MPLAB® XC32 User’s Guide for Embedded Engineers

INTRODUCTION

This document presents five code examples for 32-bit devices and the MPLAB® XC32 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 16/32 board with a PIC32MX470F512L Plug-In Module (PIM). For more information, see **Section B. "Get Software and Hardware"**.

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements   ◄──── see Section 1.1

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect 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

#define LEDS_ON_OFF 0x55   ◄──── see Section 1.3

int main(void) {

    // Port A access ◄──── see Section 1.4

    TRISA = 0x0000;       // set all port bits to be output
    LATA = LEDS_ON_OFF;   // 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 selection 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 = HS
  ```
- **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 CP = OFF
  ```

Different configuration bits may need to be set up to use another 32-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.

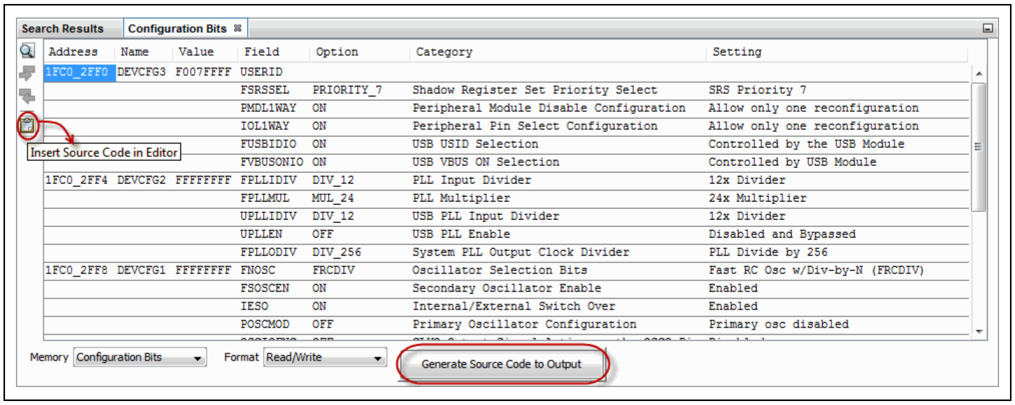
For more about configuration bits that are available for each device, see the following file in the location where MPLAB XC32 was installed:

MPLAB XC32 Installation Directory/docs/PIC32ConfigSet.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 WINDOW



Address	Name	Value	Field	Option	Category	Setting
1FC0_2FF0	DEVCFG3	F007FFFF	USERID			
			FSRSSEL	PRIORITY_7	Shadow Register Set Priority Select	SRS Priority 7
			PMDL1WAY	ON	Peripheral Module Disable Configuration	Allow only one reconfiguration
			IOL1WAY	ON	Peripheral Pin Select Configuration	Allow only one reconfiguration
			FUSBIDIO	ON	USB USID Selection	Controlled by the USB Module
			FVBUSONIO	ON	USB VBUS ON Selection	Controlled by USB Module
1FC0_2FF4	DEVCFG2	FFFFFFFF	FPLLIDIV	DIV_12	PLL Input Divider	12x Divider
			FPLLMUL	MUL_24	PLL Multiplier	24x Multiplier
			UPLLIDIV	DIV_12	USB PLL Input Divider	12x Divider
			UPLLEN	OFF	USB PLL Enable	Disabled and Bypassed
			FPLLODIV	DIV_256	System PLL Output Clock Divider	PLL Divide by 256
1FC0_2FF8	DEVCFG1	FFFFFFFF	FNOSC	FRCDIV	Oscillator Selection Bits	Fast RC Osc w/Div-by-N (FRCDIV)
			FSOSCEN	ON	Secondary Oscillator Enable	Enabled
			IESO	ON	Internal/External Switch Over	Enabled
			POSCMOD	OFF	Primary Oscillator Configuration	Primary osc disabled

Memory: Configuration Bits Format: Read/Write

Once you have the settings you want, click in your code where you want the `pragma` directives placed, before `main()`, and then click the **Insert Source Code in Editor** icon. Alternately you can click **Generate Source Code to Output** and then copy the `pragma` directives from the Output window into your code.

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 may be found in the MPLAB XC32 installation directory in the `pic32mx/include` 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.

1.3 Define Macro for LED Values

The value to be written to the LEDs, as explained in the next section, has been assigned to a descriptive macro (`LEDS_ON_OFF`), i.e., LEDs D3, D5, D7, and D9 will be on and LEDs D4, D6, D8, and D10 will be off. See **Section B.5 "Get and Set Up the Explorer 16/32 Board"** for the link to Explorer 16/32 documentation, including the board schematic.

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, `pic32mx/include/proc,` in the compiler's 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. Also, Port A has no analog I/O so all pins are digital I/O by default. For devices with ports that have analog I/O, the analog must be disabled (e.g., using the `ADxPCFT` register) to ensure digital I/O operation.

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, `TRISA` is 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. Code that has been added is red.

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect Disabled

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

#include <xc.h>

#define LEDS_ON_OFF 0x55
#define LEDS_OFF_ON 0xAA

void delay (void)
{
    int n = 50000;
    while(n>0) {n--;}
}
```

```
int main(void) {

    // Port A access
    TRISA = 0x0;    // set all port bits to be output

    while(1) {   ← see Section 2.1

        LATA = LEDS_ON_OFF; // write to port latch

        // delay value change   ← see Section 2.2
        delay();

        LATA = LEDS_OFF_ON; // write to port latch

        // delay value change
        delay();
    }
    return -1;
}
```

2.1 The `while()` Loop and Variable Values

To make the LEDs on Port A change, the 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. Therefore a `-1` is returned.

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. The function `delay()` is declared and defined above `main()` and called twice in `main()` code.

> **Note:** Do not use compiler optimizations or the delay loop will be removed (use `-O0`). See the next example for a different way to delay code execution.

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, the core timer interrupt will be used. At each interrupt, a variable value is increased and displayed on the LEDs.

The core timer is used in this example because it is consistent across all PIC32 MCUs and it increments at a constant rate (every 2 system clock cycles) with no pre/postscaler set up. Other device timers can be used for a delay, but care must be taken if other modules are also using the timer. Code that has been added is red.

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect Disabled

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

#include <xc.h>
#include <cp0defs.h>
#include <sys/attribs.h>
```

see **Section 3.1**

```
// CORE_TICK_RATE = FOSC/2/TOGGLES_PER_SEC
// FOSC/2 = Core timer clock frequency = 8MHz/2=4MHz
// TOGGLES_PER_SEC = Toggle LED x times per second; x=5
#define CORE_TICK_RATE          800000u

// Interrupt function     ←  see Section 3.2

void __ISR(_CORE_TIMER_VECTOR, IPL2SOFT) CTInterruptHandler(void)
{
    // static variable for permanent storage duration
    static unsigned char portValue = 0;
    // variables for Compare period
    unsigned long ct_count = _CP0_GET_COUNT();
    unsigned long period = CORE_TICK_RATE;

    // write to port latch
    LATA = portValue++;
    // update the Compare period
    period += ct_count;
    _CP0_SET_COMPARE(period);
    // clear the interrupt flag
    IFS0CLR = _IFS0_CTIF_MASK;
}

int main(void) {

    unsigned int stat_gie, cause_val;

    // Disables interrupts by clearing the global interrupt enable bit
    // in the STATUS register.
    stat_gie = __builtin_disable_interrupts();

    // Port A access
    TRISA = 0x0; // set all port bits to be output
    LATA = 0x0;  // clear all bits

    // Configure the core timer     ←  see Section 3.3
    // clear the CP0 Count register
    _CP0_SET_COUNT(0);
    // set up the period in the CP0 Compare register
    _CP0_SET_COMPARE(CORE_TICK_RATE);
    // halt core timer and program at a debug breakpoint
    _CP0_BIC_DEBUG(_CP0_DEBUG_COUNTDM_MASK);

    // Set up core timer interrupt     ←  see Section 3.4
    // clear core timer interrupt flag
    IFS0CLR = _IFS0_CTIF_MASK;
    // set core time interrupt priority of 2
    IPC0CLR = _IPC0_CTIP_MASK;
    IPC0SET = (2 << _IPC0_CTIP_POSITION);
    // set core time interrupt subpriority of 0
    IPC0CLR = _IPC0_CTIS_MASK;
    IPC0SET = (0 << _IPC0_CTIS_POSITION);
    // enable core timer interrupt
    IEC0CLR = _IEC0_CTIE_MASK;
    IEC0SET = (1 << _IEC0_CTIE_POSITION);
    // set the CP0 Cause register Interrupt Vector bit
    cause_val = _CP0_GET_CAUSE();
    cause_val |= _CP0_CAUSE_IV_MASK;
    _CP0_SET_CAUSE(cause_val);

    // enable multi-vector interrupts
    INTCONSET = _INTCON_MVEC_MASK;

    // enable global interrupts
    __builtin_enable_interrupts();

    while(1);

    return -1;
}
```

3.1 Additional Header Files

In addition to xc.h, other header files need to be included: cp0defs.h for CP0 macros and sys/attribs.h for ISR macros.

3.2 The Interrupt Function

For this example, CTInterruptHandler() is made into an interrupt function by using the ISR macro __ISR(v,IPL), where v is the interrupt vector for the core timer and IPL is the interrupt priority level (2) and context-saving method (via software) expressed as IPL2SOFT. For more on ISRs, see the "Interrupts" chapter of the *MPLAB XC32 C/C++ Compiler User's Guide* (DS50001686).

Within the interrupt function, the counter portValue is incremented and displayed on the LEDs.

To clear the interrupt, the CP0 Compare register must be written. The value in the Compare register will be compared to a future value of the core timer to generate the next interrupt. The current value of the core timer is found from _CP0_GET_COUNT().

Finally the interrupt flag is cleared.

3.3 Core Timer Set Up

The 32-bit core timer is initially set to zero. The Compare register is set to an initial value of the CORE_TICK_RATE. When the core timer reaches the compare value, an interrupt will be triggered.

Additionally, the core timer has been set to halt on a breakpoint to aid in debugging.

For more on the core timer, see the *PIC32 Family Reference Manual*, "Section 2. CPU for Devices with M4K® Core" (DS61113).

3.4 Core Timer Interrupt

Setting up the core timer interrupt takes several steps.

At the beginning of main code __builtin_disable_interrupts() is used to disable global interrupts. Just before the while(1) loop __builtin_enable_interrupts() is used to enable global interrupts.

The core timer interrupt flag is cleared using macros found in the device header files (accessed from xc.h).

The interrupt priority and subpriority are set using device macros. The priority here must match the priority of the interrupt function, which is 2.

The core timer and multi-vector interrupts are enabled using device macros. The interrupt vector bit in the CP0 Cause register is set using device and CP0 macros.

4. DISPLAY POTENTIOMETER VALUES ON LEDS USING AN ADC (MPLAB HARMONY)

This example uses the same device and the Port A LEDs as example 3. However, in this example, values from a potentiometer on the demo board provide Analog-to-Digital Converter (ADC) input through Port B (RB2/AN2) that is converted and displayed on the LEDs.

Instead of generating code by hand, MPLAB Harmony is used. Download the MPLAB Harmony Integrated Software Framework at:

http://www.microchip.com/mplab/mplab-harmony

The MPLAB Harmony Configurator (MHC) is an MPLAB X IDE plug-in for GUI set up of MPLAB Harmony. The plugin is 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.

This example is based on the `adc_pot` example found, in Windows®, under:

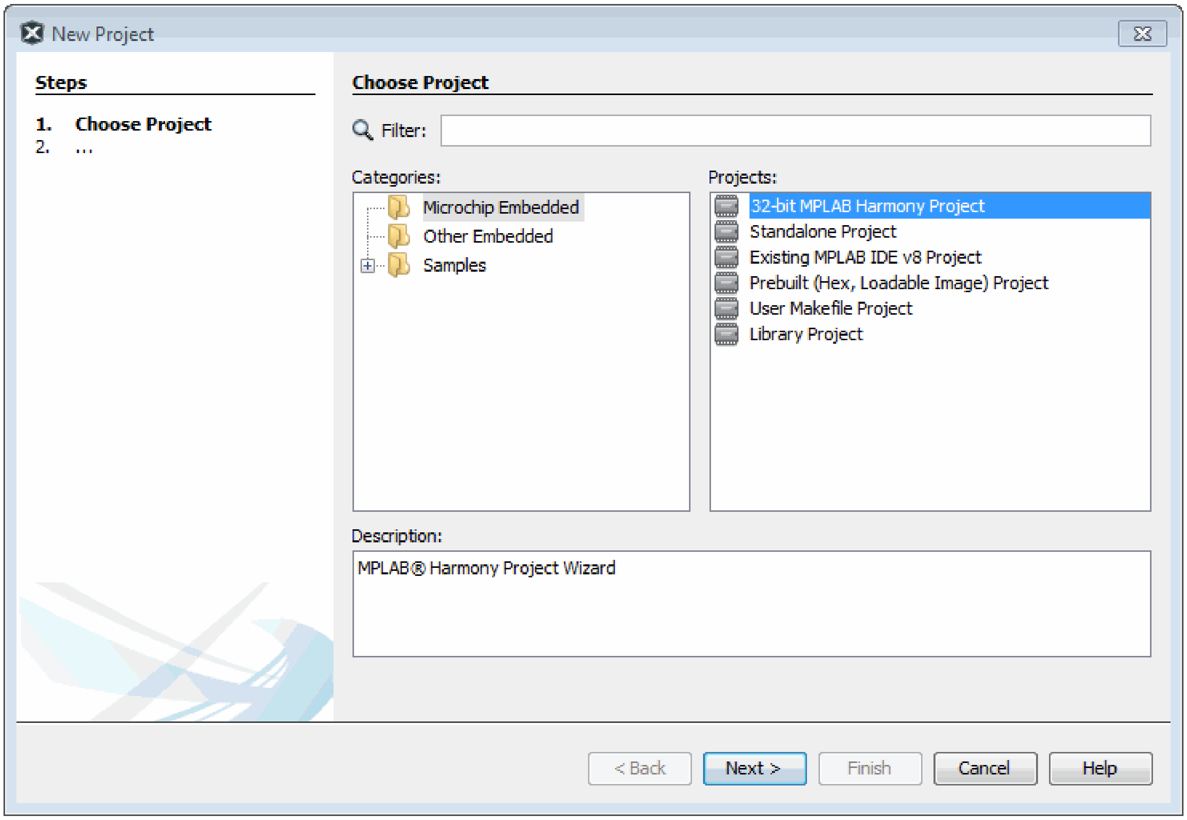
```
C:\microchip\harmony\v1_10\apps\examples\peripheral\adc\adc_pot
```

4.1 Create an MPLAB Harmony Project in MPLAB X IDE

The dialogs below are set up for this example.

