Anthony Chavez

EEE 174-CpE 185 Summer 2020

Monday, Wednesday

Lab 2

MicroChip PIC24F Development Lab

Dennis Dahlquist

Introduction:

For this lab, we will explore the capabilities of the PIC24F Curiosity Development Board and become familiar with the MPLAB X IDE. The required hardware for this lab includes the PIC24F Curiosity Development Board Part Number: DM240004 and a USB Type-A to USB Micro with power and data capability. As for the required software, we will be using the MPLAB X IDE, MPLAB XC 16, and MPLAB Code Configurator. For each part of this lab, we will follow the tutorials listed in the lab instructions and demo each one to our lab instructor.

Part 1: Digital Output: 16-bit Digital Output Example

This project will show how to program a digital output pin on a 16-bit PIC24F MCU. One of the basic functions on an MCU is to alter the state of a digital pin, simply turning on or off anything like a LED. The goal of this project is to generate code to turn on the two LEDs (LED1 and LED2) on the PIC24F Curiosity Development Board. LED1 is connected to pin RA9 and LED2 is connected to pin RA10.

First, we generated the code using the MPLAB Code Configurator (MCC) which configures the RA9 and RA10 pins as digital output pins (see Figure 1.1). RA9 will be configured to start in a high position (1) and RA10 will be initialized in a low position (0). We created a new standalone project in MPLAB X for a PIC24FJ128GA204 board (see Figures 1.2 – 1.5). Next, we opened the MCC Plugin and configured the System Resources (see Figures 1.7 and 1.8). Then we uploaded the program to the board and witnessed LED1 being ON and LED2 being OFF (see Figure 1.9) which was the expected output stated in the lab instructions. Finally, we modified the program to turn on LED2 and turn off LED1 (see Figure 1.11) by adding the code shown in Figure 1.10 to the main.c file. Please note: #include “mcc_generated_files/mcc.h” is required to be placed in any application source file which accesses MCC-generated functions.

