ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 02 / 18 / 2021

Lab 5

ABSTRACT:

A lab designed to demonstrate how to program the PIC32 microcontroller to handle an interrupt signal coming from a SPDT switch. We will be using built in interrupts to trigger our signal for the code. The purpose of the lab is to replicate lab 1, counting to 15 on LEDs, but implement an interrupt which will stop the count and turn on all LEDs for a short amount of time then return to counting. Similar to the previous labs, this will also enhance skills in reading documentations for new and existing parts.

**INTRODUCTION:**

The goal of lab 5 is to program our microcontroller with the code from lab 1 but with a few modifications. After we implement our modifications to handle the interrupt signal we will power the circuit and test it’s functionality. When the interrupt signal is triggered it’s important that the original count continues where it left off. The result of this code will be displayed on LEDs using WaveForms’ StaticIO. To perform our lab, we required a variety of materials including:

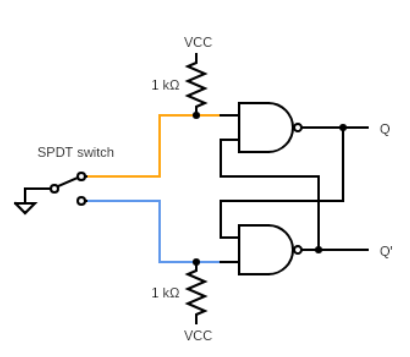
* A breadboard
* Wires
* Resistors (2x 1kΩ)
* SN74LS00 (NAND)
* SPDT Switch
* PIC32MX150F128D
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

The AD2 is going to be a partial power source and our output LEDs. Instead of using actual LEDs we will use the WaveForms software and simulate the LED output. The partial power source will power the PIC32 chip and be a 5V source for the SN74LS00 chip. However, it will not power the MPLAB Snap Debugger. It will be powered off of a 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.

To begin the lab, we will observe the circuit diagram figure 1 to show exactly how we need to wire our breadboard. The simplest part to discuss is the 4 LED outputs, pins 2, 3, 9 and 10. This is the A0-A3 line which will be our binary LED output in WaveForms StaticIO. If we were going to be utilizing real LEDs we would need resistors between the microcontroller output and the LED. Next we will discuss the SN74LS00 chip. This is a NAND chip which is used to help keep the voltage change from one side to the other side as smooth as possible. Since we will be constantly driving both inputs to HIGH we utilize the NAND chip to make our selection. This will also assist us in handing the issue of bouncing. Bouncing is the tendency for the switch to bounce rapidly between HIGH and LOW when first toggled. This is a common issue with switches and can be addressed using either hardware or software. In our diagram we are utilizing hardware by implementing a debouncer which will stabilize our input selection. Below is a better representation of our debouncer.



While this issue is simply fixed with the debouncer we run into yet another issue. The lab kit provided does not contain a SPDT switch. Thus, we had to instead use a button which differs slightly from the SPDT switch. Instead of having a constant OFF or ON position our push button has a momentary OFF or ON position depending on how our wiring is setup. We wanted to utilize a constant LOW and when the button is pressed we go to HIGH. So, we simply set one side of the button to 3.3V and the other to the output which is our pin 16 (B7). Now that we’re using a push button we need to place a pull-down resistor using CNPDBbits.CNPDB7 = 1. Other than that, this practically simplifies our circuit tremendously. We no longer need the SN74LS00 chip, the resistors, the 5V source, or the SPDT switch. Instead, we use the 3.3V power rail and a single wire to get it connected.

Our code is a bit more complex but fortunately a lot of it was harvested from lab 1 or provided in the lab manual. The new header (<sys/attribs.h>) and the two lines INTCONbits.MVEC = 1; and \_\_builtin\_enable\_interrupts(); both setup our built-in interrupts so that we can use them. We are utilizing the external interrupt 0 (INT0) which is one of the 5 interrupts on the PIC32. INT0 is the only one hard-mapped to a particular pin, pin 16.

