ECE 121
Lab 0: Hello World

I/O AND NOPS

MEL HO
STUDENT ID #: 1285020
WINTER 2021

January 09, 2021

1 Introduction and Overview

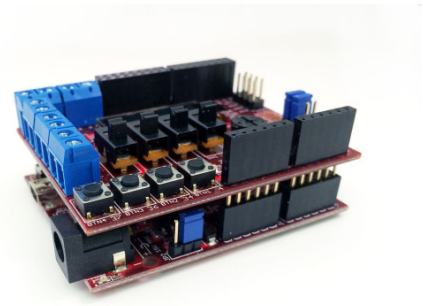


Figure 1: chipKIT Basic I/O shield with the UNO32. Photo provided by the Digilent Inc.

Lab 0 familiarizes students with how to program a microcontroller using MPLAB X. In the first part of the lab, a hello world program is constructed using the chipKit basic I/O shield and the UNO32 MCU. We then build upon that project using *non-blocking* code and NOP delays. The second part introduces exercises where the UNO32 is programmed to interact with external components in a circuit, such as LEDs and buttons. From these projects, we learn how to use a NPN transistor to drive an LED using an external power source, how to make a 5V rail using a TO-220 MOSFET, and how to observe bus contention in our microcontroller. This lab report is an overview of each exercise in Lab0 and the process used to build out each project. For more code examples, see Appendix A.

2 Hello World

2.1 Simple Digital I/O

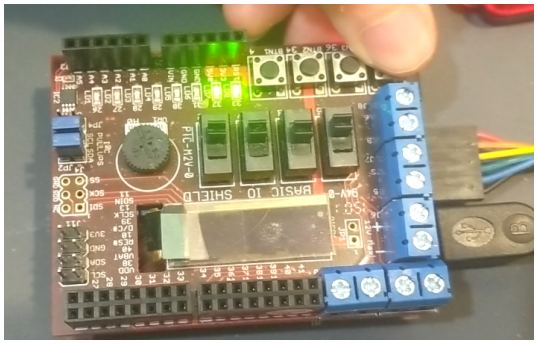


Figure 2: Hello World program for Lab0. When a button is pressed, two corresponding LEDs are lit.

Using CE13's Lab 6 as reference, a program was developed that triggered 8 LEDs using 4 input buttons. Two modular libraries were created to house the functions and macros of the **LEDs** and **buttons** provided on the UNO32 I/O shield. These files are included in the **include** and **src** directories of the **2-1_helloWorld** project.

2.1.1 LEDs

There are 8 LEDs on the basic I/O shield that are used in this program; because the LEDs span the whole set of RE registers, we can manipulate these registers to get the desired configurations for our program. The **TRISE** macro handles the Tri-state RE registers and configure the register to be either an input or an output. **LATE** are the latch RE registers which are normally used to set the output to either *HIGH* or *LOW*. Lastly, the **PORTE** macro is the Port RE registers which are used for reading the current state of the LEDs. There are three functions inside the LEDs module: `LEDS_Init()`, `LEDS_Get()`, and `LEDS_Set()` that we will briefly cover.

- `LED_Init()`: To initialize our LEDS, we set `TRISE = 0x0000` (**OUTPUT** configuration) and `LATE = 0x0000` (**OFF** state).
- `LED_Get()`: This function returns the states of the LEDs by reading **PORTE**.
- `LED_Set(x)`: The LEDs are set all at once using a hexadecimal value fed into the function. Every bit in this hexadecimal value corresponds with one of the

8 LEDs and these pins are ordered with the rightmost bit being *LED1* and the leftmost bit being *LED8*. For example, setting the LEDs using the following: `LEDS_Set(0x00FF)`, will turn every LED **ON** because its binary equivalent is `11111111b`

2.1.2 Buttons

The buttons module consists of `BUTTONS_Init()` and `BUTTONS_Check_Events()`. We use this library in order to interact with the I/O shield's built-in buttons and use them to control our LEDs. The basic I/O shield has four buttons (**BTN1-4**) which are linked to the RF1 register and RD5-7; similar to the LEDs module, we can use bit-wise operations and hex values to quickly set our buttons to the desired configurations.

