Jesse Estum
SENG5831
Lab Assignment 1
3/1/2015

Source Code
https://github.com/estu0002/EmbeddedSystemsHomework/tree/master/project3

Experiment 1
Results from serial terminal:

```
ta500
Toggle Complete
za
Zero Complete
pa
R:63 Y:62 G:61
```

Observations & Explanations:
The WCET for the loop appears to be just about right. In a perfect world, the output for the 30 second period would have put the red counter value at exactly 60. However, in order to achieve this I would have to instantaneously issue the 'za' command and start my stopwatch. I would also have to instantaneously issue the 'pa' command when my stopwatch gets to exactly 30 seconds.

Taking into consideration the human latency, I did not make any adjustments to my loop which is responsible for waiting 10ms on the red LED.

The reason for not sending any data over the serial terminal is that serial I/O on the microcontroller has a cost. This cost is in CPU cycles spent on handling the serial I/O and command processing. For this particular experiment, we are relying on CPU cycles being available to our WAIT_10MS macro in order to get the correct timing. As such, using CPU for serial I/O (or any other processing, for that matter) would throw off timing.

Experiment 2
Results from serial terminal:

```
ta500
Toggle Complete
za
Zero Complete
pa
R:59 Y:62 G:61
```

Observations & Explanations:
This output seems about right to me. The red LED is a little low because it's actually on a timer that's blinking it a little slow. With a prescaler value of 1024 the output compare would need to be at value of 625/32 = 19.53125 to run at exactly 1ms resolution. Since we're working with integers, my output compare register is set to 19. This generates almost 1028 interrupts per second, for a resolution of approximately 1.028ms.

The other deviation can be explained by the fact that all of the LED blinks are not synchronized.

Experiment 3
(This step referenced a singular ISR for the green LED. I have two ISRs tied to the green LED, so I picked just the ISR tied to TIMER1_COMPA_vect for purposes of this step.)

(A) Results from serial terminal – 60 seconds (no 90ms busy-wait anywhere):
```
ta250
Toggle Complete
za
Zero Complete
pa
R:236 Y:303 G:242
```

(B) Results from serial terminal – 60 seconds (90ms busy-wait in green LED ISR):
```
ta250
Toggle Complete
za
Zero Complete
pa
R:195 Y:302 G:241
```

(C) Results from serial terminal – 60 seconds (90ms busy-wait in yellow LED ISR):
```
ta250
Toggle Complete
za
Zero Complete
pa
R:31 Y:303 G:242
```

