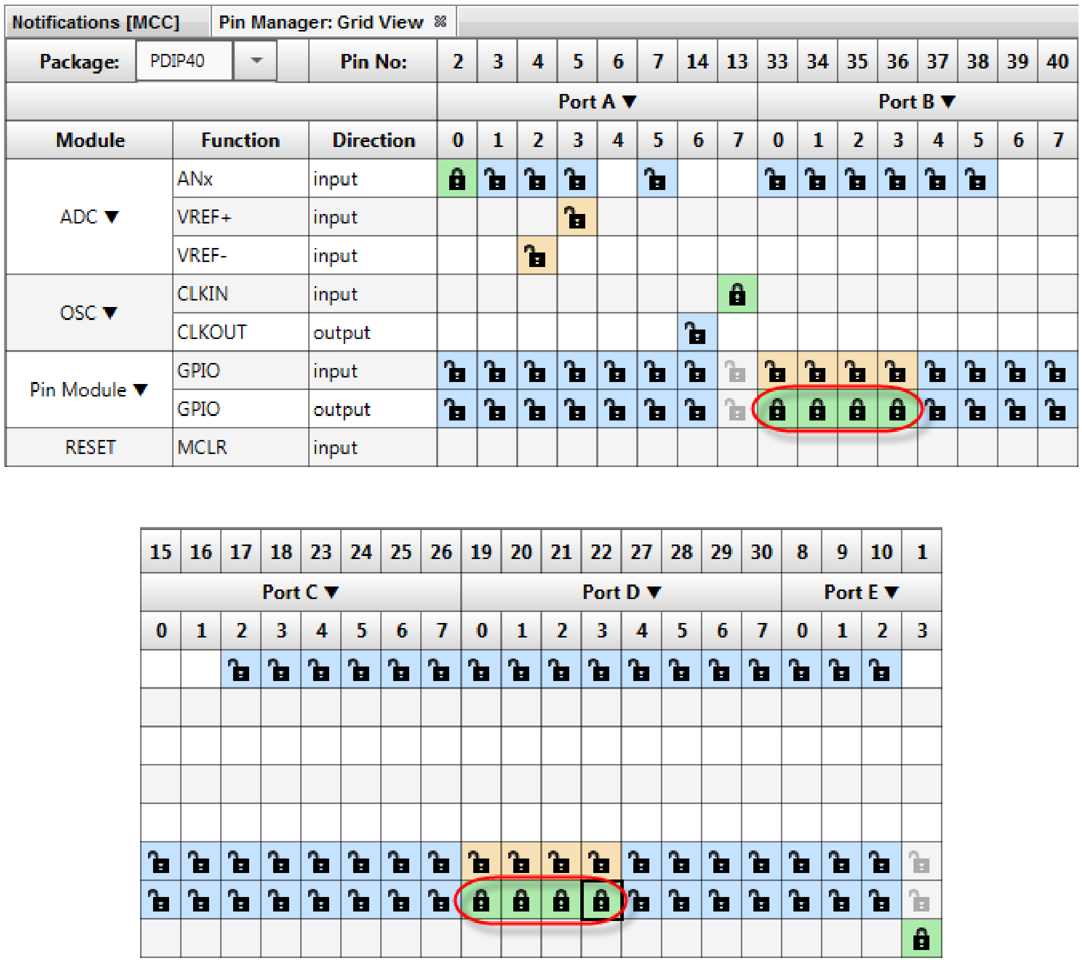
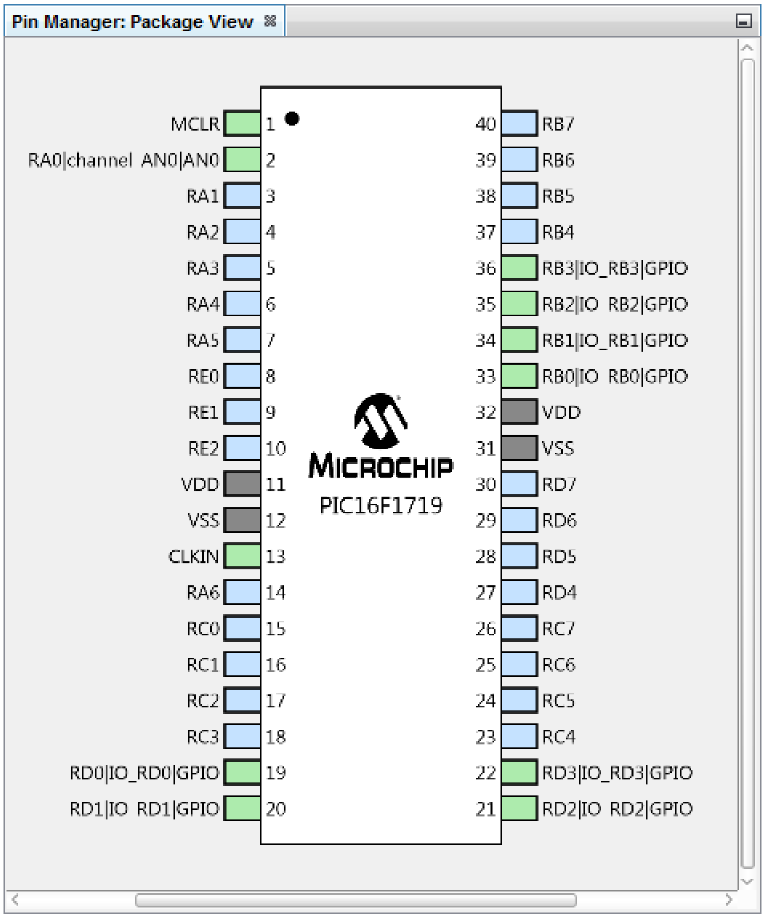