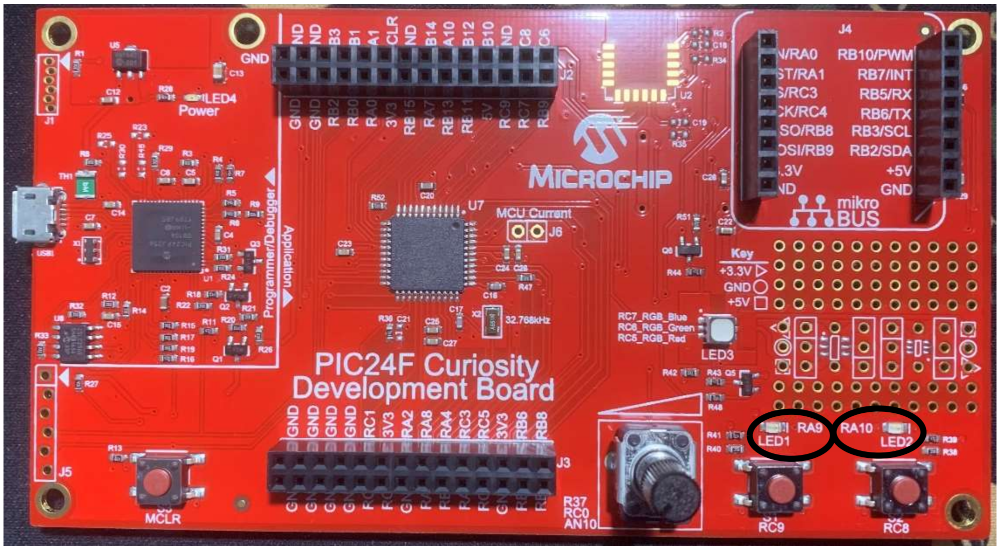


Figure 1.1: PIC24F Curiosity Development Board, Pins of Interest

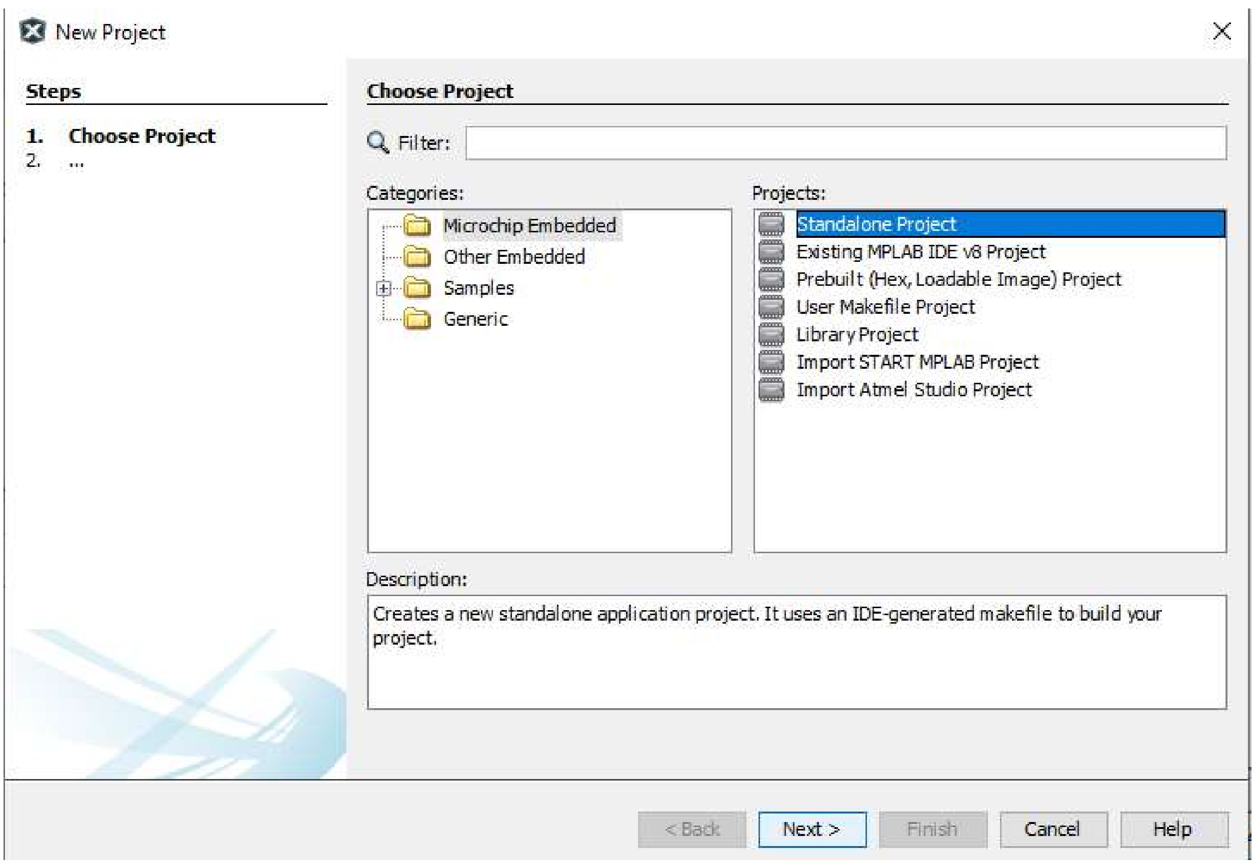


Figure 1.2: Creating a New Project

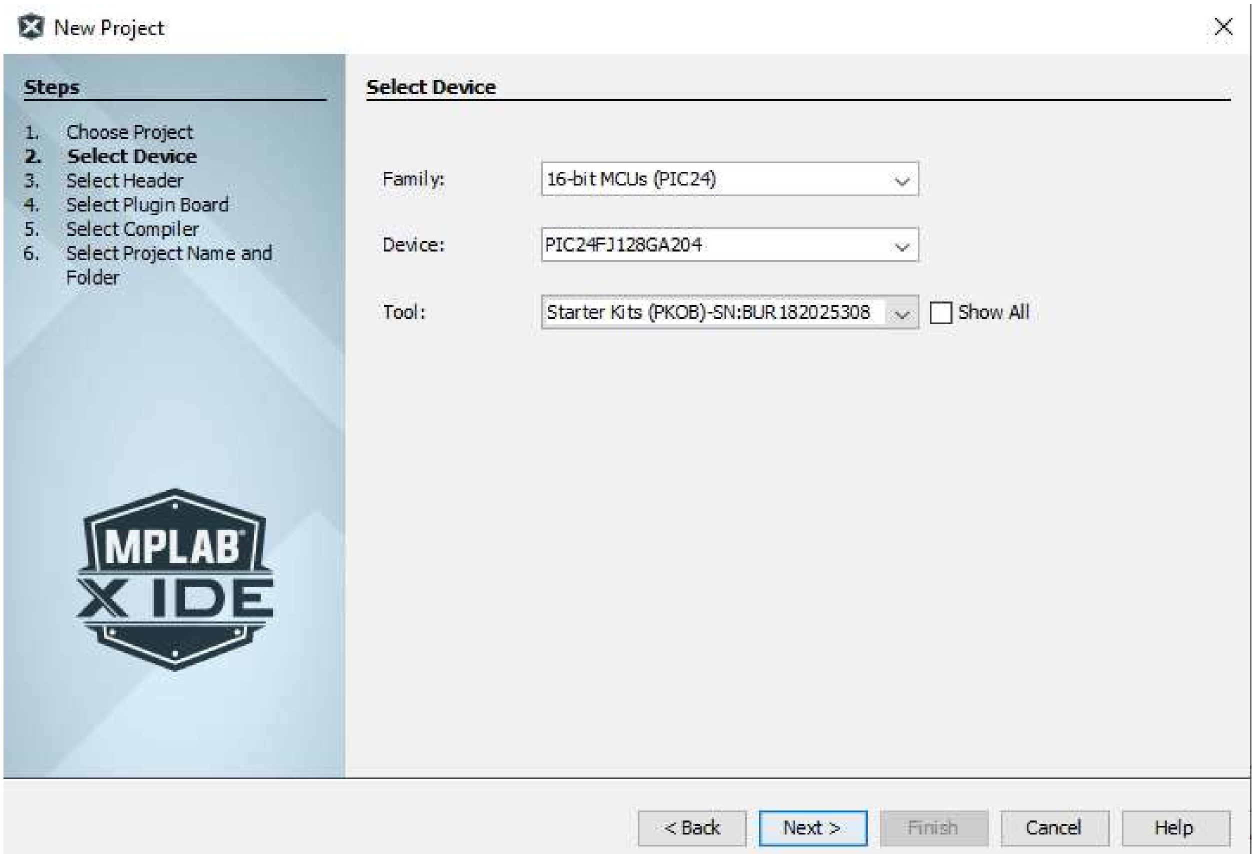


Figure 1.3: Select Device

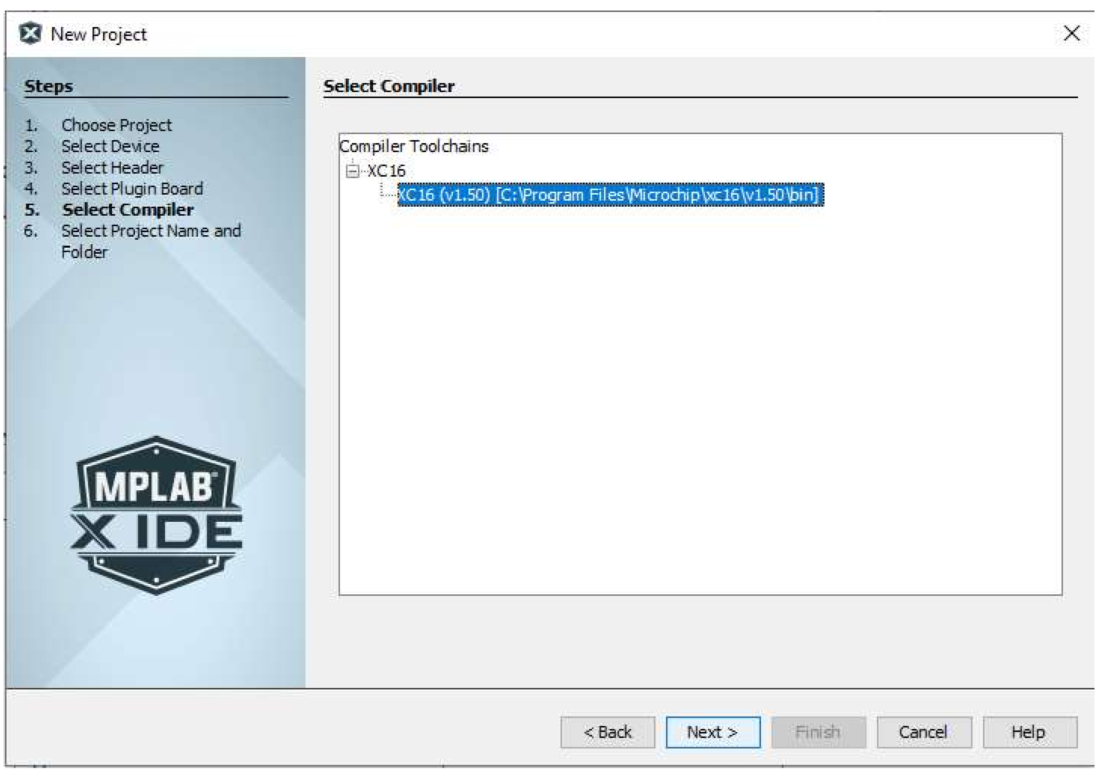


Figure 1.4: Select Compiler

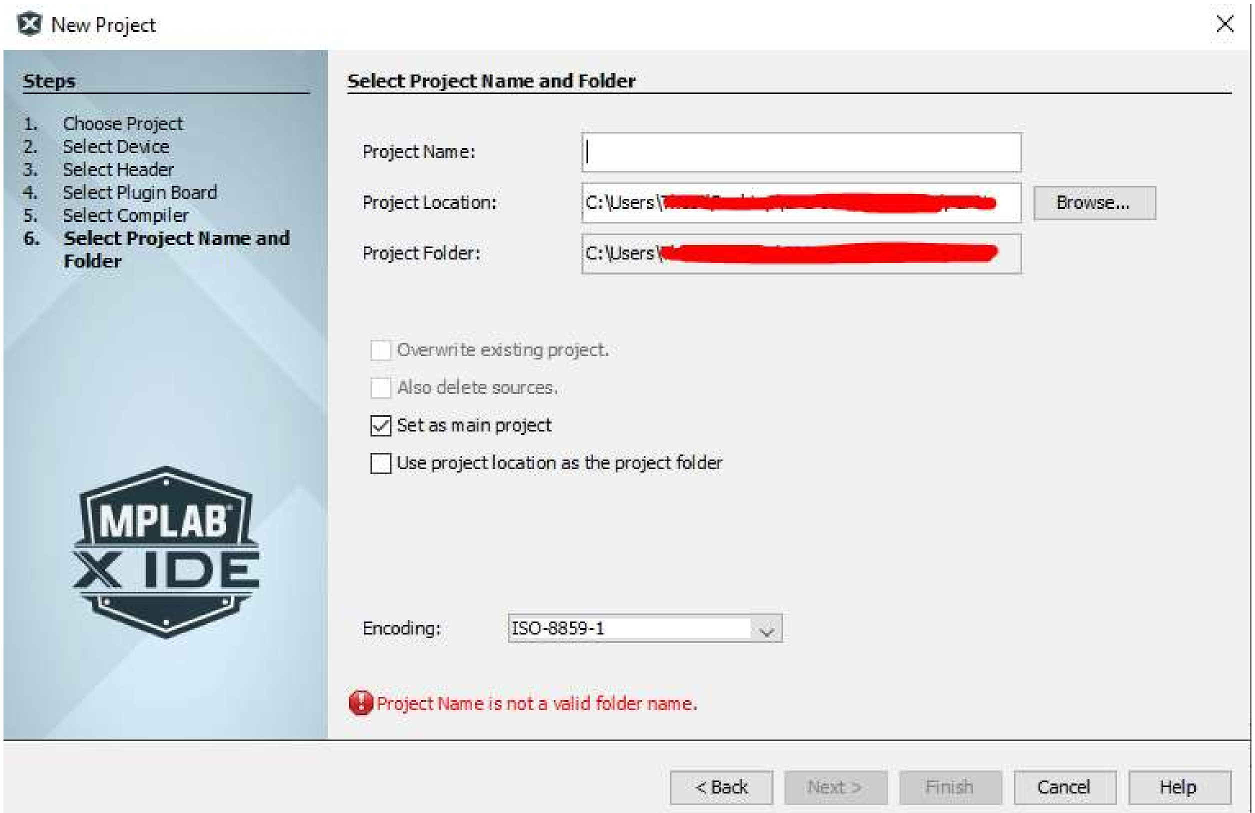


Figure 1.5: Select Project Name and Folder

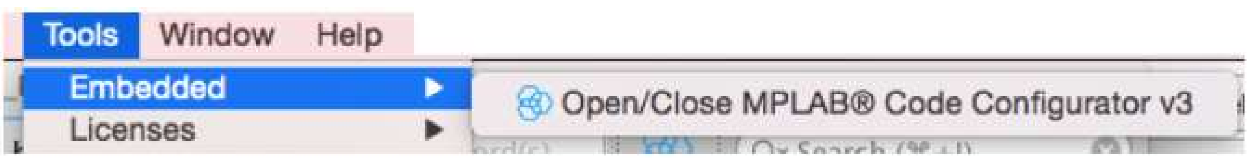


Figure 1.6: Open MCC

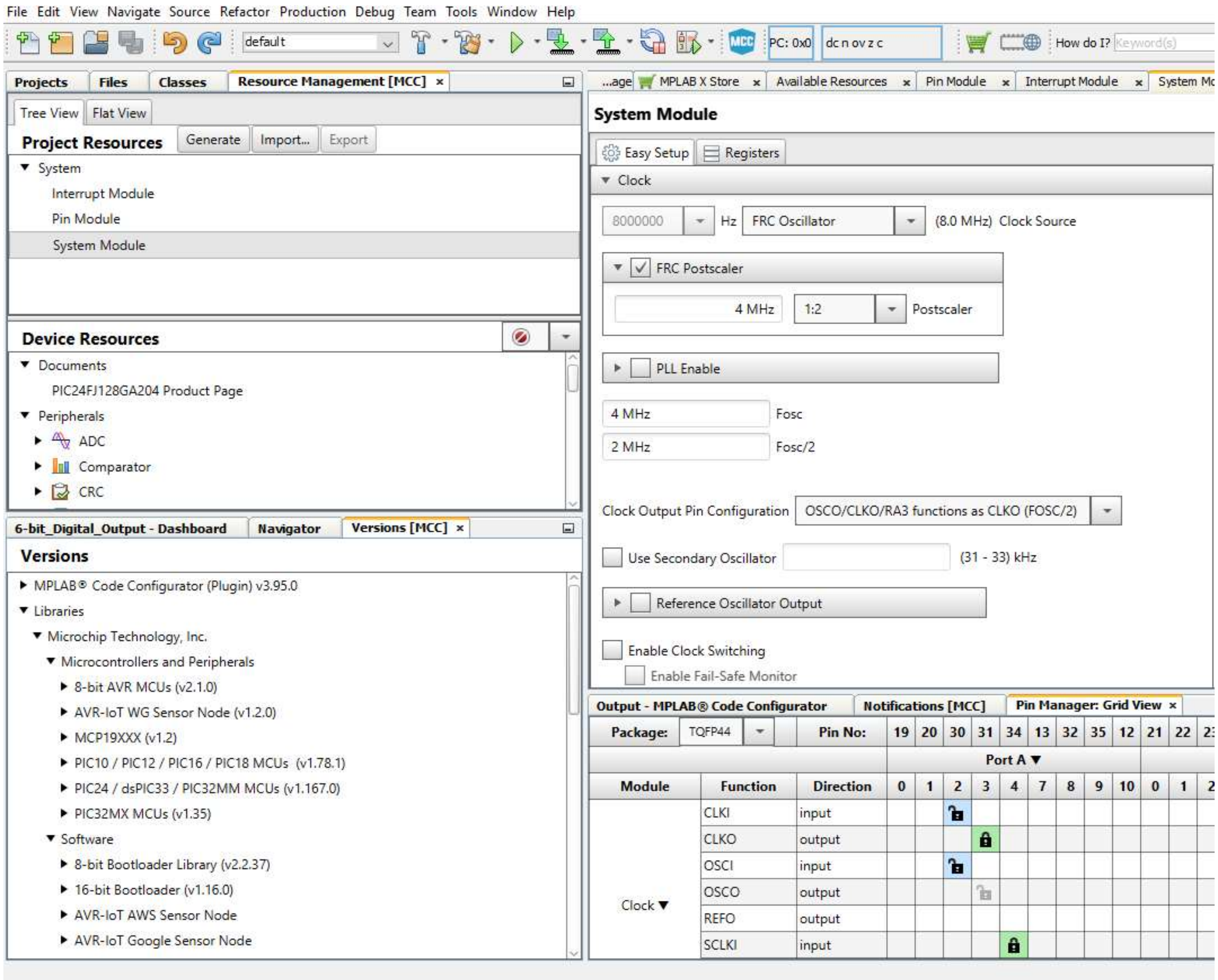


Figure 1.7: System Module Configuration

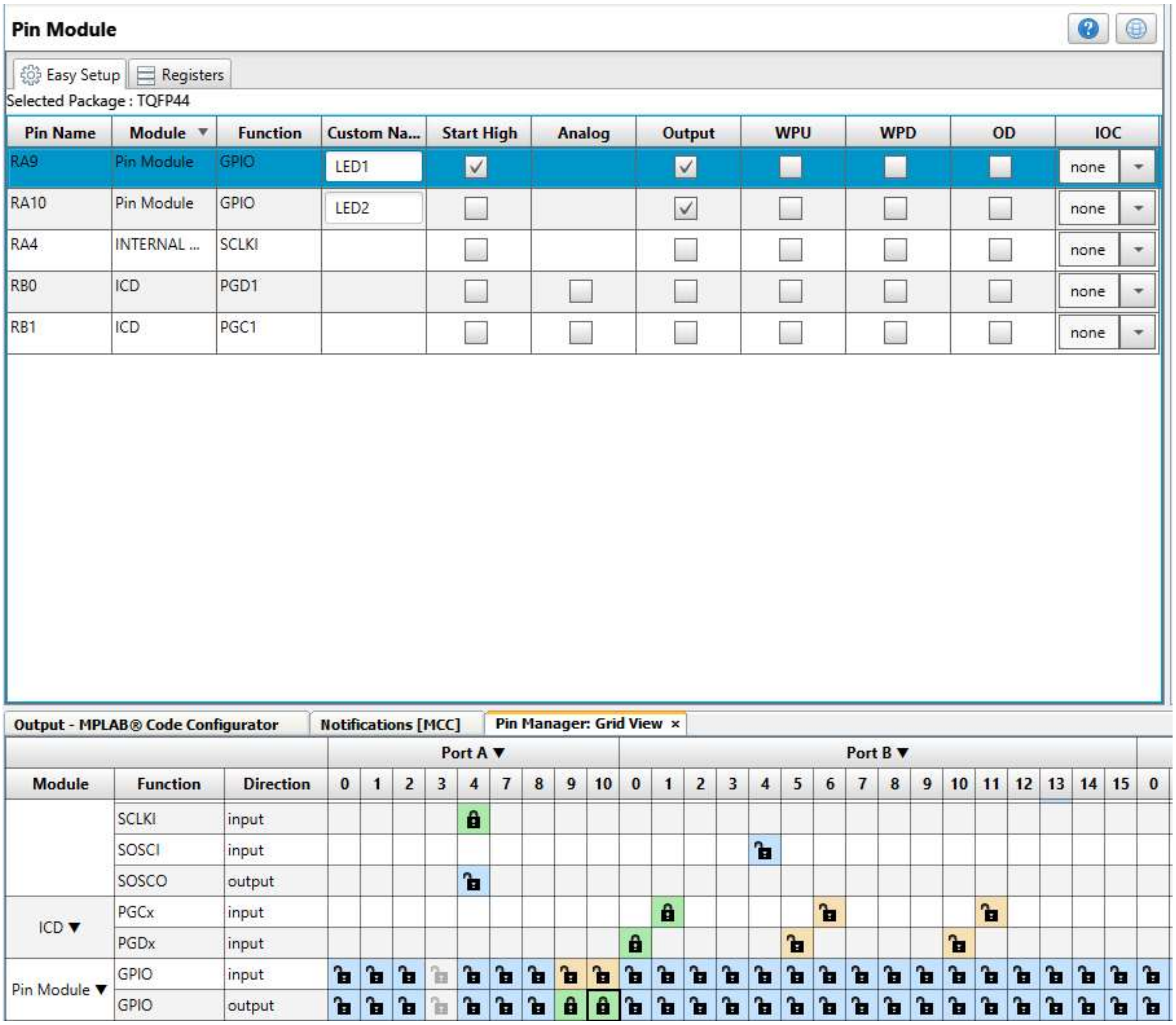


Figure 1.8: Pin Module and Pin Manager Configuration

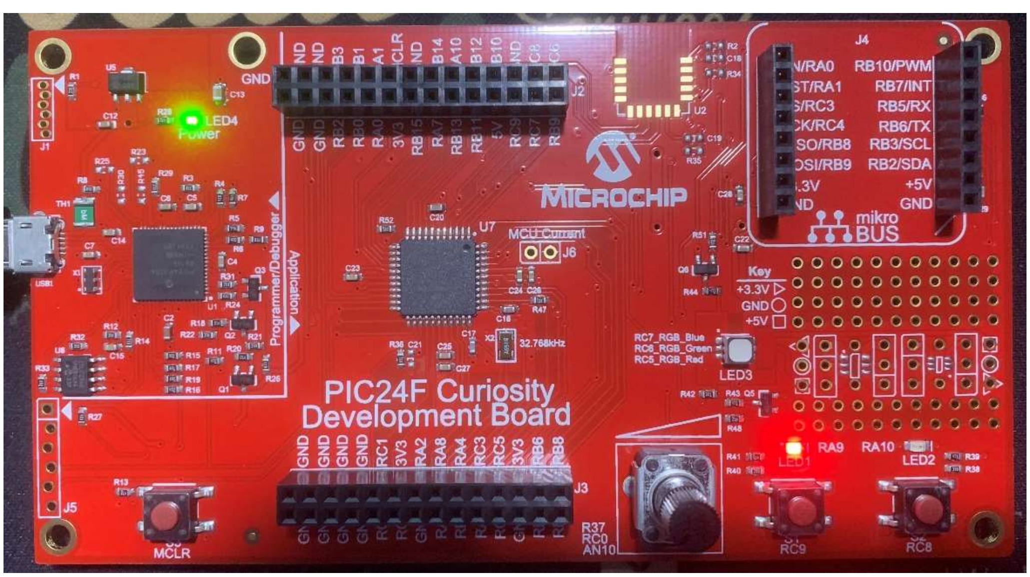


Figure 1.9: LED1 is ON and LED2 is OFF

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"
/*
                         Main application
 */
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    while (1)
    {
        LED1_SetLow();
        LED2_SetHigh();
    }

    return 1;
}
```

Figure 1.10: Code added to main.c file to turn on LED2 and turn off LED1

