ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 02 / 03 / 2021

Lab 2

ABSTRACT:

A lab designed to demonstrate programming the PIC32MX150F128D microcontroller using MPLAB X IDE and an NI ELVIS II board to fulfill requirements presented in the lab documentation. The microcontroller should read inputs and provide the expected outputs in our Digilent WaveForms application.

**INTRODUCTION:**

The goal for lab 2 is to program our microcontroller read two digital inputs and output those same values to a latch. When a button is pressed, the latch will be updated to mirror the inputs. The states of the latch’s inputs and outputs will be indicated by the StaticIO LED feature. To perform our lab, we required a variety of materials including:

* A breadboard
* Wires
* Resistor ( x1 1kΩ )
* Push Button
* PIC32MX150F128D
* SN74LS373
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

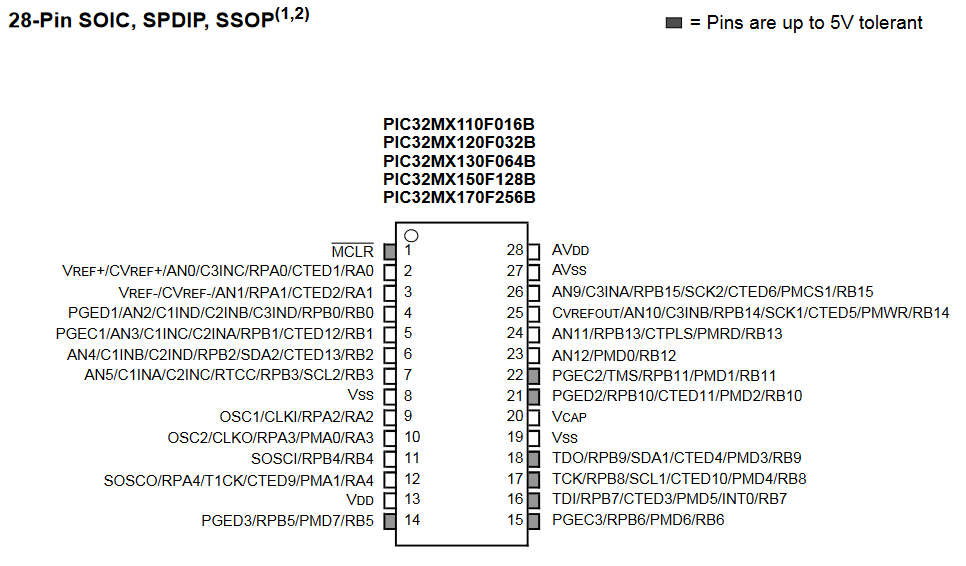
The AD2 is going to be a partial power source and our output LEDs. Instead of using actual LEDs we will use the WaveForms software and simulate the LED output. The partial power source we will power the PIC32 chip and our latch chip, but not the MPLAB Snap Debugger. It will be powered off of a Micro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

As a starting point, we want to seat both our PIC32 chip along with our DM74LS373 chip (latch) on our breadboard. The position is not critical however giving yourself enough room to work is helpful. One of the most important parts of this lab was knowing 5V tolerant pins. The reason we need to know what pins on our PIC32 are 5V tolerant is because the latch requires 5V. Below is an image of the pinout of our PIC32; the gray pins (pin 1, 14, 15, 16, 17, 18, 21, and 22) are the only 5V tolerant pins available. Thus, to complete our circuit we need to ensure that we’re not delivering more than 3.3V to any pins that are not 5V tolerant.

So, the pins we will be using to connect our PIC32 to the latch are 3, 4, and 5. These 3 pins are only digital input/output so the voltage running through them is not important. Pin 25 and 26 on our PIC32 will represent our digital input from WaveForms StaticIO. Then, we will utilize pin 21 for our push button. The push button needs to be a ACTIVE-LOW. Meaning that the microcontroller will read LOW when the button is pressed, and HIGH otherwise. The wiring up of our circuit is shown below; however, it’s worth noting that we will be using an internal pull up resistor for B10.



The input 1 and input 2 values come directly from the output of our AD2 device and will port straight in to pin 25 and 26. Our ‘clock signal’ which is representing our push button will come out of pin A1 and go into the enable pin of our latch (pin 11). Next, the digital inputs that we were referring to are going to be fed as outputs through pins B0 and B1 to pins 13 and 14. Also, we are going to be representing the outputs again in StaticIO using LEDs. As a result, we will tie the output of pins B0 and B1 to digital inputs in our AD2; the inputs you use are not significant.