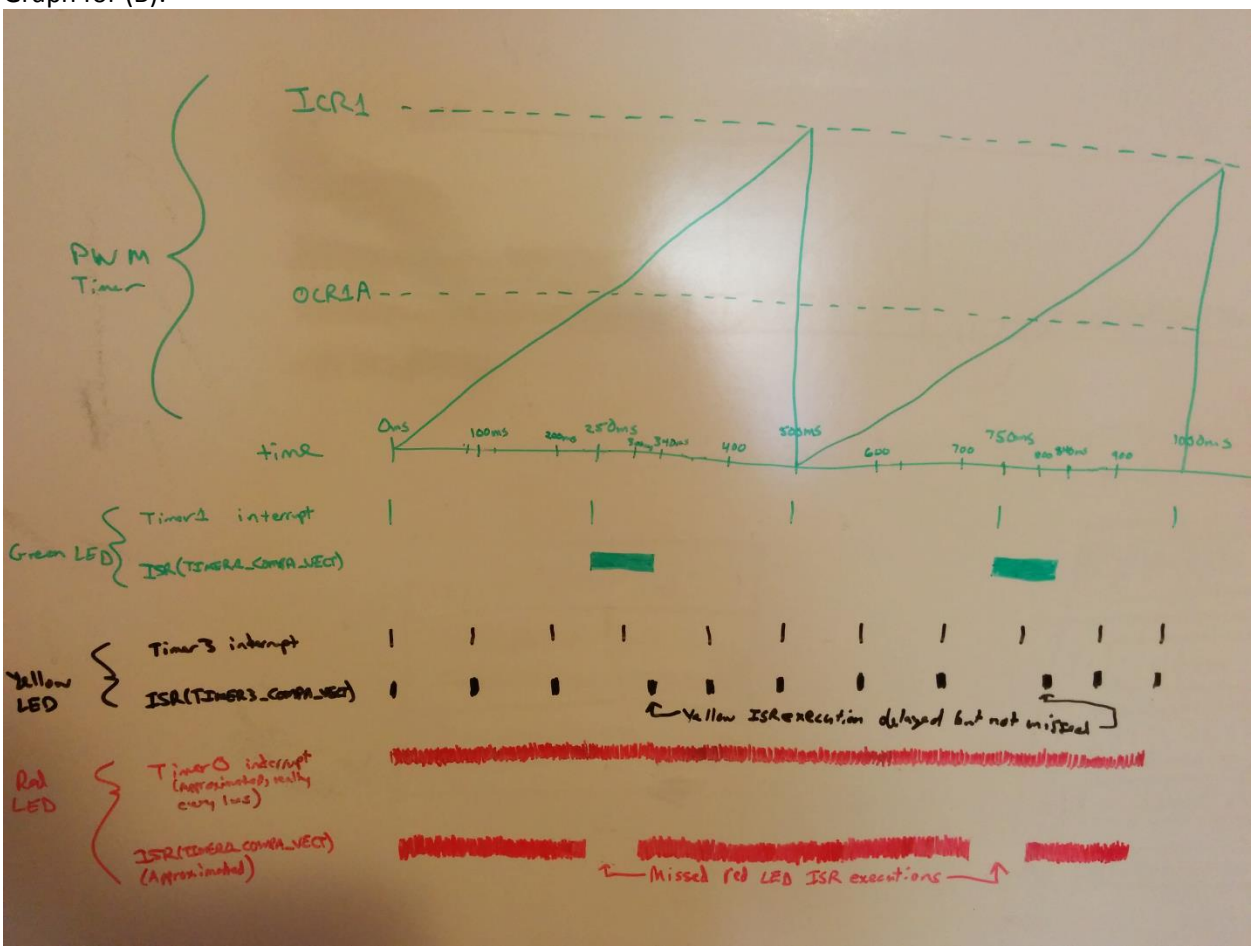
Observations & Explanations:
The first thing to note is that in the baseline test (A), the yellow LED had a higher blink count than the red and green LEDs, which were roughly equal. The reason for the higher LED count has to do with the fact that it can only blink in exact multiples of 100ms. When we set the toggle to be at 250ms, the yellow LED is actually toggling at 200ms (due to integer division). Since 250ms/200ms = 1.25, I would expect that the yellow count is approximately 1.25 times as many as the green count, which it is.

The next item of note is that the green and yellow LED counts are unchanged when adding 90ms delays to either the green or yellow LED ISR. This is because those ISRs are firing at 100ms and 250ms periods (well, technically the green LEDs have two ISRs that are firing every 500ms with a 250ms offset, but I'll ignore that) which is longer than the 90ms delay that gets introduced. Since the ISRs are firing at periods longer than the delay, none of those interrupts are getting missed (i.e., they're all still being processed) thus the counts remain unchanged. Every 5th execution of the yellow LED ISR is late, but not missed.