In MPLAB X IDE, select *File>New Project*.

FIGURE 2: NEW MPLAB HARMONY PROJECT - STEP 1



New Project

Steps

1. Choose Project
2. ...

Choose Project

Filter:

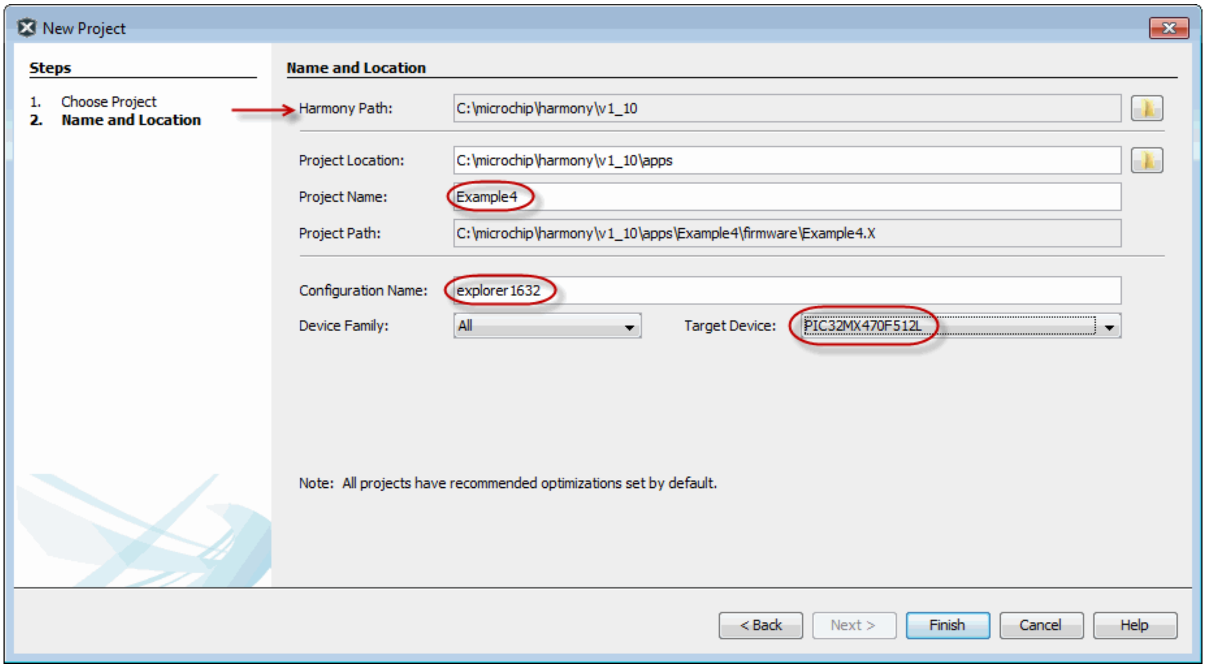
Categories: Microchip Embedded (selected), Other Embedded, Samples

Projects: 32-bit MPLAB Harmony Project (selected), Standalone Project, Existing MPLAB IDE v8 Project, Prebuilt (Hex, Loadable Image) Project, User Makefile Project, Library Project

Description: MPLAB® Harmony Project Wizard

Ensure the Harmony Path points to your installation of MPLAB Harmony.

FIGURE 3: NEW MPLAB HARMONY PROJECT - STEP 2



New Project

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Harmony Path: C:\microchip\harmony\v1_10

Project Location: C:\microchip\harmony\v1_10\apps

Project Name: Example4

Project Path: C:\microchip\harmony\v1_10\apps\Example4\firmware\Example4.X

Configuration Name: explorer1632

Device Family: All

Target Device: PIC32MX470F512L

Note: All projects have recommended optimizations set by default.

4.2 Configure the MPLAB Harmony Project

Based on your project setup, the MHC will open with some clock information already populated. Text highlighted in blue signifies changes. For this example, do not make any changes to the Clock settings.

Configure the ADC driver as shown in Figure 6 and Board Support Packages (BSP) as shown in Figure 7.

FIGURE 4: MPLAB HARMONY PROJECT AND MHC

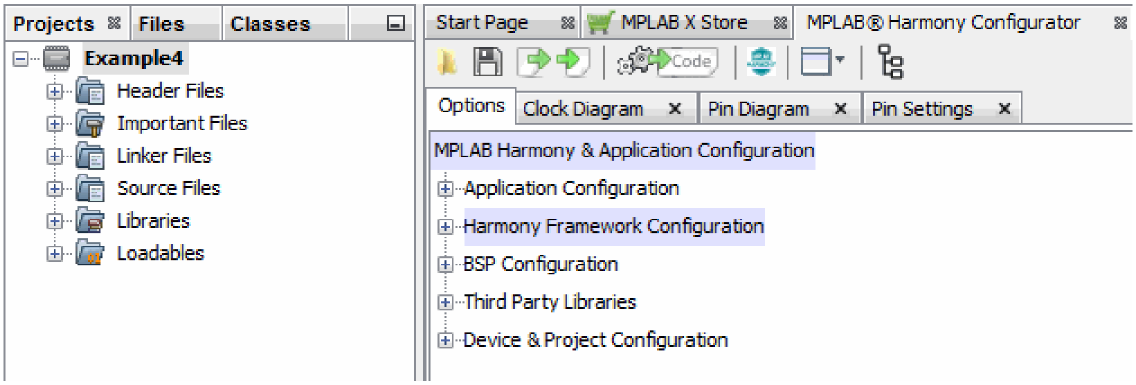
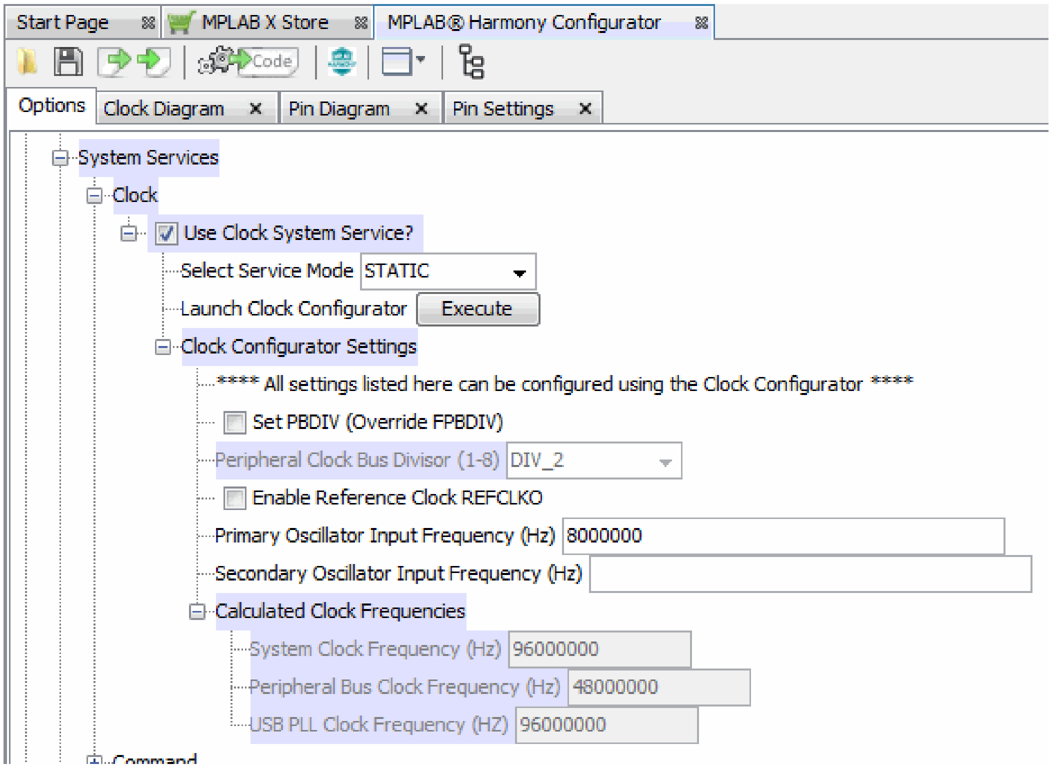


FIGURE 5: HARMONY FRAMEWORK CONFIGURATION - CLOCK



☑ Use Clock System Service?

Select Service Mode STATIC

Launch Clock Configurator

**** All settings listed here can be configured using the Clock Configurator ****

□ Set PBDIV (Override FPBDIV)

Peripheral Clock Bus Divisor (1-8) DIV_2

□ Enable Reference Clock REFCLKO

Primary Oscillator Input Frequency (Hz) 8000000

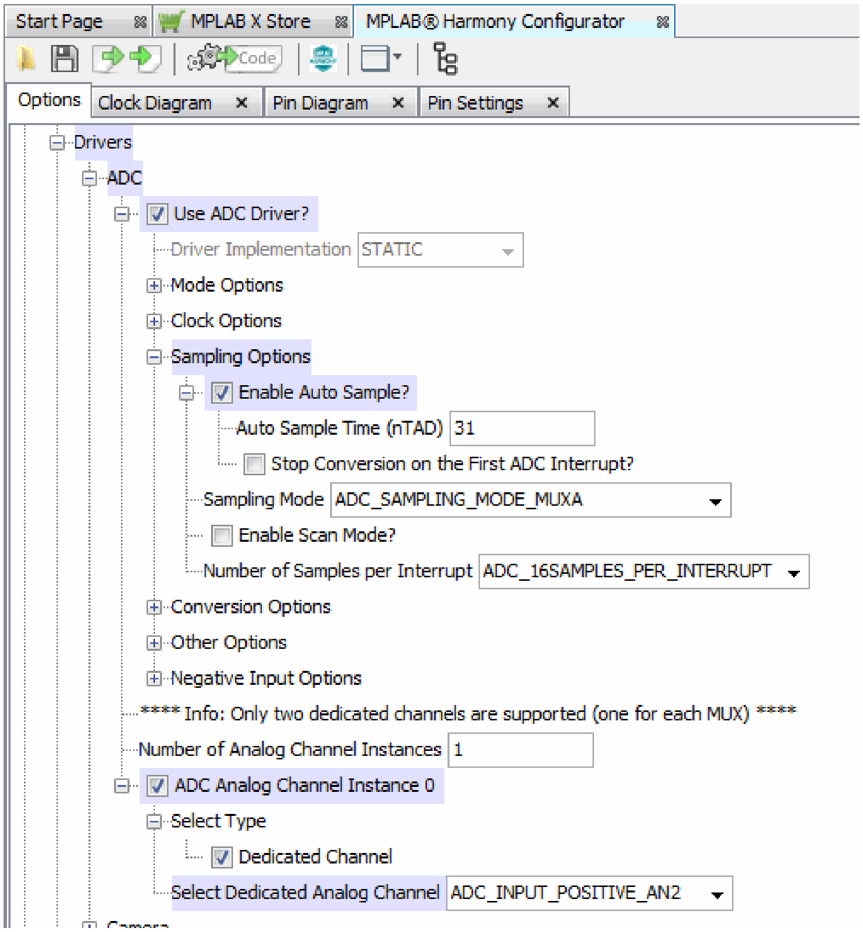
Secondary Oscillator Input Frequency (Hz)

System Clock Frequency (Hz) 96000000

Peripheral Bus Clock Frequency (Hz) 48000000

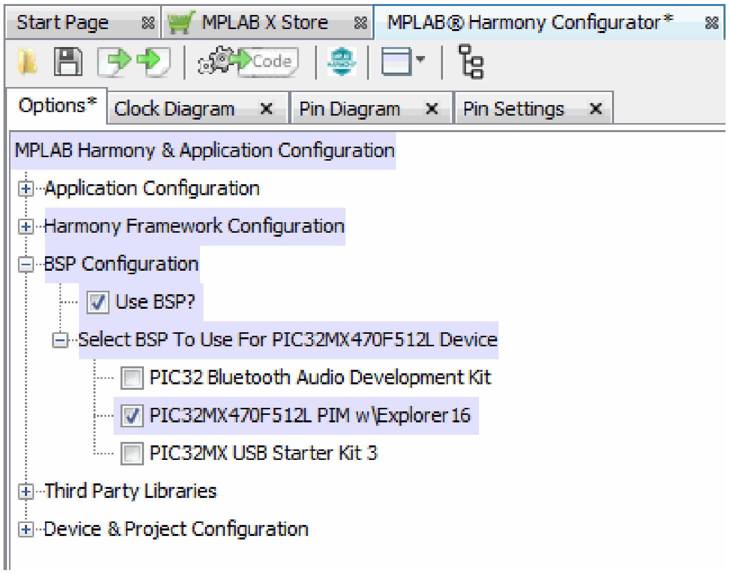
USB PLL Clock Frequency (HZ) 96000000

FIGURE 6: HARMONY FRAMEWORK CONFIGURATION - ADC DRIVER



- Drivers
  - ADC
    - ☑ Use ADC Driver?
      - Driver Implementation: STATIC
      - Mode Options
      - Clock Options
      - Sampling Options
        - ☑ Enable Auto Sample?
          - Auto Sample Time (nTAD): 31
          - □ Stop Conversion on the First ADC Interrupt?
        - Sampling Mode: ADC_SAMPLING_MODE_MUXA
        - □ Enable Scan Mode?
        - Number of Samples per Interrupt: ADC_16SAMPLES_PER_INTERRUPT
      - Conversion Options
      - Other Options
      - Negative Input Options
    - **** Info: Only two dedicated channels are supported (one for each MUX) ****
    - Number of Analog Channel Instances: 1
    - ☑ ADC Analog Channel Instance 0
      - Select Type
        - ☑ Dedicated Channel
      - Select Dedicated Analog Channel: ADC_INPUT_POSITIVE_AN2
  - Camera

