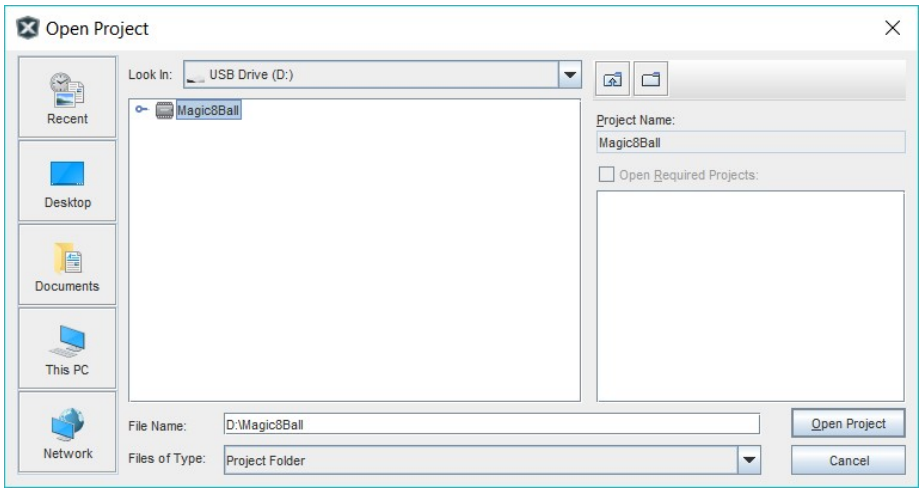
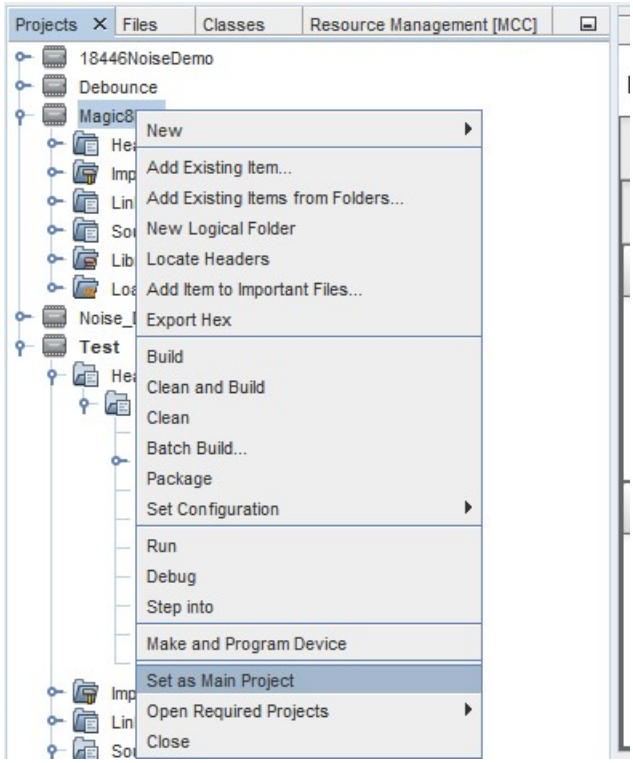
Lab 2: Button Debounce Integration