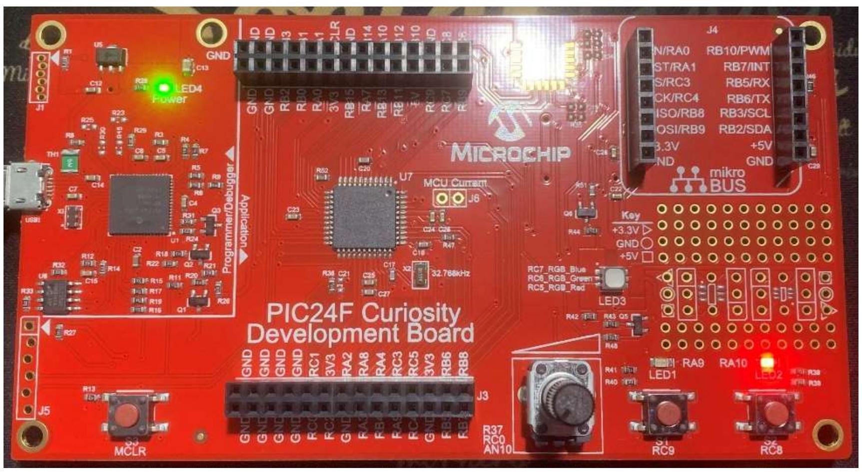


Figure 1.11: LED1 is OFF and LED2 is ON

Part 2: Digital Input: 16-bit Digital Input

This project will show how to configure and use digital inputs on a PIC24F MCU. We will be shown how to configure a pin on the MCU, which is connected to mechanical switch as a digital input. In addition, another pin will be configured as an output which is connected to an LED. The expected result of this project is when the switch is depressed, the LED will turn on and when the switch is released, the LED will turn off. We will use LED1 connected to pin RA9 and S1 connected to pin RC9 (see Figure 2.1).

First, we created a new, standalone project by following the same directions as part 1 with the exception of the Pin Module configuration (see Figures 2.2 and 2.3). Next, we modified the program to use the value of S1 to drive LED1 by adding the code provided to the main.c file (see Figure 2.4). Finally, we downloaded the program onto the board and successfully tested if LED1 turned on when the switch was pressed and turned off when the switch was released (see Figure 2.5).

