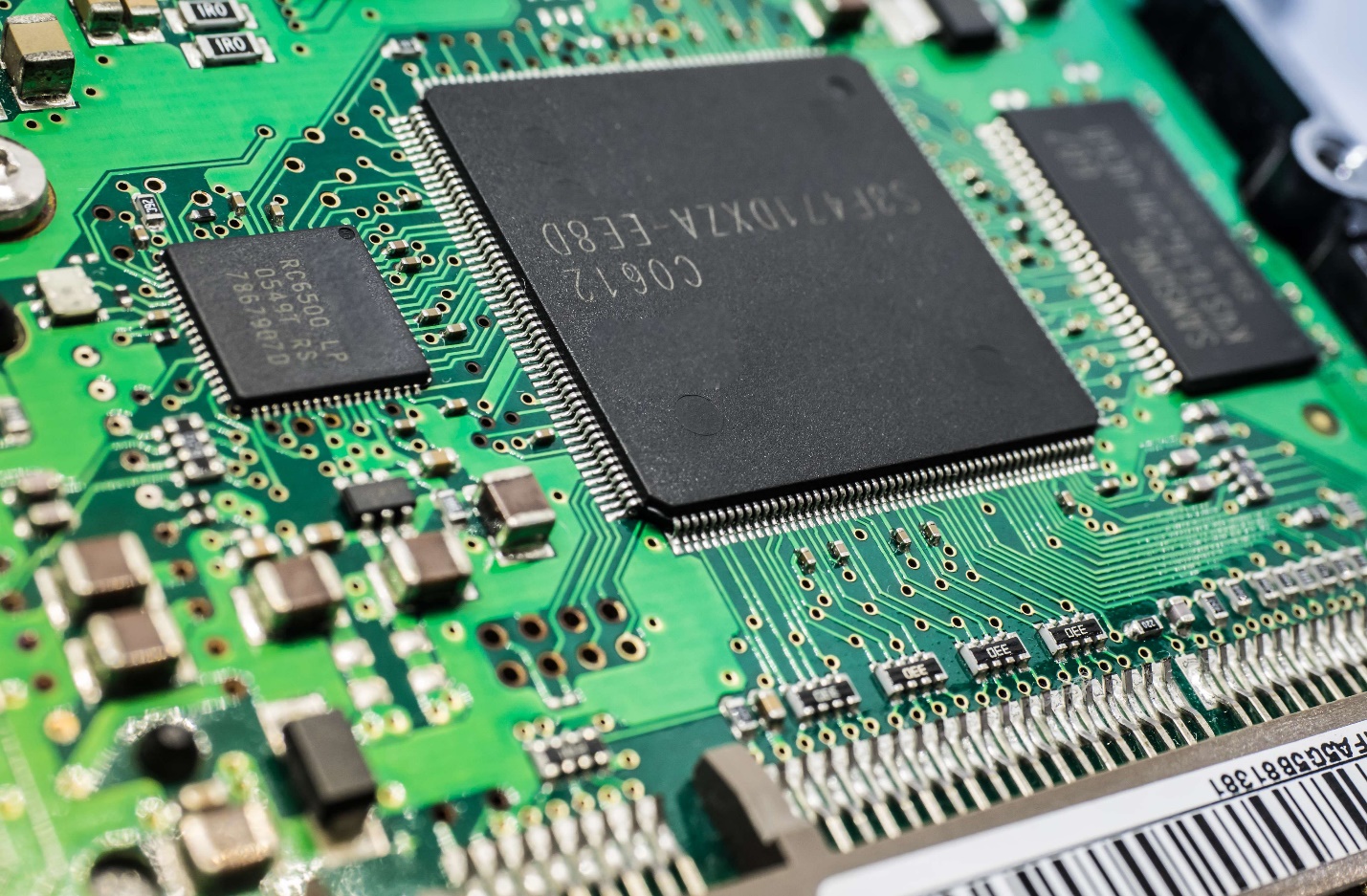
Timers Challenge

July Trendafilov(4674693)



