ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 04 / 22 / 2021

Final Project

**ABSTRACT**:

The final project was designed to have the student demonstrate their knowledge of at least three peripherals. The student was tasked with implementing three different peripherals into one central idea and to demonstrate this in front of the TA.

**INTRODUCTION:**

The goal of the final project is to develop a somewhat functional stop light that has two extra functions. In normal operating mode the stoplight will stay on LED 0 for around 15 seconds and then move to LED 1 for about 10 seconds and finally LED 2 for 10 seconds. Once it reaches LED 2 and 10 seconds elapses we will circle back around to LED 0. We can think that our LED 0 would be a green light, LED 1 would be yellow and finally LED 2 would be red. The two functions that we will be implementing are an emergency button and a Force-Sensitive Resistor which could be used for either a crosswalk or for bikers to pass on the road. When the emergency button is pressed it will flash the LEDs for a few seconds at a specific patter. If the FSR detects something then the stop light will automatically go to the LED 1 (yellow light). To perform our lab, we required a variety of materials including:

* A breadboard
* Wires
* Force-Sensitive Resistor (FSR)
* Push Button
* Resistor (x1 5.1k Ω)
* PIC32MX150F128D
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

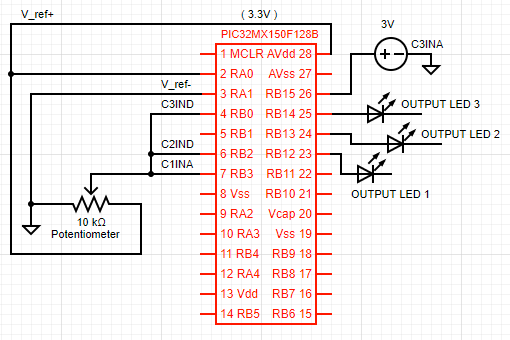
The AD2 will power the PIC32 chip along with partially powering the MPLab Snap Debugger so that it can write to the PIC32. The AD2 will also output a constant voltage of 1.5V for a reference voltage. The majority of the power for the MPLAB Snap Debugger will come from the Micro USB cord.

**EXPERIMENTAL PROCEDURES:**

We must assume that the individual reproducing this lab has already setup their breadboard in a manner that their PIC32 chip can be programmed using the MPLAB X IDE software.

This discussion is very similar to our last lab which was on transferring data from the FSR using analog to digital conversion. However, we were unable to utilize the code for lab 10 since it was having issues with what I believe is the timer. Instead we began with the code from lab 5 and modified it to our needs.

To begin we discuss how the circuit is setup in Figure 1. We first connect a 3.3V supply to A0 as a reference voltage for and then connect A1 to ground for . These are used for our FSR so that we can compare the input voltage from the FSR to the source voltage which we will select as and . Then, we connect our digital LEDs as we’ve done for the entire semester. It doesn’t really matter which pins we use for the LEDs. Pins B0 through B2 were used so that all we had to do was use the LATB command to write to our LEDs. Normally we’d need pull down resistors for LEDs but since we’re using DIO we do not need to worry about that. Next is more setup of our FSR. The FSR is setup just like how we did last lab but with a slight twist. We are interested in comparing the input voltage with another voltage and use the output compare module to find that out. Once the voltage input is over 1.5V we will set the stop light to yellow. We’re connecting the FSR similarly to how we did for lab 3 when we connected the potentiometer, using C1INA and C3INA. Since we only need to check for one voltage we only need C1INA and not C2IND or C3IND. Below is a reference image of what the lab 3 circuit looked like.



As you can see, the potentiometer setup is very similar to our FSR. The next part of our setup is the push button. The way we have it setup in the image would work totally fine but instead of doing it the way shown we used DIO from the AD2 since the push button provided only works sometimes. Once again, since we’re using DIO we do not need the pulldown resistor which normally would be required. The last connection is the reference voltage for our compare registers which is just a 1.5V DC supply from the waveform generator.

For our code is practically explains itself with the comments. We start by including the libraries to make everything much easier on our life’s. Then, we disable the secondary oscillator which enables RA4 and RB4 for use. Similarly, we disable the JTAG module which enables RB7-RB9 & RB11. Even though we do not use all of these pins it’s good practice for this lab to include these lines. The next line is not necessarily required but for good coding practice we will include it. It is a function prototype which prevents compiler errors. Our count integer is going to be a global variable so that all of our interrupts and functions can modify the value without having issues. The first function we created is a delay function which takes one input of an integer. We had developed this delay function during lab 1 and have used it since. With a slight modification we are able to have varying delays at different parts of the program. The next function is an interrupt for the timer module. All it does it increment the global variable count and then check what the value is. If its between 0 and 20 we know it’s a green light, so we output 0b1 which is 0 0 1 in binary. We do something almost identical with both yellow and red light. Th only noticeable difference is the if statement in the red-light section. The if statement checks to see if we’ve gotten to 60 yet which means it’s time to go back to the green light. Assuming the count variable reaches 60 it will reset the count variable back to 0 and then make the light green. The last line of all interrupts is the interrupt clear line. Our next function is the emergency signal interrupt or the push button interrupt. Once the push button is pressed this interrupt (INT2) is triggered which will display 1 0 1 and 0 1 0 back and forth 12 times (number of loops in for statement). Then we return back to normal operation.