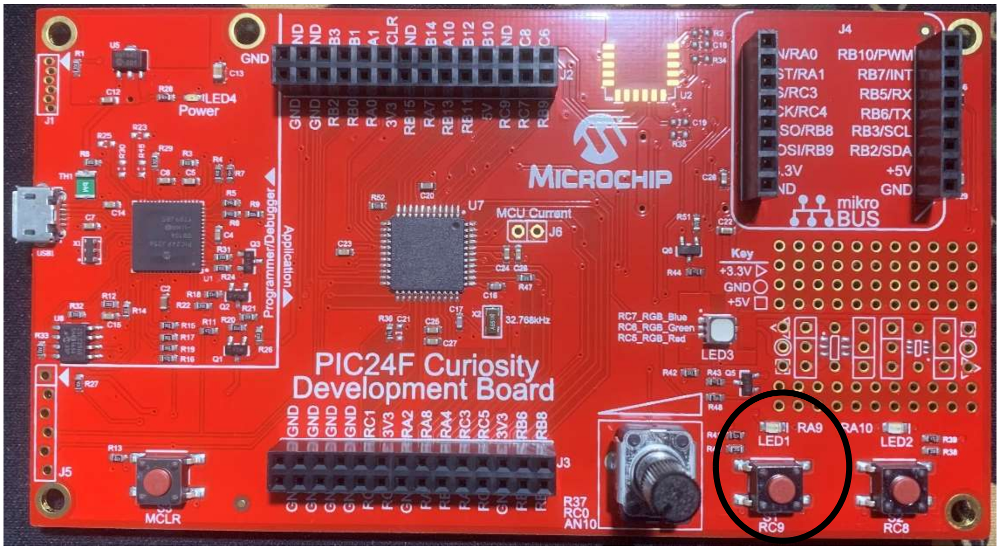
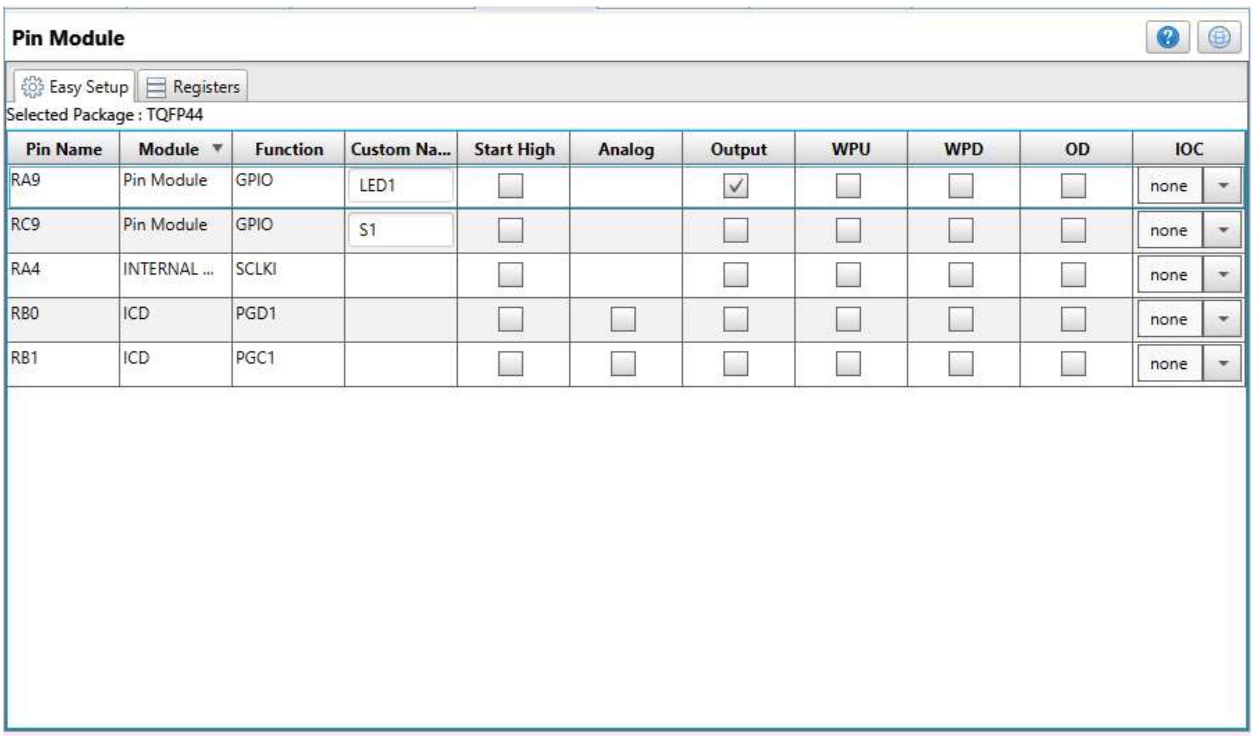


Figure 2.1: PIC24F Curiosity Development Board, Pins of Interest



Pin Module

Easy Setup | Registers

Selected Package : TQFP44

Pin Name	Module	Function	Custom Na...	Start High	Analog	Output	WPU	WPD	OD	IOC
RA9	Pin Module	GPIO	LED1	☐		☑	☐	☐	☐	none
RC9	Pin Module	GPIO	S1	☐		☐	☐	☐	☐	none
RA4	INTERNAL ...	SCLKI		☐		☐	☐	☐	☐	none
RB0	ICD	PGD1		☐	☐	☐	☐	☐	☐	none
RB1	ICD	PGC1		☐	☐	☐	☐	☐	☐	none

Figure 2.2: Pin Module Configuration

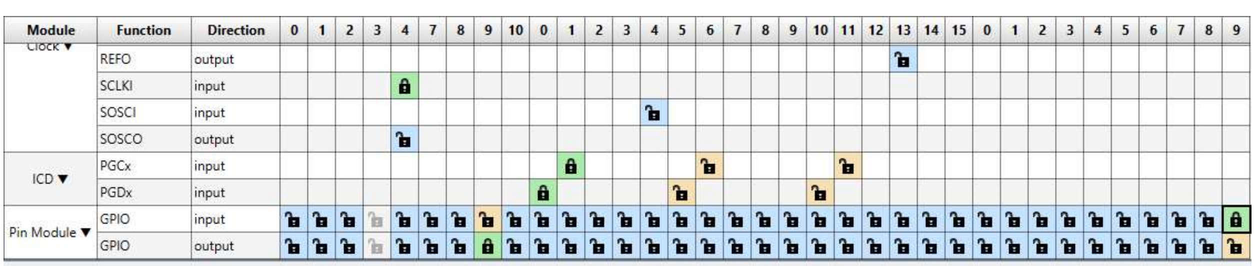


Figure 2.3: Pin Manager: Grid View Configuration

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"
/*
                          Main application
 */
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    while (1)
    {
        if (S1_GetValue())
               LED1_SetLow();
         else
               LED1_SetHigh();
    }

    return 1;
}
```