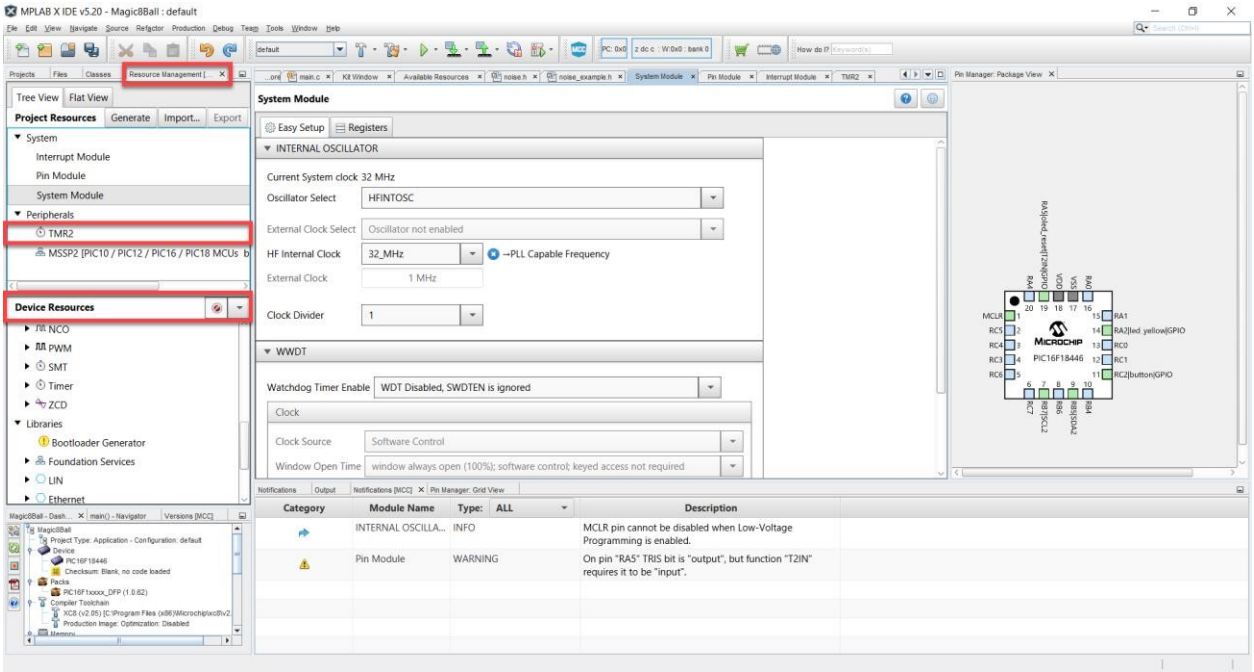
1. **Open Magic 8 Ball Project:**
    a. From File>Open Project



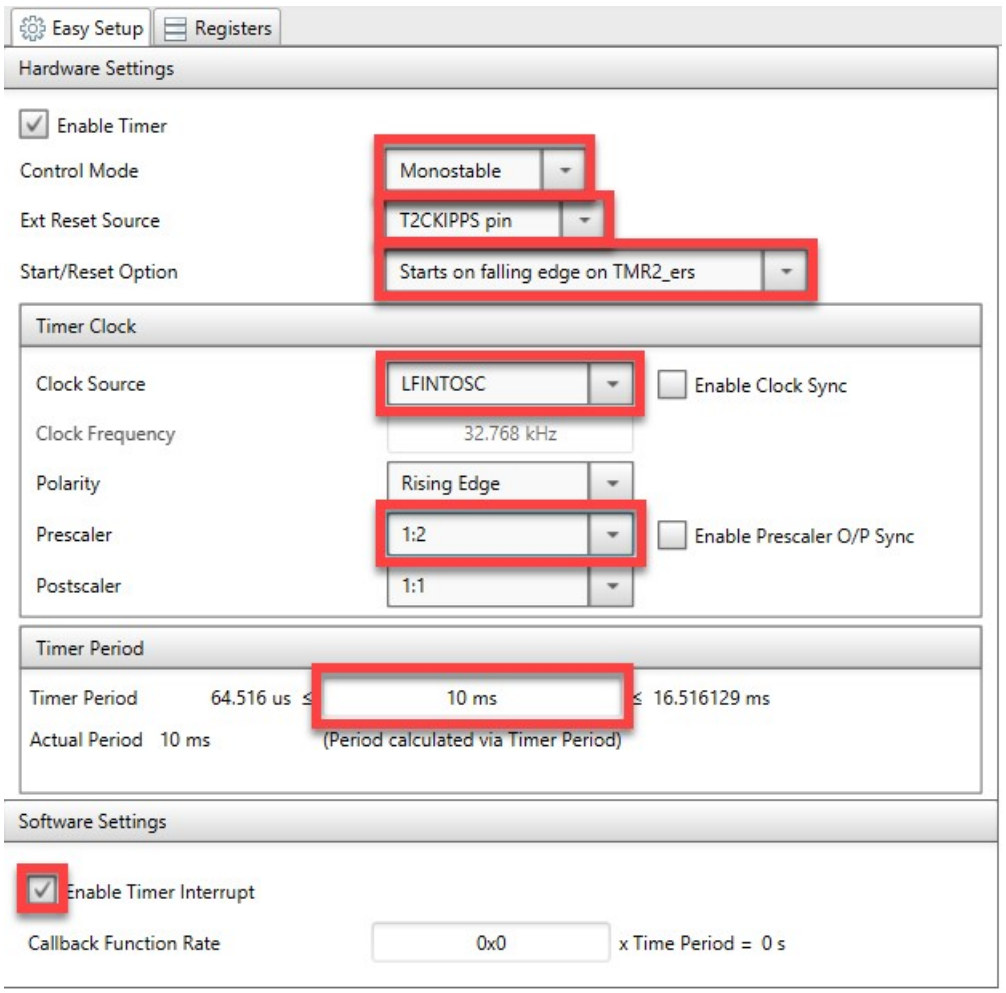
2. **Set Project as Main Project**
    a. In the left hand pane labeled projects, right click on the Magic 8 Ball project and select Set as Main Project