Next we setup our interrupt function using the \_\_ISR function. The 3 is our vector number that we found in table 7-1 of the datasheet on page 88. The code is very simple; once the interrupt is called we let all A pins to 1 (LED outputs) and then start a 700 delay and then clear the interrupt flag since it doesn’t clear itself.

Next, our delay function is almost the exact same other than implementing an input variable x which allows us to set custom delays for different parts of the code. Thus, we can count from 0 to 15 faster than our delay so that everything looks better.

Our main function starts off with setup of our interrupts. The first three lines of our main are provided. Next, we copied our code from lab 1 and used everything in the while loop.

The IPC0bits.INT0IP = 0x1 was used to set the priority control register for interrupt 0. This value was set to anything other than 0 since it’s the only interrupt being used. INTCONbits.INT0EP = 1 is the interrupt edge trigger direction. This was set to 1 so that we are looking at the rising edge input of the input from our button. Lastly, IEC0bits.INT0IE = 1 was set to enable the interrupt register of interrupt 0.

**RESULTS and DISCUSSION:**

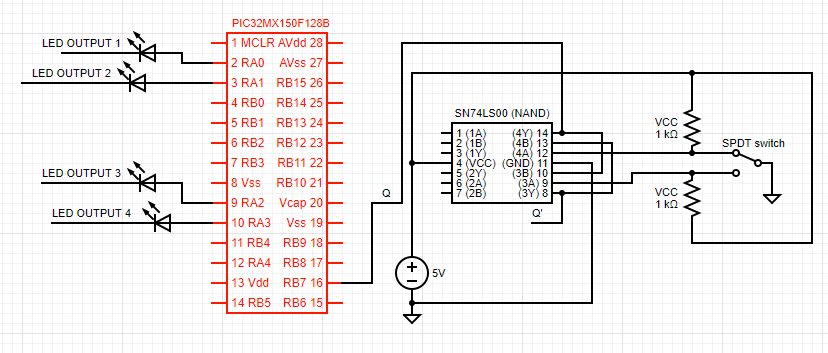
Once the microcontroller was programmed it was clear that things were working when we were watching the LED display count from 0 to 15 in binary just as we did in lab 1. Then, we pressed the button which signaled our interrupt and watched all LEDs light up for a few seconds then the LEDs returned to counting from where they left off.

One problem during this experiment was a code compiling error. For some reason when the code I wrote was compiled in the MPLAB X IDE it would produce errors. However, when you recompiled the code it would magically fix the errors and compile successfully. Thus, the solution to this problem was to recompile to code.

This week we demonstrated how to utilize interrupts and debouncing circuits. This lab is a good learning experience because almost every circuit created with switches has a tendency to bounce rapidly between HIGH and LOW when first toggled. This issue can either be handled using software or by physical hardware. We utilized hardware over software but both methods would work.

**CONCLUSION:**

To conclude, lab 5 taught us many useful techniques on utilizing the interrupts on the PIC32 microcontroller. Another useful piece of information is the idea of handing debouncing circuits. This lab also taught us more complex ways to analyze problems and how to solve them.

**FIGURES AND TABLES:**

**Figure 1: Wiring for lab 5 (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

#include <sys/attribs.h>

void \_\_ISR(3)interupt(void){

LATA = 0xF;

delay(700);

IFS0bits.INT0IF = 0;

}

void delay(int x){

int i, j;

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

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

}

int main () {

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

CFGCONbits.JTAGEN = 0;

int count = 0;

// LED Output Pins

TRISA = 0x0;

// NAND Chip input (Q)

TRISBbits.TRISB7 = 1;

CNPDBbits.CNPDB7 = 1;

IPC0bits.INT0IP = 0x1;

INTCONbits.INT0EP = 1;

IEC0bits.INT0IE = 1;

while(1){

LATA = count; //Output count to B

count++;

if(count > 15)//Restrict count to 0-15, needing only 4 bits

count = 0;

delay(300);

}

}