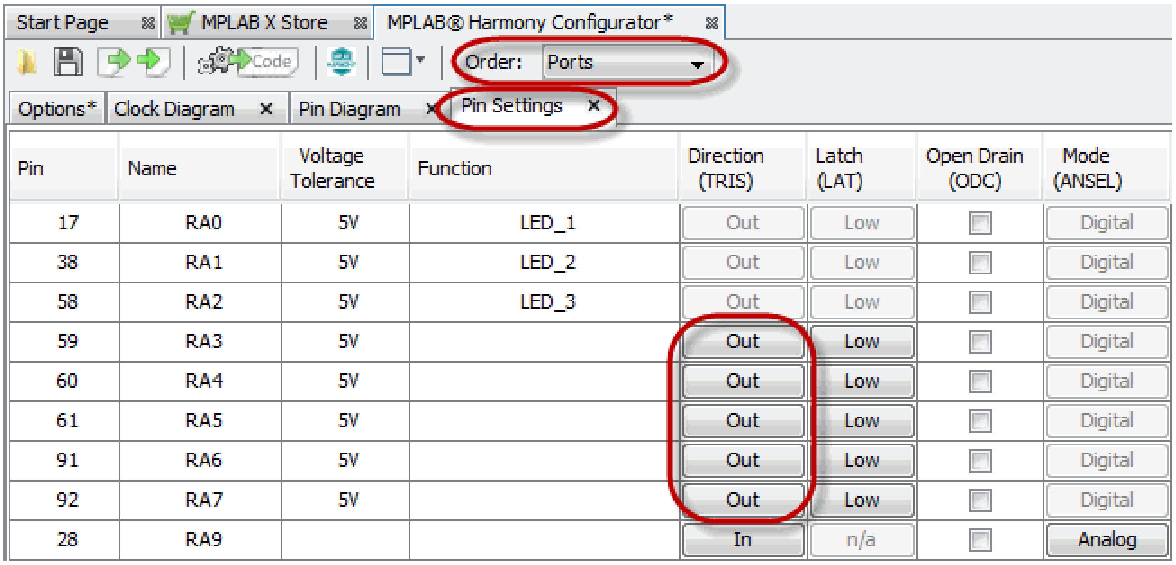
FIGURE 7: ADC PROJECT RESOURCE CONFIGURATION



MPLAB Harmony & Application Configuration

- Application Configuration
- Harmony Framework Configuration
- BSP Configuration
  - ☑ Use BSP?
  - Select BSP To Use For PIC32MX470F512L Device
    - □ PIC32 Bluetooth Audio Development Kit
    - ☑ PIC32MX470F512L PIM w\Explorer16
    - □ PIC32MX USB Starter Kit 3
- Third Party Libraries
- Device & Project Configuration

FIGURE 8: ADC PROJECT PIN SETTINGS

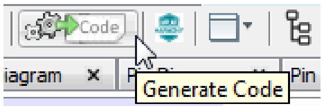


Order: Ports

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)
17	RA0	5V	LED_1	Out	Low	□	Digital
38	RA1	5V	LED_2	Out	Low	□	Digital
58	RA2	5V	LED_3	Out	Low	□	Digital
59	RA3	5V		Out	Low	□	Digital
60	RA4	5V		Out	Low	□	Digital
61	RA5	5V		Out	Low	□	Digital
91	RA6	5V		Out	Low	□	Digital
92	RA7	5V		Out	Low	□	Digital
28	RA9			In	n/a	□	Analog

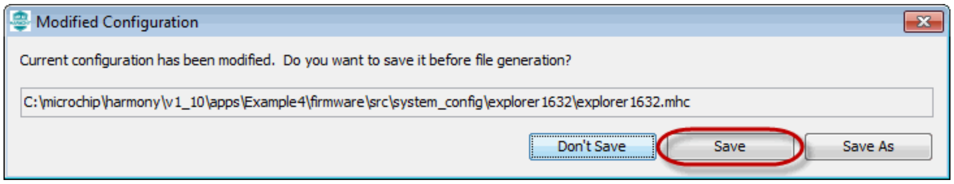
4.3 Generate Code and Edit Application Files

When MHC is set up as shown in the previous figures, click the **Generate Code** button on the **MPLAB Harmony Configurator** tab.



Save the configuration (Figure 9) and generate the project code (Figure 10).

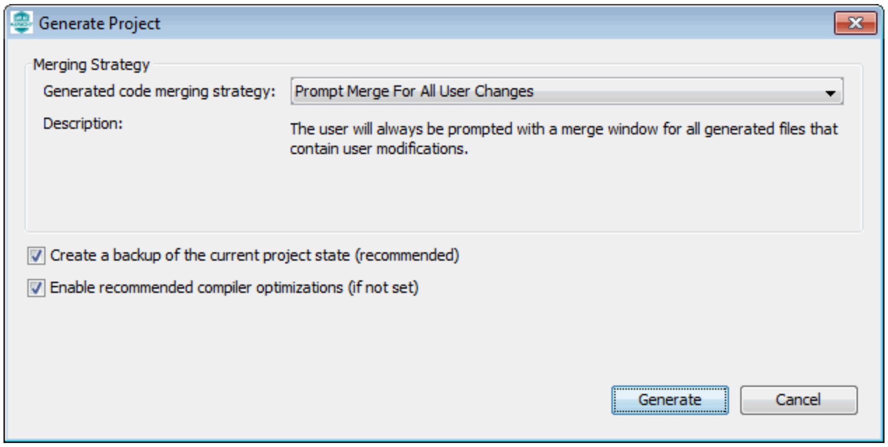
FIGURE 9: SAVE CONFIGURATION


Modified Configuration

Current configuration has been modified. Do you want to save it before file generation?

C:\microchip\harmony\v1_10\apps\Example4\firmware\src\system_config\explorer1632\explorer1632.mhc

FIGURE 10: GENERATE PROJECT CODE


Generate Project

Merging Strategy

Generated code merging strategy: Prompt Merge For All User Changes

Description: The user will always be prompted with a merge window for all generated files that contain user modifications.

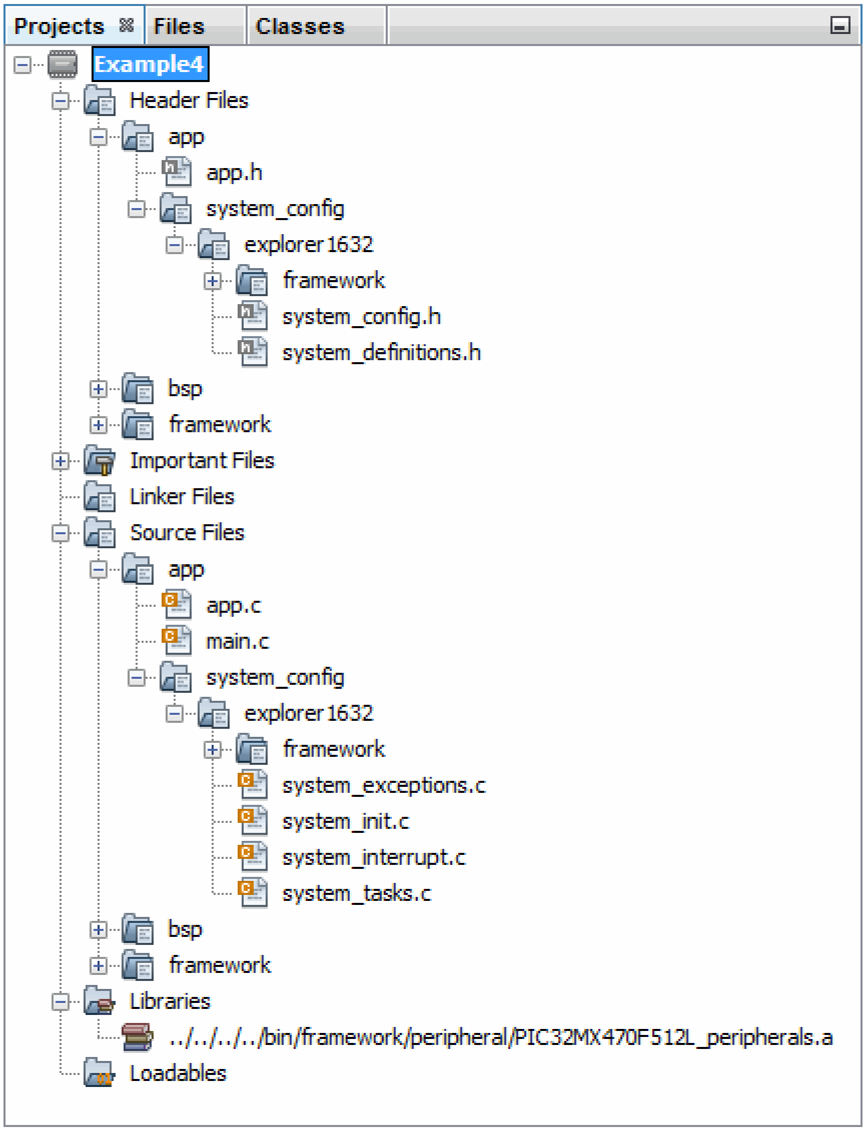
☑ Create a backup of the current project state (recommended)

☑ Enable recommended compiler optimizations (if not set)

Code generated by the MPLAB Harmony is modular, as shown in Figure 11. The application files (`app.h` and `app.c`) are the ones edited for this example.

For more information on using Flash memory, see the *PIC32 Family Reference Manual*, "Section 17. 10-Bit A/D Converter" (DS61104).

FIGURE 11: ADC PROJECT TREE FOR CODE GENERATED BY MHC



```
Example4
  Header Files
    app
      app.h
      system_config
        explorer1632
          framework
          system_config.h
          system_definitions.h
    bsp
    framework
  Important Files
  Linker Files
  Source Files
    app
      app.c
      main.c
      system_config
        explorer1632
          framework
          system_exceptions.c
          system_init.c
          system_interrupt.c
          system_tasks.c
    bsp
    framework
  Libraries
    ../../../../bin/framework/peripheral/PIC32MX470F512L_peripherals.a
  Loadables
```

4.4 app.h Modified Code

The app.h template file has been edited as shown below. Some comments have been removed, as described in < >. Code that has been added is red.

```
/********************************************************************
  MPLAB Harmony Application Header File

<See generated app.h file for file information.>

********************************************************************/

//DOM-IGNORE-BEGIN
/********************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights 
reserved.

<See generated app.h file for copyright information.>

********************************************************************/
//DOM-IGNORE-END

#ifndef _APP_H
#define _APP_H

#define ADC_NUM_SAMPLE_PER_AVERAGE  16

// ******************************************************************
// ******************************************************************
// Section: Included Files
// ******************************************************************
// ******************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

#endif
// DOM-IGNORE-END

// ******************************************************************
// ******************************************************************
// Section: Type Definitions
// ******************************************************************
// ******************************************************************

// ******************************************************************
/* Application states

  Summary:
    Application states enumeration

  Description:
    This enumeration defines the valid application states. These
    states determine the behavior of the application at various times.
*/

typedef enum
{
   /* Application's state machine's initial state. */
   APP_STATE_INIT=0,
   APP_ADC_WAIT,
   APP_ADC_DISPLAY

} APP_STATES;

// *******************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    Application strings and buffers are be defined outside
    this structure.
*/

typedef struct
{
    /* The application's current state */
    APP_STATES state;

    /* Values for the conversions */
    int potValue;
    int ledMask;

} APP_DATA;

// *******************************************************************
// *******************************************************************
// Section: Application Callback Routines
// *******************************************************************
// *******************************************************************
/* These routines are called by drivers when certain events occur.
*/

// *******************************************************************
// *******************************************************************
// Section: Application Initialization and State Machine Functions
// *******************************************************************
// *******************************************************************

/*********************************************************************
  Function:
    void APP_Initialize ( void )

  Summary:
     MPLAB Harmony application initialization routine.

<See generated app.h file for app init information.>
```

```
*/

void APP_Initialize ( void );

/*********************************************************************
  Function:
    void APP_Tasks ( void )

  Summary:
    MPLAB Harmony Demo application tasks function

<See generated app.h file for app tasks information.>

*/

void APP_Tasks( void );

#endif /* _APP_H */

//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END

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

4.5 app.c Modified Code

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

Some lines are long and wrap on the page. They have been left this way to enable cut-and-paste from this document to an editor.

```
/********************************************************************
  MPLAB Harmony Application Source File

<See generated app.c file for file information.>

*******************************************************************/

// DOM-IGNORE-BEGIN
/********************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.c file for copyright information.>

*******************************************************************/
// DOM-IGNORE-END

// ******************************************************************
// ******************************************************************
// Section: Included Files
// ******************************************************************
// ******************************************************************

#include "app.h"

// ******************************************************************
// ******************************************************************
// Section: Global Data Definitions
// ******************************************************************
// ******************************************************************

// ******************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    This structure should be initialized by the APP_Initialize
    function.

    Application strings and buffers are be defined outside this
    structure.
*/

APP_DATA appData;

// ******************************************************************
// ******************************************************************
// Section: Application Callback Functions
// ******************************************************************
// ******************************************************************

/* TODO:  Add any necessary callback functions.
*/

// ******************************************************************
// ******************************************************************
// Section: Application Local Functions
// ******************************************************************
// ******************************************************************

/********************************************************************
  Function:
    void Set_LED_Status ( void )

   Description:
        Set LEDs to display the ADC average result.
*/

void Set_LED_Status(void)
{
    int i;

    appData.ledMask = 0;

    /* Creates a mask for the LEDs, corresponding to the value read
     * from the potentiometer */
    appData.potValue >>= 7; /* 10-bit value to 3-bit value */
    for (i = 0; i <= appData.potValue; i++)
    {
        appData.ledMask |=  1<<(i);
    }
   /* Write the mask to the LEDs */
    SYS_PORTS_Write( PORTS_ID_0, PORT_CHANNEL_A,
        (PORTS_DATA_MASK)appData.ledMask );
}

// ******************************************************************
// ******************************************************************
// Section: Application Initialization and State Machine Functions
// ******************************************************************
// ******************************************************************

/********************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}

/*********************************************************************
  Function:
    void APP_Tasks ( void )

  Remarks:
    See prototype in app.h.
*/

void APP_Tasks ( void )
{

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT: ← see Section 4.6
        {
            /* Enable ADC */
            DRV_ADC_Open();

            appData.state = APP_ADC_WAIT;

            break;
        }

        /* Display pot value on LEDs*/
        case APP_ADC_DISPLAY: ← see Section 4.7
        {
            Set_LED_Status();
            appData.state = APP_ADC_WAIT;

            break;
        }

        /* Wait for ADC */
        case APP_ADC_WAIT: ← see Section 4.8
        {
            /* Wait for conversion*/
            if (DRV_ADC_SamplesAvailable())
            {
                int i;

                //Read data
                for(i=0;i<ADC_NUM_SAMPLE_PER_AVERAGE;i++)
                    appData.potValue +=
                        PLIB_ADC_ResultGetByIndex(ADC_ID_1, i);
                appData.potValue = appData.potValue /
                    ADC_NUM_SAMPLE_PER_AVERAGE;

                appData.state = APP_ADC_DISPLAY;
            }

            break;
        }

        /* The default state should never be executed. */
        default:
```

```
        {
            /* TODO: Handle error in application's state machine. */
            break;
        }
    }
}

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

4.6 Application State - APP_STATE_INIT

When the tasks loop begins, the application is in its initial state. In this case, the ADC is enabled in the auto-sampling mode. Then the application state is changed to wait (APP_ADC_WAIT). Application states are defined in app.h.