3. **Add a Timer 2 Peripheral**
    a. Click on the Resource Manager tab in the left-hand corner. Navigate to the device resources pane.
    b. In the device resources pane click on **Peripherals**->**Timer**->**TMR2** to add the peripheral to your project



4. **Configure the Timer 2 peripheral**
    - **a.** Set control mode to **Monostable**
    - **b.** Set external reset source to T2CKIPPS (this allows the timer to be triggered by a specified pin)
    - **c.** Set Start/Reset Option to Starts on falling edge (the push button signal goes low when pressed, therefore we would like the timer to begin counting on the falling edge of this signal)
    - **d.** Set Clock Source to LFINTOSC
    - **e.** Set Prescaler to 1:2 (this will allow our timer period to have a range from 64us to 16.4 ms)
    - **f.** Set Timer Period to 10ms (this will cause the timer to generate an interrupt 10ms after the button is pressed, this will allow all other bouncing signals to be ignored)
    - **g.** Lastly enable timer interrupts



5. **Connect Push Button**
   a. According to the PIC18F18446 Curiosity Nano User Guide, the push button is connected to pin RC2. Therefore navigate to the pin manager and tie T2IN to pin RC2

Notifications | Output | Pin Manager: Grid View | Notifications [MCC]

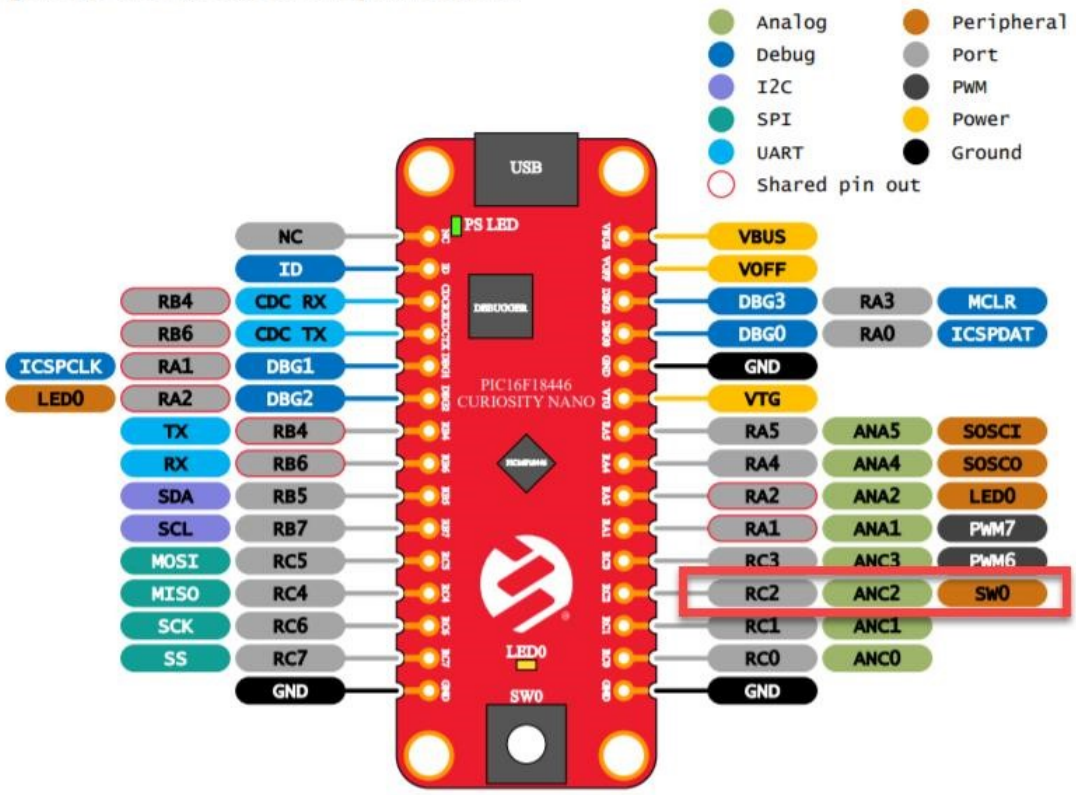
Package:	QFN20	Pin No:	16	15	14	1	20	19	10	9	8	7	13	12	11	4	3	2	5	6
			Port A ▼						Port B ▼				Port C ▼							
Module	**Function**	**Direction**	0	1	2	3	4	5	4	5	6	7	0	1	2	3	4	5	6	7
MSSP2 ▼	SCL2	in/out																		
	SDA2	in/out																		
OSC	CLKOUT	output																		
Pin Module ▼	GPIO	input																		
	GPIO	output																		
RESET	MCLR	input																		
TMR2	T2IN	input																		

4.1.1 PIC16F18446 Curiosity Nano Pinout

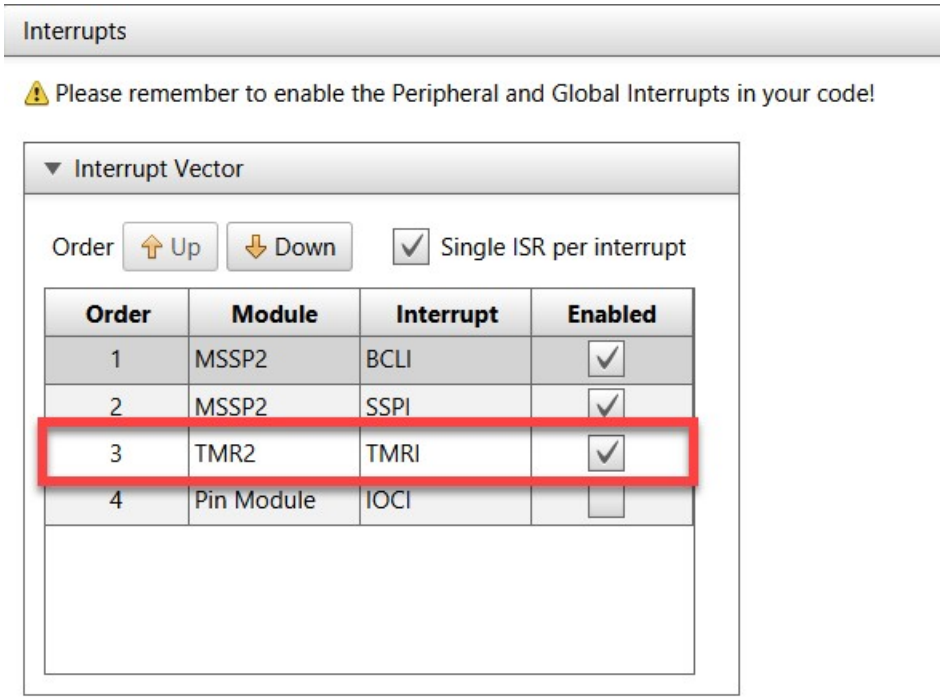
All of the PIC16F18446 I/O pins are accessible at the edge connectors on PIC16F18446 Curiosity Nano. The image below shows the kit pinout.