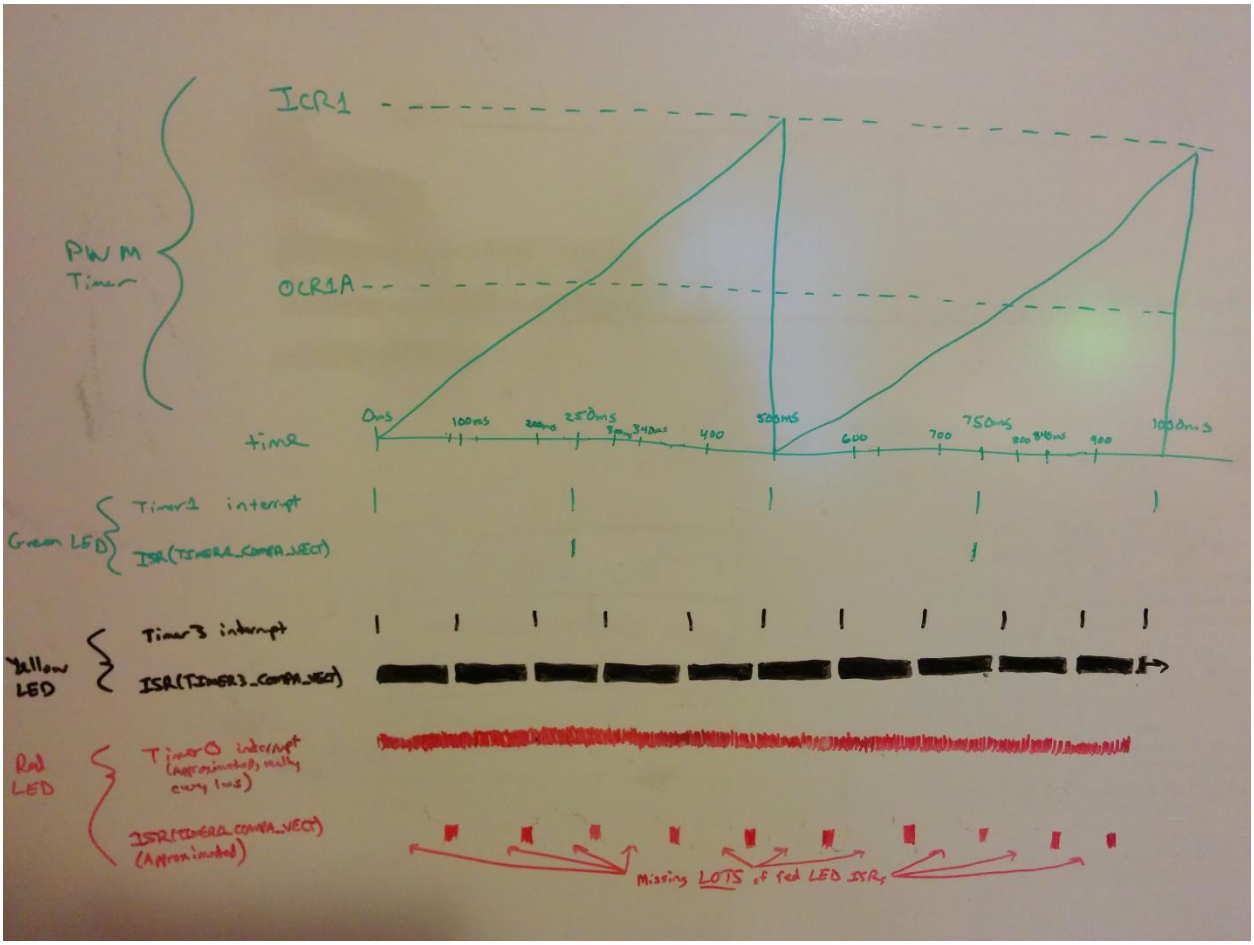
The interesting action happens with the red LED. When the busy-wait is in the green LED ISR (B), the number of red toggles is slightly diminished (reduced from 236 to 195). This is because the red LED ISR is unable to run when the green LED ISR is running, which happens somewhat infrequently. When the

busy-wait is in the yellow LED ISR (C), the number of red toggles is greatly diminished (reduced from 236 to 31). This is because the red LED ISR is unable to run when the yellow LED ISR is running, which happens more frequently.

Graph for (B):



Graph for (C):



Experiment 4

(A) Results from serial terminal – 60 seconds (110ms busy-wait in green LED ISR):

```
ta250
Toggle Complete
za
Zero Complete
pa
R:186 Y:303 G:243
```

(B) Results from serial terminal – 60 seconds (110ms busy-wait in yellow LED ISR):

Unable to use serial interface

Observations & Explanations:

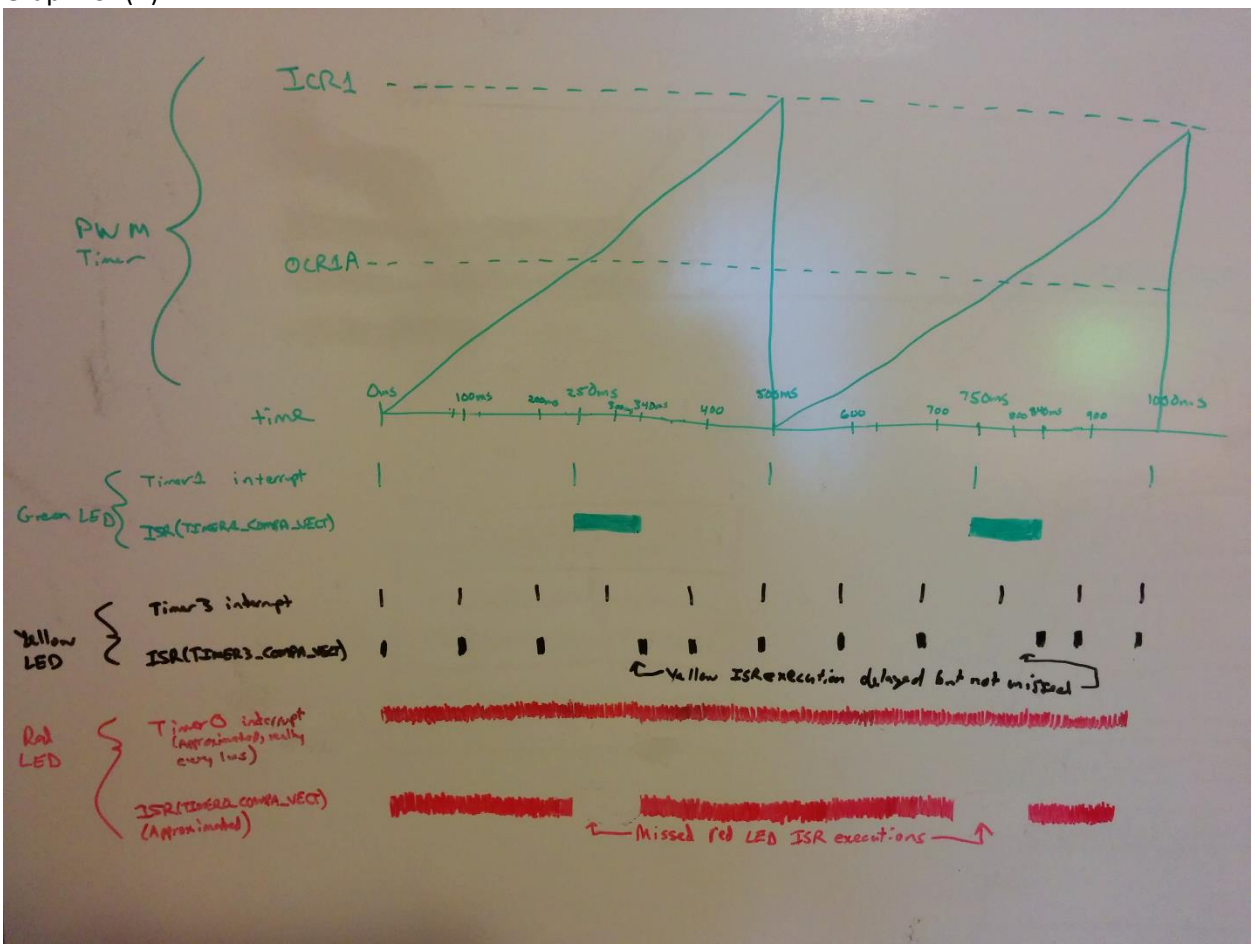
With the 110ms busy-wait in the green LED ISR (A), the results are very similar to (B) in the previous experiment. The net result is a slight reduction in the number of times that the red LED ISR can run, which reduces its count from 195 (90ms busy-wait in green LED ISR) to 186 (110ms busy-wait in green LED ISR). Comparing the two experiments, the yellow LED toggle count stays the same. This is because the yellow LED ISR is made late (every 5th execution, see graph) but never missed.

When the 110ms busy-wait is moved into the yellow LED ISR (B), the serial interface becomes unusable. This is because the 110ms busy-wait exceeds the period of the yellow LED ISR (set at 100ms) and therefore there is *always* an interrupt queued for the yellow LED ISR. Since there's always an interrupt queued, the CPU spends its time constantly processing this ISR and has no available time for the serial processing. The serial processing is not done in an interrupt, which always has a lower priority than the interrupt.

