Shammah Thao

EEE 174 - CpE 185 Lab Section #2

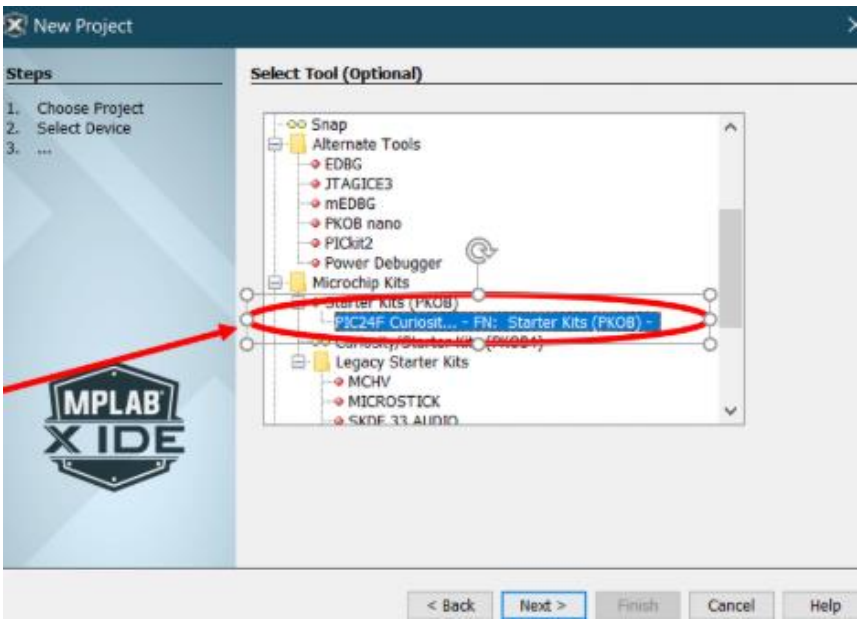
Monday & Wednesday

Lab 2: Microchip Development System

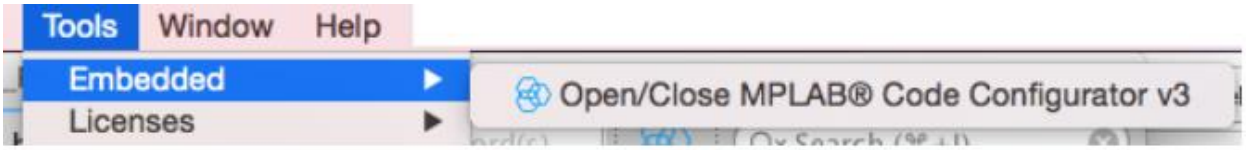
Dahlquist

Part 1: 16-bit digital output Example

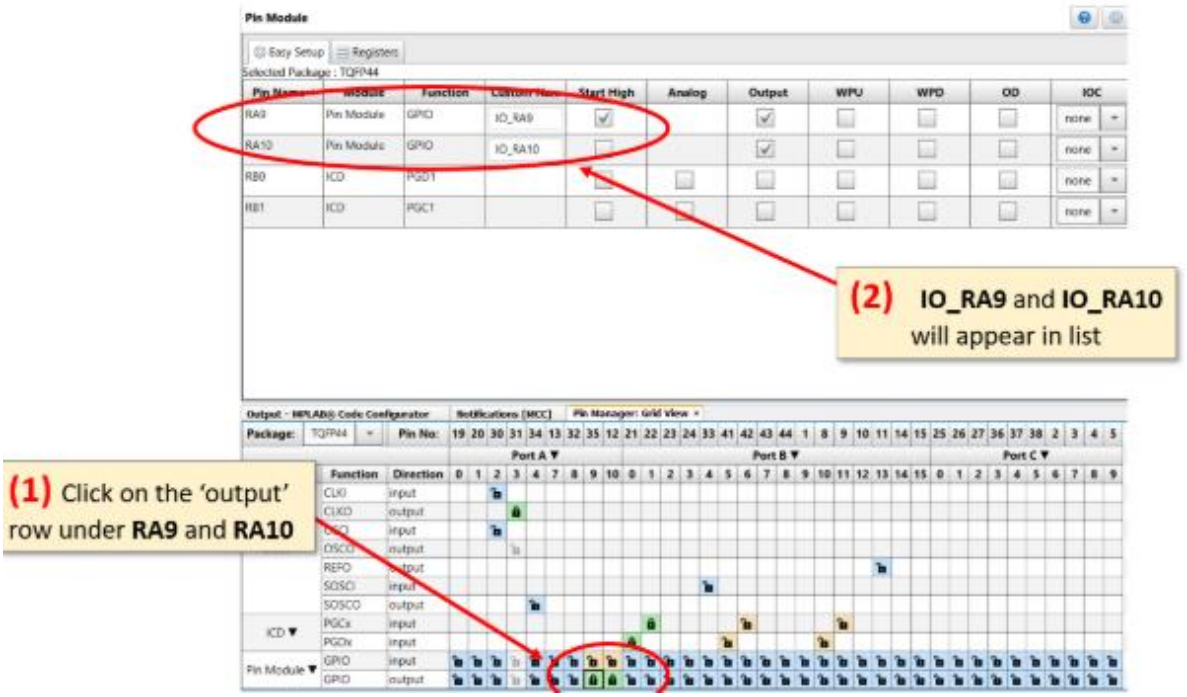
To start doing this lab we would have to start by downloading the software MPLAB X IDE, MPLAB XC 16 and install MPLAB Code Configurator. After that we would then have to open up MPLAB X IDE and start a new project.