Figure 2.4: Code added to main.c file to turn on LED1 when switch is pressed

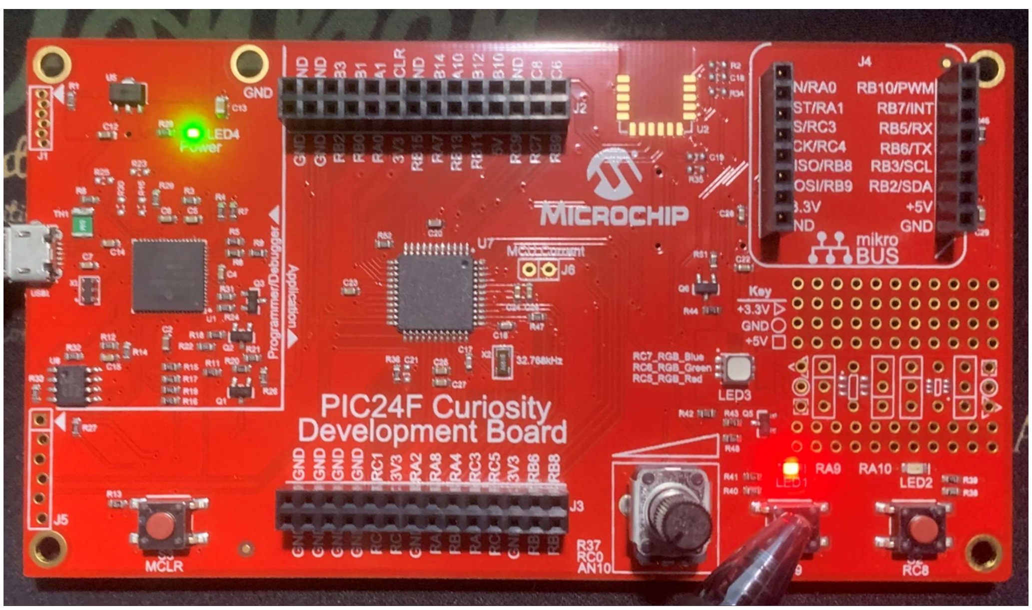


Figure 2.5: LED1 is ON when switch is pressed

Part 3: Timer: Programming the PIC24/dsPIC33 Timer and Programming the PIC24/dsPIC33 Timer Using Interrupts

The first project will show how to configure and use an internal timer on a PIC24 16-bit MCU. We will learn how to configure one of the timers to overflow every ½ second and have the code monitor the timer's overflow flag. The expected behavior of this project is each time the timer overflows the program will toggle one of the LEDs on the development board.

First, we created a new, standalone project by following the same directions as part 1 with the exception of the Pin Module configuration (see Figures 3.2 and 3.3). In addition, we added a Timer Configuration (see Figure 3.4) with the Fosc set at 2 MHz. However, since Timer1 will be unable to generate a ½ second period as the 16-bit counter will overflow before ½ second has passed. Therefore, we will length the period window for this timer by pre-scaling the system clock by 64 to down the timer. Next, we added the provided code (see Figure 3.5) to the main.c file and run program on the development board. We noticed LED1 changing every ½ second from the ON state to OFF state which is the correct result we wished to achieve.

The second project will have the same result as the first project, but we will be using interrupts. First, we created a new, standalone project by following the same directions as the previous project above, but we added an interrupt feature. While setting up the Timer1, we check the box "Enable Timer Interrupt" (see Figure 3.6) and make sure the Interrupt Module shows Timer1 has priority (see Figure 3.7). Then we added the provided code in Figure 3.8 to the main.c file and ran the code on the development board. We noticed LED1 having the same output as the previous project which indicates we successfully completed the project.

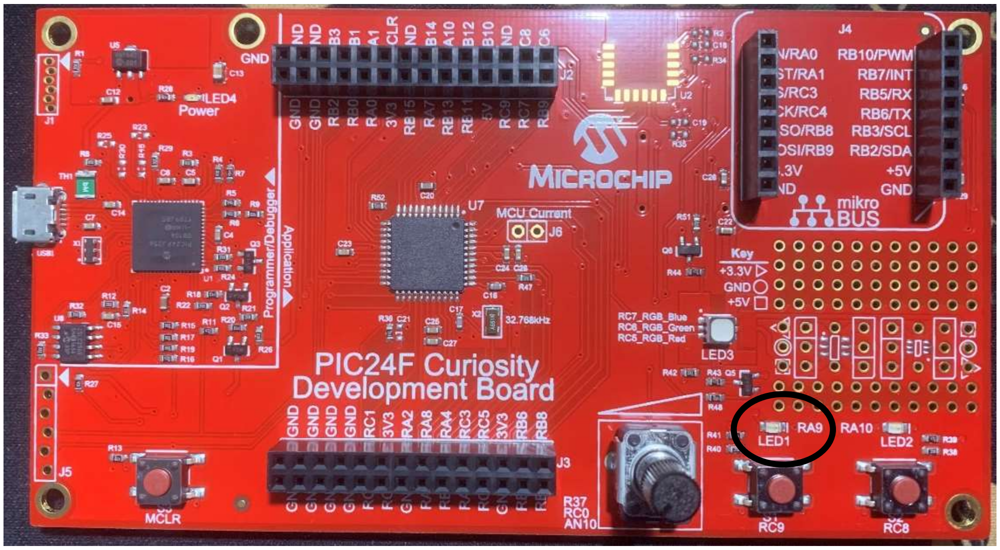


Figure 3.1: PIC24F Curiosity Development Board, Pins of Interest

Pin Name	Module	Function	Custom Na...	Start High	Analog	Output	WPU	WPD	OD	IOC
RA9	Pin Module	GPIO	LED1	☐		☑	☐	☐	☐	none
RA4	INTERNAL ...	SCLKI		☐		☐	☐	☐	☐	none
RB0	ICD	PGD1		☐	☐	☐	☐	☐	☐	none
RB1	ICD	PGC1		☐	☐	☐	☐	☐	☐	none