FIGURE 8: ADC PROJECT GPIO PIN RESOURCE - PACKAGE

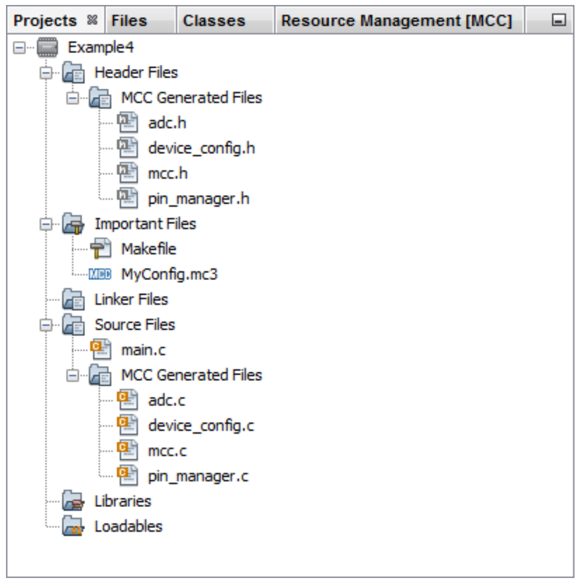


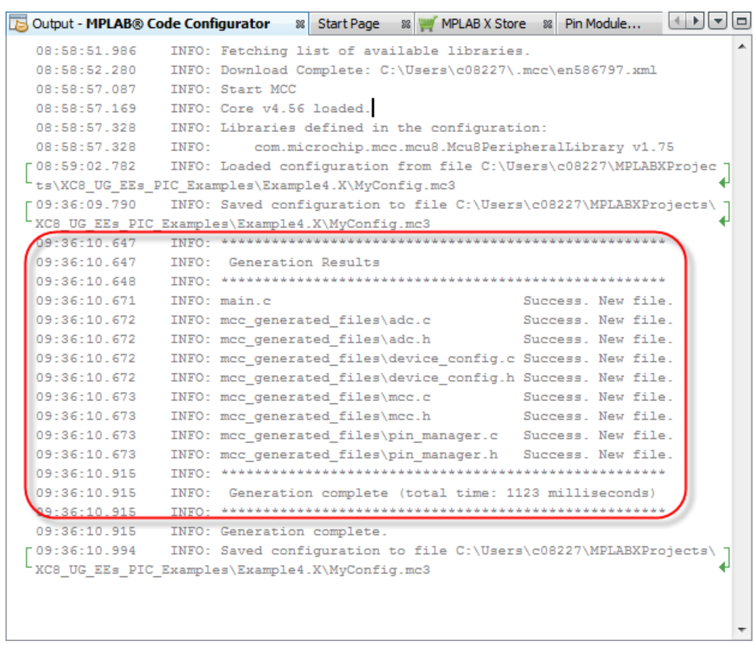
 © 2015-2018 Microchip Technology Inc.

Once the 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.

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 9: 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 red.

```
/**
  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"

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

    // <No interrupts used - see generated main.c file for code.>

    while (1) {

        // Select A/D channel        see Section 4.2

        ADC_SelectChannel(channel_AN0);

        // Start A/D conversion
        ADC_StartConversion();

        // Wait for ADC to complete        see Section 4.3

        while(!ADC_IsConversionDone());

        // Write to Port Latches        see Section 4.4

        LATD = ADRESH; // RD[0:3] = LED[0:3]
        LATB = (ADRESH >> 4); // RB[0:3] = LED[4:7]

    }
}
/**
 End of File
 */
```


© 2015-2018 Microchip Technology Inc.