Contents

[Revision history 2](#_Toc120830816)

[Challenge FreeRTOS and Timers 3](#_Toc120830817)

[Conclusion 6](#_Toc120830818)

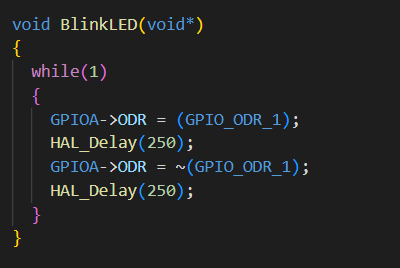
# Revision history

|  |  |
| --- | --- |
| 29.11.2022 | Added Challenge section to POC document |
| 01.12.2022 | Removed POC section (only Challenge) |
| 04.12.2022 | Added pictures of logic analyzer |

# Challenge FreeRTOS and Timers

For the challenge I have one LED, one servo controlled by PWM and on rotary encoder. The rotary encoder is reused from a previous project, while there are some new concepts here (PWM) and RTOS. The LED is blinking at 0.5 Hz on a separate thread and the servo is controlled by the rotary encoder on a different thread. The threads don’t share resources, so we don’t have synchronization.

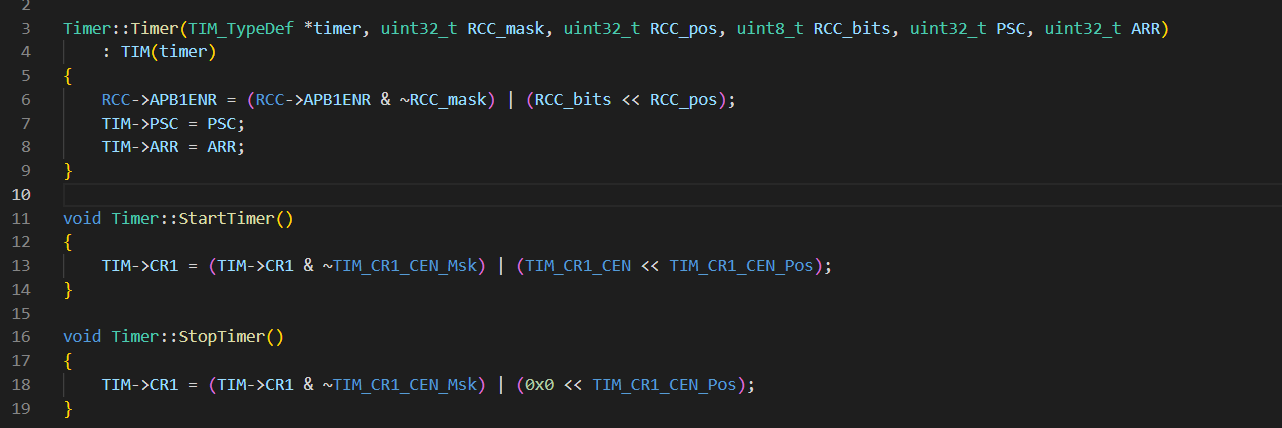
The first part is the Thread for the LED:



Here there is an infinite loop which toggles a LED which is not blocking the other thread.

The second part is the Servo. The new concepts introduced here are timers and PWM. This is done with a class for timers and servos which look like:

The timers class member functions are:



There is a constructor and a start stop method. In the constructor we have 3 important lines:  
1. To enable the timer we want to use in register RCC->APB1ENR (I use TIM2 in this example).

