SENG 5831 – Lab Assignment #1

For the purposes of this lab, I will do my best to consider toggling at 1 Hz (or any frequency) to mean that the LED will turn on and off once per second, meaning the LED will be on for 500ms, and then off for 500ms.

# Question #1

For the tuning of my 10ms for loop, I first ran an empty for loop many time to determine how long it took the average empty for loop to execute. Once I had that value, I was able to plug that into the #define section and run the loop. I ran the #define loop and number of times, each time printing out the number of milliseconds the loop took. I was able to then modify my \_\_ii value to get my loop to run in an average of 9,999 microseconds, or 9.999 milliseconds. The value I ended up with for \_\_ii is 5563. This is a close to 10ms as I was able to get. Using a value of 5562 gave me an average loop time of 9,997 microseconds, and a value of 5564 gave me an average loop value of 10,003 microseconds.

Using a value of 5563, my LEDs blinked 31 times over 30 seconds. I believe this was a little bit off because in my original evaluation, I had to calculate the amount of time the loop was being run, and then output that value to the screen. It was impossible to do the analysis without counting those functions as part of the time. In the loop for this problem, there is no printing to the screen, or really anything else of consequence going on. We are setting the LEDs so that will also take some time, but we are always going to be doing something, so it perhaps will be impossible to truly get the correct timing of that loop through experimentation. However, we’ll give it our best shot.

To determine the best value for this loop, I’ll simply increase the value until we get to the closest value of 30 blinks over 30 seconds. To do this, I’ll start the timer when the LEDs turn on, and try and get as close to the LED blinking the 31st time at exactly 30 seconds. That should be 30 blinks in 30 seconds.

When setting \_\_ii to 5564, we were still getting 31 LED blinks comfortably. A value of 5565 was still fast enough to get us 31 LED blinks. Fast forward a bit, it looks like the original estimate was pretty far off. I finally started getting pretty close with a \_\_ii value of 5700. The LED blinking was staying pretty close to the seconds ticking on my timer, at least as best I could tell when looking at them.

As we can see, the value can be affected by the simplest of things, like printing to the LCD. If we were to allow menu navigation, those would all be things that would affect the actual timing of our loop. As a matter of fact, the fact this problem wanted us to toggle all three LEDs in this loop probably has a greater affect than just toggling one LED because we are doing three times the work outside the loop, thus taking three times as long for non-loop related tasks.

# Question #2

Using the software timers, I sent over a command of “ta1000.” This will set all LEDs to toggle with a period of 1 Hz. I entered the “za” command, and hit enter on the console at the “exact” same time as starting my stop watch. I think did my best to hit enter on a “pa” command when the timer hit 30 seconds. The values on the output were green at 60, yellow at 60, and, as mentioned in the problem, red at 59. It looks like my timers and menu are set up mostly correct. On with the rest of the lab!

# Question #3

OK, so here’s where this homework got tricky. When setting the frequency to 2 Hz, the yellow light is unable to blink at that speed because it is a 100ms timer. Therefore, when running this test without any busy-wait for-loop, the LED counts were 120 for the green, 150 for the yellow, and 119 for the red. This can be explained by the fact that 500 / 2 is going to be 250, but because the yellow LED interrupt is at 100ms, the closest we could get is 200ms, which would put us at 2.5 Hz, which would result in 150 counts per half minute.

On to the 90ms busy-wait for-loop. With the 90ms loop in both the PWM compare interrupt and the PWM overflow interrupt, the results of the counters are green at 240, yellow at 300, and red at 153. Now, moving the 90ms busy-wait into the 16-bit timer ISR resulted in counters of green at 240, yellow at 300, and red at an abysmal 26 toggles.

The poor red LED takes the most abuse here. The red LED interrupt is being fired at a frequency of 1 kHz, or once every 1ms. This means that it is going to be the most susceptible to delays in the code. When in the green interrupt, the delay is only getting called on match and overflows, meaning it is about a 90ms delay 4 times a second. This doesn’t affect the yellow interrupt because it is being fired at 100ms, meaning even if a yellow interrupt first during the 90ms, there is still plenty of time when the 90ms is done to handle the interrupt. The red interrupt on the other hand, is being missed about 89 times during each 90ms delay. The flag for each interrupt can only be set and cannot account for multiple interrupt firings during the delays. That means, the interrupt fired 90 times during the 90ms delay, but the system only sees it as a single interrupt. Therefore, we could guess that in a single second, approximately 356 interrupts are being missed.

