CMPE 12 Lab Report # 6

John Allard
Lab Section #2

December 12th, 2014

1 Overview

Laboratory #6 had us once again working with the PIC32 Microcontrollers and their IO shields. This time, instead varying the timing between leds on the IO shield, we were instructed to turn our microcontroller into a timing device. This device makes use of two of the built in switches on the shield for input, as well as the Oled output display to display the current time count to the user. In order to make this work, we had to design our own interrupt routine based on a peripheral timer that is built into the microcontroller. We also had to design other routines to keep track of the current time and enable/disable the interrupts. Reading from the switches was almost identical to one of the tasks from lab #5. I decided to design most of my program logic in C, because I am more familiar with it than MIPS. This means I have kept the assembly language to the minimum amount required (implementing the 4 required functions). I use flags in the main `while` loop to maintain knowledge of the current timer state (running, stopped, reset). All of this functionality is discussed further in the following sections.

2 Main Program

For the main program I just built off of the template that was provided to us. I turned the while loop into a finite-state-machine like object that changed what occured based on the value of the switches on the IO shield. If the switches were set to timer-on, I would check to see if a 1000 milliseconds has passed, if so, I create the new time string, clear the output display, and write the time string. It was difficult to ensure that the display didn't flicker, I had to make sure that the clear and write routines were adjacent to eachother at each separate call place. To make this simpler I introduced a new function `printTime` that takes the current minutes and seconds of the timer and does all of the necessary printing. On top of this I use a series of flags to keep track of the previous state (reset, stopped, running). This allows me to achieve the required fucntionality.

3 Timer Setup

I began my coding by writing the `T1Setup` function. This function is responsible for setting up the interrupts based on a specific timing interval. I begin by saving all of the save registers as usual, then I begin configuring the different interrupt flags as designated in the lab instruction sheet. This includes clearing the `T1CON` register, setting the interval with which I want the timer to count to before sending an interrupt signal, setting the interrupt enable flag, clearning the interrupt-set flag, and finally enabling the timer. To perform these operations I work with the SET and CLR offset registers, which allows me to avoid loading, masking, and saving to set individual bits. I calculated the timer routine by adjusting both the PR1 value and the value that I used to increment the `milliseconds` variable until it was within $\pm 3\%$ of the value of the timer on my phone.

4 Interrupt Routine

I tried to keep the interrupt service routine as short as possible, otherwise my timing function would grow increasingly more innacurate as time went on. To do this, I simply increment the value of the `milliseconds` variable, reset `TMR1` value to zero, then reanable interrupts by setting the appropriate bit in the `IFS0` register. After this, a simple call to `ERET` and my interrupt was finished. I of course had to also save all registers to the stack in the interrupt routine, this is because the interrupt can be called at any time, which means I could unexpectedly clobber a user's variables and they would be none the wiser.

5 Timer Control

The timer is controlled by two simple functions, `T1Start` and `T1Stop`. These simply enable and disbale interrupts, respectively, by setting the appropriate bits in the `IEC0` register and by stopped/starting the actual timer. These functions are pretty bare because most of the logic of the timer is contained inside of the `main.c` file.

6 Conclusion

This lab was a good way to apply all of the knowledge that we have gained in this class into an useful computing device. Seeing how I can combine the C programming language and assembly to create a neat timing device has made me appreciate assembly more than I used to, and seeing all of your code come together to make something that works is a fun experience. I did experience some bugs throughout the process of completing this lab though, they are listed below.

At one point in the developement, my timer started taking more and more time with each increasing second. I looked for bugs for well over an hour and ended up just resetting my code base to an earlier version and the bug never showed up again. Thank goodness for version control systems! I did also have quite a bit of trouble getting all of the correct bits set in the setup routine. After much trial and error, I was able to get it right and just kept with it, but it definitely showed me how much studying I still have to do before this final. Lastly, I had some trouble getting the display to not flicker. I have improved this significantly but it will still flicker one every 20-30 seconds, and I have not found a way to change this. My clear and putString functions are adjacent to one another, so I don't know where to go from here.