4.7 Application State - APP_ADC_DISPLAY

Once an ADC value has been captured in APP_ADC_WAIT, the value is displayed by calling the function Set_LED_Status() in the Local Functions section. This function displays the ADC value from the potentiometer (appData.potValue) onto the LEDs using a mask (appData.ledMask). These variables are defined in app.h.

Once the function returns, the application state is changed back to APP_ADC_WAIT to wait for another sample.

4.8 Application State - APP_ADC_WAIT

After initialization (APP_STATE_INIT), the application waits for a pot value to be converted. Then the ADC value is assigned to the variable appData.potValue for display on the LEDs in the APP_ADC_DISPLAY case. ADC_NUM_SAMPLE_PER_AVERAGE is defined in app.h.

5. DISPLAY POTENTIOMETER VALUES ON LEDS USING AN ADC (MCC)

This example uses the same device and the Port A LEDs as example 3. However, in this example, values from a potentiometer on the demo board provide Analog-to-Digital Converter (ADC) input through Port B (RB2/AN2) 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:

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

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

FIGURE 12: ADC PROJECT RESOURCES - SYSTEM MODULE

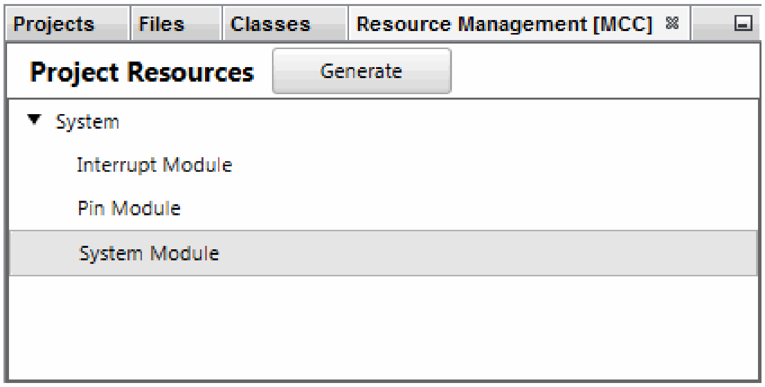
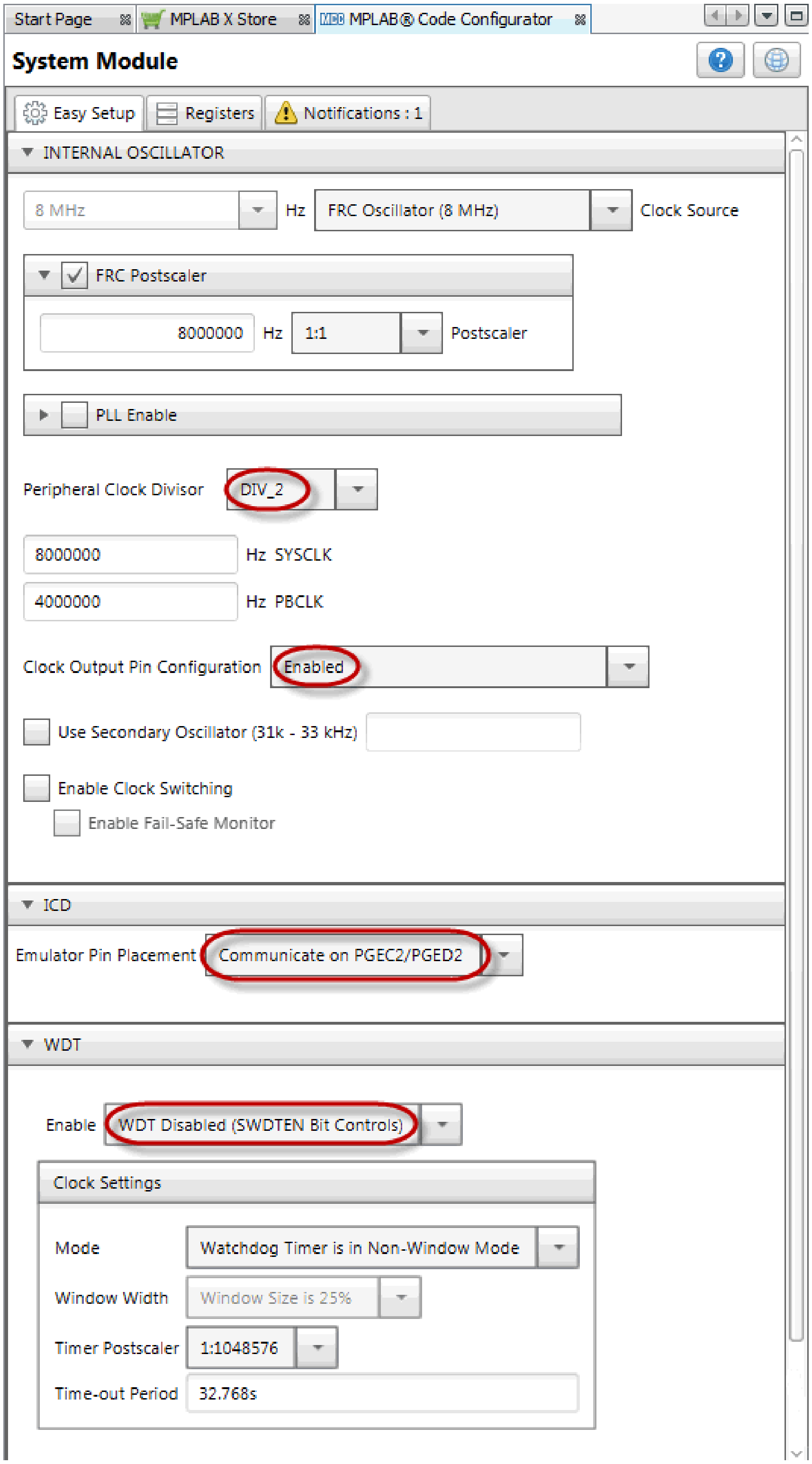


FIGURE 13: ADC PROJECT SYSTEM MODULE EASY SETUP



8 MHz Hz FRC Oscillator (8 MHz) Clock Source

☑ FRC Postscaler

8000000 Hz 1:1 Postscaler

□ PLL Enable

Peripheral Clock Divisor DIV_2

8000000 Hz SYSCLK

4000000 Hz PBCLK

Clock Output Pin Configuration Enabled

□ Use Secondary Oscillator (31k - 33 kHz)

□ Enable Clock Switching

□ Enable Fail-Safe Monitor

Emulator Pin Placement Communicate on PGEC2/PGED2

Enable WDT Disabled (SWDTEN Bit Controls)

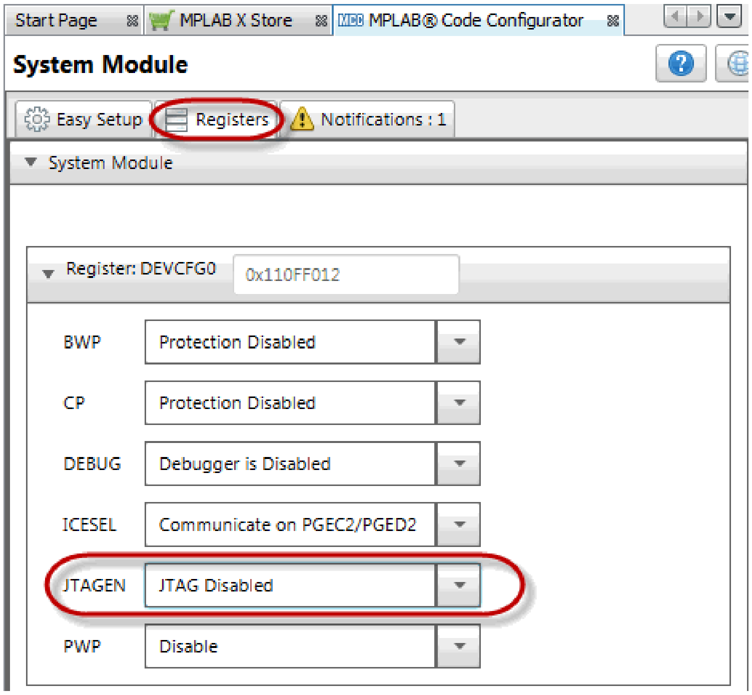
Mode Watchdog Timer is in Non-Window Mode

Window Width Window Size is 25%

Timer Postscaler 1:1048576

Time-out Period 32.768s

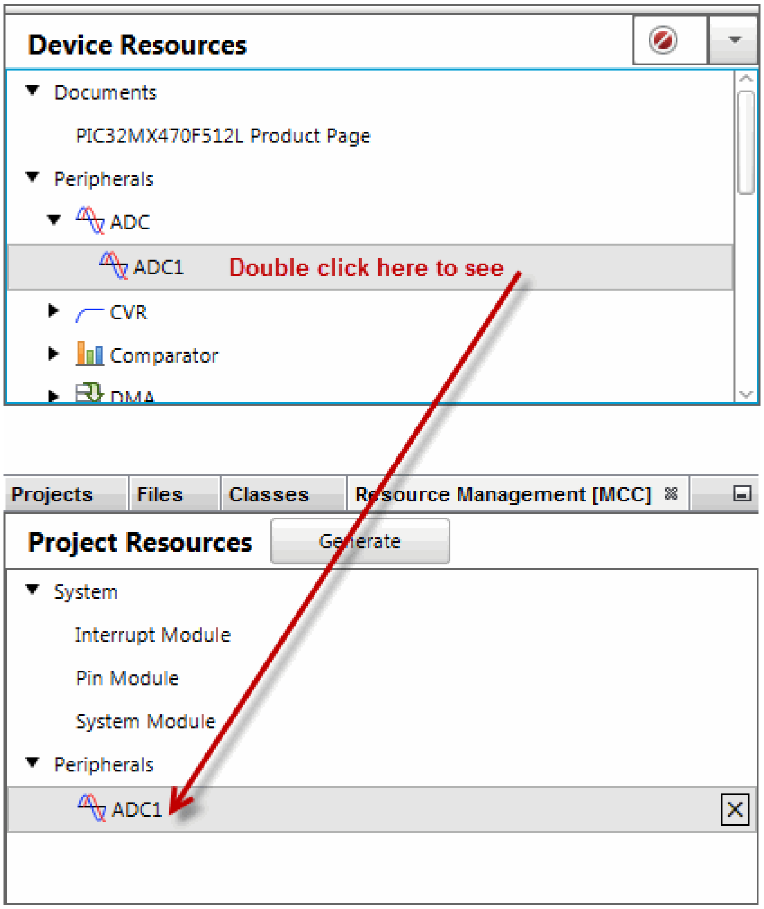
FIGURE 14: ADC PROJECT SYSTEM MODULE REGISTERS



Register: DEVCFG0 0x110FF012

BWP	Protection Disabled
CP	Protection Disabled
DEBUG	Debugger is Disabled
ICESEL	Communicate on PGEC2/PGED2
JTAGEN	JTAG Disabled
PWP	Disable

FIGURE 15: ADC PROJECT RESOURCES - ADC1



Device Resources

- Documents
  - PIC32MX470F512L Product Page
- Peripherals
  - ADC
    - ADC1
  - CVR
  - Comparator
  - DMA

Double click here to see

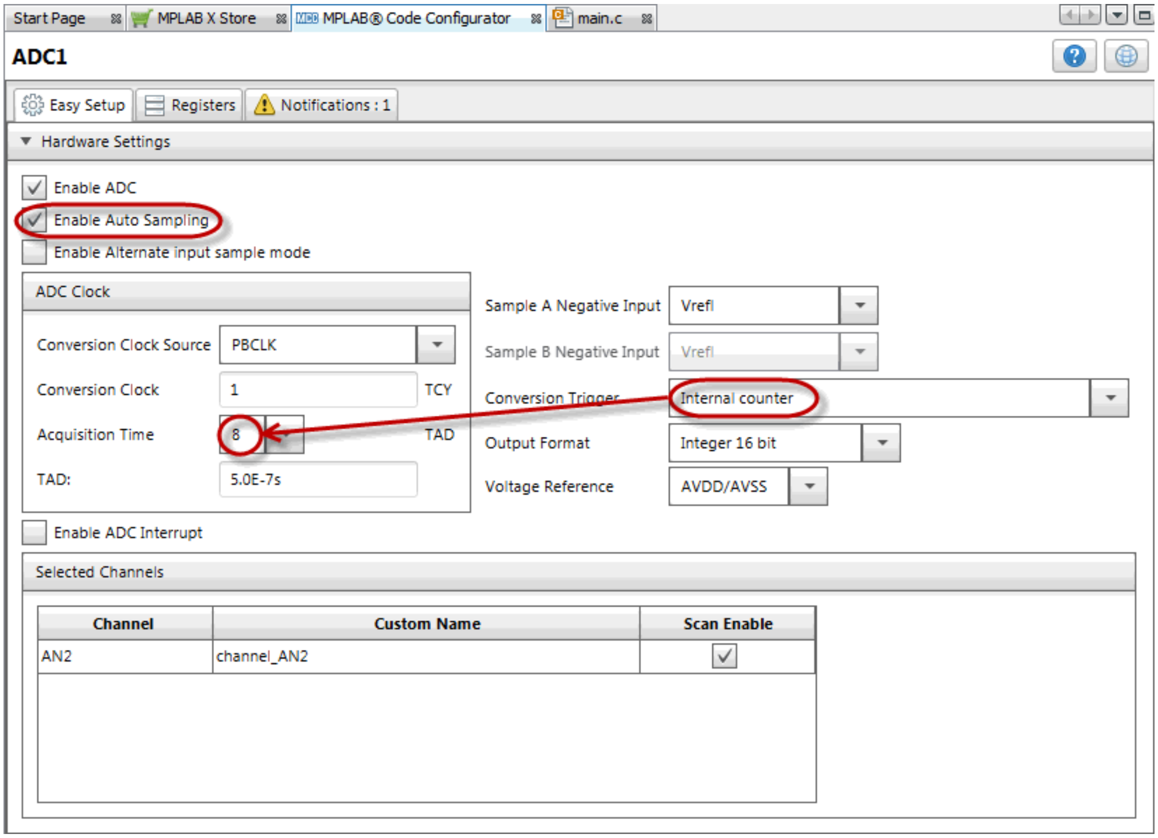
Projects | Files | Classes | Resource Management [MCC]

Project Resources

Generate

- System
  - Interrupt Module
  - Pin Module
  - System Module
- Peripherals
  - ADC1

FIGURE 16: ADC PROJECT ADC1 EASY SETUP



ADC1

▼ Hardware Settings

☑ Enable ADC

☑ Enable Auto Sampling

□ Enable Alternate input sample mode

ADC Clock

Conversion Clock Source: PBCLK

Conversion Clock: 1 TCY

Acquisition Time: 8 TAD

TAD: 5.0E-7s

Sample A Negative Input: Vrefl

Sample B Negative Input: Vrefl

Conversion Trigger: Internal counter

Output Format: Integer 16 bit

Voltage Reference: AVDD/AVSS

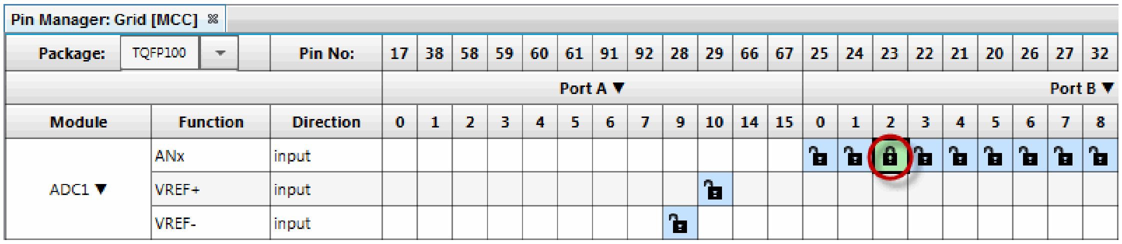
□ Enable ADC Interrupt

Selected Channels

Channel	Custom Name	Scan Enable
AN2	channel_AN2	☑

RB2 to AN2 map displays after selection is made in Figure 17.

