Mathias Eriksen
mterikse@ucsc.edu
Lab 06 : Basic IO on a Microcontroller
November 7 2022

Summary:

In summary, this lab consited of implementing various behaviors on the provided microcontroller. The specific behaviors were outlined in the lab doc. Each behavior was centered around excecuting simple input and output using the I/O shield on the microcontroller. The main concepts for this lab were interrupts, event driven programming, analog to digital conversion, as well as more basic concepts such as bit masking, variable scopes, and timers.

Methods:

My approach to this lab was exactly the same as the others, start by reading the lab doc, and build from the bottom up, getting each file perfectly correct before moving on to the next. I was very busy this week, so most of the work was done by myself this weekend. The lab in total took me probably around 10-12 hours of total work.

The very first thing I did was implementing the LED functions file. This was quite simple using the lab doc since it stated how to initiate the LEDs using TRISE, and set their value using the 8 bit value of LATE.

Next, it was pretty easy for me to implement the timer functionality using the frequency of the peripheral clock to count the seconds, and I had no real problems with this file since it simply consisted of raising events at set intervals and handling them with simple IF statements. Most important thing here was to use no magic numbers by defining relevant variables with meaningful names in the beginning of the file.

In order to implement the bounce_switch file, it was a little more complex since it needed a movement variable that controlled the LEDS, and we had to handle the cases where the LED is moving and meets the end of the row. In this case I just had to switch the movement variable to the other direction. All of the things handling the movement was quite easy for me since it drew on knowledge I already had from CSE 100.

Within the adc reading file for this lab, I encountered more trouble than the previous files. The most difficult part of this file was using the OLED commands that were provided in order to print items to the screen. This was difficult becasue on string had to be correctly formatted and loaded into the OledDrawString() function. I ended up solving this issue by using the sprintf() function to load the string into a buffer that could be printed using the function. The other difficult part of this file was handling the edge cases, where the max and minimum readings need to reach 100 and 0 percent smoothly. This was handled simply by including an if statement which adjusts the stored value when the read value is at the potentiometers maxima or minima.

The hardest file for me to implement was definitely the Buttons.c file. This was because configuring the if statements with each boolean expression took some time to wrap my head around. It also took some thinking to determine what the result value should be, since the state value from the buttons is a 4 bit number, and the return value must be an 8 bit number which can handle two cases at once. I was stuck for quite a while debugging, and my issue was just that nothing was happening when I pressed the buttons. My issue ended up being that my variable which held the stored previous state of the buttons was not static, and thus it was being reset in every function call. Once I realized this issue and made the scope of the variable fit its application, everything worked well. It was pretty simple to implement the functions from Buttons.c into the bounce_buttons file using some simple if statements and LED toggles.

Finally, I enjoyed doing the extra credit the most since it tied together all of the files into a nice end product. It was simple to implement as well, since all of the functions were already written. In order to use the potentiometer to change the LED bounce speed, you simply had to make the stored percentage of the potentiometer from the adc function the countdown variable for the switch timer. Then, to make the button presses stop/start the LED bounce, I simple created a togglable enable bit using a bitwise XOR on a single bit. This enable bit would simply be included as a condition for the bitwise shift which was responsible for moving the LEDS. Finally, in order to make the speed increase as the ADC voltage increased, I had to use the expression (100 - potentiometer percentage) since a greater percentage would have lead to a greater countdown time and thus a slower bounce. This matches the lab doc asking for a clockwise movement to increase the speed.

Results:

I enjoyed this lab the most of all of the labs that I have done so far in this class, since there was a real world application in the microcontroller programming. And, since I am taking ECE 121 next quarter, I applied myself a lot to this lab since I will have to know a lot about how to program microcontrollers for that class. My favorite part was the extra credit, where everything came together, and if I were to do the lab again, I would probably make sure to read the BOARD header file before I started coding, since a lot of the information needed was in that file.

Feedback:

The biggest issue I had with this lab was that the extra credit example video provided does not match the behavior outlined in the lab manual whatsoever. I'm not sure if the extra credit file has been altered and the video has just not been updated, but it was quite frustrating to not be able to check if my behavior matched the expected behavior correctly. The button presses in the video demo do not freeze and unfreeze the LEDs, it just overlays the toggled LEDS from bounce_buttons with the bouncing LEDs.