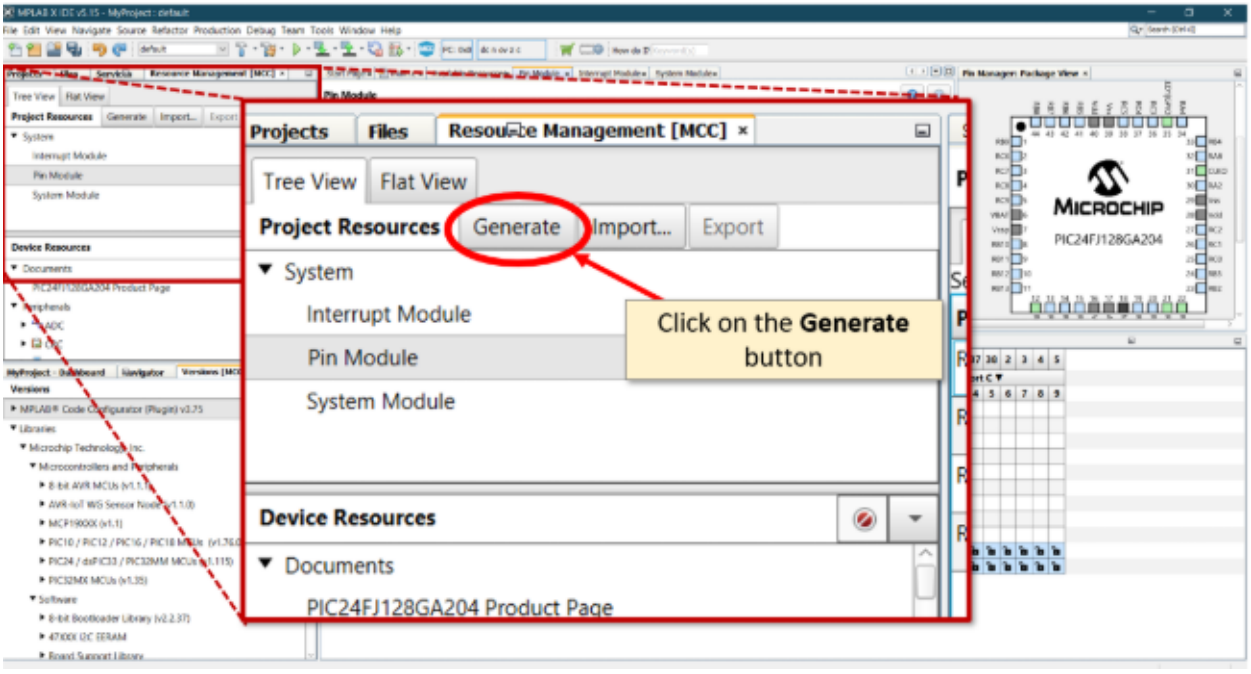
After going through the process of creating the project, we then must open up the code configurator from the tools bar



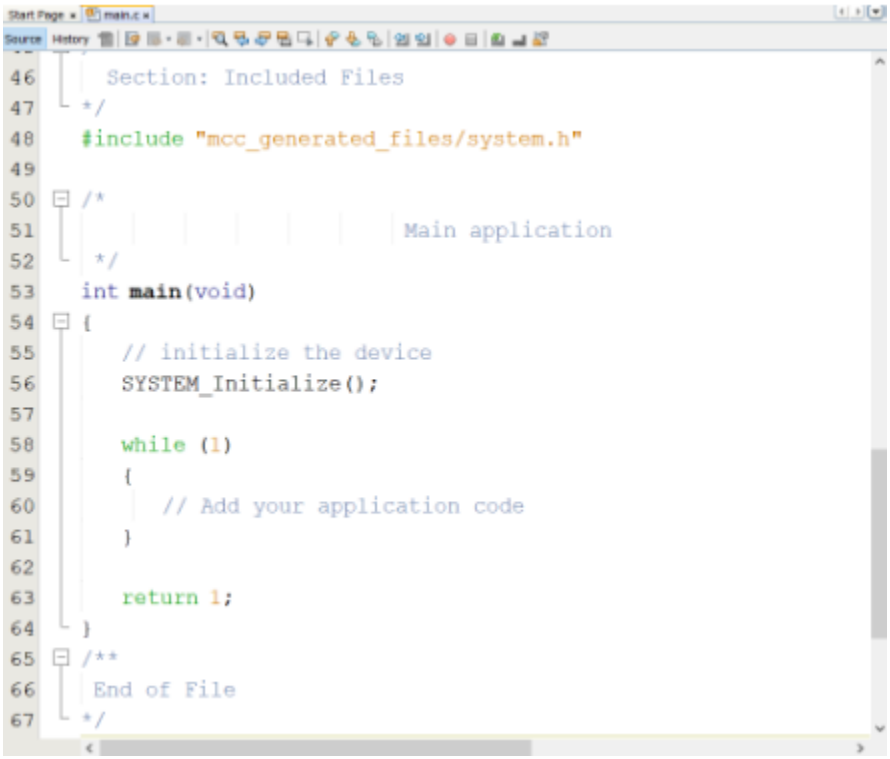
Then after that we would have to set the pin module to be like the one from the example.



When the pin module matches to what was needed, we then generated the code from the left-hand side.

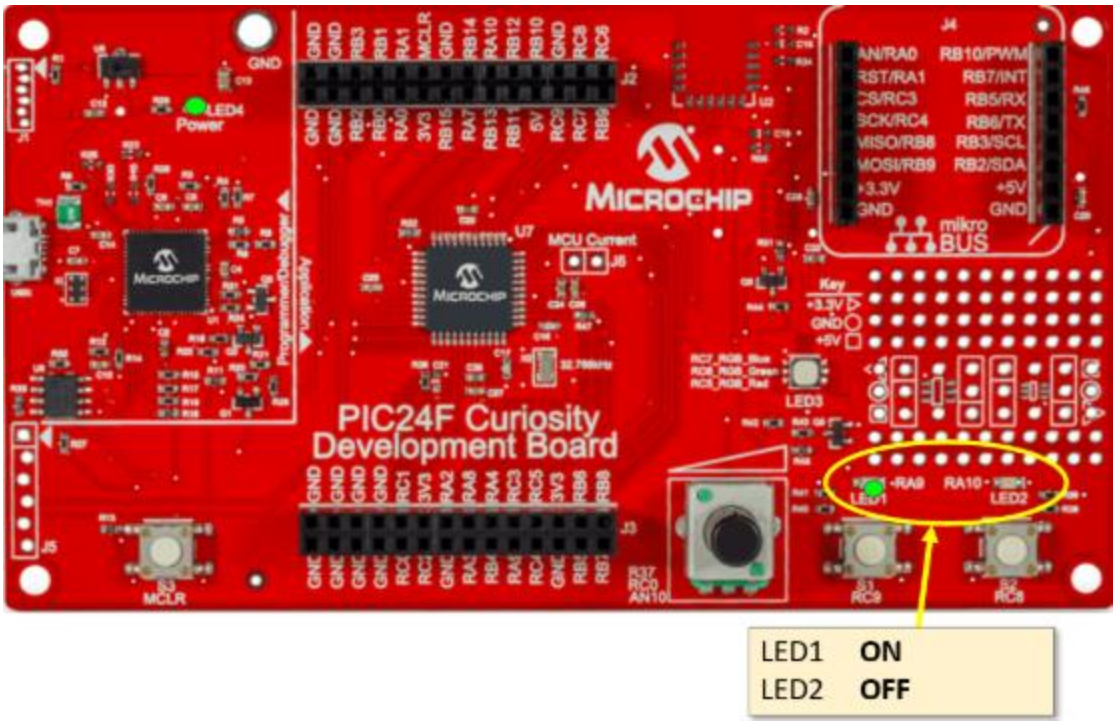


We then got the main file called MCC.h from the generated files, which we then added a given line of code to run.