Now, when the delay is put into the yellow interrupt handler, we are running the delay much more than when in the green interrupt handler. Again, the green and yellow interrupts are being triggered at 250 and 100ms respectively, so with the 90ms delay, we can see that they aren’t being affected. The red interrupt however, is being crippled in this scenario. Instead of running the 90ms delay only four times per second, we are now running it a whopping 10 times per second. This means that the 89 interrupt misses are now also happening 10 times per second instead of four, meaning we are missing 890 red interrupts per second. This explains why the red LED slows down with the delay in the green interrupt handler, and slows to a crawl with the delay in the yellow interrupt handler.

# Question #4

When the 110ms delay was in the green ISR handlers, the results were very similar to when the 90ms delay was in the green ISR handler. The toggle counts were green at 240, yellow at 300, and the red at 134. Based on the explanation above, you might think the yellow LED should be affected. This can be explained by the fact that the yellow ISR handler has priority over the green ISR handler. Both interrupts will fire at 0ms. Because yellow gets higher priority, it will get handled. Then the green will get handled, and while the delay is happening, a single yellow ISR will fire. That gets handled when the green ISR is done. Next, the yellow will fire at 200ms and the green will fire at 250ms. No problem there because the yellow fired at 300ms will get handled at 360ms when the green finishes up. This process will start over at 500ms where the yellow ISR will once again get precedence over the green ISR. The red is again missing many ISR handlers while the delay is happening, and its toggle count is affected accordingly.

Now, when the delay is put into the yellow ISR handler, we’ve got a different story. As stated in the problem, my menu didn’t work. I was able to get some counts by hand. The green LED blinked the required 120 times, the yellow 112 times, and the red didn’t blink once. The green is unaffected because its ISR is still only firing every 250ms. The 110ms is not enough time to cover up more than a single ISR, which will get handled when the delay is finished. The yellow slowing down can be explained by the fact that the ISR can now only get triggered a maximum of once every 110ms, meaning one in every 11 triggers will be missed, and our counting will slow down accordingly. The red LED doesn’t blink at all. This, and the menu not working can be explained for the same reason. Both of those events are handled in the cyclic executive. This has the lowest priority, and because we are constantly executing the green, yellow, and red ISR handlers, the main cyclic executive never gets processor time, meaning the released red LED task, and the handling of the input never happens.

# Question #5

With the 510ms delay in the green ISR handlers, the menu once again didn’t work. I then had to count LED toggles by hand. The green LED blinked 120 times, meaning it toggled about 240 times in a minute. The other two LEDs didn’t blink at all. The green LED wasn’t affected at all because the interrupts are in two different handlers. This means that even if a full green period happens, both interrupts would get handled, because there are two separate interrupts, one for compare match, and one for overflow. The yellow LED would come to a halt because there would be no extra time to handle the yellow ISR handler. The red one, of course, can get no time on the cyclic executive to ever toggle.

With the delay in the yellow ISR handler, the green LED toggled 240 times, and the yellow LED toggled about 22 times. Once again, in this case, the menu didn’t work. I think the results here can be explained by the fact that the green handlers are going quickly enough where there is time for the yellow ISR handler to get serviced. The yellow LED slowed down considerably because during the 510ms, we are missing a lot of yellow interrupts, meaning our LED should slow down considerably, but at least still get serviced. The cyclic executive is once again not getting serviced, meaning we are missing out on red LED toggling and the menu.

# Question #6

With the delay in the green ISR handler, and an sei() call at the top, the green and yellow LED toggled each 240 times, pretty much alternating. When the green LED was on, the red LED was off, and vice versa. Once again, the red LED and menu were not functional. I believe the results here can be explained once again by the yellow ISR having a higher priority over the green. It is able to get serviced because it gets triggered through the sei() call, and the system then handles that interrupt. The menu and red LED are again not handled because the cyclic executive is not allowed any CPU time.

With the delay and sei() call in the yellow interrupt handler, the green interrupt fired 240 times in one minute. For the first three or so minutes, the yellow LED did not flash. After the three minutes, the LED began to flash. Then, after flashing for a minute or so, it would go dormant again and repeat this cycle. This is the most interesting of all cases. It makes sense that the yellow LED would blink less because as the higher priority, the delay is in its loop, meaning it is going to miss a bunch no matter what. Now, I’m not totally sure what’s happening when it fires up and all of a sudden starts blinking. It appears at that time to be blinking at about the same rate as the green LED of 240 toggles per minute, except that the yellow LED did seem a bit more erratic in its blinking. The only thing I can think is that at some point, the interrupts line up just right, for a period of time, which allows the yellow interrupt to fire more frequently.