- `BUTTONS_Init()`: `TRISE = 0x00E0` and `TRISF = 0x0002` (**INPUT** configuration) are set.
- `BUTTONS_Check_Events()`: This function checks to see if a change has happened from the previous button configuration and returns a `int8_t` value that consists of all of the new button states. It has a 4 sample debouncer implemented to prevent false positives from occurring from a noisy button input.

Using these two libraries, the microcontroller was programmed to trigger a set of LEDs depending on the button pressed. For example, button 1 would trigger LED1 and LED2 of the I/O shield, and button 3 would trigger LED5 and LED6. Buttons could be pressed simultaneously to turn on multiple LEDs without any distinct lag.

2.2 NOP

While our hello world program is sufficient, it utilizes a bad practice called, **code blocking**. Code blocking halts the processor from proceeding until a condition has been met, slowing down the program significantly. By adding a NOP (no operation), we can delay the processor for a clock cycle allowing. In this section of the lab, a NOP for loop function is implemented and used to blink an LED every 5ms. To determine the number of iterations needed in our for loop, the initial delay set was slowed so that it could be visibly seen. Then using a stopwatch, the average time it took for the LED to turn on and off was taken and converted into 5 milliseconds. For this experiment, **500000** iterations equated approximately a second. Dividing this number by 1000, we convert our value into milliseconds. Finally, we times our new integer by 5 to produce the desired 5ms. After confirming the period on an oscilloscope, the LED then was set to blink with a delay of a second for demonstration purposes. Please see Appendix A for the source code.

2.3 Improved Hello World

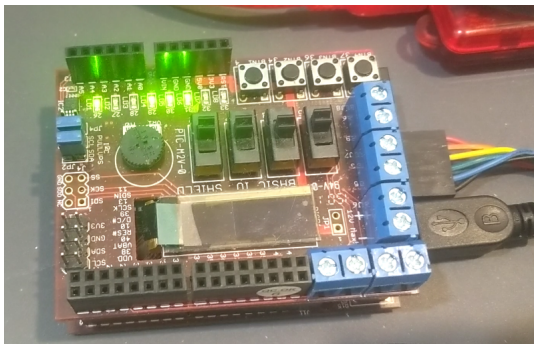


Figure 3: Improved Hello World program.

Combining what we learned from implementing Hello World and NOPs, an improved version of the hello world program was implemented based off of the pseudo-code provided in Lab0.

Algorithm 1: `Improved Hello World Program Flow`

initialize all software and hardware modules ;
initialize loop counter and LEDbits variables ;

while *1* **do**
    write LEDbits to LEDs ;
    increment LEDbits ;
    if *button pressed* **then**
        reset LEDbits ;
    end
    if *LEDbits* > *0xFF* **then**
        reset LEDbits ;
    end
    Delay using NOP loop ;
end

Figure 4: Improved Hello World Pseudo-code provided in Lab0 manual.

Using multiple delays, the LEDs on the UNO32 basic I/O shield increment up in binary until they roll over every 250ms. The counter is reset by pressing any of the four buttons present on the shield and the microcontroller checks for these inputs every 5ms when running its loop. Now that we have become familiar with the basic coding practices using MPLAB X, let's start controlling external peripherals using the UNO32.

3 Digital Inputs/Output (I/O)

3.1 Driving External LEDs

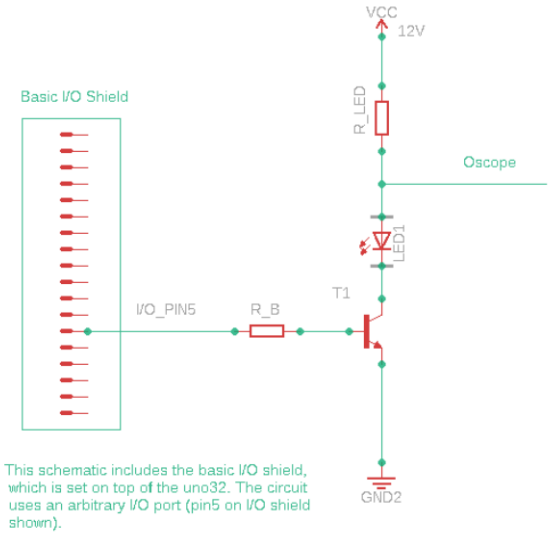


Figure 5: Schematic for a 12V driven LED using an I/O pin with a NPN transistor. In our final circuit, two of these circuits are created to control each LED (red and green).