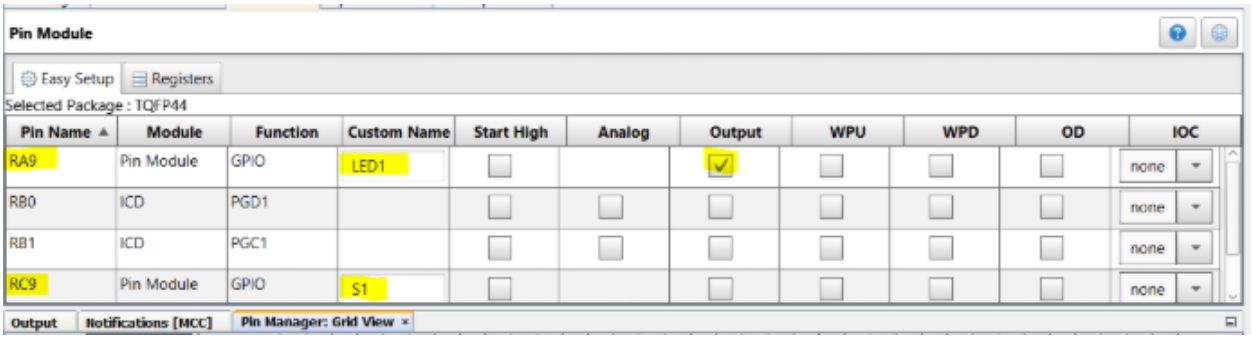
```
46      Section: Included Files
47  */
48  #include "mcc_generated_files/system.h"
49
50  /*
51                          Main application
52   */
53  int main(void)
54  {
55      // initialize the device
56      SYSTEM_Initialize();
57
58      while (1)
59      {
60          // Add your application code
61      }
62
63      return 1;
64  }
65  /**
66   End of File
67  */
```

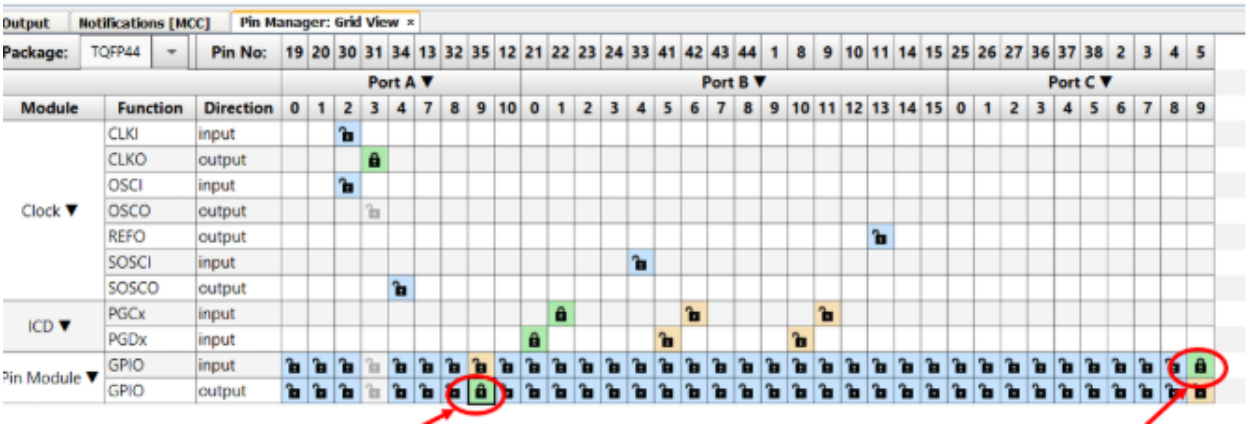
After putting in the code and building the program, we should be able to see the led on the board change. The led 1 should be lit up constantly without touching it and led 2 should be blank with no light on.



Part 2: 16-bit digital input

For this part of the lab, we basically did the same thing as the previous part. We made a new project then started modifying the pin modules.

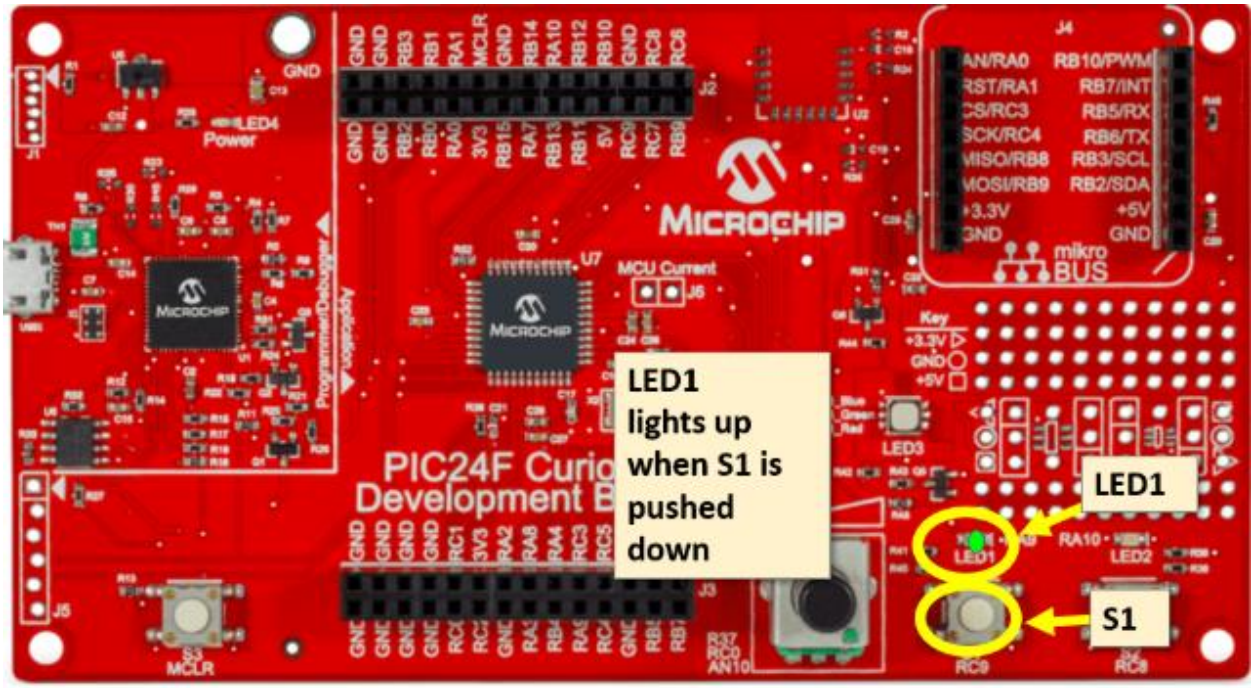




After we fix the pin module, we then generated the code again to get the mcc.h file and we modify that too with the code given to use in the instruction.