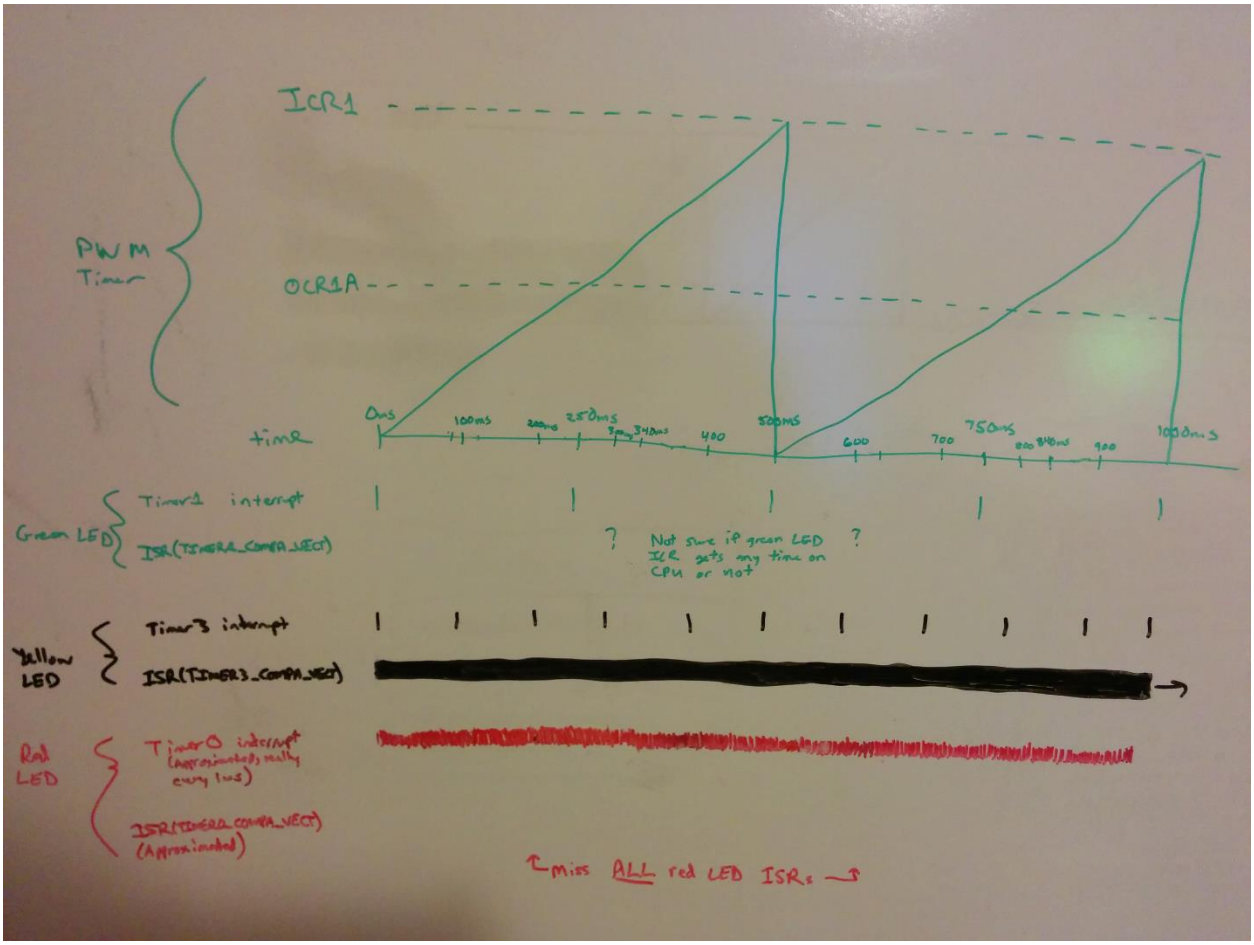
Since I couldn't get counts from the serial interface, I am forced to visually observe the LEDs. I observed the green LED blinking at its default rate of 500ms on-time. This makes sense because the green LED is receiving control directly from the PWM generator which produces a signal in hardware. The green LED ISR can get zero CPU time and the green LED will still blink. It's not clear to me if the green LED ISR is getting opportunity to run or not, because it's unclear to me what the relative priority of the green LED ISR and the yellow LED ISR are.