Figure 3.2: Pin Module Configuration

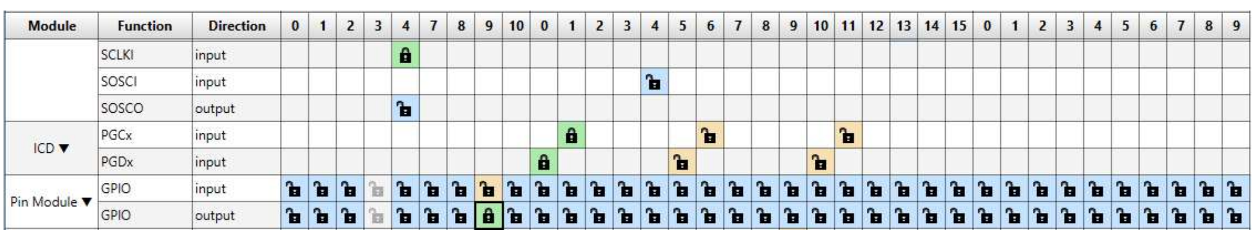


Figure 3.3: Pin Manager: Grid View Configuration

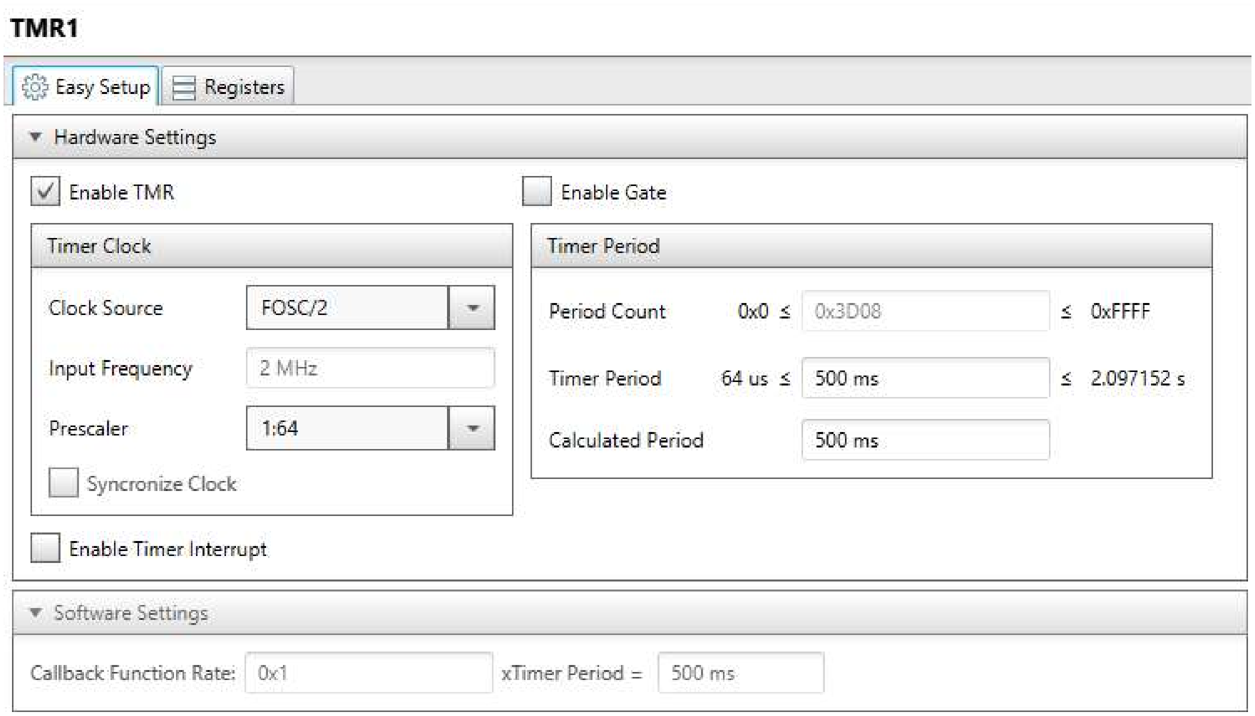


Figure 3.4: Timer1 Configuration Without Interrupt

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"
/*
                         Main application
 */
int main(void)
{
   // initialize the device
    SYSTEM_Initialize();
    TMR1_Start();   // ####### Start the timer ####
    while (1)
    {
        if (IFS0bits.T1IF)
        {
            LED1_Toggle();
            IFS0bits.T1IF = 0 ;
        }
    }
    return 1;
}
```

Figure 3.5: Code added to main.c file to change the state of LED1 every ½ second

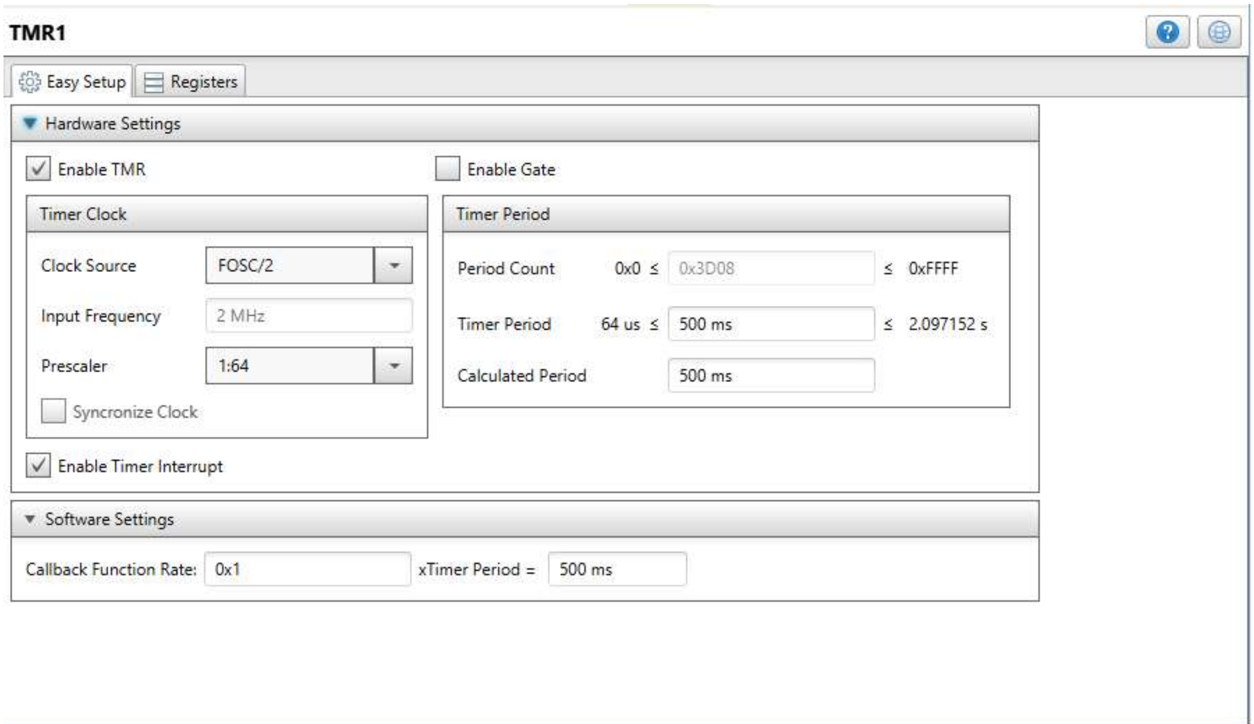


Figure 3.6: Timer1 Configuration With Interrupt

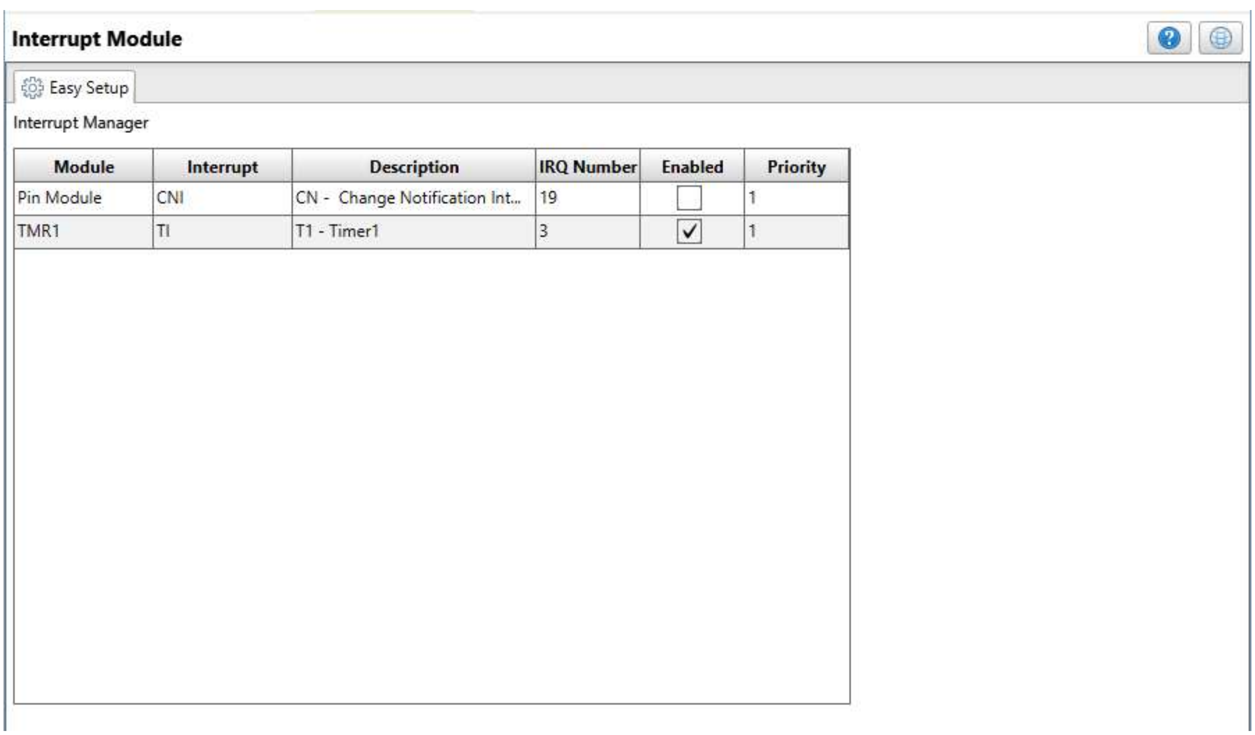


Figure 3.7: Interrupt Module

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"  //##### must be added #####

void My_ISR(void)                     //##############
{
    LED1_Toggle();
}
int main(void)
{
    SYSTEM_Initialize();
    TMR1_SetInterruptHandler(My_ISR) ;  //###########
    TMR1_Start();                       //###########
    while (1)
         {
         }
    return 1;
}
```

