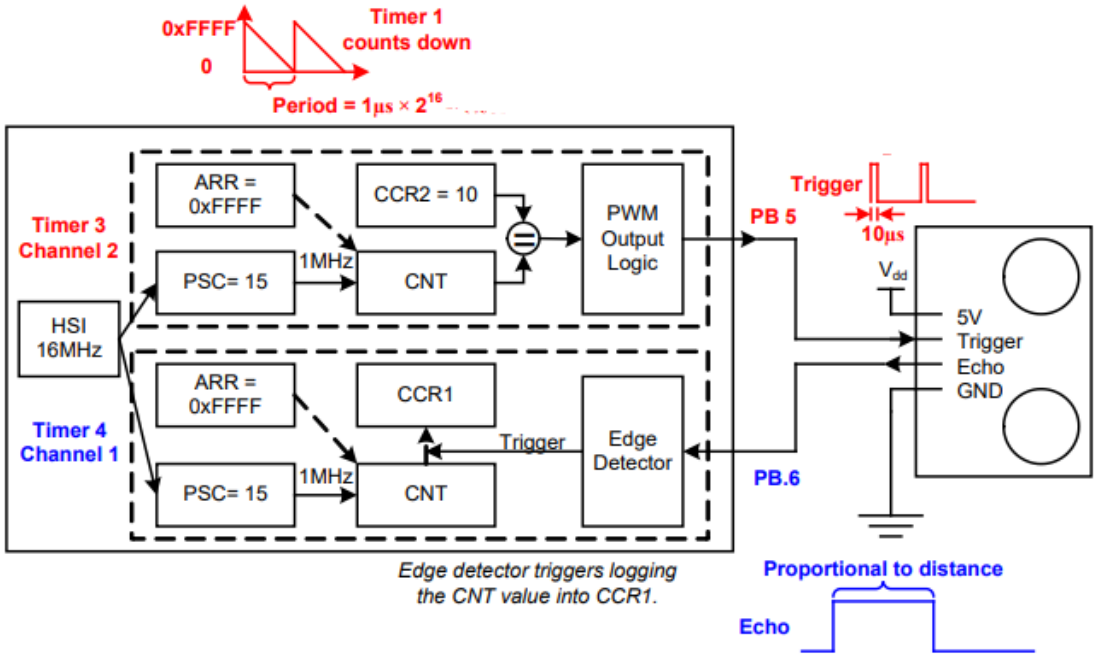
ES LAB 5: Timers Input.



Student: Johnson Domacasse
Date: 11 Nov 2023
Student#: 4471709
Teacher: Suzana Andova

Introduction:

Timers are built-in tools that can be used to generate time base generations, producing PWM signals, measuring pulse width and more. There are three types of timers: the Basic timers, the general purpose timers and the advanced timers. For the sake of this Lab we will be using general purpose timers to produce PWM signals and measure the pulse widths.