The yellow LED was blinking at approximately the default rate (100ms), although I know it has to be off a bit due to the 110ms busy-wait introduced to its ISR. The red LED came on and never blinked. This is expected since the red LED ISR isn't getting an opportunity to run.

Graph for (A):



Graph for (B):



Experiment 5

(A) Results from serial terminal – 60 seconds (510ms busy-wait in green LED ISR):

```
ta250
Toggle Complete
za
Zero Complete
pa
R:1 Y:60 G:242
```

(B) Results from serial terminal – 60 seconds (510ms busy-wait in yellow LED ISR):

Unable to use serial interface

Observations & Explanations:

For the case where a 510ms busy-wait is added to the green LED ISR (A), the yellow and red LED counts are greatly reduced. The green LED count seems to be almost the same (reduced by 1 from baseline). I believe this is because the green ISR fires every 500ms. Since the ISR takes ~510ms to execute (thanks to the 510ms busy-wait), this delays the green LED by 10ms each second. Over the 60 second experiment period, this results in an offset of 600ms, which likely causes one missed run of the green LED ISR.

Since the yellow count came out at 60 and it's toggling every 200ms, that means that the ISR ran 120 times. This suggests to me that the ISR for the yellow LED must have a higher priority than the ISR for the green LED.

The red count of 1 means that either (1) the red LED ISR ran 250 times or (2) that it happened to toggle within the 60 second period that I ran the experiment in.

For the case where a 510ms busy-wait is added to the yellow LED ISR (B), the behavior is the same as the previous experiment. I believe this is for the same reason – essentially because the yellow LED timer fires more frequently (every 100ms) than its ISR takes to execute (probably a hair over 510ms). In these circumstances the yellow LED ISR is essentially always running on the CPU.

(Graphs for Experiment 5 omitted due to similarity to previous experiments.)

Experiment 6

(A) Results from serial terminal – 60 seconds (510ms busy-wait in green LED ISR preceded by `sei()` call):

```
ta250
Toggle Complete
Unable to use serial interface
```

(B) Results from serial terminal – 60 seconds (510ms busy-wait in yellow LED ISR preceded by `sei()` call):

```
Unable to use serial interface
```

Observations & Explanations:

For the case where a 510ms busy-wait is added to the green LED ISR and the for loop for the busy-wait is preceded by a `sei()` call (A), I observed some very bizarre behavior. As soon as I issued the ta250 command, the red LED blinked one time and the yellow LED also stopped blinking. The green LED continued to blink regularly, which is expected because it is blinking due to a hardware generated signal.

The bizarre part is that after some amount of time, the yellow LED started blinking again. I don't know exactly how long it was, but it was upwards of 5 minutes of runtime. Based on these results, I decided to repeat the case after resetting the hardware. On the second trial, the yellow and green LEDs were both blinking after issuing the ta250 command. I observed them for 6+ minutes and this behavior continued.

I have no idea how to explain this. My best guess is that there's some sort of race condition going on and that the behavior of the LEDs depends on the system state when the ta250 command is processed.

For the case where a 510ms busy-wait is added to the yellow LED ISR and the for loop for the busy-wait is preceded by a `sei()` call (B), the serial interface was immediately unusable. The system came up with the green LED blinking at the default frequency (again, expected due to hardware signal generation), the red LED in an on state, and the yellow LED in an off state. The red and yellow LEDs did not toggle state.

I think I can explain this behavior. What's going on here is that the WCET of the ISR servicing the yellow LED timer is >510ms. Since this timer interrupts every 100ms and the ISR can be interrupted, the ISR execution never gets past the 510ms for-loop at the beginning of the ISR.

The red LED never toggles state because its actual signal toggling happens in the cyclic executive (while loop inside the main function). Due to the long running ISRs, all of the CPU time is spent running ISRs and the while loop inside the main function never gets any time on the CPU.