For the first section of part three, we designed a circuit to drive two different colored external LEDs using two output pins on the UNO32 as shown in Fig.6. In order to control our 12V powered LEDs using our I/O pins on our Uno32, we needed to calculate a **load resistor** and a **base resistor** for our circuit. This can be done using Ohm's law. Since we want to limit our load current to 30mA for our LED, we calculate the following:

$$\boxed{\frac{12V-2V-0.2V}{0.03A} \approx 326.66\Omega} \tag{1}$$

where **12V** is our **power source**, **2V** is the **LED forward voltage**, and **0.2V** is the **collector voltage**. The closest resistor value (330Ω) was chosen as our load resistor. Next, we calculate our base resistor with a current limit of 20mA:

$$\boxed{\frac{3.3V-0.6V}{0.02A} = 135\Omega} \tag{2}$$

where **3.3V** is the **output voltage** of our I/O pin and **0.6V** is our **base voltage**. 150Ω resistor was chosen for the base resistor as it was the closest resistor value available.

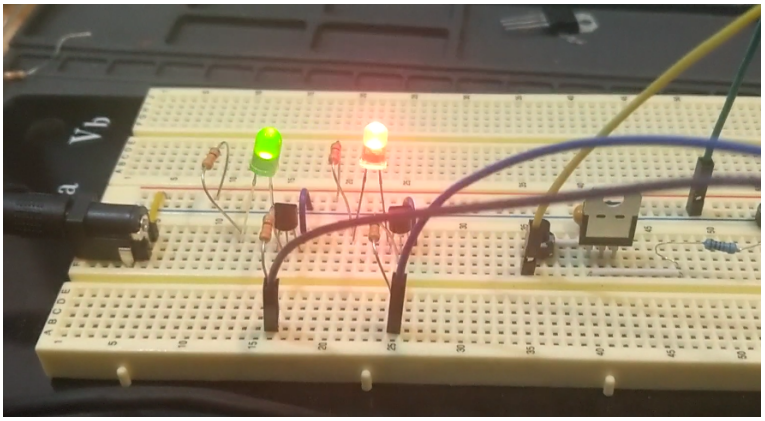


Figure 6: Physical circuit of the External LEDs program.

Pin 5 (green LED) and 6 (red LED) were chosen for this program with the red LED blinking at a frequency of **10Hz** and the green LED blinking at a **5Hz**. A button (pin 9) is also added to this circuit and switches the frequencies of the two LEDs whenever it is pressed down. This button uses a 5V rail created using the circuit shown in Fig.7 as toggle flag for the microcontroller to determine the state of the button.

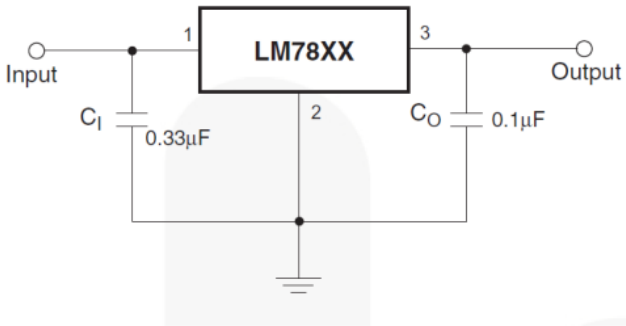
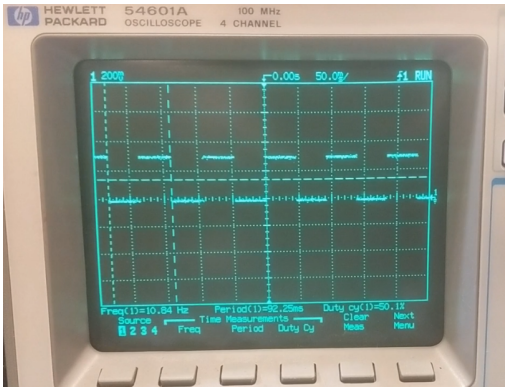
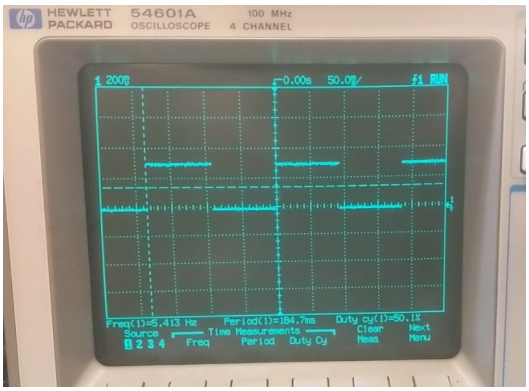


