Jesse Belanger

EC 330

Homework 1

2/10/15

Starting with the Blink\_using\_WDT code as a foundation, I had the necessary methods for setting up a timer and using it to efficiently toggle an LED at a set interval. From there, I knew that I would have to take control of and manipulate that basic blinking function in order to output SOS.

My first order of business was to figure out how to control the number of flashes. Since the WDT handler simply reverses the state of the LED every time that it’s called, I figured it would need to be called twice for a complete flash (off 🡪 on, on 🡪 off), so six passes through the handler would be sufficient for three flashes. This lead me to create a global *flash* variable that would be initialized to six, decremented with each pass through the handler, and then reset after it reached zero. In this manner, each flash of unit T would have a built in pause of unit T.

After I worked that out, the mechanism for controlling pauses became simple. It is basically the same idea, except the LED is not toggled when the handler is called. I created another global variable – *stall* – to handle the pauses. It gets reset to different values depending on whether the pause is between letters or at the end of the word.

Based on my capability to control various length flashes and pauses, the SOS message seemed to naturally break down into four components: S, O, the pause between letters, and the long pause at the end of the message. I decided that the best way to transition between these components would be with a state machine. Each of the components became one state, and I created a global variable called *state* to hold that information. When I looked at how the states would follow from one to another, I saw that not only the present state, but the previous state, were necessary to determine the proper next state. For that I created a global variable *prev*.

Whenever the WDT handler was called and the blink\_counter was zero, the function would check to see what state the machine was in and act accordingly. As long as the flash or stall variables were greater than zero, the correct behavior would occur. As soon as the pertinent variable reached zero, another piece of logic would detect the change and use knowledge of the previous state to set the new state. Then next time the handler was called it would be with a different state and different behavior would happen.

The last thing I had to figure out was how to implement long flashes. These were especially tricky because the on periods would be 3T, but the off periods only T. I went back to my understanding that each call of the WDT handler only switches the state of the LED, and then doesn’t re-enter the handler until the blink\_counter has been depleted again. I determined that if I made different logic for on toggles than off toggles, I could make the interval larger for on periods than off periods.

For hardware, I used LED1 on the board, connected to P1.0.