ES LAB 4: Timers Output.

Student: Johnson Domacasse
Date: 21 Okt 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.

Procedure:

Part A:

This part of the Lab has us turn on an LED with 75% brightness. In order to do this we first need to set up the timer and then make sure it outputs PWM signals. We will use **TIMER2**.

For setting up the timer, you need to configure a few registers within your program. We do this in the "Timer_Config" function. First we need to configure our auto-reload register and our prescaler with the correct values. The following equation was used:

$$PWM\ Frequency => \frac{Clock\ Frequency}{Desired\ Frequency} = value(PSC \times ARR)$$

The clock frequency that will be used for this implementation is 72 MHz. Our desired frequency is 1KHz, or 1 millisecond. Our final value is then 72000. Which was then split between the prescaler and the ARR. 720 for the prescaler and 100 for the ARR. Next up we needed to configure the compare value. This is the value that will send out the PWM depending on how bright you want the LED to be. The following formula was used:

$$PWM\ Duty => \frac{CCRy}{ARR} \times 100$$

Where y is the channel that will be used.

The reason why we chose 100 as our ARR value is because, it makes it easier when configuring the duty value. In this case if we want a duty value of 75% then our CCRy value would be 75.

We need to set up this channel as PWM output channel is well. Through reference manual, we can see that we need to set 2 bits. In this case it is bit 2 and bit 1. This is done in the CCMx register. Once that is finished, we can enable the channel.

Finally we need to set up our pin. We check table 14 in the STM32F303RE datasheet. We are using channel 2 in this case. So we see that we have to use PA1 with AF1. For that we have a separate function called "System_Config". This is where set up the MODE register to be in alternate function and set the AF1 in the AF register. See figure 1.

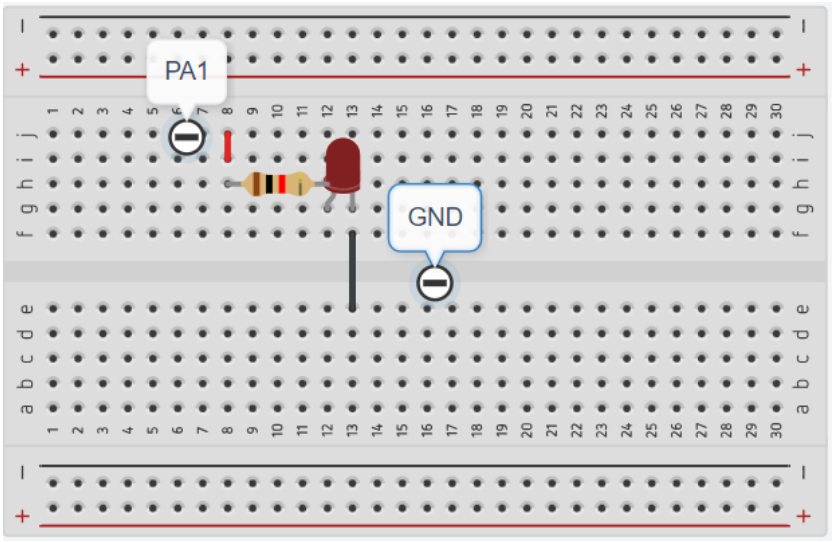


Figure 1: LED circuit

Results:

The light is indeed turning on. We didn't know if it was the right brightness, so we put a different lower value in the CCRy. For example, 20. When ran again the brightness is indeed significantly lower.

Part B:

This part of the lab has us pulse an LED by changing its brightness from 20% to 100% and back. For this implementation we were allowed to change the CCRy directly and then putting a delay in between. We are reusing some code of our SysTick implementation to simulate a delay. Here we made a simple delay function using the SysTick.

In our main loop then we set the CCRy value to 20, then we use the SysTick delay with a value of 500 milliseconds. Then we set the CCRy value to 100 with another SysTick delay after with the same value.

Results:

Once the code is uploaded, we can see that the LED is pulsing by changing its brightness by going to a low brightness and a high brightness. We wanted to see how this behaviour looked using a logic analyser. See figure 2 for the updated circuit. The results are difficult to show in this document so I will demonstrate them in the live demo. Essentially. Every .5 seconds you will see a change in the waveform on the logic analyser. I expect this because every .5 seconds, the CCRy value changes. On top of that, you will notice that the duty cycles (when waveform rises) matches the CCRy values as well.