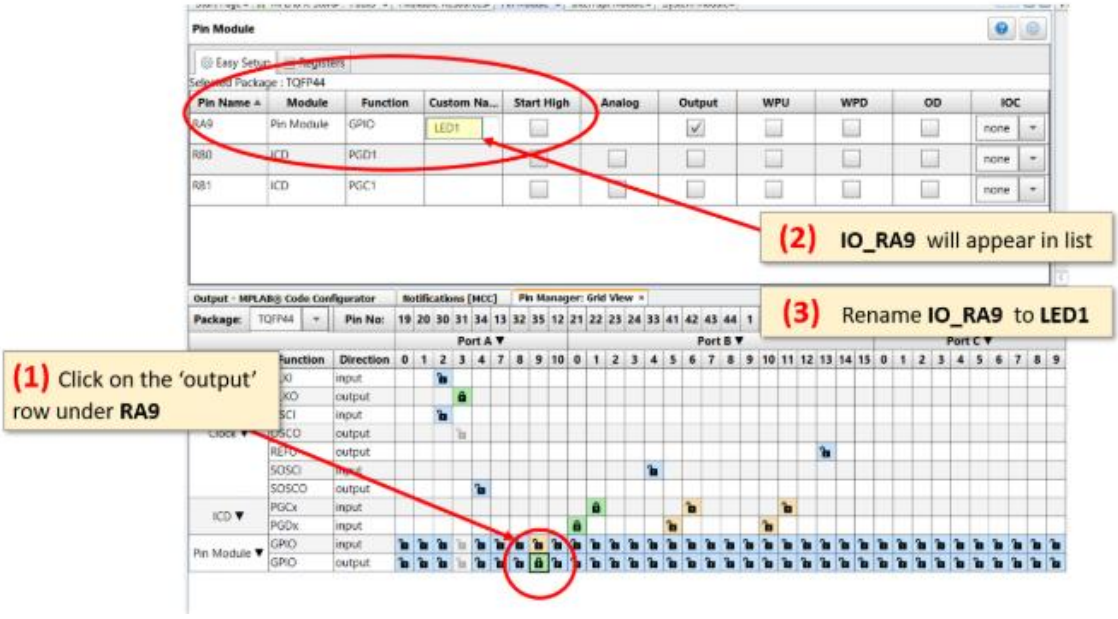
```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"
/*
                        Main application
 */
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

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

    return 1;
}
```

After putting the code in, we then build the code and ran it, which gave us the following result. The led1 should be on when the button is pushed and led should be off no matter what. Led 1 only turn on when the button is pressed.