4.2 Select A/D Channel and Start Conversion

From the `adc.c` module, use the function:

```
void ADC_SelectChannel(adc_channel_t channel)
```

The variable `channel` is of typedef `adc_channel_t` defined in `adc.h`. For this example, pot input is on RA0, so select `channel_AN0`.

Start the A/D conversion using the function:

```
void ADC_StartConversion()
```

4.3 Wait for ADC to compete

From the `adc.c` module, use the function:

```
bool ADC_IsConversionDone()
```

This function returns the negated value of the `ADCON0bits.GO_nDONE` bit (defined in the device header file). However, the actual value of this bit is desired in the `main while` loop, so the return value is negated again.

4.4 Write to Port Latches

As only 8 LEDs are available, just the value from ADRESH is displayed. The lower bits are displayed via `LATD` on LEDs 0 through 3, and the upper bits are shifted so they can be displayed via `LATB` on LEDs 4 through 7.

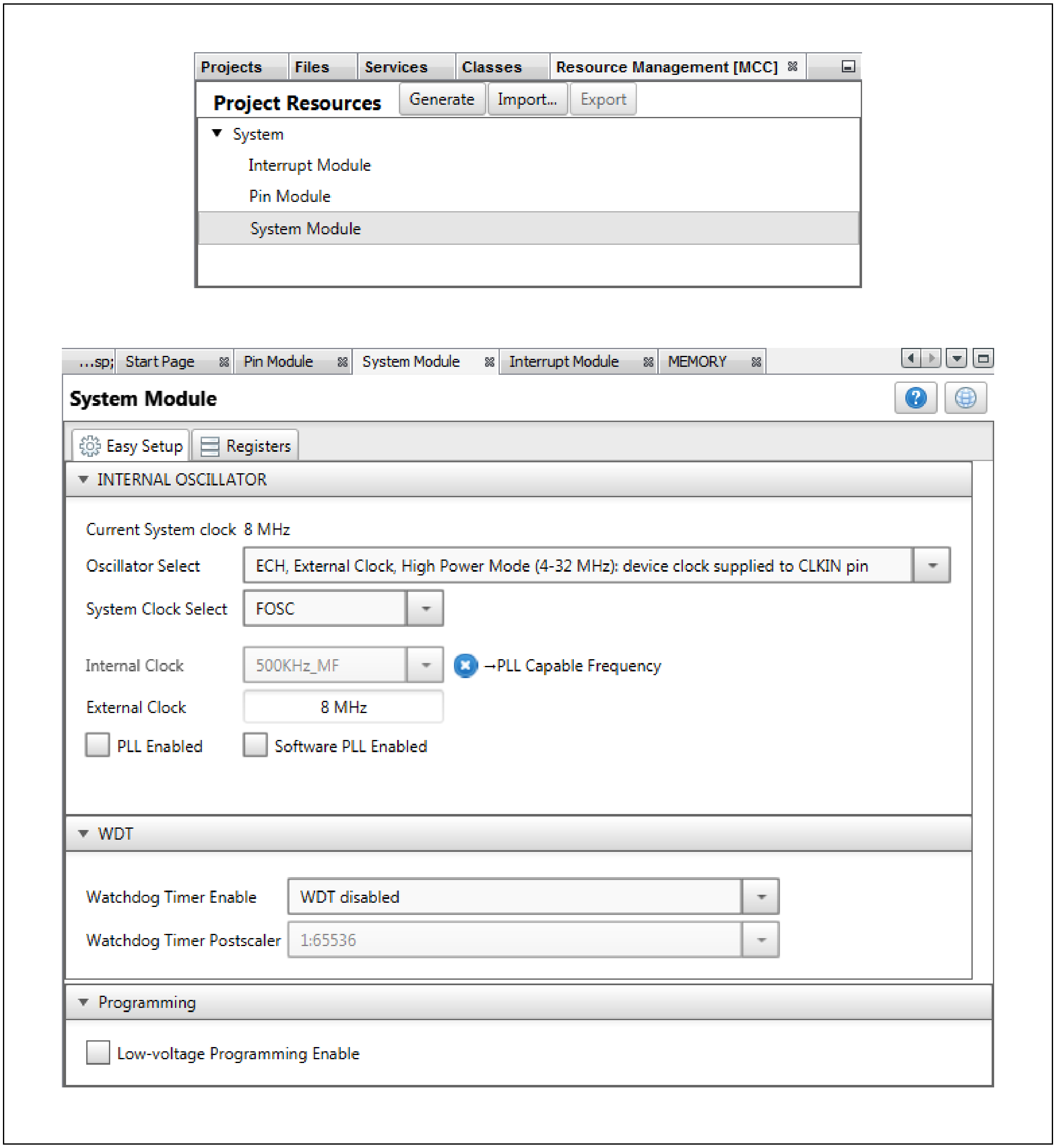
5. DISPLAY EEPROM DATA VALUES ON LEDS

This example uses another Microchip device, the PIC16F1939 MCU, to demonstrate how to write to and read from EEPROM Data (EEData). Read values are displayed on Port D and Port B LEDs.