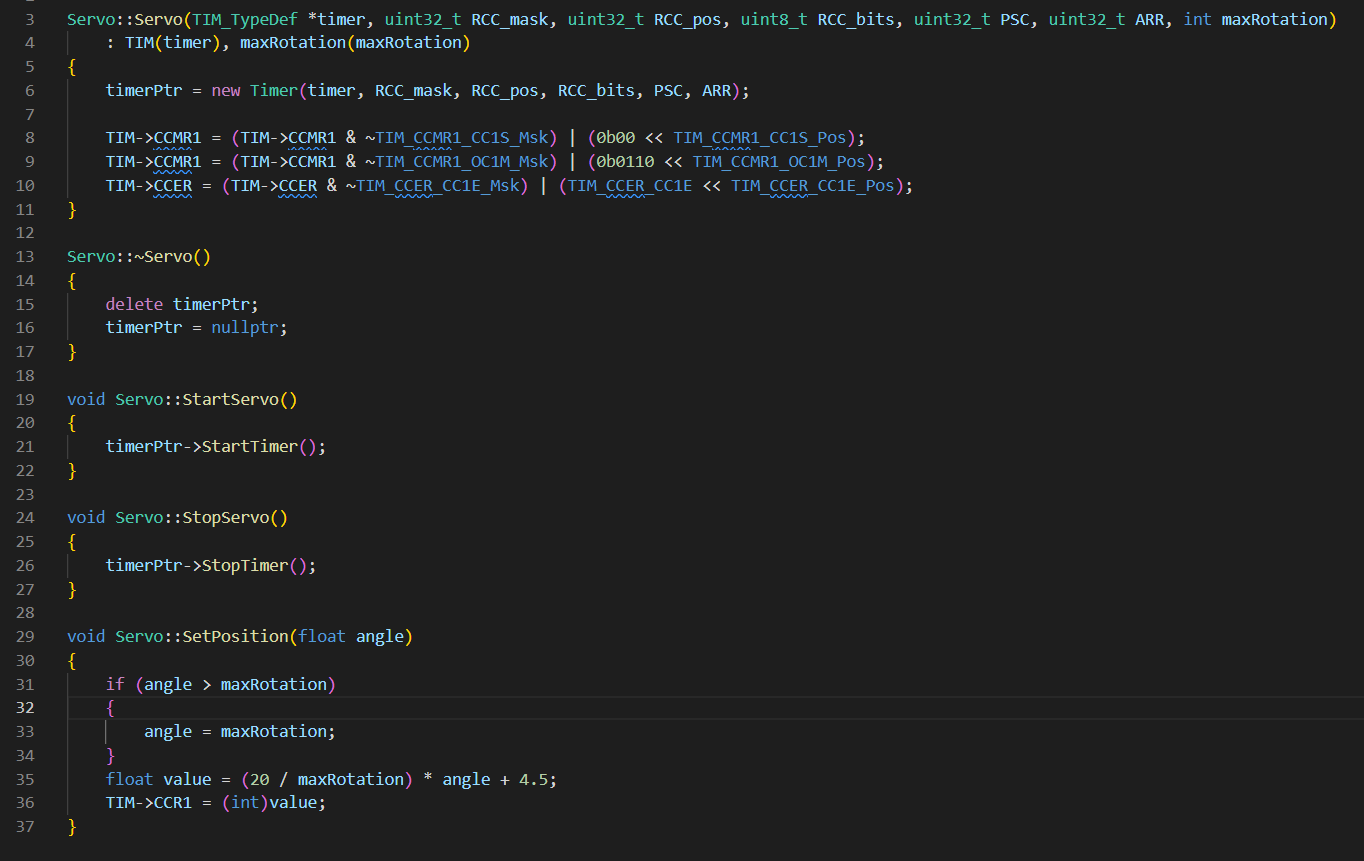
2. To set a prescaler for the timer so that it doesn’t count very fast. In this example I will set it to 1kHz.