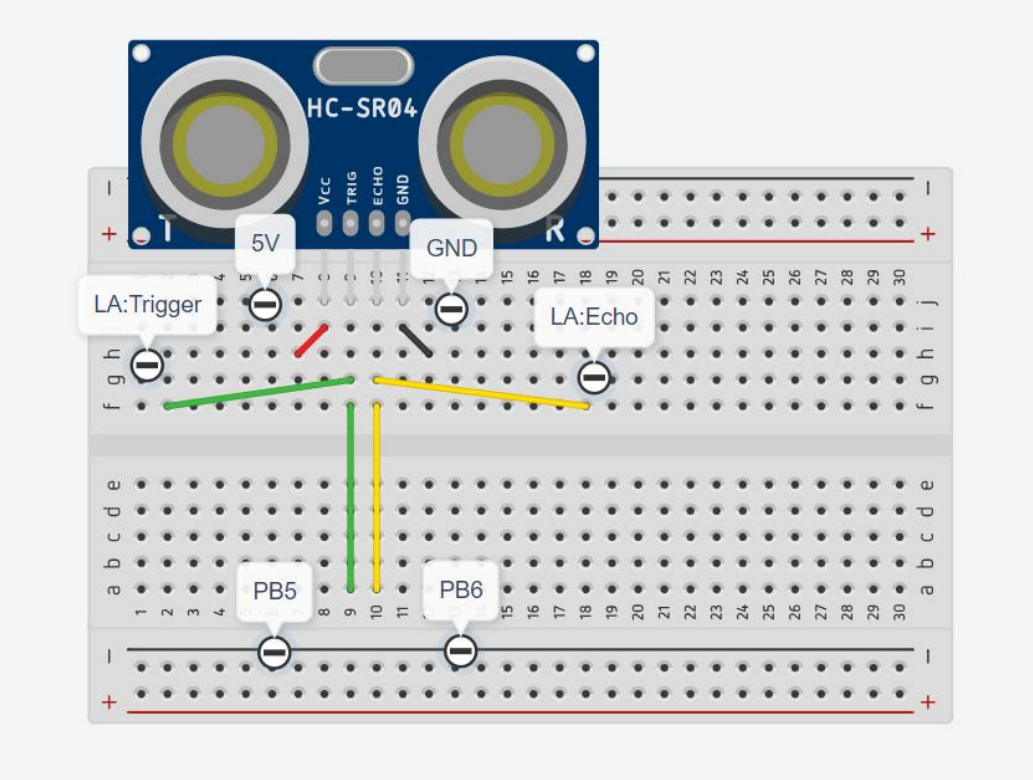


Figure 1: Circuit used

Procedure:

Going into this research, we will assume that the pins are correctly placed. The pins are also correctly setting using their appropriate clock, their correct modes, and the correct AFRL are set. See figure 1 for the circuit used.

Trigger:

To begin, I first started by configuring my echo pin to generate PWM signals. In my case, most of the work is already done. I use the same functionality as I do from the previous assignment. It was a matter of changing some values. For example in order to slow down the clock, I set the PSC to 15. This way the system clock is cut down from 16MHz down to 1MHz. then we set the ARR to maximum values. Then we continuously set pulse the timer with 10 microseconds by setting the CCRy register

with 10. Further testing using logic analyser give me the following values:

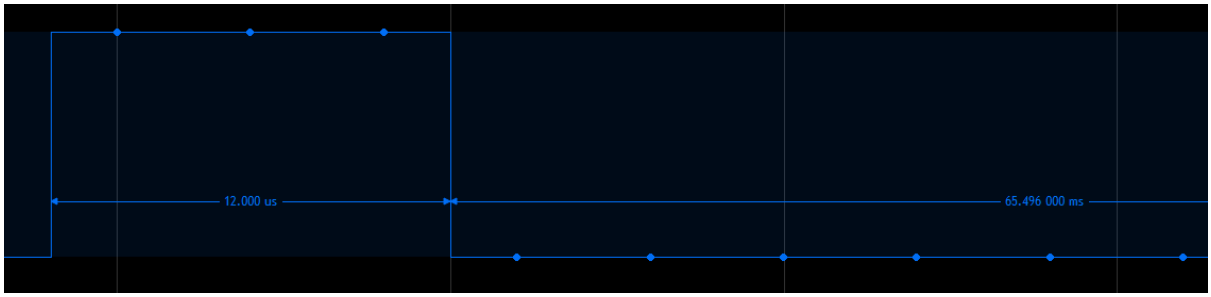


Figure 2: pulse width (duty cycle) of 10 microseconds.

So we now know that the pin is pulsing atleast 10 microseconds. Although sometimes It is 8 microseconds but the sensor still works with this pulse.

Echo:

Now that our trigger pin is correct, we need to configure the echo pin. Some of the conditions we need to program were given to us in the document. What was needed extra were the following: The interrupt functionalities, the input pre scaler value to capture all of transitions, and to enable the timer. Now that this is configured we will test it again with the logic analyser.