RA0 and RA3 are only available at the edge connector in the debugger section as long as the cut straps on the bottom are not cut.

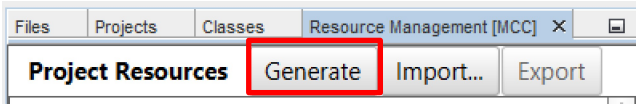
Figure 4-1. PIC16F18446 Curiosity Nano Pinout



6. **Configure Interrupt Module:** In the Project Resources pane click on Interrupt Module
   a. Ensure that TMR2 Interrupt is enabled



7. **Generate Code:** Click on generate code from top of the MCC window



8. **Edit main.c:** Now go back to main.c. Make the following changes:
    a. Inside of the main loop **before the system initialize function call,** make an extern variable named buttonPressed. (This variable will be changed by the ISR of our timer. Therefore it needs to be an extern variable so that it will be visible by our Timer.c file. In this example I am using an integer, however you can easily use a Boolean type variable if you choose to)

```
extern int buttonPressed; // Added

// initialize the device
  SYSTEM_Initialize();

  // When using interrupts, you need to set the Global and Periphe
  // Use the following macros to:

  // Enable the Global Interrupts
  //INTERRUPT_GlobalInterruptEnable();

  // Enable the Peripheral Interrupts
  //INTERRUPT_PeripheralInterruptEnable();
```

    b. Next inside of the while one loop comment out the following lines:
        i. If((button_GetValue()==0))
        ii. { **this is the bracket following the if statement mentioned in line i**
        iii. Button_press_timer++
        iv. If((button_press_timer>BUTTON_ACTION_TIMING)&&(get_button==0))
        v. { **this is the bracket following the if statement mentioned in line iii**

```
121
122     while (1)
123      {
124          // Add your application code
125 //      if((button_GetValue()==0))
126 //          {
127 //          button_press_timer++;
128 //          if((button_press_timer>BUTTON_ACTION_TIMING)&&(get_button==0))          //1000ms
129 //              {
130                 state_machine=state_machine++;
131                 if(state_machine>STATE_MACHINE_MAX)
132                     {
133                     state_machine=STATE_MACHINE_TEMPERATURE;
134                     }
135                 led_yellow_SetLow();
```

c. Remove closing bracket of if statement line 151

```
139                 show_icon_timer=0;              //reset timer
140                 if(state_machine==STATE_MACHINE_TEMPERATURE)
141                     {
142                     show_picutre(PICTURE_INDEX_TEMP_ICON);
143                     }
144                 if(state_machine==STATE_MACHINE_MOTION)
145                     {
146                     show_picutre(PICTURE_INDEX_8BALL_ICON);
147                     }
148                 if(state_machine==STATE_MACHINE_LOGO)
149                     {
150                     show_picutre(logo_index);
151                     }
152                 get_button=1;
153                 //}
154             }
```

d. Check if button has been pressed:

i. Replace the commented out code with a single if statement.

If((buttonPressed == 1)){

This if statement checks to see if the button pressed variable has been changed. This variable will be modified by the Timer 2 interrupt service routine. If it has then the code will continue to the state machine.

```
127 //          button_press_timer++;
128 //          if((button_press_timer>BUTTON_ACTION_TIMING)&&(get_button==0))          //1000ms
129 //              {
130         if((buttonPressed == 1))
131         {
132                 state_machine=state_machine++;
133                 if(state_machine>STATE_MACHINE_MAX)
134                     {
135                     state_machine=STATE_MACHINE_TEMPERATURE;
136                     }
137                 led_yellow_SetLow();
138                 state_machine_timer=0;          //  clear timer when chaged state
139                 show_icon_timer=0;              //reset timer
140                 if(state_machine==STATE_MACHINE_TEMPERATURE)
141                     {
142                     show_picutre(PICTURE_INDEX_TEMP_ICON);
143                     }
144                 if(state_machine==STATE_MACHINE_MOTION)
145                     {
146                     show_picutre(PICTURE_INDEX_8BALL_ICON);
```