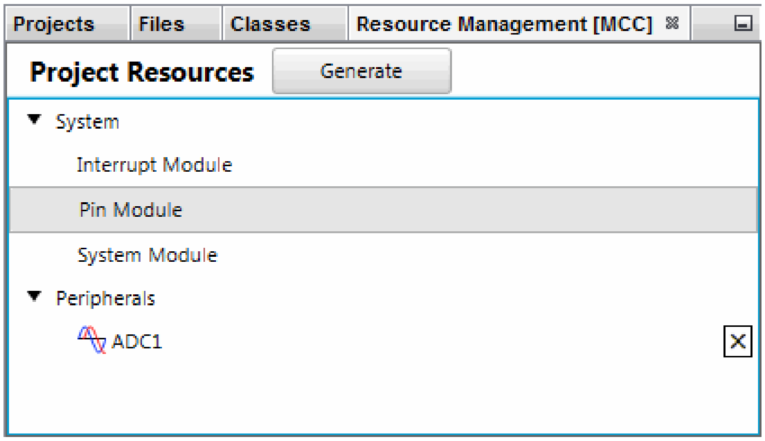
FIGURE 17: ADC PROJECT ADC1 PIN RESOURCE



Pin Manager: Grid [MCC]

Package:	TQFP100	Pin No:	17	38	58	59	60	61	91	92	28	29	66	67	25	24	23	22	21	20	26	27	32
			Port A ▼												Port B ▼								
Module	Function	Direction	0	1	2	3	4	5	6	7	9	10	14	15	0	1	2	3	4	5	6	7	8
ADC1 ▼	ANx	input													🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓
	VREF+	input										🔓											
	VREF-	input									🔓												

FIGURE 18: ADC PROJECT RESOURCES - PIN MODULE

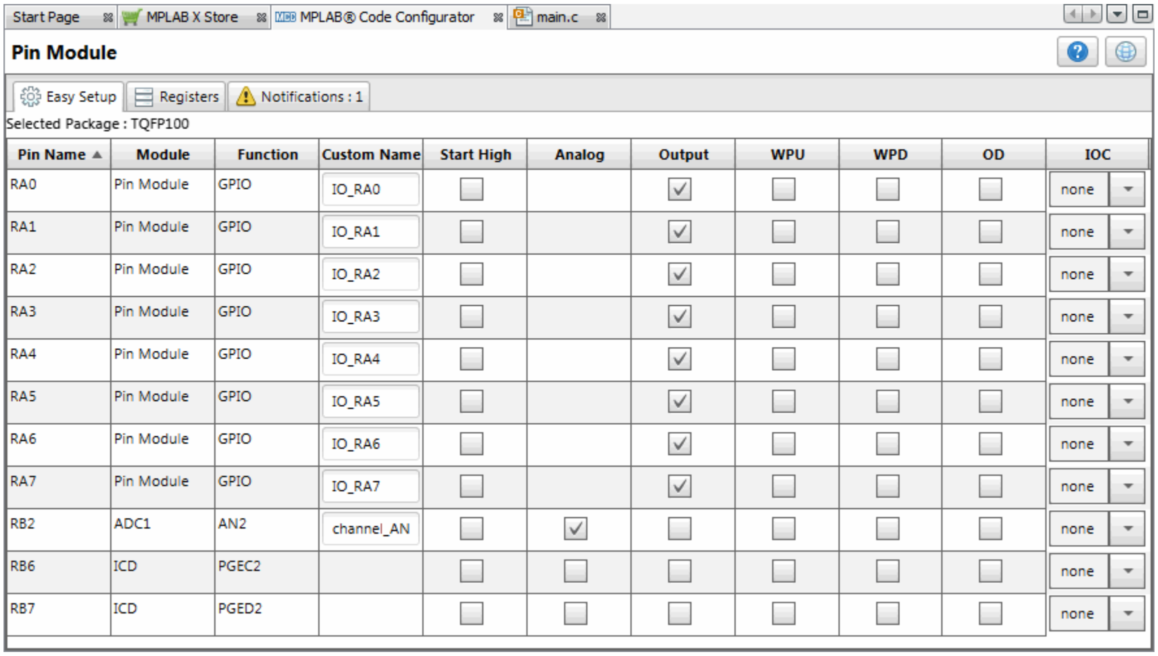


Projects | Files | Classes | Resource Management [MCC]

Project Resources
Generate

- System
  - Interrupt Module
  - Pin Module
  - System Module
- Peripherals
  - ADC1 ☒

FIGURE 19: ADC PROJECT PIN MODULE EASY SETUP



Pin Module

Selected Package : TQFP100

Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	WPD	OD	IOC
RA0	Pin Module	GPIO	IO_RA0	□		☑	□	□	□	none
RA1	Pin Module	GPIO	IO_RA1	□		☑	□	□	□	none
RA2	Pin Module	GPIO	IO_RA2	□		☑	□	□	□	none
RA3	Pin Module	GPIO	IO_RA3	□		☑	□	□	□	none
RA4	Pin Module	GPIO	IO_RA4	□		☑	□	□	□	none
RA5	Pin Module	GPIO	IO_RA5	□		☑	□	□	□	none
RA6	Pin Module	GPIO	IO_RA6	□		☑	□	□	□	none
RA7	Pin Module	GPIO	IO_RA7	□		☑	□	□	□	none
RB2	ADC1	AN2	channel_AN	□	☑	□	□	□	□	none
RB6	ICD	PGEC2		□	□	□	□	□	□	none
RB7	ICD	PGED2		□	□	□	□	□	□	none

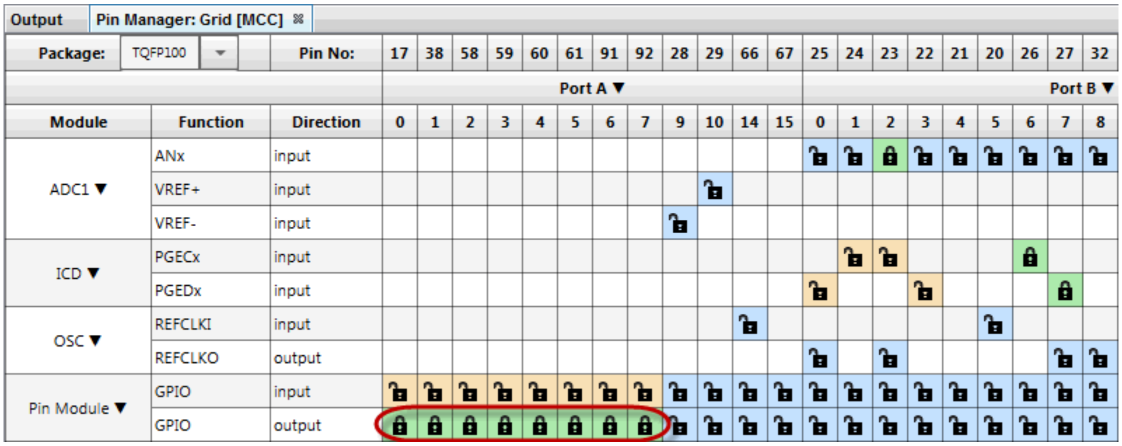
Pins RA0:7 will appear in the window above when they are selected in Figure 20.

RB2 was previously selected in Figure 17.

RB6 and RB7 are selected per PGEC2/PGED2 in Figure 13.

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

FIGURE 20: ADC PROJECT PIN RESOURCES



Output | Pin Manager: Grid [MCC]

Package:	TQFP100	Pin No:	17	38	58	59	60	61	91	92	28	29	66	67	25	24	23	22	21	20	26	27	32
			Port A ▼												Port B ▼								
Module	**Function**	**Direction**	0	1	2	3	4	5	6	7	9	10	14	15	0	1	2	3	4	5	6	7	8
ADC1 ▼	ANx	input													🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓
	VREF+	input										🔓											
	VREF-	input									🔓												
ICD ▼	PGECx	input														🔓	🔓				🔒		
	PGEDx	input													🔓			🔓				🔒	
OSC ▼	REFCLKI	input											🔓							🔓			
	REFCLKO	output													🔓		🔓					🔓	🔓
Pin Module ▼	GPIO	input	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓

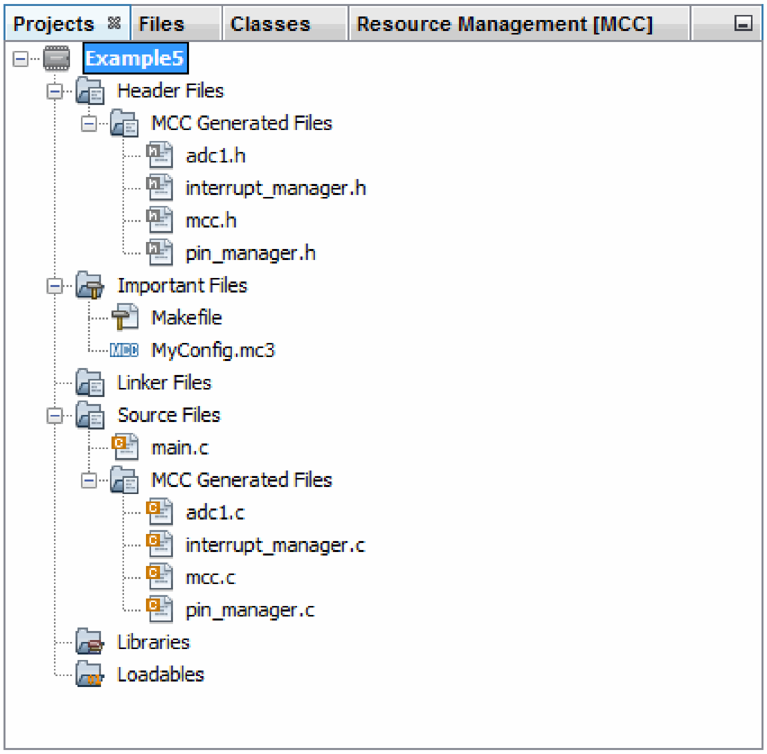
When the code is configured as shown in the previous figures, click the **Generate** button in the "Project Resources" window (Figure 12). Code generated by the MCC is modular. Therefore main, system and peripheral code are all in individual files. Also, each peripheral has its own header file.

Interrupt Manager files are generated to catch potential errors. Although no interrupts will be used in this application, these 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.

For more information on using Flash memory, see the *PIC32 Family Reference Manual*, "Section 17. 10-Bit A/D Converter" (DS61104).

FIGURE 21: ADC PROJECT TREE FOR CODE GENERATED BY MCC



- Example5
  - Header Files
    - MCC Generated Files
      - adc1.h
      - interrupt_manager.h
      - mcc.h
      - pin_manager.h
  - Important Files
    - Makefile
    - MyConfig.mc3
  - Linker Files
  - Source Files
    - main.c
    - MCC Generated Files
      - adc1.c
      - interrupt_manager.c
      - mcc.c
      - pin_manager.c
  - Libraries
  - Loadables

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 that has been added to `main()` is 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"

unsigned int value = 0;

/* Creates a mask for the LEDs, corresponding
 * to the value read from the potentiometer */
unsigned int Mask_Value(unsigned int pot_value){
    int i;
    unsigned int mask_value = 0;

    pot_value >>= 7; /* 10-bit value to 3-bit value */
    for (i = 0; i <= pot_value; i++)
    {
        mask_value |=  1<<(i);
    }

    return mask_value;
}

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

    while (1) {

        // Wait for conversion  ◄──── see Section 5.2
        // and then get result
        while(!ADC1_IsConversionComplete());
        value = ADC1_ConversionResultGet();

        // Mark value              ◄──── see Section 5.3
        value = Mask_Value(value);

        // Write to Port Latch/LEDs  ◄──── see Section 5.4
        LATA = value;