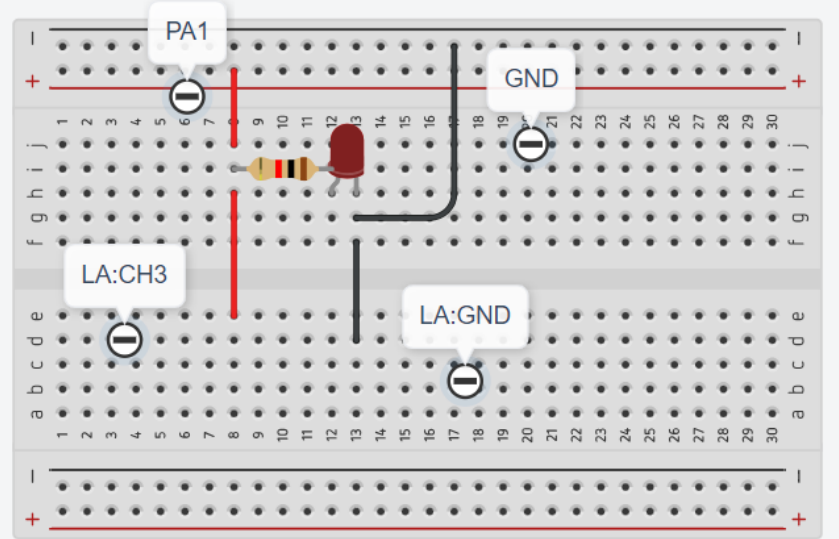


Figure 2: logic analyser added.

Part C:

The final part of this Lab has us turn a servo motor in a few angles using the pulse milliseconds. We need to first start by calculating new values for our ARR and PSC registers.

Initially our desired frequency was 1kHz but for this part the period needs to happen at 50Hz or 20ms. We calculate the new ARR and PSC values using the same formula as before:

$$PWM\ Frequency => \frac{Clock\ Frequency}{Desired\ Frequency} = value(PSC \times ARR)$$

Our clock frequency remains the same, but our desired frequency goes to 50 from 1000. Our total values is then 1440000. If we put 100 in the ARR value that leaves us with 14400 to put in the PSC register. Once checked on the logic analyser, it can be confirmed that our period is roughly 20ms.

The next part is to calculate the values used to change our servo angle. Initially how it worked, is you would pass the exact percentage in value to the CCRy to get the desired percentage duty cycle. But in order for us to get values of 0.5ms accuracy we cannot use 100 as our ARR value, since we can't use values with decimal points. In order to solve this we put new values of 10000 in the ARR and 144 in the PSC registers. You can also do 1000 in the ARR and 1440 in the PSC just as long as the total value matches the calculated PWM frequency. Now we can get to the actual values for the servo angles.

Here are some results I found. Table 1 shows the calculated values I want within my period. Table 2 shows values I need to put in to my CCRy value.

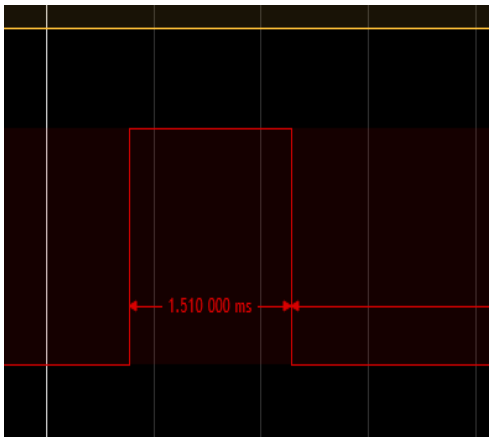
Value (ms)	Period (ms)	%-total (%)
1.0	20	5.0
1.5	20	7.5
2.0	20	10.0

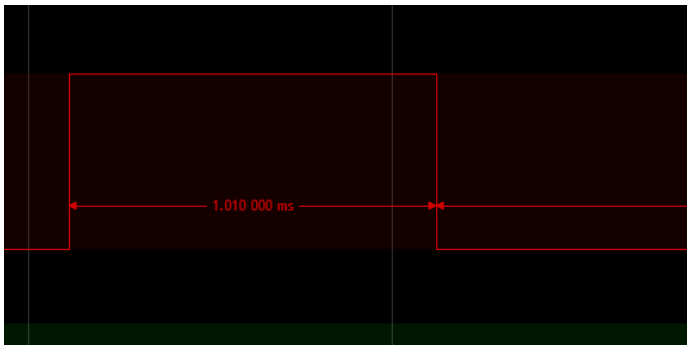
Table 1: calculated percentage based on period.