Figure 7: The circuit used to create a 5V rail using the 12V external source in our original circuit diagram. The input of this schematic is attached to the 12V source and the output is connected to the button.

We can see the expected behavior from Fig. 8, which shows two oscilloscope traces of how the LED frequency changes once the button is pressed down. For the source code of this project, see Appendix A.



((a)) The green LED oscillating at 5Hz before the button switch is pressed.



((b)) The green LED oscillating at 10Hz after the button switch is pressed.

Figure 8: Oscilloscope traces of the change between one LED in this external LED circuit once the button is pressed.

3.2 Bus Contention

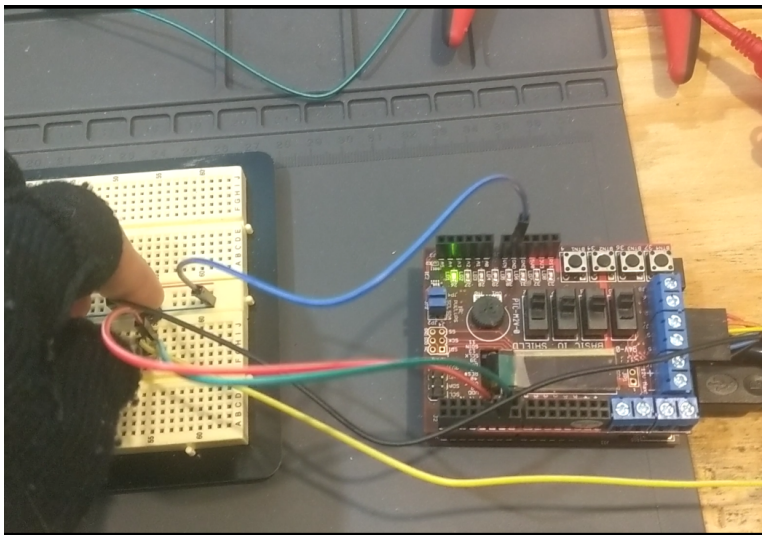


Figure 9: Bus contention circuit. When the button is pressed, an LED is turned on to indicate the bus connection has been lost.

For final part of Lab0, we designed a bus contention circuit with a switch as seen in Fig.10. The purpose of this exercise is to familiarize us with the process of bus arbitration and how to read the difference between the output of the bus and what is read by an input pin; this experiment highlights the issue with having a bus that

attempts to place values on it simultaneously, causing errors.

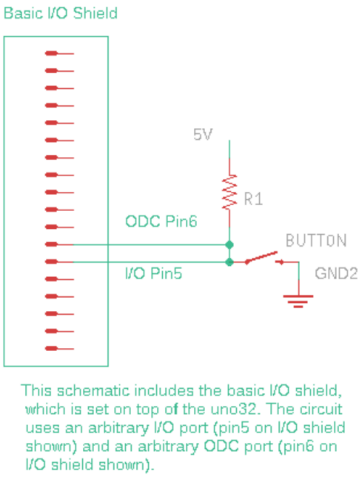
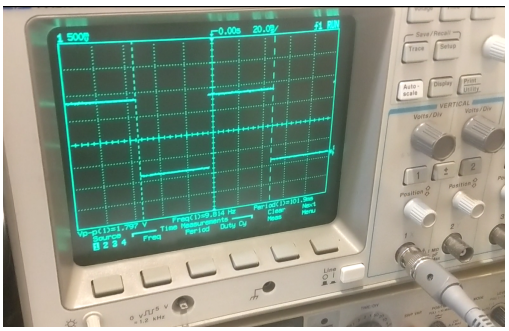
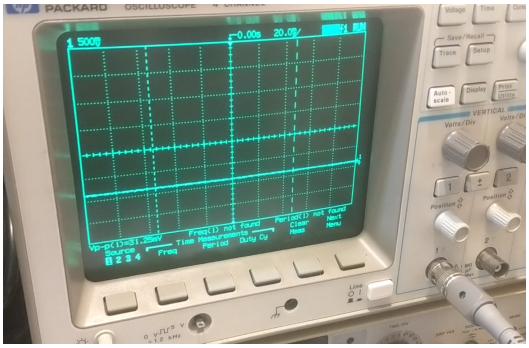