Pin 20 of our latch chip will be driven high with 5V and in 10 and 1 will be tied to ground. The output of our latch, when the button is pressed, will come from pins 12 and 15 which will feed into our AD2 on whatever input you select.

The software written was very simple. First we need to declare what pins we’re using and whether they’re inputs our outputs. We are using pins B14, B15, and B10 as input pins and A1, B0 and B1 as output pins. Then, we need to declare our pull up resistor on pin B10 for the push button; so we set it to 1.

Next, we need to declare that our pins are digital and not analog, so we set all B ping and pin A1 to 0 using ANSEL.

Lastly is our while loop which is how our push button works. As long as the circuit is powered this while loop will execute taking whatever value is on B15 and set B0 to that value. Similarly, we will take whatever value is on B14 and set B1 to that value. Next, we need to check and see if the push button is pressed. If it is pressed then we will set pin A1 to 1 (enable) otherwise A1 to 0. It’s important to remember that our button is ACTIVE-LOW which means that it’s only ‘active’ when the value is low or 0. We could easily make this an active high circuit with very small adjustments.

**RESULTS and DISCUSSION:**

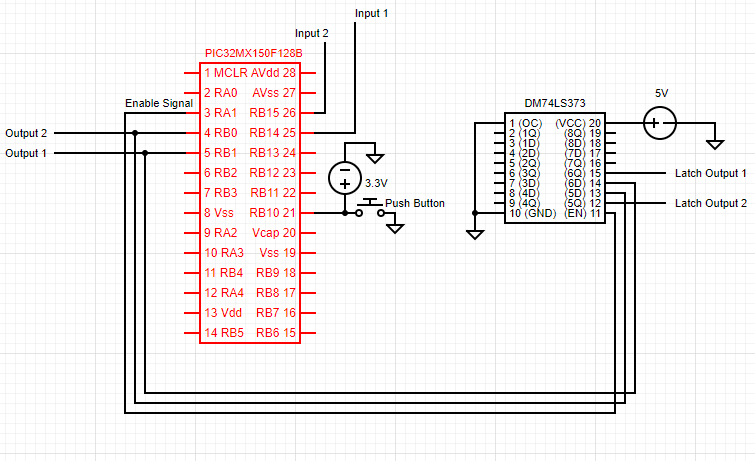
Once the microcontroller was programmed we noticed that the result of whatever was passed in on the input was replicated on our Output 1 and Output 2 line and only replicated onto Latch Output 1 and Latch Output 2 when we pressed the button. This was the exact output that we were expecting to see.

One problem was that a physical resistor was used instead of a pull up internal resistor and the circuit did not provide any results. It took a slight amount of physical measuring with our DMM to determine that we were not receiving the 3.3V we were expecting on pin B10 to keep everything happy. After we removed the resistor and added the internal resistor using CNPUBbits.CNPUB10 = 1 everything started working.

This week we demonstrated how to utilize push buttons, internal resistors (push up), more complex StaticIO and using different voltages for different elements. This lab is a good learning experience because buttons and resistors will be abundant in the real work application.

**CONCLUSION:**

To conclude, lab 2 taught us many useful techniques for utilizing different voltages for different elements, using physical push buttons in our circuit, how to program to interface with those push buttons, and issues to look out for.

**FIGURES AND TABLES:**

**Figure 1: Wiring for lab 2 (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

int main(void){

TRISBbits.TRISB14 = 1;

TRISBbits.TRISB15 = 1;

TRISBbits.TRISB10 = 1;

TRISAbits.TRISA1 = 0;

TRISBbits.TRISB0 = 0;

TRISBbits.TRISB1 = 0;

CNPUBbits.CNPUB10 = 1;

ANSELB = 0;

ANSELAbits.ANSA1 = 0;

while(1){

LATBbits.LATB0 = PORTBbits.RB15;

LATBbits.LATB1 = PORTBbits.RB14;

if (PORTBbits.RB10 == 0){

LATAbits.LATA1 = 1;

}

else {

LATAbits.LATA1 = 0;

}

}

}