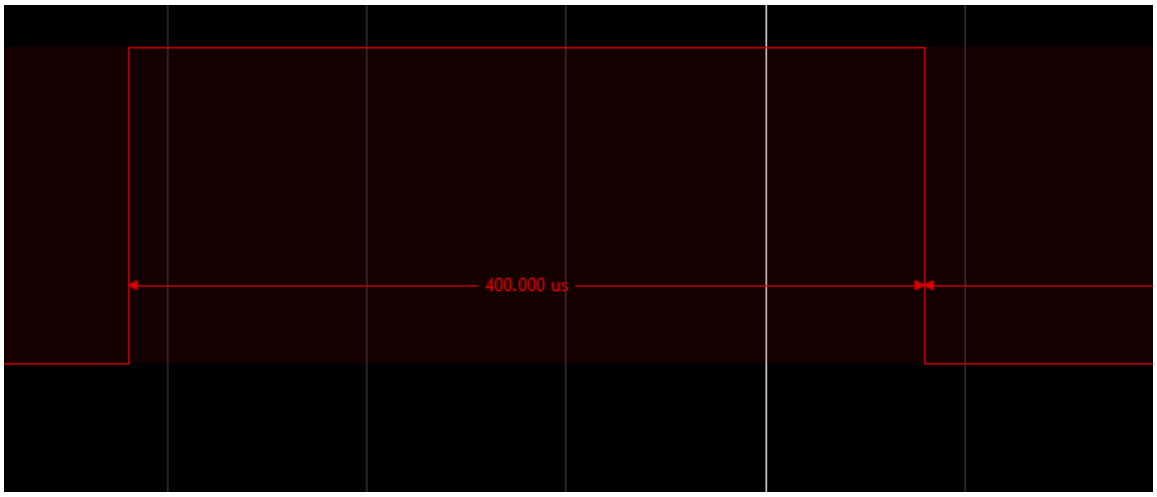


Figure 3: Pulse width that is being read on input pin.

As you can see the pulse width of the input pin is 400 microseconds. At the time of measurements the sensors is approximately 6 to 8 centimetres from a surface. We use the following formula to convert it to a readable values:

$$Distance\ (in\ cm) = \frac{Time\ (microseconds)}{58}$$

If we do 400 divided by 58 we get 6.8965… which will be rounded up to 7. So we now know that our sensor is reading.

Distance:

Now I want to print the values so they can be seen to a normal user. This I can do in the interrupt handler. Using the Serial monitor and UART we can print out the values. We first check which channel has triggered the interrupt. The we check both the rising and galling edges of the pulse

width. Finally we use a function to calculate and return the distance to us. This distance is then printed to our serial monitor so we can see.

Conclusion

My results for this are what I was expecting. One thing I noticed was for example if you have multiple channels open on the timer, you can have multiple interrupts being triggered. In a way, depending on which pin captured first, in interrupt from that pin (channel on specific timer) can be generated.

On another note, this assignment gave me some practice with the serial print function of this microcontroller.

Finally this assignment gave me a good idea of what needs to be done, in order for our vehicle in the robot project to stop moving. Using the same setup as here, we can determine when a wall is too close, or if we can move forward.

Explanation on the Interrupt section.

I have figure out why the code works. In a sense, the first capture will always be missed. Basically, the line that checks if condition with:
the TIM4->CCER register (which has been set to 1 in the setup function) and the TIM_CCER_CC1P macro (which has always been set to 1) has been **set is always failing on the first iteration.** In other words, the first value you receive is always going to be 0 (when run with the calculate distance function).

The reason the program doesn't fail is because at the end when you perform the XOR-operation on the macro, it will change it to 0. Then in the second iteration the resulting value when compared with both of them would be 0. So the function would return true, and the logic behind the program can now function properly.

So in order to fix this issue, I performed a similar logic to what I had with Mohammed using states. I know have a "logic level" variable, that will be initialized at 0 resembling low signal logic. I then replace the condition of the if statement to check if the logic level is either low or high. If it is high, then the next trigger will be a falling edge which will then put the variable to low.