    }
    return -1;
}
```

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

5.2 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 *PIC32 Family Reference Manual,* "Section 17. 10-bit Analog-to-Digital Converter (ADC)" (DS61104).

5.3 ADC Conversion Result Mask

Since only 8 LEDs are available, and the ADC conversion result is 10-bit, the conversion result in the variable `value` is masked via the function `Mask_Value()` for displaying values in three-bit groups on the LEDs.

5.4 Write to Port Latch and LEDs

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

6. DISPLAY FLASH MEMORY VALUES ON LEDS (MPLAB HARMONY)

This example uses the same device and the Port A LEDs as example 4. However, in this example, values are written to and read from Flash (Non-Volatile) memory and the success (LED2) or failure (LED0) of these operations is displayed.

Instead of generating code by hand, MPLAB Harmony is used. For information on how to download and install the MPLAB Harmony Integrated Software Framework and MPLAB Harmony Configurator (MHC) MPLAB X IDE plug-in, see **Section 4. "Display Potentiometer Values on LEDs Using an ADC (MPLAB Harmony)"**.

For information on creating an MPLAB Harmony project in MPLAB X IDE, see **Section 4.1 "Create an MPLAB Harmony Project in MPLAB X IDE"**. For this example, name the project "Example6".

This example is based on the Flash driver application found under (e.g., for Windows OS):

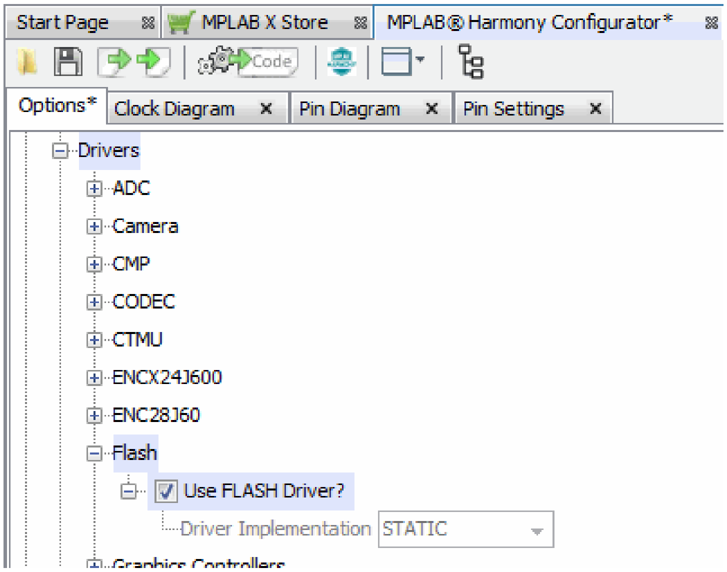
```
C:\microchip\harmony\v1_10\apps\examples\peripheral\flash\flash_modify
```

6.1 Configure the MPLAB Harmony Project

Based on your project setup, the MPLAB Harmony Configurator (MHC) will open with some clock information already populated. Blue highlight signifies changes. For this example, do not make any changes to the Clock settings.

Configure the Flash driver as shown in Figure 22 and Board Support Packages (BSP) as shown in Figure 23.

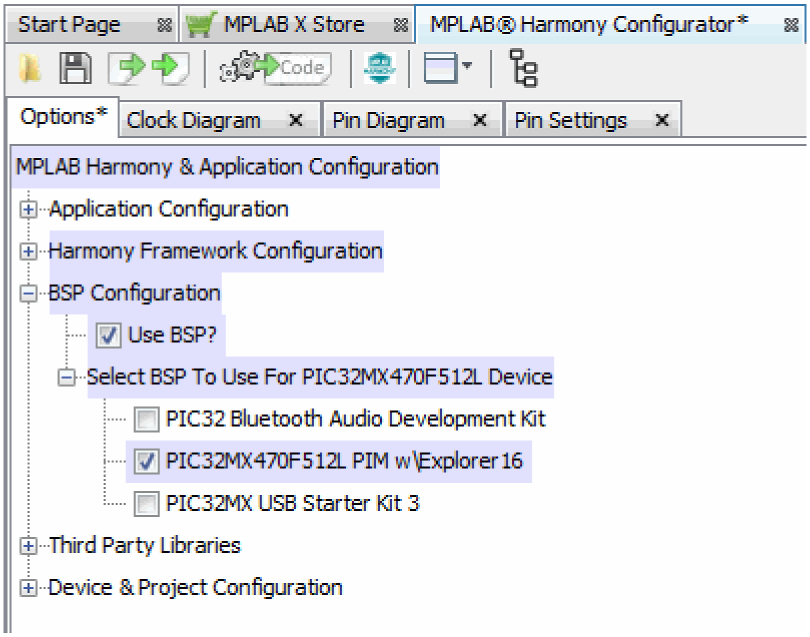
FIGURE 22: HARMONY FRAMEWORK CONFIGURATION - FLASH DRIVER



☑ Use FLASH Driver?

Driver Implementation STATIC

FIGURE 23: FLASH PROJECT RESOURCE CONFIGURATION

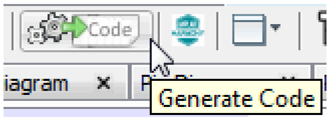


MPLAB Harmony & Application Configuration

- Application Configuration
- Harmony Framework Configuration
- BSP Configuration
  - ☑ Use BSP?
  - Select BSP To Use For PIC32MX470F512L Device
    - □ PIC32 Bluetooth Audio Development Kit
    - ☑ PIC32MX470F512L PIM w\Explorer16
    - □ PIC32MX USB Starter Kit 3
- Third Party Libraries
- Device & Project Configuration

6.2 Generate Code and Edit Application Files

When MHC is setup as shown in the previous figures, click the **Generate Code** button on the **MPLAB Harmony Configurator** tab.

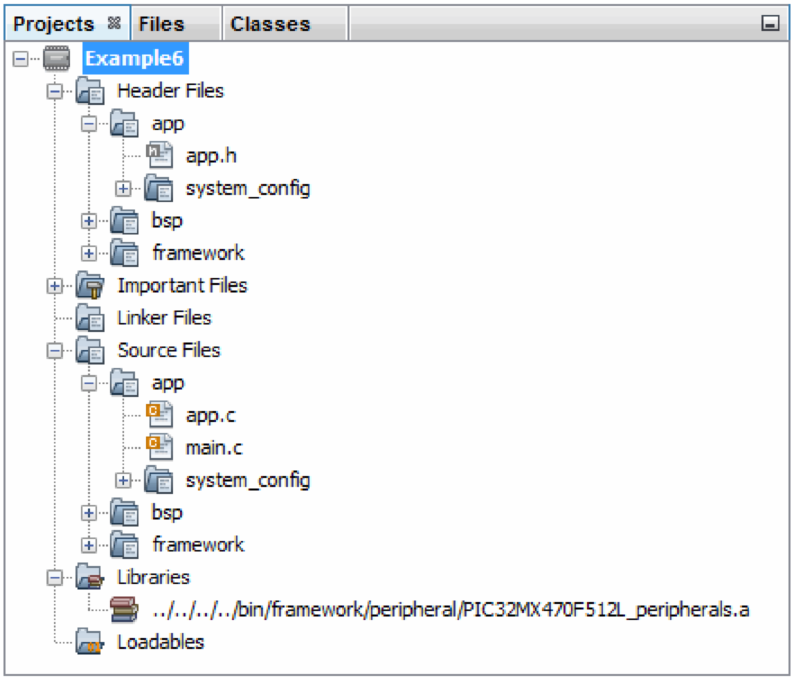


Save the configuration and generate the project code as per **Section 4.3 “Generate Code and Edit Application Files”**.

Code generated by the MPLAB Harmony is modular, as shown in Figure 24. The application files (`app.h` and `app.c`) are the ones edited for this example.

For more information on using Flash memory, see the *PIC32 Family Reference Manual*, “Section 5. Flash Programming” (DS60001121).

FIGURE 24: FLASH PROJECT TREE FOR CODE GENERATED BY MHC



```
Projects   Files   Classes
Example6
  Header Files
    app
      app.h
      system_config
    bsp
    framework
  Important Files
  Linker Files
  Source Files
    app
      app.c
      main.c
      system_config
    bsp
    framework
  Libraries
    ../../../../bin/framework/peripheral/PIC32MX470F512L_peripherals.a
  Loadables
```

6.3 app.h Modified Code

The app.h template file has been edited as shown below. Some comments have been removed, as described in < >. Code that has been added is red.

```
/********************************************************************
  MPLAB Harmony Application Header File

<See generated app.h file for file information.>

********************************************************************/

//DOM-IGNORE-BEGIN
/********************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.h file for copyright information.>

 *******************************************************************/
//DOM-IGNORE-END

#ifndef _APP_H
#define _APP_H

#define USERLED_SUCCESS             LED_2 //D5 on Explorer 16/32
#define USERLED_ERROR               LED_0 //D3 on Explorer 16/32

// ******************************************************************
// ******************************************************************
// Section: Included Files
// ******************************************************************
// ******************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

#endif
// DOM-IGNORE-END

// ******************************************************************
// ******************************************************************
// Section: Type Definitions
// ******************************************************************
// ******************************************************************

#define APP_DATABUFF_SIZE         (sizeof(databuff) /
sizeof(uint32_t))

/* Row size for device is 2Kbytes */
#define APP_DEVICE_ROW_SIZE_DIVIDED_BY_4
(DRV_FLASH_ROW_SIZE/4)
```

```
/* Page size for device is 16Kbytes */
#define APP_DEVICE_PAGE_SIZE_DIVIDED_BY_4
(DRV_FLASH_PAGE_SIZE/4)

#define APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE  (unsigned int)
0x9D008000
#define APP_PROGRAM_FLASH_BASE_ADDRESS  (unsigned int *)
APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE

// *****************************************************************
/* Application states

  Summary:
    Application states enumeration

  Description:
    This enumeration defines the valid application states.  These
states
    determine the behavior of the application at various times.
*/

typedef enum
{
   /* Application's state machine's initial state. */
   APP_STATE_INIT=0,
   APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE,
   APP_STATE_NVM_ERASE_COMPLETION_CHECK,
   APP_STATE_NVM_WRITE_START,
   APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK,
   APP_STATE_NVM_ERROR_STATE,
   APP_STATE_NVM_SUCCESS_STATE,

} APP_STATES;


// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    Application strings and buffers are be defined outside this
structure.
 */

typedef struct
{
    /* The application's current state */
    APP_STATES state;
    DRV_HANDLE flashHandle;

} APP_DATA;


// *****************************************************************
// *****************************************************************
// Section: Application Callback Routines
// *******************************************************************
// *******************************************************************
/* These routines are called by drivers when certain events occur.
*/

// *******************************************************************
// *******************************************************************
// Section: Application Initialization and State Machine Functions
// *******************************************************************
// *******************************************************************

/*********************************************************************
  Function:
    void APP_Initialize ( void )

  Summary:
     MPLAB Harmony application initialization routine.

<See generated app.h file for app init information.>

*/

void APP_Initialize ( void );


/*********************************************************************
  Function:
    void APP_Tasks ( void )

  Summary:
    MPLAB Harmony Demo application tasks function

<See generated app.h file for app tasks information.>

 */

void APP_Tasks( void );