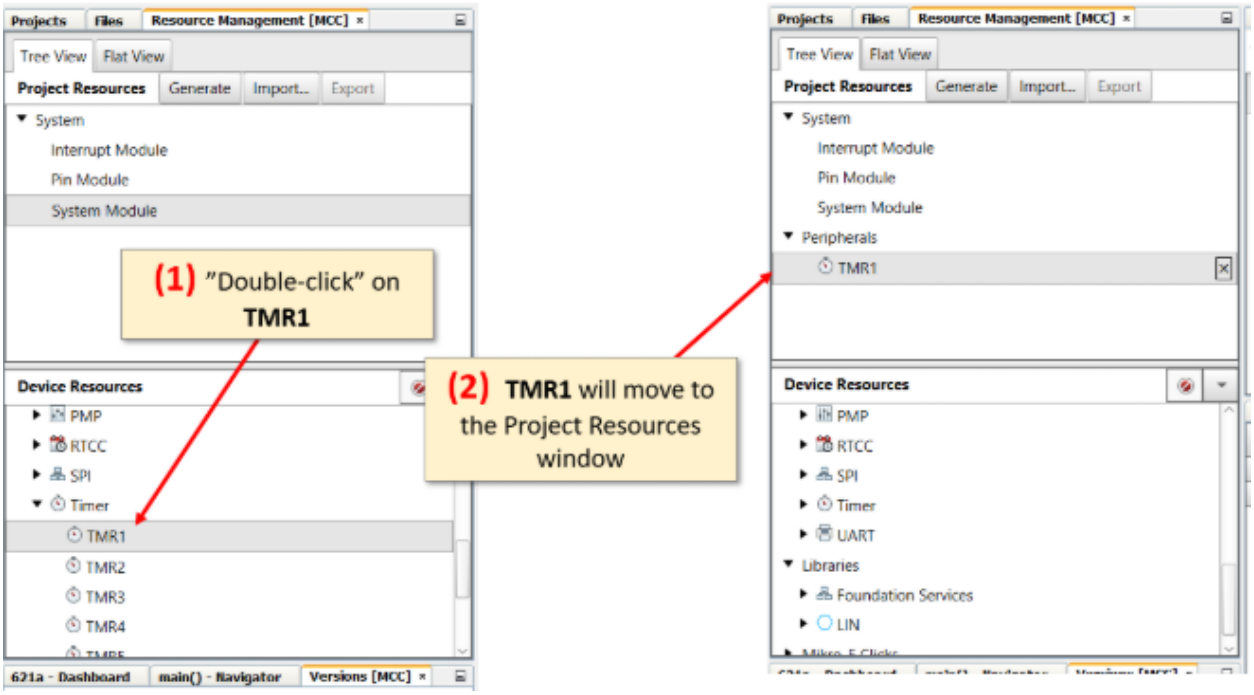


Part 3 A: Programming the PIC24/dsPIC33 Timer

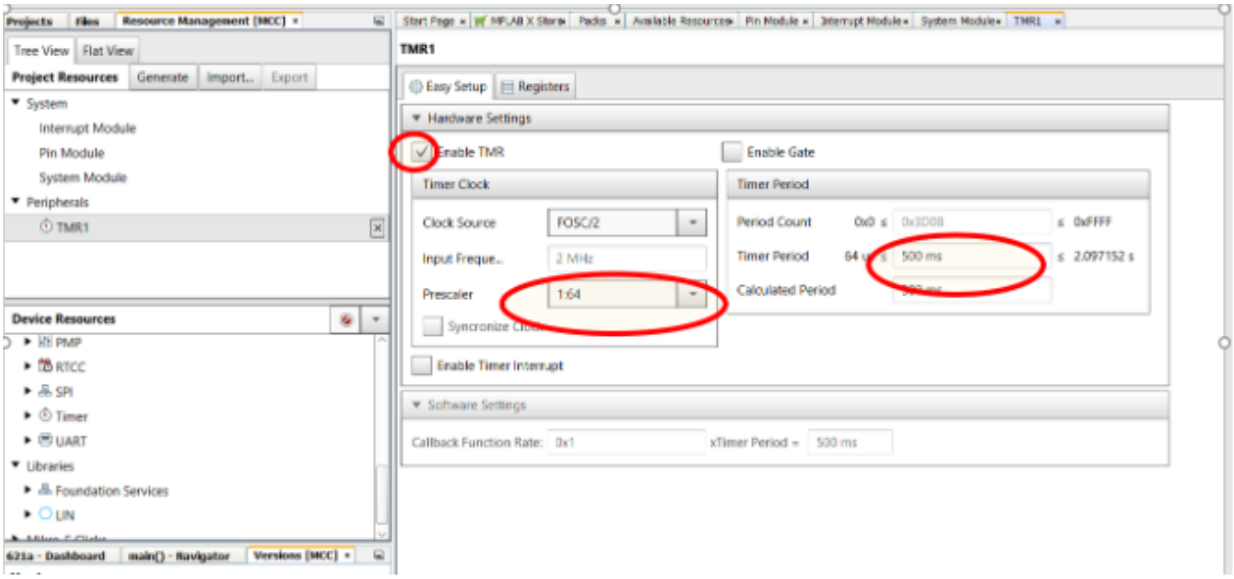
For this part we are implementing a timer into our code which would make the led flashes over time. We first build a new project then modify the pin modules again.



We then added the timer into the system and started setting it up.



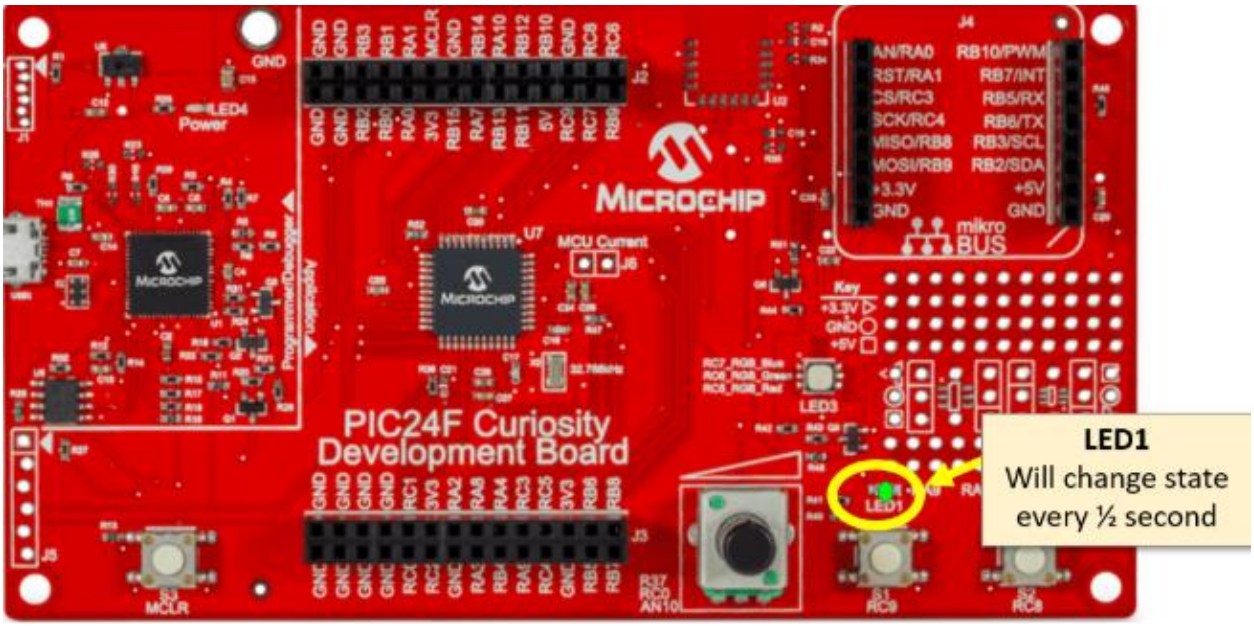
We then set the timer to operate a certain interval and how flash it should work.



After this, we get the timer down we generate the code for the mcc.h file. Which added the following code to the main method.