Figure 3.8: Code added to main.c file to change the state of LED1 every ½ second

Part 4: ADC Input

For this project, we will learn how to configure and use an internal Analog-to-Digital Converter (ADC) on a PIC24F MCU to change the speed of a blinking LED. We will configure a PIC24 timer to blink an LED on the demo board at an interval determine by the timer's Period register. The demo board used has a digital potential (pot) connected to one of the MCU's analog capable pins. When the pot is moved, the resulting voltage on the analog pin will change. Our objective is to write code which triggers the ADC to read the value of the attached pot and place that value into the timer's Period register (see Figure 4.1). The expected result of this project is to witness the speed in which the LED blinks will change as the pot is rotated.

First, we created a new, standalone project by following the same directions as part 1. Second, we added a Timer1 and a ADC and configured the settings as shown in Figures 4.2 – 4.6. Next, we added the provided code shown in Figure 4.7 to the main.c file and downloaded the code onto the board. We noticed that as the digital potential on the demo board was turned to the right, the LED light flashes very quickly and finally flashes at ½ second.

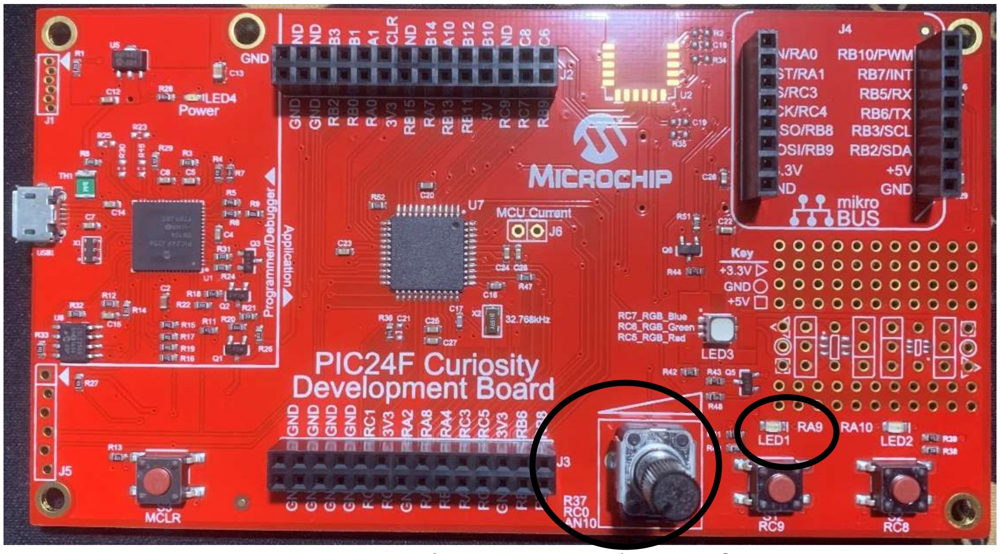


Figure 4.1: PIC24F Curiosity Development Board, Pins of Interest

Pin Name	Module	Function	Custom Na...	Start High	Analog	Output	WPU	WPD	OD	IOC
RA9	Pin Module	GPIO	LED	☐		☑	☐	☐	☐	none
RA4	INTERNAL ...	SCLKI		☐		☐	☐	☐	☐	none
RB0	ICD	PGD1		☐	☐	☐	☐	☐	☐	none
RB1	ICD	PGC1		☐	☐	☐	☐	☐	☐	none
RC0	ADC1	AN10	channel_AN	☐	☑	☐	☐	☐	☐	none