#endif /* _APP_H */

//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END

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

6.4 app.c Modified Code

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

Some lines are long and wrap on the page. They have been left this way to enable cut-and-paste from this document to an editor.

```
/*******************************************************************
  MPLAB Harmony Application Source File

<See generated app.c file for file information.>

 *******************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.c file for copyright information.>

 *******************************************************************/
// DOM-IGNORE-END


// *****************************************************************
// *****************************************************************
// Section: Included Files
// *****************************************************************
// *****************************************************************

#include "app.h"

// *****************************************************************
// *****************************************************************
// Section: Global Data Definitions
// *****************************************************************
// *****************************************************************

/*****************************************************
 * Initialize the application data structure. All
 * application related variables are stored in this
 * data structure.
 *****************************************************/

/* Array in the KSEG1 RAM to store the data */
uint32_t databuff[APP_DEVICE_ROW_SIZE_DIVIDED_BY_4]
__attribute__((coherent, aligned(16)));

// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    This structure should be initialized by the APP_Initialize
function.

    Application strings and buffers are be defined outside this
structure.
*/

APP_DATA appData;

// ******************************************************************
// ******************************************************************
// Section: Application Callback Functions
// ******************************************************************
// ******************************************************************

/* TODO:  Add any necessary callback functions.
*/

// ******************************************************************
// ******************************************************************
// Section: Application Local Functions
// ******************************************************************
// ******************************************************************


/* TODO:  Add any necessary local functions.
*/


// ******************************************************************
// ******************************************************************
// Section: Application Initialization and State Machine Functions
// ******************************************************************
// ******************************************************************

/********************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}


/********************************************************************
  Function:
    void APP_Tasks ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Tasks ( void )
{
    unsigned int x;
    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */      ← see Section 6.5
        case APP_STATE_INIT:
          appData.flashHandle = DRV_FLASH_Open(DRV_FLASH_INDEX_0,
              intent);
          appData.state = APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE;
          break;

        /* Fill data buffer, clear LEDs,      ← see Section 6.6
         * and begin erase page */
        case APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE:
          for (x = 0; x < APP_DATABUFF_SIZE; x++)
          {
              databuff[x] = x;
          }
          BSP_LEDOff(USERLED_SUCCESS);
          BSP_LEDOff(USERLED_ERROR);

          /* Erase the page which consist of the row to be written */
          DRV_FLASH_ErasePage(appData.flashHandle,
              APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE);
          appData.state = APP_STATE_NVM_ERASE_COMPLETION_CHECK;
          break;

        /* Check for erase complete */      ← see Section 6.7
        case APP_STATE_NVM_ERASE_COMPLETION_CHECK:
          if(!DRV_FLASH_IsBusy(appData.flashHandle))
          {
              appData.state = APP_STATE_NVM_WRITE_START;
          }
          break;

        /* Write row of Flash */      ← see Section 6.8
        case APP_STATE_NVM_WRITE_START:
          /* Erase Success */
          /* Write a row of data to PROGRAM_FLASH_BASE_ADDRESS,
           * using databuff array as the source */
          DRV_FLASH_WriteRow(appData.flashHandle,
              APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE, databuff);
          appData.state =
              APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK;
          break;

        /* Check for write complete      ← see Section 6.9
         * and verify write operation */
        case APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK:
          if(!DRV_FLASH_IsBusy(appData.flashHandle))
          {
              /* Verify that data written to flash memory is valid
               * (databuff array read from kseg1) */
              if (!memcmp(databuff,
                 (void *)KVA0_TO_KVA1(APP_PROGRAM_FLASH_BASE_ADDRESS),
                 sizeof(databuff)))
              {
                  appData.state = APP_STATE_NVM_SUCCESS_STATE;
              }
              else
              {
                  appData.state = APP_STATE_NVM_ERROR_STATE;
              }
          }
          break;

        /* Write Failure */  ◄────  see Section 6.10
        case APP_STATE_NVM_ERROR_STATE:
          /*stay here, nvm had a failure*/
          BSP_LEDOn(USERLED_ERROR);
          BSP_LEDOff(USERLED_SUCCESS);
          break;

        /* Write Success */  ◄────  see Section 6.11
        case APP_STATE_NVM_SUCCESS_STATE:
          BSP_LEDOn(USERLED_SUCCESS);
          BSP_LEDOff(USERLED_ERROR);
          break;

    }
}


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

6.5 Application State - Initial State

When the tasks loop begins, the application is in its initial state (`APP_STATE_INIT`). In this case, the Flash driver is initialized and the state is changed to the next state (`APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE`). Application states are defined in `app.h`.

6.6 Application State - Fill Data Buffer & Erase Page

Once initialization is complete, actions in preparation for write are performed. First, a data buffer is filled with values that will be written to Flash memory (The data buffer is defined in "Section: Global Data Definitions"). Second, LEDs specifying success and failure are cleared (These values are set in `app.h`). Third, erase of a Flash memory (NVM) page is begun. Finally the application state is changed to wait for the page erase to complete (`APP_STATE_NVM_ERASE_COMPLETION_CHECK`).

6.7 Application State - Page Erase Complete

This state waits for the page erase begun in the previous state to complete. Once it does, the application state is changed to begin Flash memory (NVM) write (`APP_STATE_NVM_WRITE_START`).

6.8 Application State - Write Row of Flash Memory

A write of a row of the erased page in Flash memory is now begun. Values in the data buffer will be written to this row. The application state is then changed to wait for the write to finish and verify the result (`APP_STATE_NVM_WRITE_COMPLETION_-CHECK_AND_VERIFY_CHECK`).

6.9 Application State - Write Row Complete and Verify

This state waits for the row write begun in the previous state to complete. Once it does, the write is verified against the values in the data buffer. If the write is a success, the application state is changed to `APP_STATE_NVM_SUCCESS_STATE`. If the write failed, the application state is changed to `APP_STATE_NVM_ERROR_STATE`.

6.10 Application State - Error State

If the Flash memory has failed to be written to, an error state is entered. LED 3 (D3) on the demo board is lit to show that an error has occurred.

6.11 Application State - Success State

Once the Flash memory is successfully erased and written, a success state is entered. LED 5 (D5) on the demo board is lit to show that application execution was successful.

7. DISPLAY FLASH MEMORY VALUES ON LEDS (MCC)

This example uses the same device and the Port A LEDs as example 5. However, in this example, values are written to and read from Flash (Non-Volatile) memory and the success (LED2) or failure (LED0) of these operations is displayed.

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 5. "Display Potentiometer Values on LEDs Using an ADC (MCC)"**.

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

FIGURE 25: FLASH PROJECT RESOURCES - SYSTEM MODULE

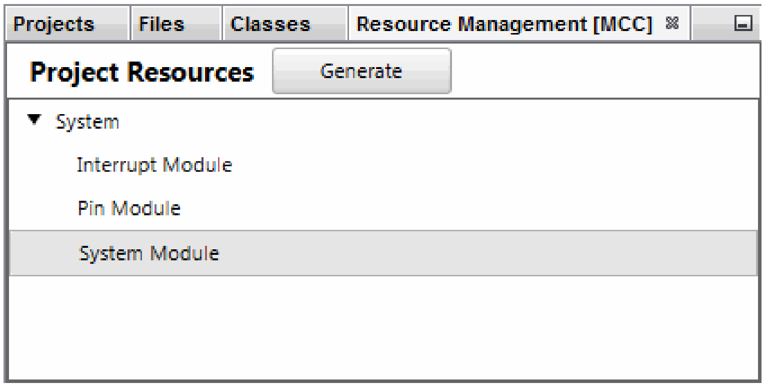
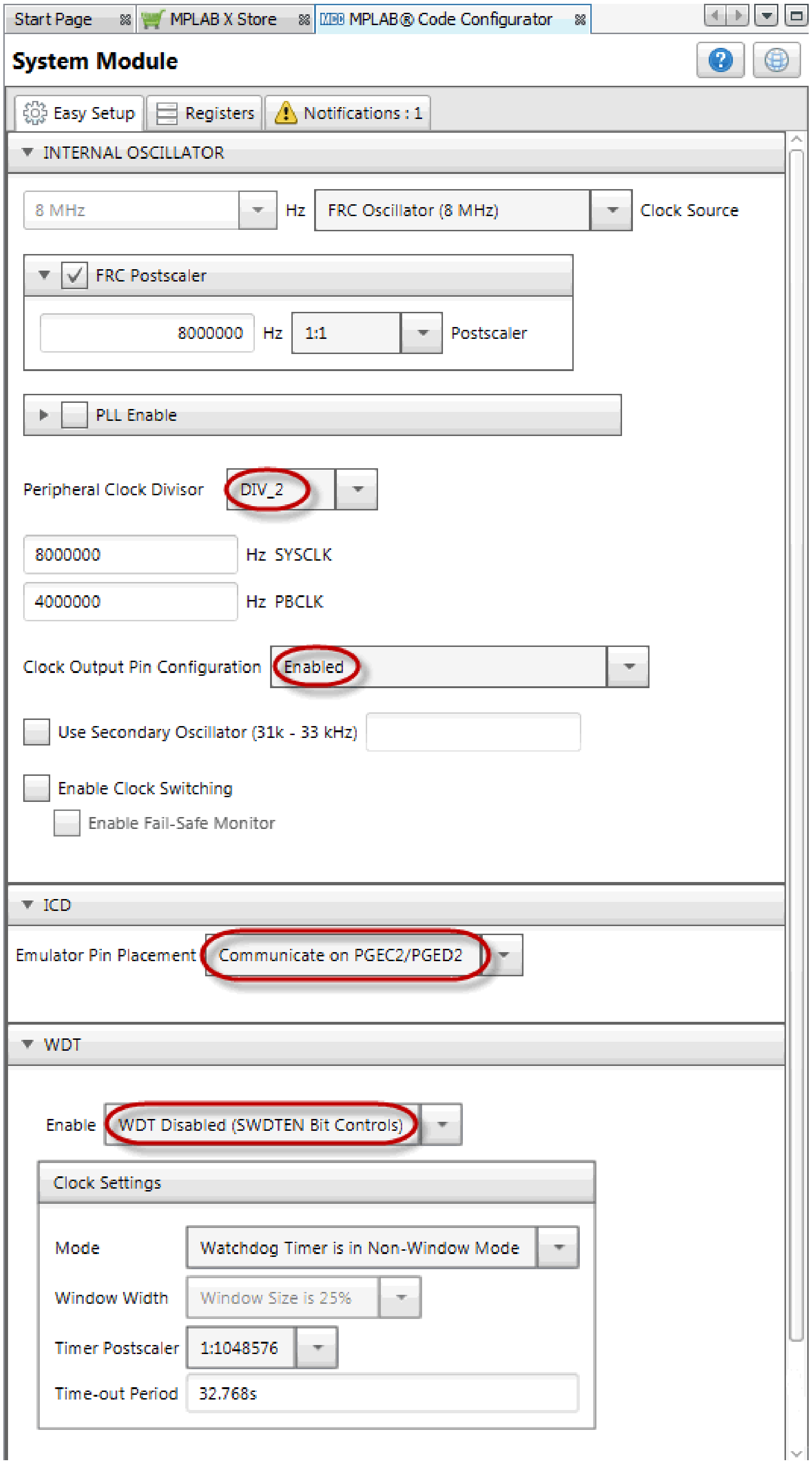


FIGURE 26: FLASH PROJECT SYSTEM MODULE EASY SETUP



System Module

INTERNAL OSCILLATOR

8 MHz Hz FRC Oscillator (8 MHz) Clock Source

☑ FRC Postscaler

8000000 Hz 1:1 Postscaler

□ PLL Enable

Peripheral Clock Divisor DIV_2

8000000 Hz SYSCLK

4000000 Hz PBCLK

Clock Output Pin Configuration Enabled

□ Use Secondary Oscillator (31k - 33 kHz)

□ Enable Clock Switching

□ Enable Fail-Safe Monitor

ICD

Emulator Pin Placement Communicate on PGEC2/PGED2

WDT

Enable WDT Disabled (SWDTEN Bit Controls)

Clock Settings

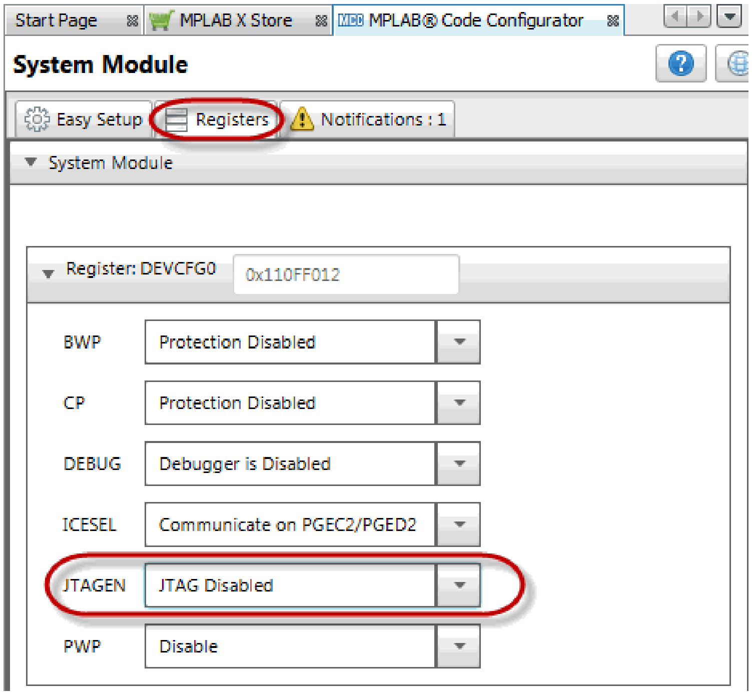
Mode Watchdog Timer is in Non-Window Mode

Window Width Window Size is 25%

Timer Postscaler 1:1048576

Time-out Period 32.768s

FIGURE 27: FLASH PROJECT SYSTEM MODULE REGISTERS



Register: DEVCFG0 0x110FF012

BWP Protection Disabled

CP Protection Disabled

DEBUG Debugger is Disabled

ICESEL Communicate on PGEC2/PGED2

JTAGEN JTAG Disabled

PWP Disable

FIGURE 28: FLASH PROJECT RESOURCES - NVM

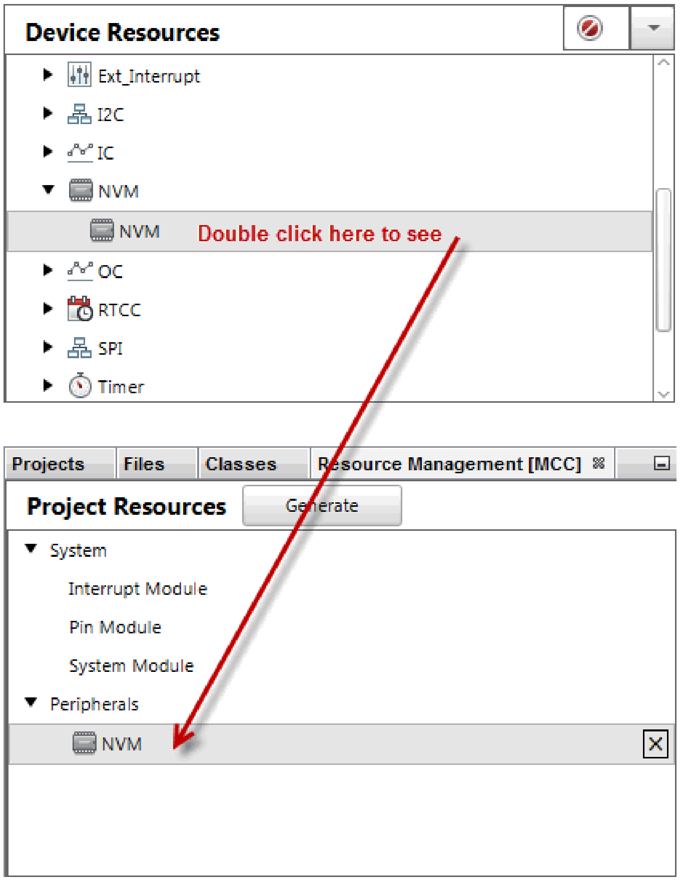
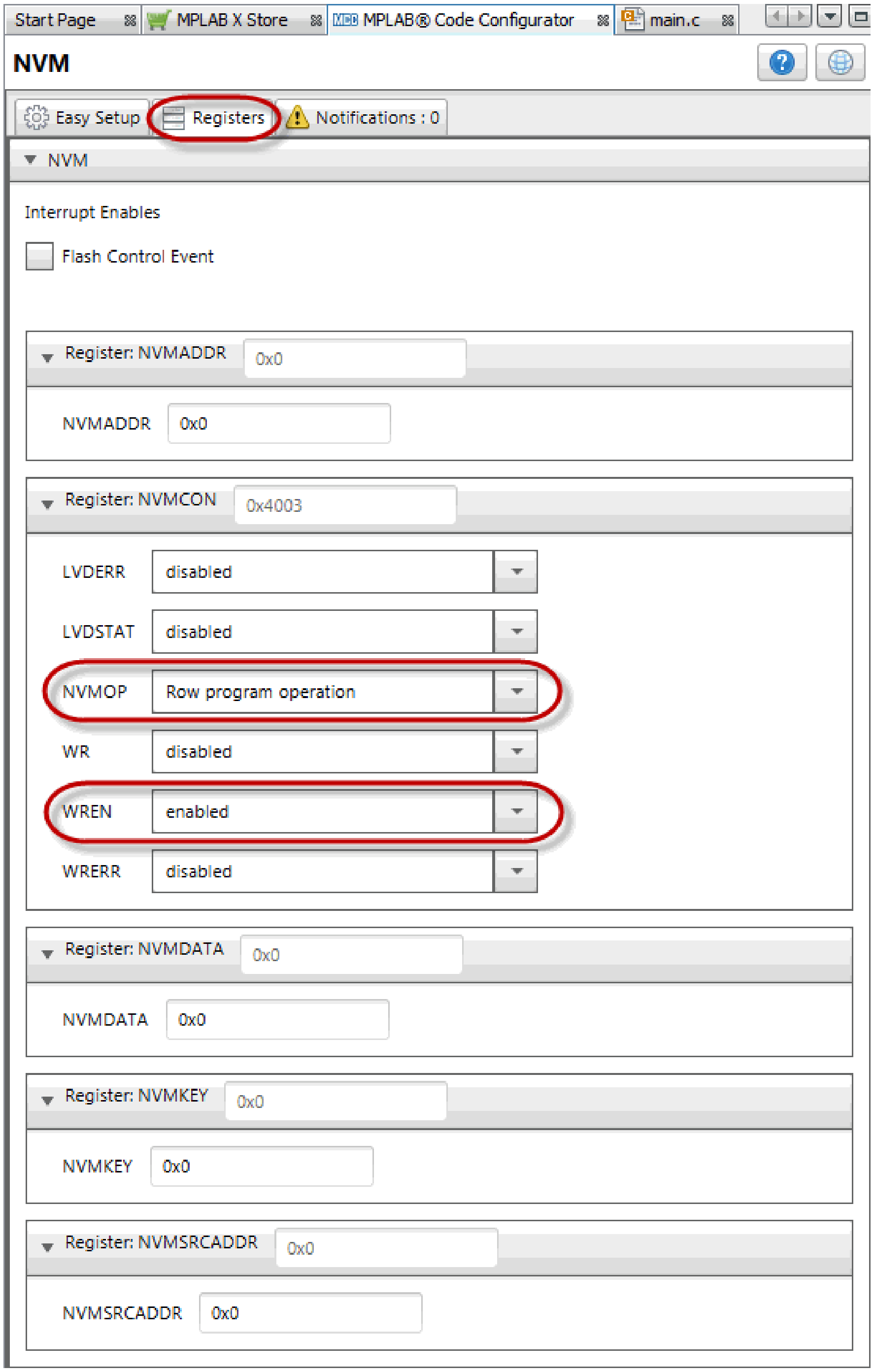


FIGURE 29: FLASH PROJECT NVM REGISTERS



Interrupt Enables

□ Flash Control Event

Register: NVMADDR 0x0

NVMADDR 0x0

Register: NVMCON 0x4003

LVDERR disabled

LVDSTAT disabled

NVMOP Row program operation

WR disabled

WREN enabled

WRERR disabled

Register: NVMDATA 0x0

NVMDATA 0x0

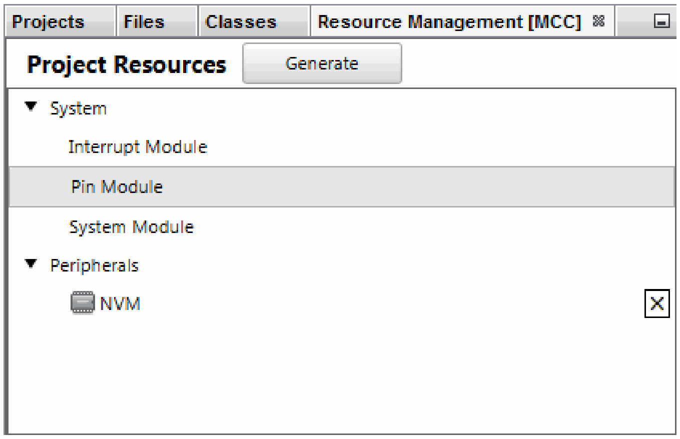
Register: NVMKEY 0x0

NVMKEY 0x0

Register: NVMSRCADDR 0x0

NVMSRCADDR 0x0

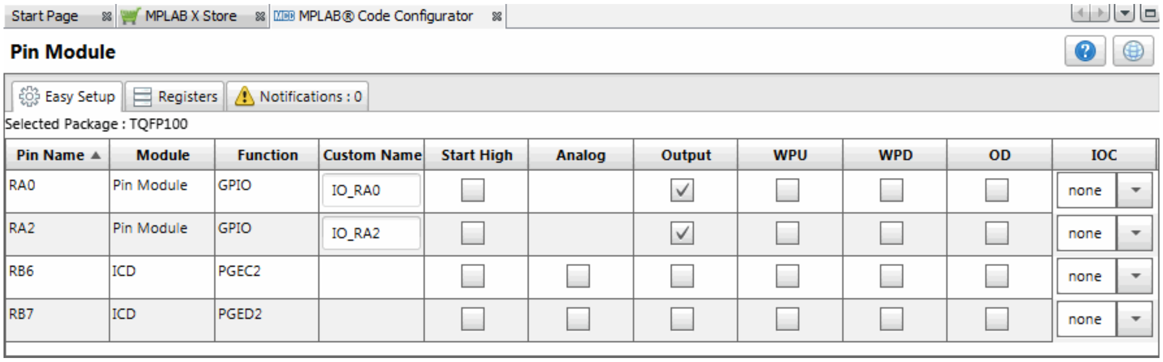
FIGURE 30: FLASH PROJECT RESOURCES - PIN MODULE



Project Resources

- System
  - Interrupt Module
  - Pin Module
  - System Module
- Peripherals
  - NVM ☒

FIGURE 31: FLASH PROJECT I/O PIN CONFIGURATION



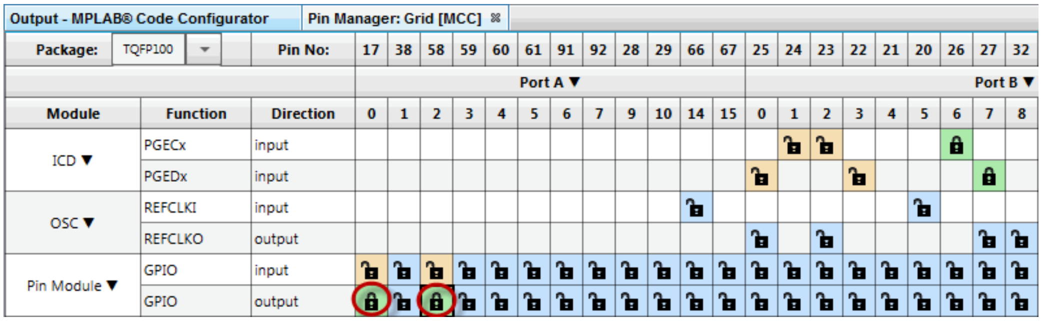
Pin Module

Selected Package : TQFP100

Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	WPD	OD	IOC
RA0	Pin Module	GPIO	IO_RA0	□		☑	□	□	□	none
RA2	Pin Module	GPIO	IO_RA2	□		☑	□	□	□	none
RB6	ICD	PGEC2		□	□	□	□	□	□	none
RB7	ICD	PGED2		□	□	□	□	□	□	none

Pins RA0 and RA2 will appear in the window above when they are selected in Figure 32.
RB6 and RB7 are selected per PGEC2/PGED2 in Figure 26.
Once visible in the window, pin configurations may be viewed or selected for each pin.

FIGURE 32: FLASH PROJECT I/O PIN RESOURCES



Package:	TQFP100	Pin No:	17	38	58	59	60	61	91	92	28	29	66	67	25	24	23	22	21	20	26	27	32
			Port A ▼												Port B ▼								
Module	Function	Direction	0	1	2	3	4	5	6	7	9	10	14	15	0	1	2	3	4	5	6	7	8
ICD ▼	PGECx	input														🔓	🔓				🔒		
	PGEDx	input													🔓			🔓				🔒	
OSC ▼	REFCLKI	input											🔓							🔓			
	REFCLKO	output													🔓		🔓					🔓	🔓
Pin Module ▼	GPIO	input	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
	GPIO	output	🔒	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓

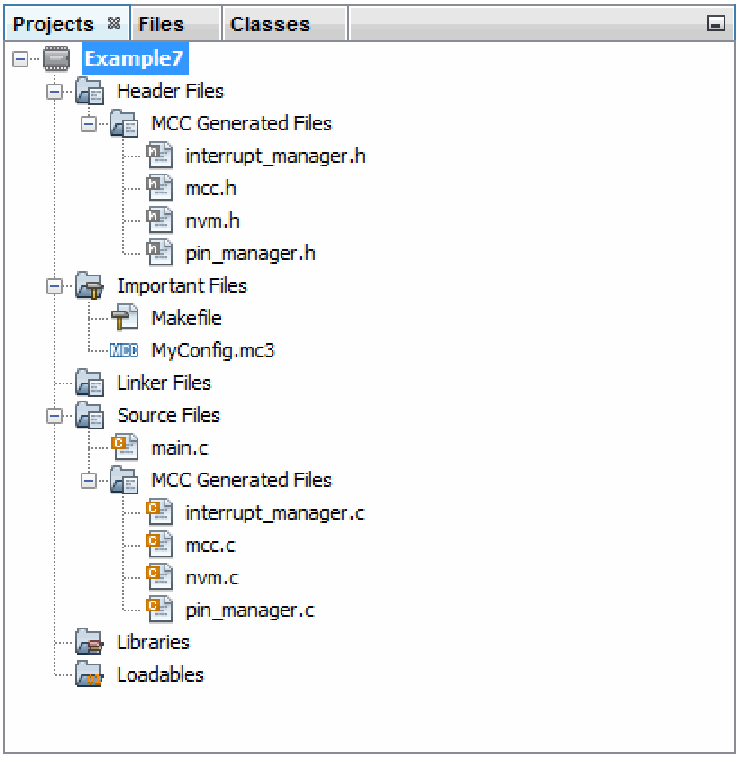
When the code is configured as shown in the previous figures, click the **Generate** button in the "Project Resources" window (Figure 12). Code generated by the MCC is modular. Therefore main, system and peripheral code are all in individual files. Also, each peripheral has its own header file.

Interrupt Manager files are generated to catch potential errors. Although no interrupts will be used in this application, these 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.

For more information on using Flash memory, see the *PIC32 Family Reference Manual*, "Section 5. Flash Programming" (DS60001121).

FIGURE 33: FLASH PROJECT TREE FOR CODE GENERATED BY MCC



- Example7
  - Header Files
    - MCC Generated Files
      - interrupt_manager.h
      - mcc.h
      - nvm.h
      - pin_manager.h
  - Important Files
    - Makefile
    - MyConfig.mc3
  - Linker Files
  - Source Files
    - main.c
    - MCC Generated Files
      - interrupt_manager.c
      - mcc.c
      - nvm.c
      - pin_manager.c
  - Libraries
  - Loadables

7.1 `main.c` Modified Code

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