Figure 10: Bus Contention circuit. The resistor value R1 pull-up is arbitrarily chosen as a 10K Ω resistor. This makes our default state to be set to 1. Once the button is pressed, our state changes to 0.

To demonstrate this, we set one of our output pins to be an **open-drain output** using the **ODCx register**. By setting our output pin to open-drain, we can now send out a control signal using an external voltage source. For our purposes, we recycled our 5V rail designed in 3.1 to be the voltage used to power a 10Hz square wave produced by our output. Pin 6 and 5 are re-purposed for this project to be the ODC (open-drain output) pin and an input pin respectively. A program was then designed to check when the bus connection is lost whenever the button is pressed and turns on a LED to indicate this break. From Fig.11, it is clear that the bus connection is lost every time the button is pressed.



((a)) Oscilloscope trace when the bus is still synchronized.



((b)) Oscilloscope trace when the bus connection is lost.

Figure 11: Oscilloscope traces of the change in voltage once the button triggers a bus loss.

4 Conclusion

Lab 0 covers the basics of how to code, compile, and flash the UNO32 using MPLAB X and C. This is done through exercises which us the LEDs and buttons on the basic I/O shield and external components provided by BELS. Through these programs, we learn how to assign and program individual pins from the UNO32 and also concepts such as NOPs, non-blocking coding practices, and bus contention.