Again, MPLAB Code Configurator (MCC) is used to generate most 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 A/D"**.

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

FIGURE 10: EEDATA PROJECT SYSTEM RESOURCE CONFIGURATION



© 2015-2018 Microchip Technology Inc.

FIGURE 11: EEDATA PROJECT MEMORY RESOURCE CONFIGURATION

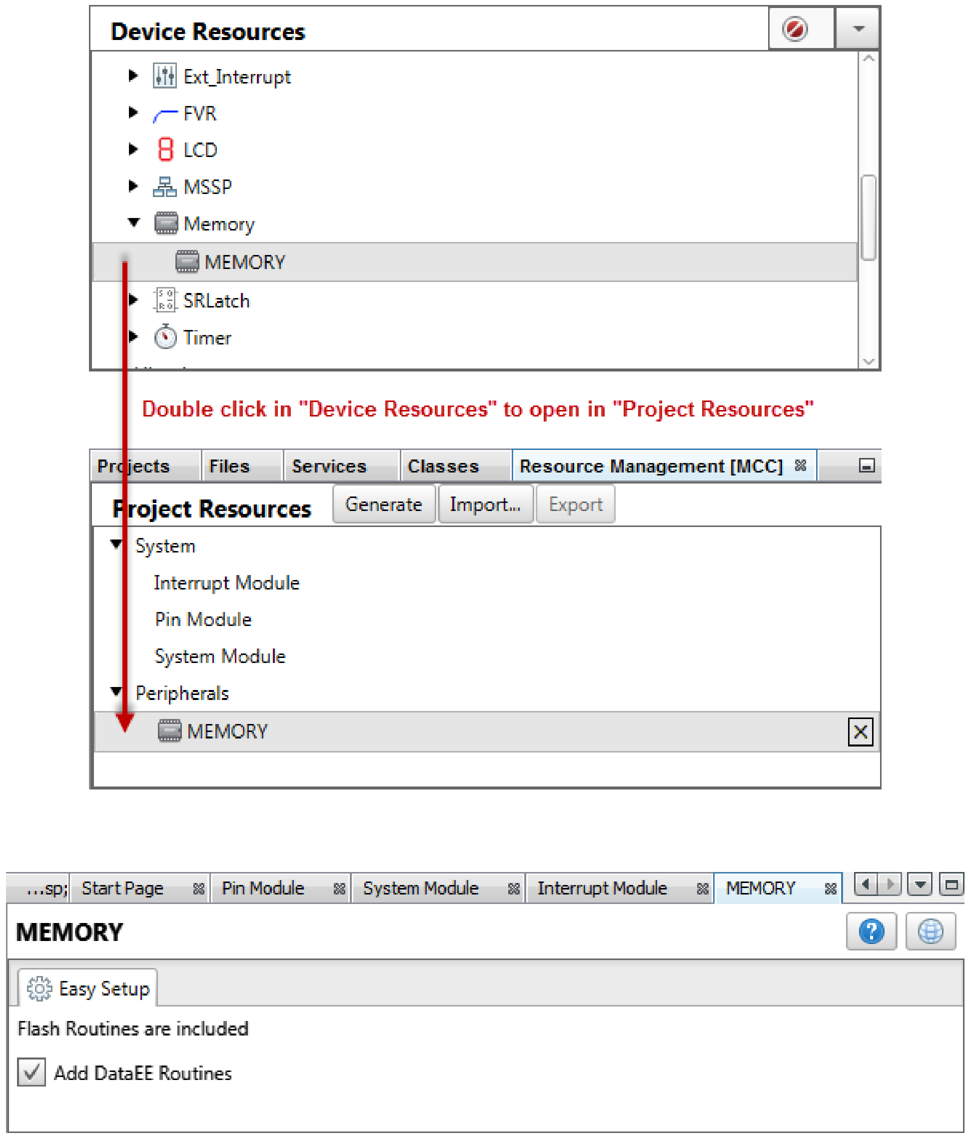
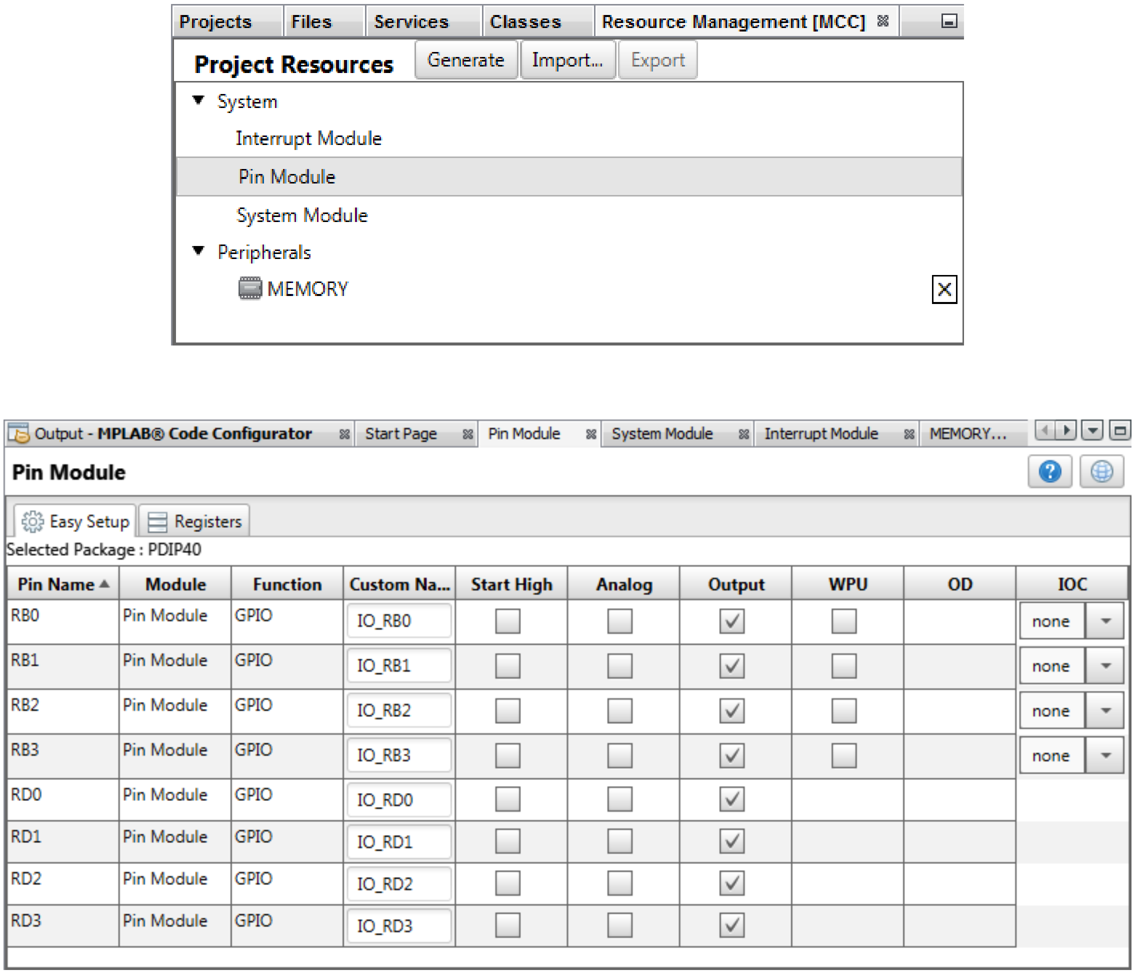