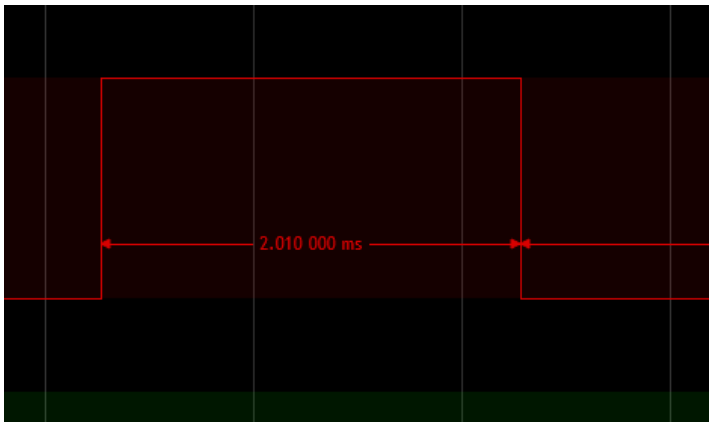
%-value (%)	ARR value	CCRy value
5.0	10000	500
7.5	10000	750
10.0	10000	1000

Table 2: calculated CCRy based on percentage.

The results when passed to the register are the following for the 0°, -90° and 90° angles:







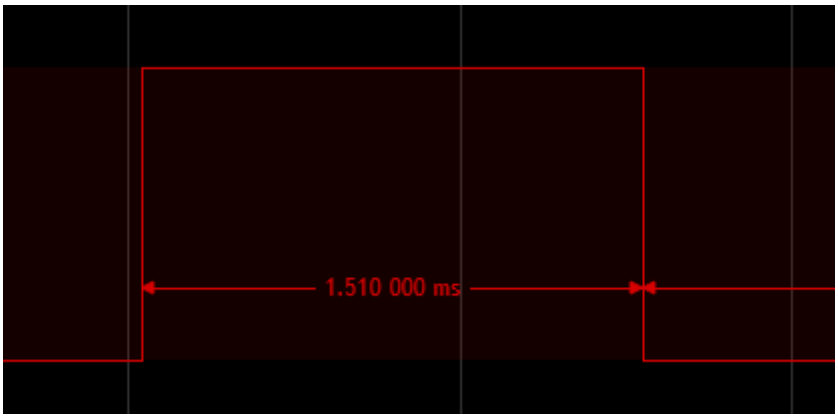
The only issue is now the servo doesn't turn with 90°, but only turns at with an angle of 45°. So I need to play around with the CCRy values in order to achieve 90°. Finally, after some trial and error, these are the values I ended up with the following:

old CCRy value	New CCRy value	
1000	1210	Turns to 90°
0	0	Stays the same (0°)
500	290	Turns to -90°

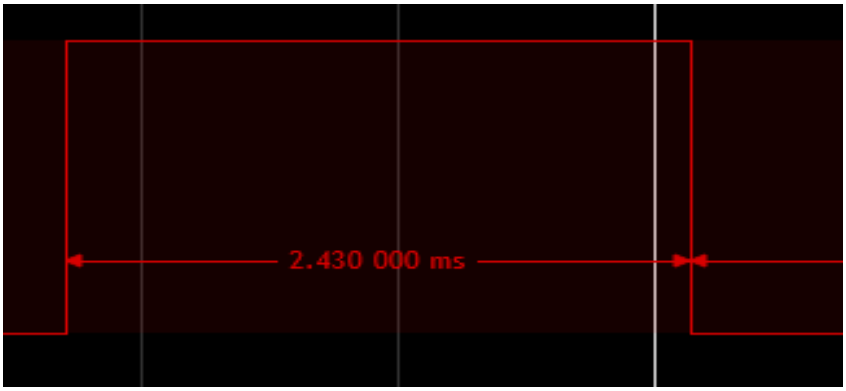
Table 3: new CCRy values corresponding with angles.

Results:

In the end I am left with a servo that does what is expected. It first goes to 0° then 90° then -90°. This was after I change some values. The 0 CCRy value stayed the same because this was the centre point. Both the other two CCRy values have changed with around 210. This causes that the servo now turns approximately to the correct 90° angles. Under the logic analyser you can also see how these values have changed. These are the final duty cycles within the 20ms period.



0°



90°

-90°