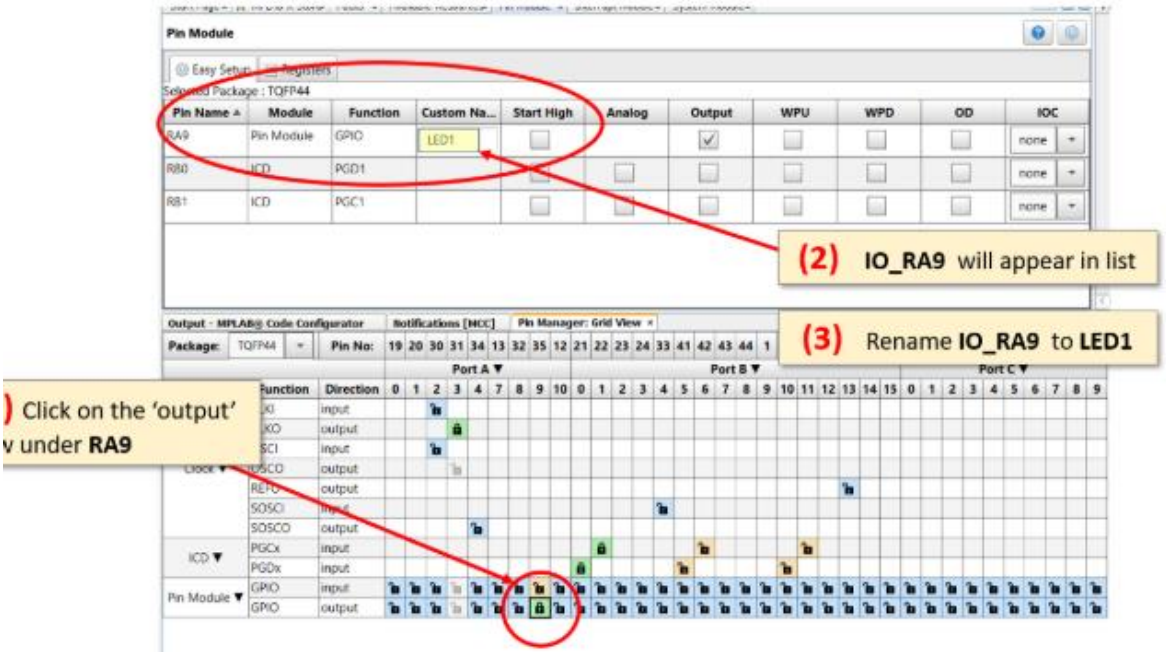
```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"  //##### must be added #####
/*
                        Main application
 */
int main(void)
{
   // initialize the device
    SYSTEM_Initialize();
    TMR1_Start();   // ####### Start the timer ####
    while (1)
    {
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

After this we build the program and ran it to show the result onto the board. It would show the LED light blinking at a constant rate without you needing to hold the button

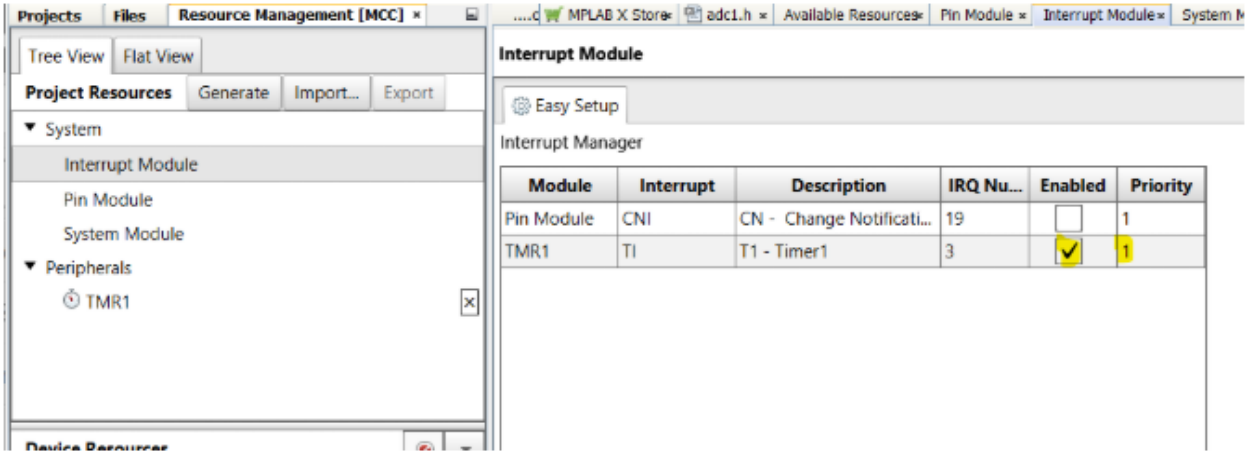