FIGURE 12: EEDATA PROJECT PIN RESOURCE CONFIGURATION



Pins RB0:3 and RD0:3 will appear in the window above when they are selected in Figure 13. Once visible in the window, pin configurations can be selected for each pin.

© 2015-2018 Microchip Technology Inc.

FIGURE 13: EEDATA PROJECT GPIO PIN RESOURCE- GRID

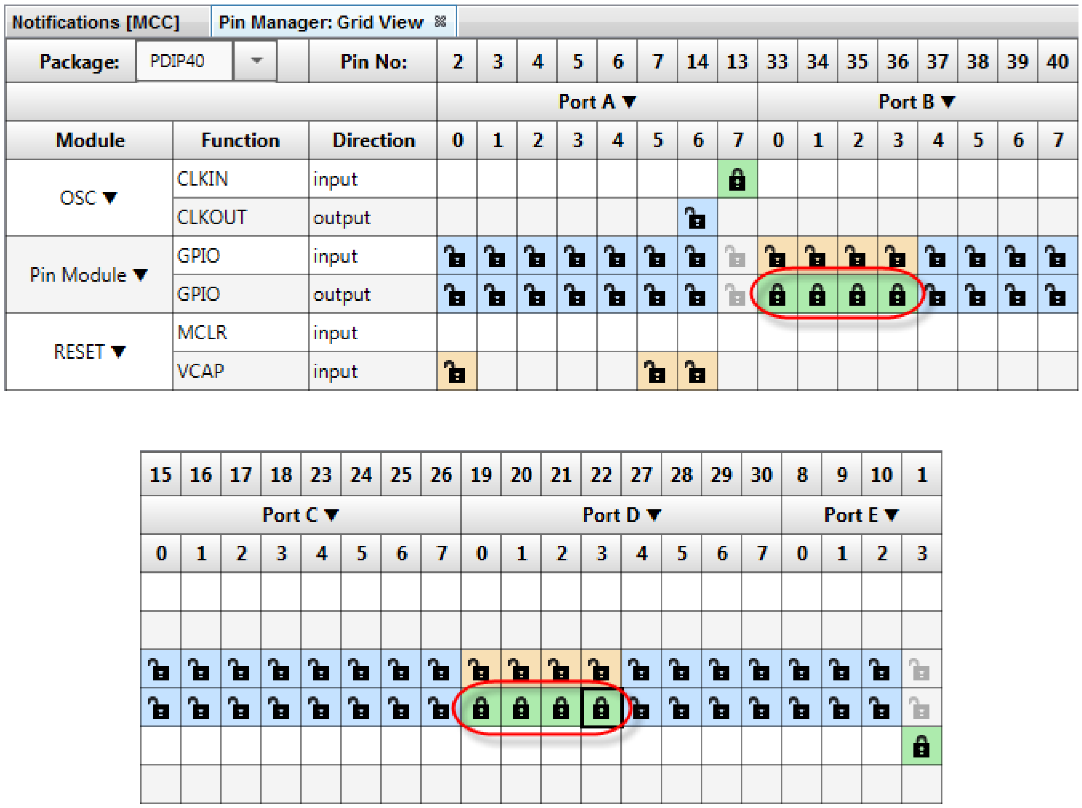
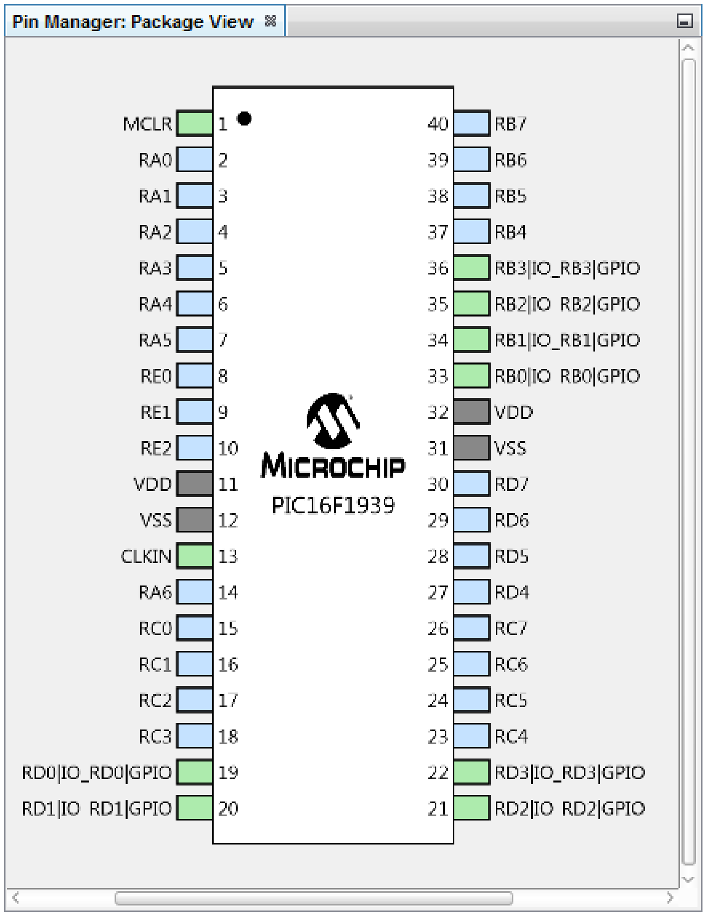