Appendix

A. Source Code

```
/*
 * File:   helloworld.c
 * Author: Mel Ho
 *
 * Basic Hello world which turns on LEDS using buttons from the I/O shield.
 * Created on January 5, 2021, 8:09 PM
 */
#include "xc.h"
#include "leds.h"
#include "Buttons.h"
#include "BOARD.h"
#include "stdio.h"
#define TRUE 1
#define LD12 0x0003
#define LD34 0x000C
#define LD56 0x0030
#define LD78 0x00C0
int main(void)
{
    uint8_t ledConfig = 0;
    uint8_t buttonEvents = 0;
    BOARD_Init();
    LEDS_Init();
```

```
    BUTTONS_Init();
    while (TRUE){
        buttonEvents = BUTTONS_Check_Events();
        ledConfig = LEDS_Get();
        if (buttonEvents) {
            if(buttonEvents & BUTTON_EVENT_1DOWN)
                LEDS_Set(ledConfig | LD12);
            else if (buttonEvents & BUTTON_EVENT_1UP)
                LEDS_Set(ledConfig & ~LD12);
            if(buttonEvents & BUTTON_EVENT_2DOWN)
                LEDS_Set(ledConfig | LD34);
            else if (buttonEvents & BUTTON_EVENT_2UP)
                LEDS_Set(ledConfig & ~LD34);
            if(buttonEvents & BUTTON_EVENT_3DOWN)
                LEDS_Set(ledConfig | LD56);
            else if (buttonEvents & BUTTON_EVENT_3UP)
                LEDS_Set(ledConfig & ~LD56);
            if(buttonEvents & BUTTON_EVENT_4DOWN)
                LEDS_Set(ledConfig | LD78);
            else if (buttonEvents & BUTTON_EVENT_4UP)
                LEDS_Set(ledConfig & ~LD78);
    }
}
}
```

```
/*
 * File:   Buttons.c
 * Author: Mel Ho
 *
 * Button Library for UNO32 I/O shield
 * Created on January 08, 2021, 2:08 PM
 */
#include <stdint.h>
#include <xc.h>
#include "Buttons.h"
#include "BOARD.h"
#include "leds.h"
#define DOWN 0x000F
#define UP 0x0010
#define TRUNCATE 0x00E0
uint8_t truncateBits = 0x00E0;
uint8_t down = 0x000F;
uint8_t up = 0x0010;
uint8_t bufferBTN1 = 0x0000;
uint8_t bufferBTN2 = 0x0000;
uint8_t bufferBTN3 = 0x0000;
uint8_t bufferBTN4 = 0x0000;
uint8_t btn1State = BUTTON_EVENT_1UP;
uint8_t btn2State = BUTTON_EVENT_2UP;
uint8_t btn3State = BUTTON_EVENT_3UP;
uint8_t btn4State = BUTTON_EVENT_4UP;
void BUTTONS_Init()
{
    TRISD |= 0x00E0;
    TRISF |= 0x0002;
}
uint8_t BUTTONS_Check_Events()
{
    bufferBTN1 = bufferBTN1 << 1;
    bufferBTN1 |= BTN1;
    bufferBTN1 &= ~TRUNCATE;
    bufferBTN2 = bufferBTN2 << 1;
    bufferBTN2 |= BTN2;
    bufferBTN2 &= ~TRUNCATE;
    bufferBTN3 = bufferBTN3 << 1;
    bufferBTN3 |= BTN3;
    bufferBTN3 &= ~TRUNCATE;
    bufferBTN4 = bufferBTN4 << 1;
    bufferBTN4 |= BTN4;
    bufferBTN4 &= ~TRUNCATE;
    if ((bufferBTN1) == DOWN) {
      btn1State = BUTTON_EVENT_1DOWN;
    }
    if ((bufferBTN1) == UP) {
      btn1State = BUTTON_EVENT_1UP;
    }
    if ((bufferBTN2) == DOWN) {
      btn2State = BUTTON_EVENT_2DOWN;
    }
    if ((bufferBTN2) == UP) {
      btn2State = BUTTON_EVENT_2UP;
    }
```

```
    if ((bufferBTN3) == DOWN) {
      btn3State = BUTTON_EVENT_3DOWN;
    }
    if ((bufferBTN3) == UP) {
      btn3State = BUTTON_EVENT_3UP;
    }
    if ((bufferBTN4) == DOWN) {
      btn4State = BUTTON_EVENT_4DOWN;
    }
    if ((bufferBTN4) == UP) {
      btn4State = BUTTON_EVENT_4UP;
    }
    uint8_t buttonConfig = BUTTON_EVENT_NONE;
    buttonConfig = buttonConfig | btn1State | btn2State | btn3State | btn4State;
    return buttonConfig;
}
```

```
/*
 * File:   leds.c
 * Author: Mel Ho
 *
 * LED Library of UNO32 I/O shield
 * Created on December 19, 2012, 2:08 PM
 */
#include <stdint.h>
#include <xc.h>
#include "Buttons.h"
#include "BOARD.h"
#define OUTPUT 0x0000
#define OFF 0x0000
void LEDS_Init()
{
    TRISE = OUTPUT;
    LATE = OFF;
}
uint8_t LEDS_Get()
{
    uint8_t leds = LATE;
    return leds;
}
void LEDS_Set(uint8_t x)
{
    LATE = x;
}
```

```
/*
 * File:   NOP.c
 * Author: Mel Ho
 *
 * NOP blinks an LED with a delay of approximately a second. This was estimated