e. **Reset buttonPressed variable:** Once the state machine has been executed, we need to reset the buttonPressed variable so that it can be set again once the button is pressed.

```
                    {
                    state_machine=STATE_MACHINE_TEMPERATURE;
                    }
                led_yellow_SetLow();
                state_machine_timer=0;          //  clear timer when chaged state
                show_icon_timer=0;              //reset timer
                if(state_machine==STATE_MACHINE_TEMPERATURE)
                    {
                    show_picutre(PICTURE_INDEX_TEMP_ICON);
                    }
                if(state_machine==STATE_MACHINE_MOTION)
                    {
                    show_picutre(PICTURE_INDEX_8BALL_ICON);
                    }
                if(state_machine==STATE_MACHINE_LOGO)
                    {
                    show_picutre(logo_index);
                    }
                get_button=1;
                buttonPressed = 0;          //Added
                //}
            }
        else
```

9. **Declare and set buttonPressed variable in TMR2.c:**
   a. TMR2.c can be found under Magic8Ball->Source Files-> MCC Generated Files-> TMR2.c
   b. Open this file and declare a variable named buttonPressed and set it equal to 0. (Ensure that it is the same name and data type of the extern variable that you declared in main.c)

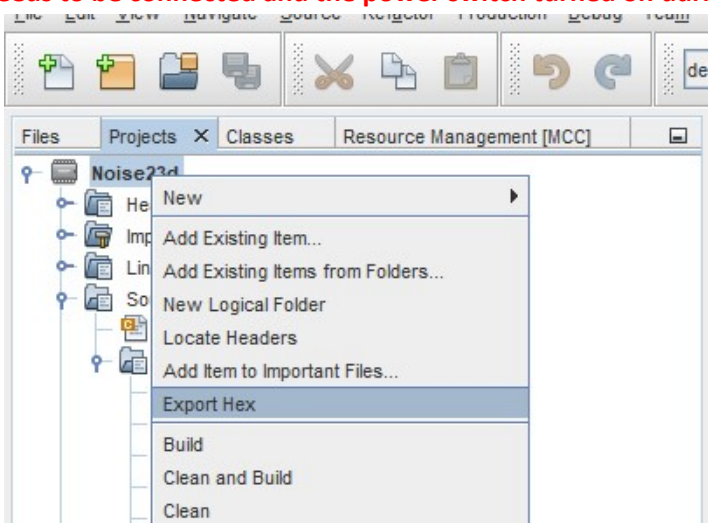
```
 */

#include <xc.h>
#include "tmr2.h"

/**
  Section: Global Variables Definitions
 */
int buttonPressed = 0;

void (*TMR2_InterruptHandler)(void);

/**
  Section: TMR2 APIs
 */

void TMR2_Initialize(void)
```

c. Locate the TMR2_DefaultInterruptHandler function (found at the bottom of the file) and set buttonPressed to 1

```
181 void TMR2_SetInterruptHandler(void (* InterruptHandler)(void)){
182     TMR2_InterruptHandler = InterruptHandler;
183 }
184
185 void TMR2_DefaultInterruptHandler(void){
186     // add your TMR2 interrupt custom code
187     // or set custom function using TMR2_SetInterruptHandler()
188     buttonPressed = 1;
189 }
```

10. **Compile and Program:** Go to the top level of your project and right click. In the dropdown menu you will see **Export Hex**. It will then ask you to provide a file name. Do this and click save.

Note: Your device needs to be connected and the power switch turned on during this step



11. **Finished!:** You should now be able to press your on board push button to change modes. This is the same behavior as the original code. However, we are now able to implement a push button debounce without occupying the CPU's resources. Additionally due to the fact that we are using an interrupt to trigger the start of the switch case, the device has become more responsive.