FIGURE 14: EEDATA PROJECT GPIO PIN RESOURCE - PACKAGE

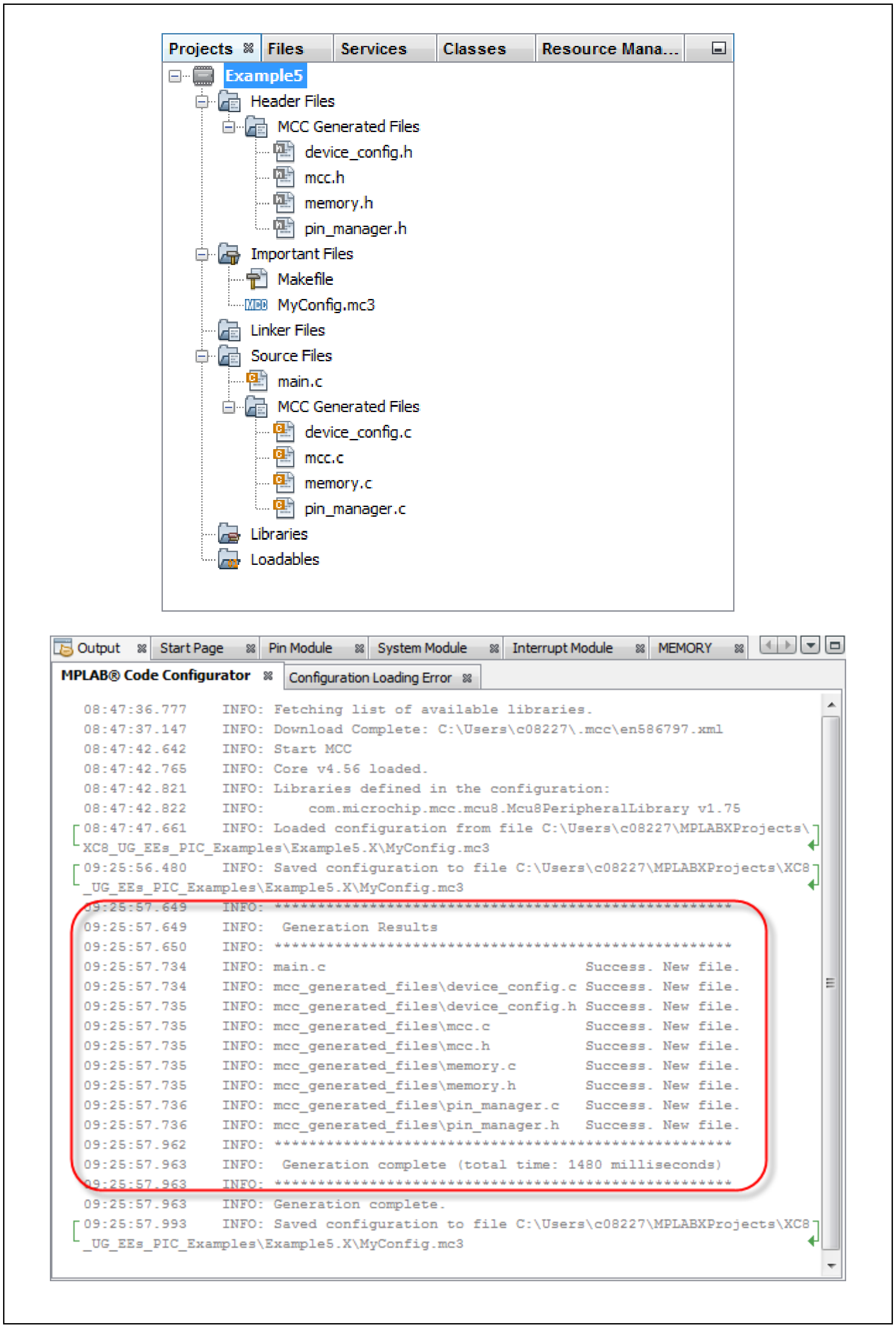


© 2015-2018 Microchip Technology Inc.

When the 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.

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 15: EEDATA PROJECT TREE FOR CODE GENERATED BY MCC



5.1 `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 is in red.

```
/**
  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"

#define NUM_EE_VALUES 64
#define INSTR_CYCLE_DELAY 25000

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

    // <No interrupts used - see generated main.c file for code.>

    // Declare RAM array, loop variable          see Section 5.2

    volatile uint8_t RAMArray[NUM_EE_VALUES];
    uint8_t i;

    // Write initial values to EEPROM Data          see Section 5.3
    PIR2bits.EEIF = 0x0; // clear write flag

    for(i=0; i<NUM_EE_VALUES; i++){
        DATAEE_WriteByte(_EEADRL_EEADRL_POSN + i, i);
        while(!PIR2bits.EEIF);  // check for write finished
        PIR2bits.EEIF = 0x0;
    }

    while(1){
        // Read from EEPROM and display          see Section 5.4
        for(i=0; i<NUM_EE_VALUES; i++){
            RAMArray[i] = DATAEE_ReadByte(_EEADRL_EEADRL_POSN + i);
            LATD = RAMArray[i]; // RD[0:3] = LED[0:3]
            LATB = (RAMArray[i] >> 4); // RB[0:3] = LED[4:7]
            _delay(INSTR_CYCLE_DELAY); // delay value change
        }
```


© 2015-2018 Microchip Technology Inc.

```
        // Write to EEPROM in reverse order
        for(i=0; i<NUM_EE_VALUES; i++){
            DATAEE_WriteByte(_EEADRL_EEADRL_POSN +
              (NUM_EE_VALUES - 1) - i, RAMArray[i]);
            while(!PIR2bits.EEIF);  // check for write finished
            PIR2bits.EEIF = 0x0;
        }

    };

}
/**
 End of File
 */
```

5.2 EEData Associated Variables

Variables used to store data from an EEData read or write must match the types specified in the read/write function prototype, referenced from `mcc.h`, and found in `memory.h`:

```
void DATAEE_WriteByte(uint8_t bAdd, uint8_t bData);
uint8_t DATAEE_ReadByte(uint8_t bAdd);
```

From `stdint.h` (also referenced), `uint8_t` is the same as `unsigned char`.