 * using a stopwatch and an oscilloscope.
 * Created on January 9, 2021, 6:24 PM
 */
#include "xc.h"
#include "BOARD.h"
#include "leds.h"
#define NOPS_FOR_1S 500000
#define TRUE 1
#define LED1_ON 1
#define LED1_OFF 0
void NOP_delay_1s();
int main(void)
{
    BOARD_Init();
    LEDS_Init();
    while (TRUE)
    {
        NOP_delay_1s();
        LEDS_Set(LED1_ON);
        NOP_delay_1s();
        LEDS_Set(LED1_OFF);
    }
}
void NOP_delay_1s()
{
    int i;
    for(i=0; i < NOPS_FOR_1S;i++)
    {
        asm("nop");
    }
}
```

```
/*
 * File:   improvedHelloWorld.c
 * Author: Mel Ho
 *
 * improvedHelloWorld counts up in binary with leds incrementing
 * until a button is pressed or the leds go over the max iteration.
 * Created on January 9, 2021, 6:41 PM
 */
#include "xc.h"
#include "BOARD.h"
#include "Buttons.h"
#include "leds.h"
#define TRUE 1
#define MAX_ITERATION 0x00FF
#define NOPS_FOR_MS 6000 / 43
uint8_t LEDBits = 0;
int count = 0;
void NOP_delay(int k);
int main(void)
{
    BOARD_Init();
    BUTTONS_Init();
    LEDS_Init();
    TRISDbits.TRISD1 = 0;
    while(TRUE){
        LEDS_Set(LEDBits);
        LEDBits++;
        if (LEDBits > MAX_ITERATION)
            LEDBits = 0;
        NOP_delay(250);
    }
}
void NOP_delay(int k)
{
    int i;
    for(i=0; i < NOPS_FOR_MS * k;i++)
    {
        asm("nop");
        if (i % 500 == 0)
        {
            if (BTN1 | BTN2 | BTN3 | BTN4)
                LEDBits = 0;
        }
    }
}
```

```
/*
 * File:   externalLeds.c
 * Author: Mel Ho
 *
 * externalLEDs drives two external LEDS, one red and one green, with two different frequencies
 * 5Hz and 10Hz.
 * Created on January 11, 2021, 1:36 PM
 */
#include "xc.h"
#include "BOARD.h"
#include "leds.h"
#include "Buttons.h"
#define TRUE 1
#define FALSE 0
#define MAX_ITERATION 0x00FF
#define FIFTY_MS 6000
#define ONEHUNDRED_MS 6000 * 100 / 50
#define RED_LED LATDbits.LATD2
#define GREEN_LED LATDbits.LATD1
#define BUTTON PORTDbits.RD3
int i;
int main(void)
{
    BOARD_Init();
    /* TRISx Initializations */
    ///// LEDs /////
    // RED LED | pin 6
    TRISDbits.TRISD2 = 0;
    // GREEN LED | pin 5
    TRISDbits.TRISD1 = 0;
    ///// Button /////
    // push button | pin 9
    TRISDbits.TRISD3 = 1;
    /* LATx Initializations */
     ///// LEDs /////
    // RED LED | pin 6
```

```
    LATDbits.LATD2 = 0;
    // GREEN LED | pin 5
    LATDbits.LATD1 = 0;
    while(TRUE){
        if ((i % FIFTY_MS == 0) && (BUTTON == TRUE))
        {
            GREEN_LED ^= 1;
        }
        else if ((i % FIFTY_MS == 0) && (BUTTON == FALSE))
        {
            RED_LED ^= 1;
        }
        if ((i % ONEHUNDRED_MS == 0) && (BUTTON == TRUE))
        {
            RED_LED ^= 1;
        }
        else if ((i % ONEHUNDRED_MS == 0) && (BUTTON == FALSE))
        {
            GREEN_LED ^= 1;
        }
        i++;
    }
}
```

```
/*
 * File:   busContention.c
 * Author: Mel Ho
 *
 * busContention generates a 10Hz square wave off of an output pin
 * set to open drain driven by a 5V signal. It then uses another input pin
 * Created on January 11, 2021, 6:50 PM
 */
#include "xc.h"
#include "BOARD.h"
#include "leds.h"
#define TRUE 1
#define LED1_ON 1
#define LED1_OFF 0
#define FIFTY_MS 6000 * 50/30 //For some reason I had to shift delay to maintain 10Hz
// Pin 6
#define ODC_OUTPUT LATDbits.LATD2
// Pin 5
#define INPUT PORTDbits.RD1
int i;
uint8_t buffer = 1;
int main(void) {
    BOARD_Init();
    LEDS_Init();
    ODCDbits.ODCD2 = 1;
    TRISDbits.TRISD2 = 0;
    TRISDbits.TRISD1 = 1;
    LATDbits.LATD2 = 0;
    while(TRUE)
    {
        if (i % FIFTY_MS == 0)
            ODC_OUTPUT ^= 1;
        if (INPUT != ODC_OUTPUT)
                LEDS_Set(LED1_ON);
        else
            LEDS_Set(LED1_OFF);
        i++;
    }
}
```