```
/**
  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"

// Program Flash Physical Addresses:  0x1D00_0000 - 0x1D07_FFFF
// Program Flash Virtual Addresses:   KSEG0: 0x9D00_0000 - 0x9D07_FFFF
//                                    KSEG1: 0xBD00_0000 - 0xBD07_FFFF
#define NVM_PROGRAM_PAGE 0xbd008000

unsigned int databuff[128];

/*
                         Main application
 */
int main(void) {

    unsigned int x;

    // initialize the device
    SYSTEM_Initialize();

    // Fill databuff with some data
   for(x =0; x < sizeof(databuff); x++)
       databuff[x] = x;

   // Erase second page of Program Flash  ◄──── see Section 7.2
   NVM_ErasePage((void *)NVM_PROGRAM_PAGE);

   // Write 128 words starting at  ◄──── see Section 7.3
   // Row Address NVM_PROGRAM_PAGE
   NVM_WriteRow((void *)NVM_PROGRAM_PAGE, (void*)databuff);

   // Verify data matches  ◄──── see Section 7.4

   if(memcmp(databuff, (void *)NVM_PROGRAM_PAGE, sizeof(databuff)))
   {
       // If not turn led0 on to indicate an error
       IO_RA0_SetHigh();
   }

    else {
       // If true turn led2 on to indicate success
       IO_RA2_SetHigh();
    }
    while (1) {
       // End of program
    }

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

7.2 Erase Page of Flash

The smallest section of Flash memory that can be erased is a page.

Find the NVM_ErasePage() function in the `nvm.c` file.

7.3 Write Row of Flash

The contents of databuff will be written into a row of Flash memory.

Find the NVM_WriteRow() function in the `nvm.c` file.

7.4 Verify Write and Display Data on LEDs

The data written is compared to the contents of databuff. If the content does not match, LED0/D3 is lit to signify an error. If the content matches, LED2/D5 on the Explorer 16/32 board is lit to signify a success.

A. RUN CODE IN MPLAB X IDE

For examples 1, 2, and 3, create a project as follows:

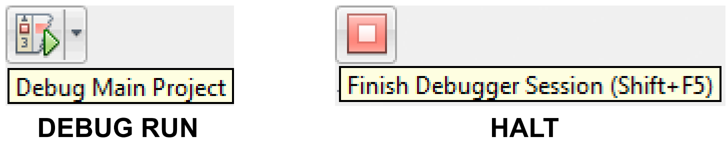
1. Launch MPLAB X IDE.
2. From the IDE, launch the New Project Wizard (*File>New Project*).
3. Follow the screens to create a new project:
   a) **Choose Project:** Select “Microchip Embedded”, and then select “Standalone Project”.
   b) **Select Device:** Select the example device.
   c) **Select Header:** 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 XC32 (*latest version number*) [`bin` *location*]. If you do not see a compiler under XC32, 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 XC32 and MPLAB X IDE documentation for details
   g) **Select Project Name and Folder:** Name the project.
4. Right click on the project name in the Projects window. Select *New>Empty File*. The New Empty File dialog will open.
5. Under “File name”, enter a name.
6. Click **Finish**.
7. Cut and paste the example code from this user’s guide into the empty editor window and select *File>Save*.

For examples 4 and 6, create a project as specified in **Section 4.1 “Create an MPLAB Harmony Project in MPLAB X IDE”**. Then, set up the MHC, generate code and edit the application as specified.

For examples 5 and 7, follow steps 1 through 3, above. Then set up the MCC, generate code and edit the application as specified.

Finally, select Debug Run to build, download to a device, and execute your code. View program output on the LEDs. Click Halt to end execution.

FIGURE 34: TOOLBAR ICONS



B. GET SOFTWARE AND HARDWARE

For the MPLAB XC32 projects in this document, the Explorer 16/32 board with a PIC32 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 XC32 C Compiler

MPLAB X IDE v3.55 and later can be found at:

http://www.microchip.com/mplabx

The MPLAB XC32 C Compiler v1.42 and later can be found at:

http://www.microchip.com/mplabxc

B.2 Get MPLAB Harmony and Configurator Plugin

MPLAB Harmony Configurator v1.0.10.xx and later can be found in MPLAB X IDE:

Tools>Plugins, **Available Plugins** tab.

MPLAB Harmony v1.10 and later can be found at:

http://www.microchip.com/mplab/mplab-harmony

B.3 Get the MPLAB Code Configurator (MCC)

The MCC v3.26 and later can be found at:

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

B.4 Get PIC® MCU Plug-in Module (PIM)

The PIC MCU PIM used in the examples is available on the Microchip Technology web site:

PIC32MX470F512L: http://www.microchip.com/MA320002-2

B.5 Get and Set Up the Explorer 16/32 Board

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

http://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.6 Get Microchip Debug Tools

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

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

B.7 Get Example Code

The code examples discussed in this document are available for download at:

http://www.microchip.com/mplabxc

under the **Documentation** tab. Place the MPLAB Harmony examples in this folder:

```
C:\microchip\harmony\v1_10\apps
```

NOTES:

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

ISBN: 978-1-5224-1540-4

MICROCHIP

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

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 - Guangzhou
Tel: 86-20-8755-8029

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: 86-21-3326-8000
Fax: 86-21-3326-8021

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: 886-7-213-7830

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

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

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

France - Saint Cloud
Tel: 33-1-30-60-70-00

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: 47-7289-7561

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

11/07/16