5.3 Write to EEData

EEData is written twice in this example: first to initialize values in EEData memory and second to change the data for dynamic display.

Writing to EEData takes more than one cycle, so a write-complete flag is used to determine when the write is done (`PIR2bits.EEIF`). The flag is cleared initially, and again, after each time the write completes. (This flag must be cleared in software.)

5.4 Read from EEData

After EEData is written, memory values are read into a RAM array and then displayed on Port D and Port B LEDs. The values in the RAM array are used in this write loop to change the values in EEData memory.

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 Example 2) to slow execution.

© 2015-2018 Microchip Technology Inc.

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," and then select "Standalone Project."
   b) **Select Device:** Select the example device.
   c) **Select Header:** None.
   d) **Select Tool:** Select your hardware debug tool, SNxxxxxx. If you do not see a serial number (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 XC8 (*latest version number*) [`bin` *location*]. If you do not see a compiler under XC8, ensure the compiler is correctly installed and that MPLAB X IDE is aware of it (*Tools>Options*, **Embedded** button, **Build Tools** tab). See MPLAB XC8 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 box, click on the "XC8 Compiler" category, select the "Preprocessing and messages" option category, and check "Use CCI syntax." Click the **OK** button.

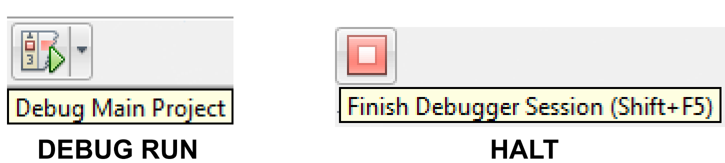
A.3 Debug the Examples

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 opens.
   b) Under "File name," enter a name (e.g., example*n)*, where *n* is the example number.
   c) Click **Finish**. The file opens 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 the `main.c` file 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 16: TOOLBAR ICONS



© 2015-2018 Microchip Technology Inc.

B. GET SOFTWARE AND HARDWARE

For the MPLAB XC8 projects in this document, the Explorer 8 board (with either a PIC16F1719 or PIC16F1939 MCU) 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 XC8 C Compiler

MPLAB X IDE v5.10 and later can be found at:

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

The MPLAB XC8 C compiler v2.00 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® MCUs

The PIC MCUs used in the examples are available at:

https://www.microchip.com/PIC16F1719

https://www.microchip.com/PIC16F1939

B.4 Get and Set Up the Explorer 8 Board

The Explorer 8 development kit (DM160228) is available at:

https://www.microchip.com/DM160228

Jumpers were set up as shown in the following tables.

TABLE 1-1: JUMPER SELECTS FOR PROJECTS

Jumper	Selection	Description
J2	BRD+5V	Power board from power supply (not USB)
J14	+5V	Device Power level
J24	Open	+5V used (not 3.3V)
J7	Closed	Enable LEDs on Port D <RD0:3>
J21	Closed	Enable LEDs on Port B <RB0:3>
J36	OSC1 to RA7	OSC1 CLKIN (8MHz External Oscillator)
J37	OSC2 to RA6	OSC2 CLKOUT (8MHz External Oscillator)
J51	PGD to RB7	ICSPDAT
J52	PGC to RB6	ISCPCLK

TABLE 1-2: JUMPER SELECTS NOT USED

Jumper	Selection	Description
JP2	Closed	LCD not used
J22, J23, J53, J54	Open	LCD not used
J15, J16	Open	Digilent Pmod™ Connectors not used
J43, J44, J45, J46, J47	Open	mikroBUS not used
J41, J42, J48, J49, J50	Open	mikroBUS not used
J4, J31	VCAP	RA5, RA4 not used

© 2015-2018 Microchip Technology Inc.

B.5 Get Microchip Debug Tools

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

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

© 2015-2018 Microchip Technology Inc.

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.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.** Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

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.

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

ISO/TS 16949

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2018, Microchip Technology Incorporated, All Rights Reserved.

ISBN: 978-1-5224-4032-1

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

© 2015-2018 Microchip Technology Inc.