Hardware Timers and Interrupts Learning Activity

Spencer Hernandez - **ECE 3430** - October 5, 2022

I. Intro

The purpose of this lab was to become familiar with port register interrupts, hardware interrupts, and polling.

II. Part A

This part was done in two parts: configuring TimerA0 and implementing the TimerA0 interrupt handler. To configure the timer, we had to first clear whatever values were in the control registers, set the value that we want to count up to in the capture compare register, enable the timer interrupt, enable the TimerA0 interrupt in the ISER, assign the interrupt a priority, and then correctly divide the clock frequency such that it is 500 ms. To implement the interrupt handler, all we have to do is clear the interrupt flag, and then toggle the LEDs such that we turn the current LED that is on to off, and then turn the LED that was off to on. Experimental Verification can be seen below in Figure 1.

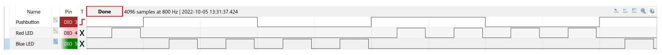


Fig. 1. Verification for Part A

III. Part B

For This section, instead of using a port register interrupt to change which LED is blinking, we will poll the interrupt flag in the control register. Verification can be seen below in Figure 2.

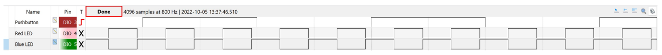


Fig. 2. Verification for Part B

IV. Part C

This part was done by using a 32-bit hardware timer to create a delay. This was done using one-shot mode, which causes the timer to count down from a specific value. To verify, a square wave with a 500 ms period was run to compare with the generated signal to prove that the generated signal has a similar period. Verification is shown below in Figure 3

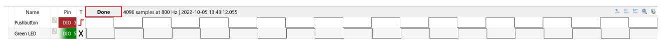
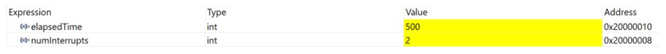


Fig. 3. Verification for Part C

V. Part D

For this section, in order to verify that the code is working correctly, a square waveform with a 1 kHz frequency was used as the input for P2.4, and the clock frequency was divided such that it was 1 MHz. Therefore, after one of the two interrupts were called (since both edges cause an interrupt), the elapsed time would be 500 ms. This value is calculated by reading the number in the capture compare register, and subtracting it with the previous time. Figure 4 below shows that this number is stored in this elapsed time variable. Note that the numInterrupts variable is simply used so that we know we are measuring the elapsed time after 2 interrupts and not hundreds or thousands of interrupts.



Expression	Type	Value	Address
elapsedTime	int	500	0x20000010
numInterrupts	int	2	0x20000008

Fig. 4. Verification for Part D