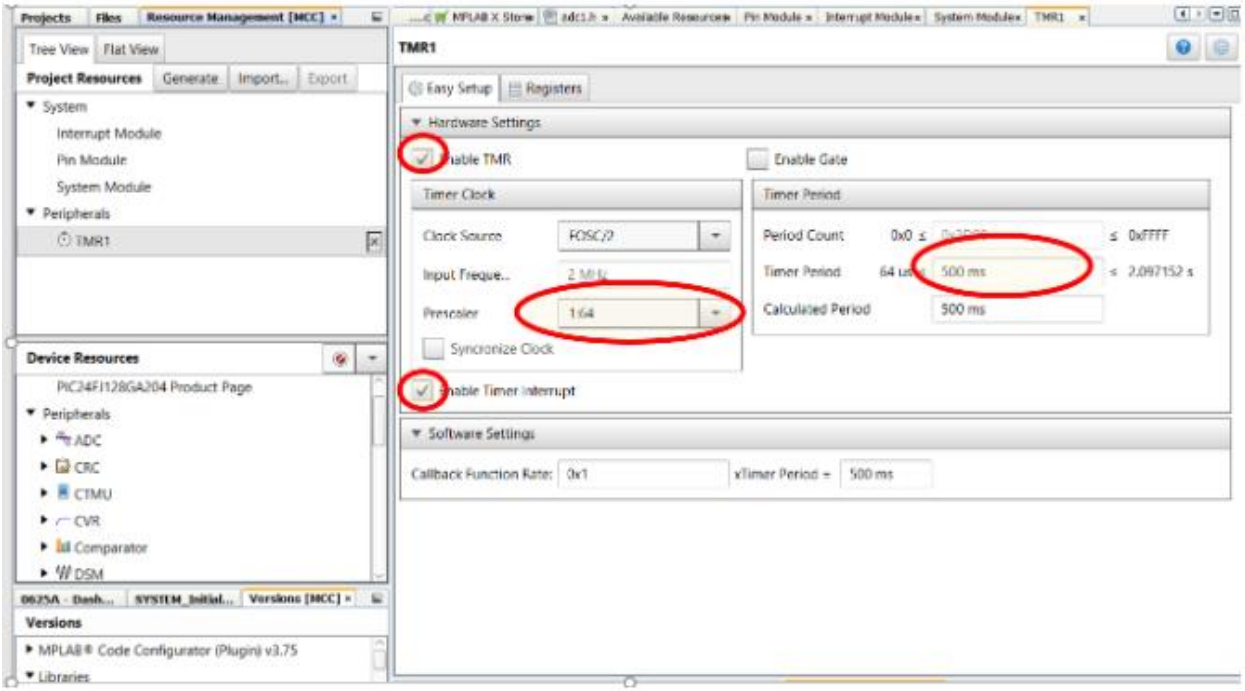
Part 3 B: Programming a PIC24/dsPIC33 Timer Using Interrupts

We had to do the same thing as the other part, where we make a new project, and modify the pin module to match the example.



We then added the interrupts and also a timer into the program.

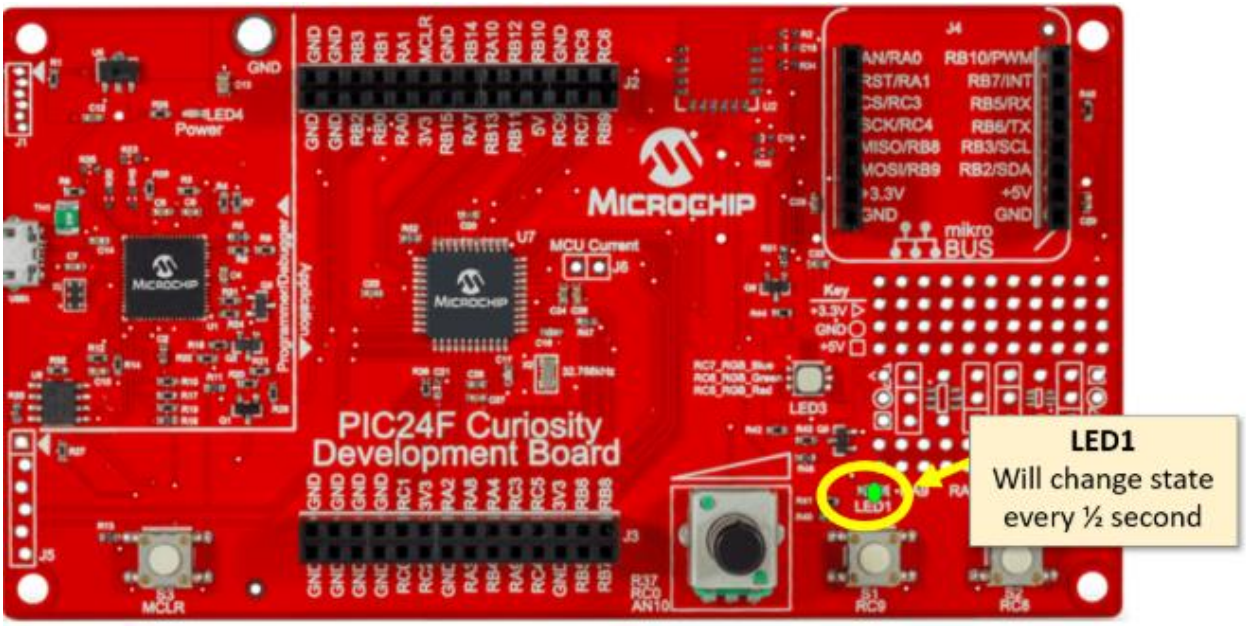




After which we then generated the code and added the give code into the main method of the mcc.h file

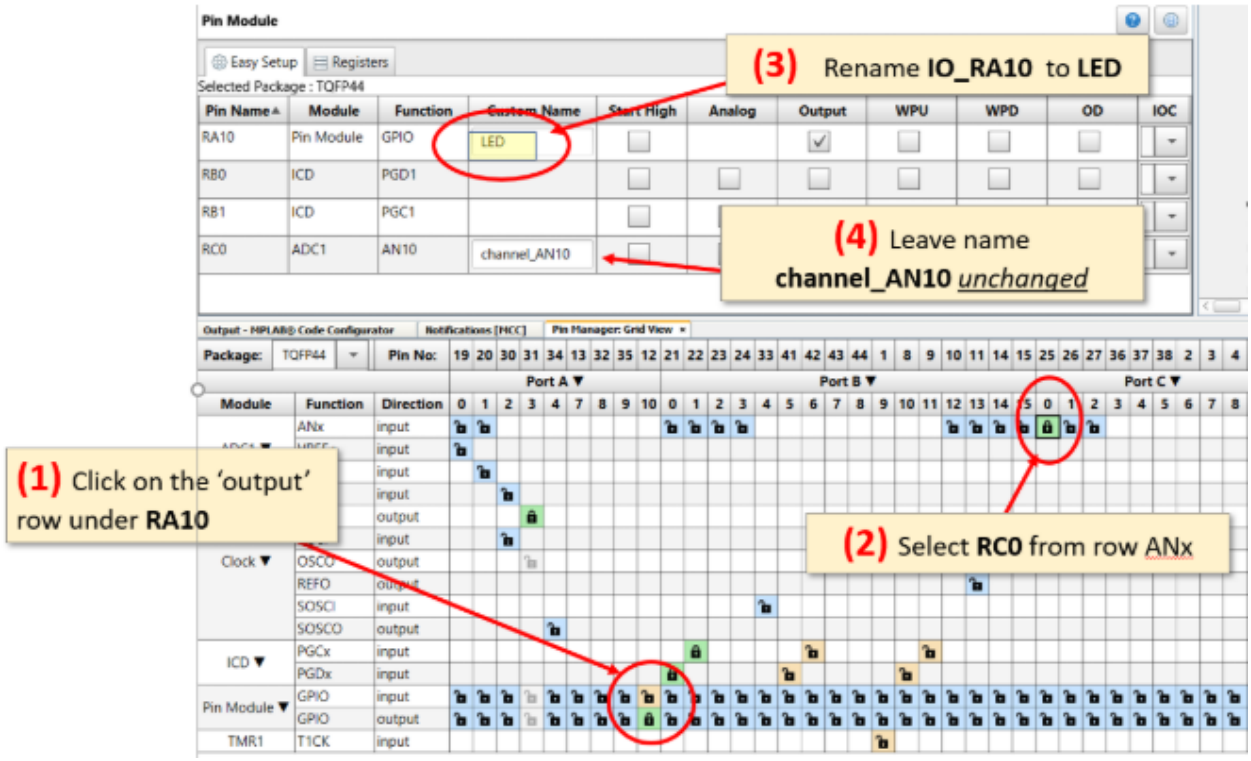
```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/mcc.h"  //##### must be added #####