Figure 4.2: Pin Module Configuration

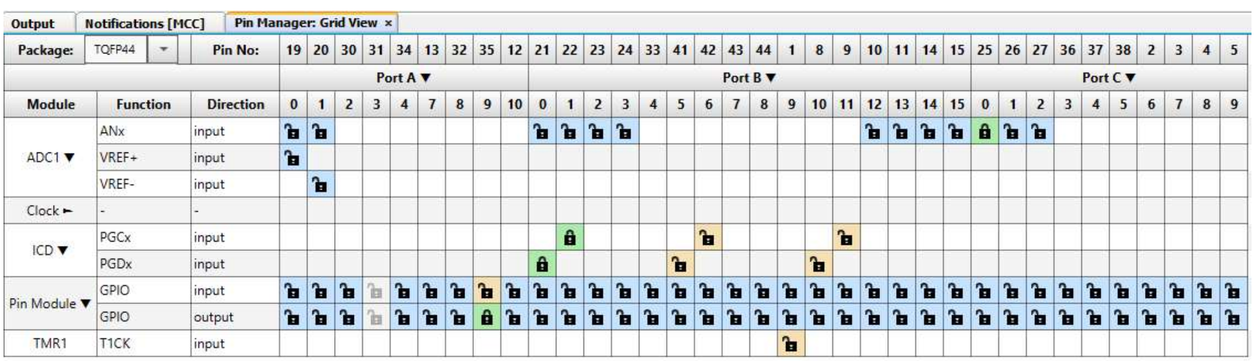


Figure 4.3: Pin Manager: Grid View Configuration

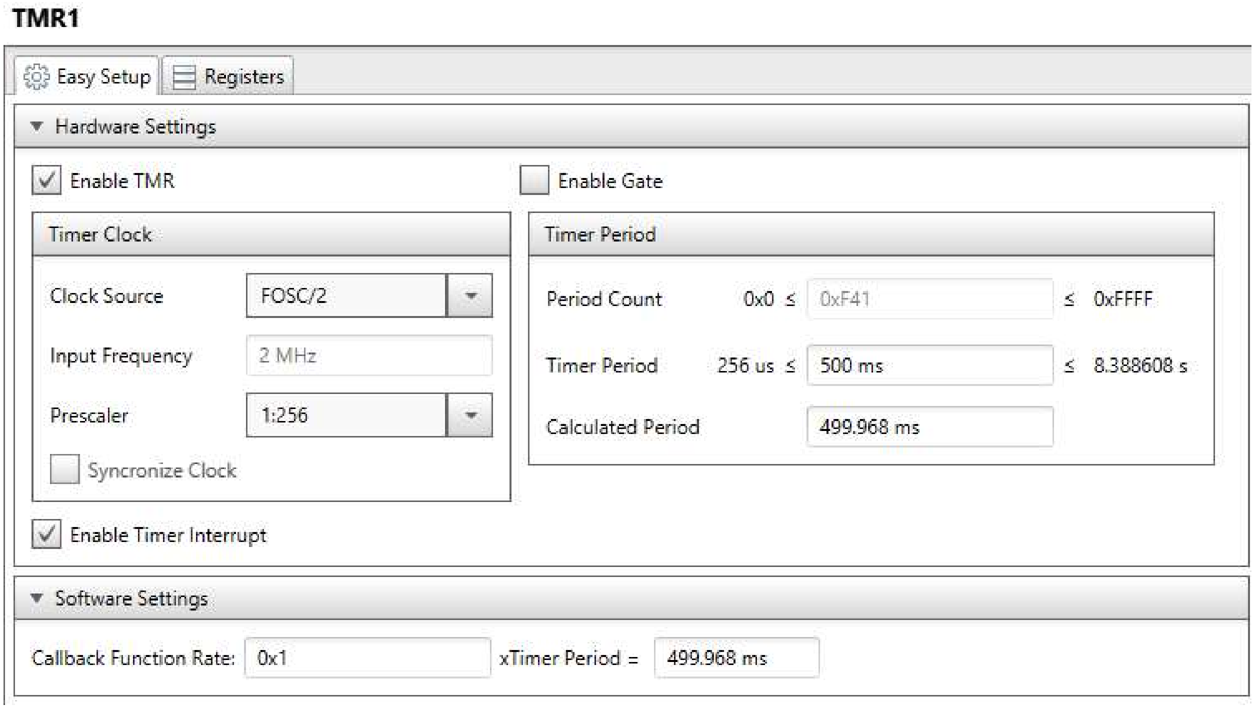
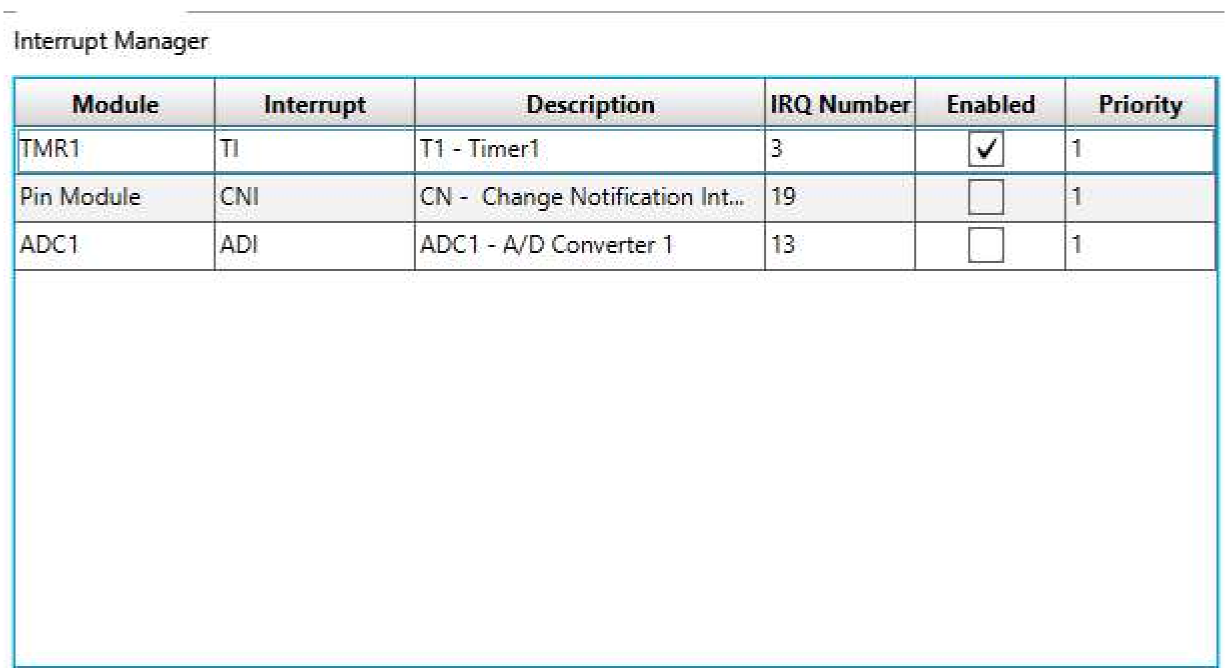


Figure 4.4: Timer1 Configuration


Interrupt Manager

Module	Interrupt	Description	IRQ Number	Enabled	Priority
TMR1	TI	T1 - Timer1	3	✓	1
Pin Module	CNI	CN - Change Notification Int...	19		1
ADC1	ADI	ADC1 - A/D Converter 1	13		1

Figure 4.5: Interrupt Module Configuration

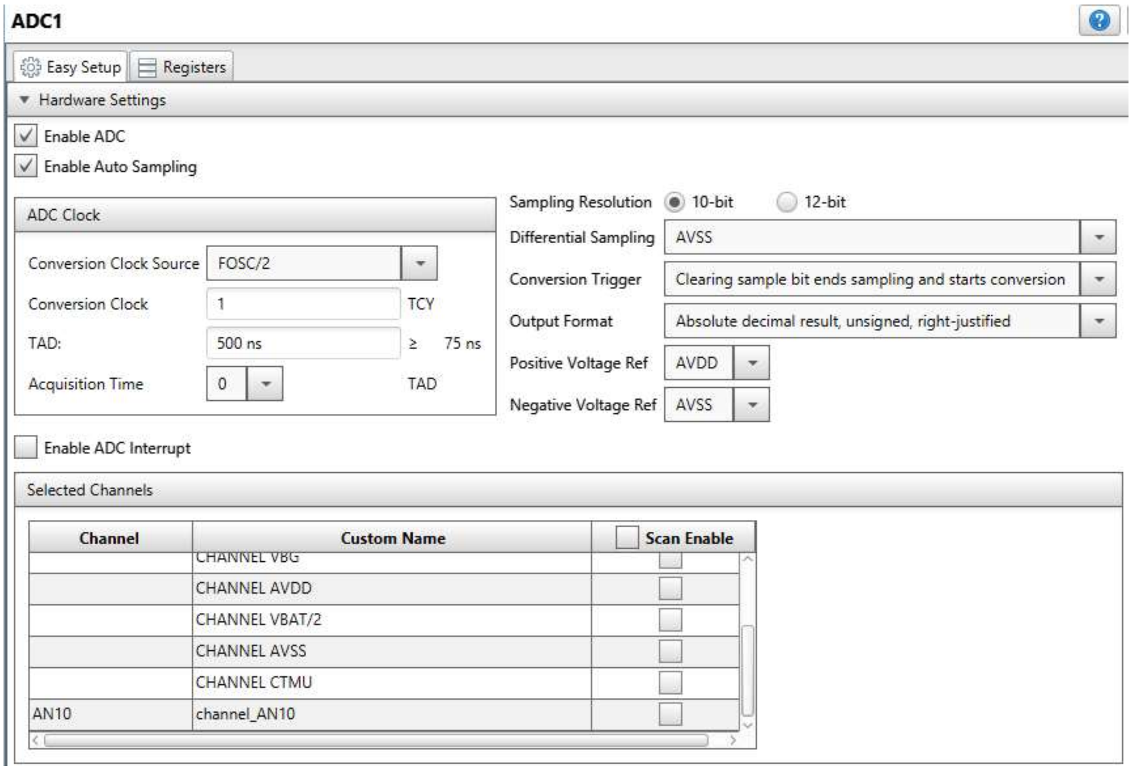


Figure 4.6: Analog-to-Digital (ADC) Converter Configuration

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"  //##### must be added #####

void My_ISR(void)
{
    LED_Toggle();
    ADC1_SoftwareTriggerDisable();      // trigger next conversion
}

int main(void)
{
    SYSTEM_Initialize();
    ADC1_ChannelSelect(channel_AN10);
    ADC1_SoftwareTriggerDisable();      // begin first conversion
    TMR1_SetInterruptHandler(My_ISR);
    TMR1_Start();
    while (1)
    {
        if (ADC1_IsConversionComplete(channel_AN10)){
            if (ADC1_ConversionResultGet(channel_AN10)==0)
                TMR1_Period16BitSet(1);
            else
               TMR1_Period16BitSet(ADC1_ConversionResultGet(channel_AN10));
        }
    }
        return 1;
}
```

Figure 4.7: Code added to main.c file

Conclusion:

For this lab, we will explored the capabilities of the PIC24F Curiosity Development Board and became familiar with the MPLAB X IDE. We learned how to set up a new project, configure the pin out module, the timer module, the interrupt module, and the ADC module. This lab was fairly straightforward and found the explanations of each feature when configuring the project as well as the code extremely helpful in understanding the concepts of each part. Two problems I ran into during the lab included not being able to find the MCC plugin initially and using the wrong software in which I was using the MPLAB X IPE software instead of the MPLAB X IDE. However, this lab only took me a few hours to complete and I have a better understanding of how to use the PIC24F Development Board for future projects.