The main function starts like many other programs by enabling interrupts and MVEC. Next we start setting up our pins. TRISB will make the B pins either inputs or outputs while TRISA will do the same but for the A pins. Next we use ANSELA and ANSELB to make A and B pins analog enabled. These are just for pins where the voltage flowing through them is an important reading. Next we setup our compare output register using the same settings at lab 3. Next we setup the interrupt for the emergency button and then finally the interrupt for the timer. We clear the timer register and load the period register with 1’s. After everything is enabled we start to see everything work as expected when the stop light starts rotating between signals. The last line of the program is the while loop which checks the input from the FSR; if the FSR is being pressed then we set the light to yellow. Otherwise, we just keep rolling.

**RESULTS and DISCUSSION:**

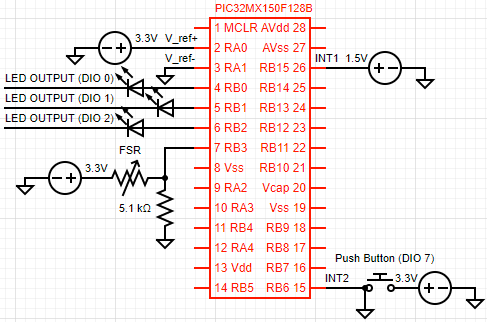
Once the microcontroller was programmed it was apparent that things were working when we observed the stop light change from green to yellow and finally to red. Then, when we pressed our digital emergency button it started flashing the LEDs and then returned back to where it was. Lastly, we applied force to the FSR and the light would automatically move to yellow no matter where it was in the process.

One problem during this experiment was that I could not get the FSR to work along with the timer and other interrupts. I was using code from lab 10 to try and get the FSR to work but I believe something to do with auto sampling was causing issues with the timer. When I would power the chip, it would just stay on whatever LED it started on and all buttons/FSR would do nothing. I changed my plan and went to using the code from lab 3 which worked much better. When I finished implementing the code from lab 3 the emergency button, FSR and timer worked together with no issues.

The final project demonstrated how to research how to implement multiple modules together when you try to make a more complex program. Being able to adapt to what registers you have available or what pins you can use will be important for future jobs. This project helped gain more skills in reading datasheets and troubleshooting.

**CONCLUSION:**

To conclude, the final project truly challenged us to take what we’ve learned this semester and work towards something better. As we started combining code we realized many issues with previous code and more efficient ways to do things. Having the knowledge of what is going on inside of the microcontroller helps a lot with programming it from an overhead view. As with many other labs, this one teaches us how to debug our circuit to check if everything is working. This lab also taught us more complex ways to analyze problems and how to solve them.

**FIGURES AND TABLES:**

**Figure 1: Wiring for final project (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

#include <sys/attribs.h>

#pragma config FSOSCEN = OFF //Disable secondary Oscillator (enable RA4 and RB4)

#pragma config JTAGEN = OFF //Disable JTAG (Enable RB7-RB9 & RB11)

void delay(int x);

int count = 0;

void delay(int x){ // Simple delay function so that we can see what is going on

int i, j;

for(i = 0; i < x; i++)

for(j = 0; j < x; j++);

}

void \_\_ISR(12)Timer(void){

count++;

if (count >= 0 && count < 20) { // GREEN LIGHT

LATB = 0b1; // 0 0 1

delay(200);

}

else if (count >= 20 && count <= 40) { // YELLOW LIGHT

LATB = 0b10; // 0 1 0

delay(200);

}

else if (count > 40 && count < 61) { // RED LIGHT

LATB = 0b100; // 1 0 0

if (count == 60) {

count = 0;

LATB = 0b1; // 0 0 1

} else delay(200);

}

IFS0bits.T3IF = 0; // Clear interrupt flag

}

void \_\_ISR(11)interupt(void){ // INT2 Interrupt (Emergency Signal)

int i;

for (i = 0; i < 11; i++) {

LATB = 0x5; // 1 0 1

delay(250);

LATB = 0x2; // 0 1 0

delay(250);

}

IFS0bits.INT2IF = 0;

}

int main () {

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

// LED Output Pins

TRISB = 0x8048; // INT1 & INT2 Input | LED Output

TRISA = 1; // A0 (V\_ref+) & A1 (V\_ref-) Input

ANSELA = 1; // Analog A pins Enabled

ANSELB = 0x8008; // Analog B15 & B3 Enabled

INT2R = 0b0001; // INT2 is set to B6

CM1CONbits.ON = 1;

CM1CONbits.CPOL = 0;

CM1CONbits.CREF = 0;

CM1CONbits.CCH = 0x3;

// Set the comparator reference voltage (CV\_REF)

CVRCONbits.ON = 1;

CVRCONbits.CVRSS = 1;

CVRCONbits.CVRR = 0;

CVRCONbits.CVR = 0xF;

IPC2bits.INT2IP = 0x7;

INTCONbits.INT2EP = 1;

IEC0bits.INT2IE = 1;

T2CONbits.ON = 0; // Disable Timer2

T2CONbits.TCKPS = 0x000; // Utilize a prescale value of 1:1

T2CONbits.T32 = 1; // 32-Bit Timer

T2CONbits.TCS = 0; // Internal peripheral clock

TMR2 = 0x00; //Clear contents of the timer register

PR2 = 0xFFFF; // Load the Period register with the value 0xFFFF

IEC0bits.T3IE = 1; // Enable Timer3 interrupts

IFS0bits.T3IF = 0; // Clear the Timer3 interrupt status flag

IPC3bits.T3IP = 1; // Polarity level for Timer3

T2CONbits.ON = 1; // Enable Timer2

while(1)

{

if (CMSTATbits.C1OUT == 1) count = 25; // When FSR is pressed set light to yellow

}

}