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

After building and running the code we would get the same result as Part 3 A, a constant blinking led that involves intervals.

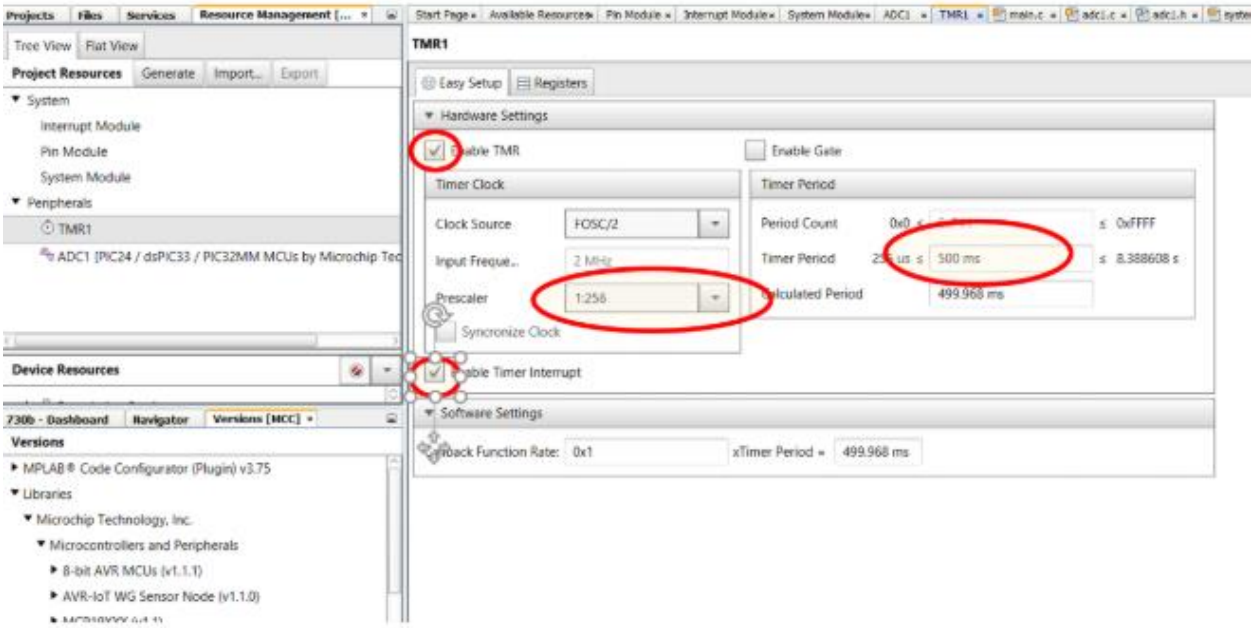


Part 4: PIC24F ADC Input Example

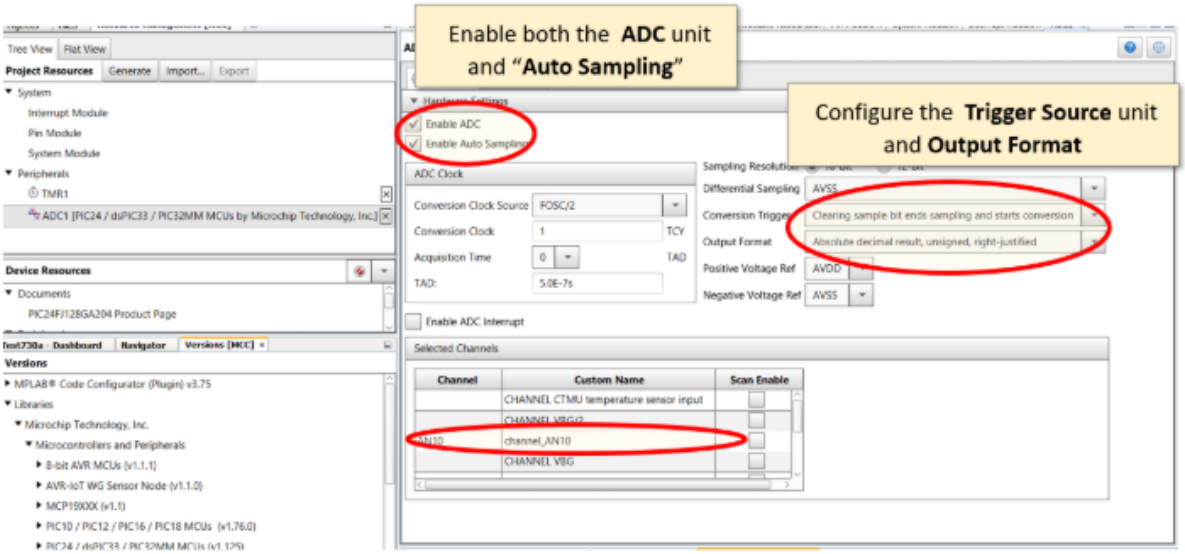
For this part, we would have to do an extra step compared to the other. Mostly everything is similar what we have done before about setting up. We first have made another project again then create the pin module



We then added the timer in.



We then configure the adc unit, adding it and enable it.

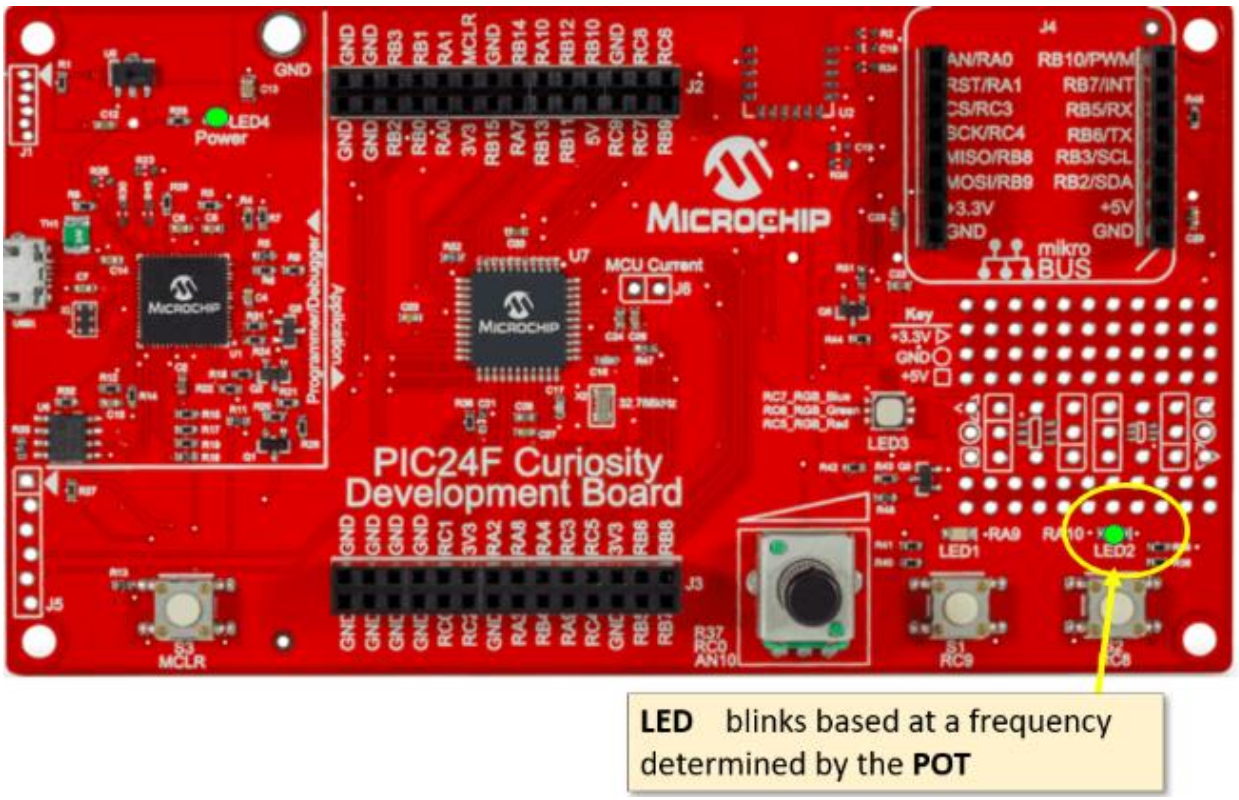


After this we then generate the code and then add the given code to our main method of the cc file.

```
#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));
            // application code goes here
            }
          }
        return 1;
      }
```

After building the program to check if there is any error, we then run the program to see the result.

We should be able to control how fast the led should be blinking based on the POT



Conclusion:

For this lab, it was straight forward and repetitive, but because of this repetitiveness I was able to learn more about the PIC24F board. It has interesting feature the we would be able to mess around with.