3. To set a value at which the timers goes back to 0 and starts counting again

Then we have two functions that start and stop the timer object:

1. To do that we either write 1 or 0 to the first bit of register CR1.

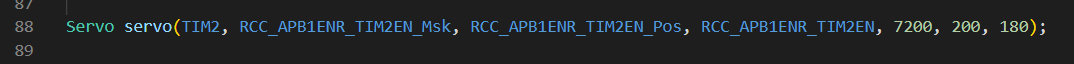
Then the other class servo looks like:

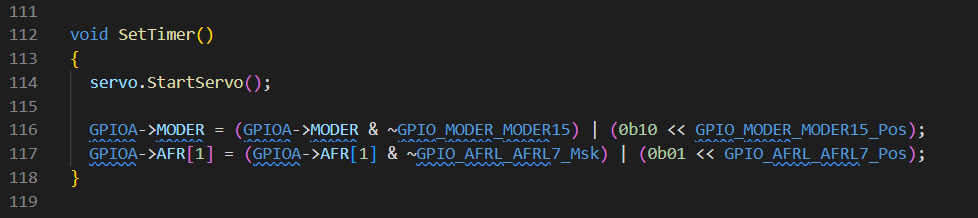


There is a constructor, start and stop and “SetPosition” function. In the constructor we initialize the timer with an object of the timer class. Then we extend the setup of this timer to enable PWM. This is done with 3 operations:

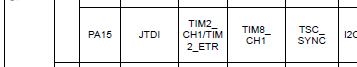
1. First, we must configure that we will use the register as output. (line 8).
2. Second, we set the timer in PWM mode 1 which is up-counting(line 9).
3. Third, we set the capture/compare channel 1 to output by writing to the corresponding bit.

After we have the classes for the timer and PWM generation for the servo we can initialize our PWM timer as:

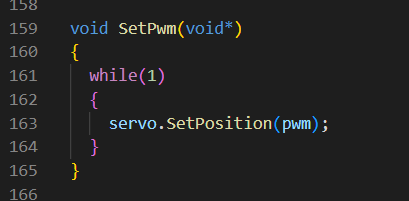




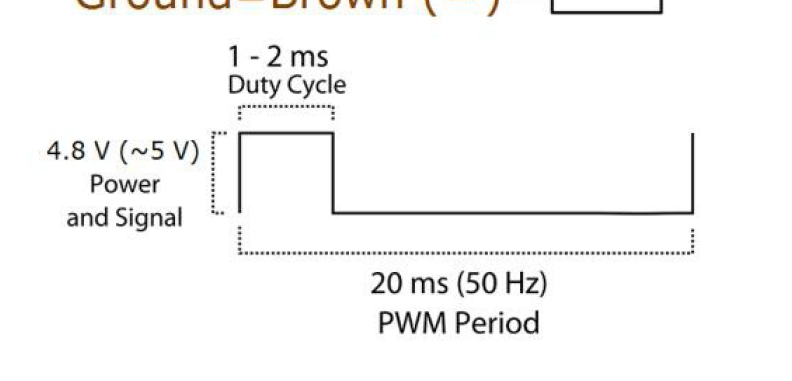
We have a global variable servo which we start in this function. Then in order for the PWM to work, a pin with the correct alternating function must be set. In this case we use GPIOA pin 15 as it can be connected to TIM2 CH1 which we previously have set. To use the tim function, we toggle the second bit of the register (line 117).

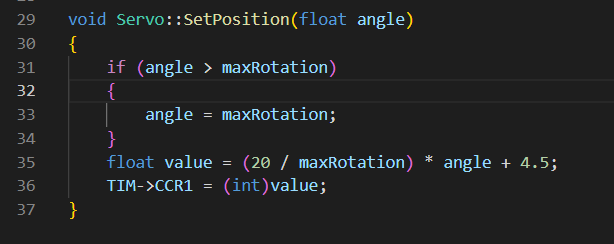


Lastly we have our thread for setting the angle using PWM.



Here in a different thread we use set the angle using our PWM variable which has values [0 – 180] because we use a servo which can rotate only 180°. To control the servo we must generate the correct PWM. To do so in the spec for the servo there is written that the PWM duty cycle must be between 1-2 ms while the whole duty cycle is 20ms (50Hz). The SetPwm is in a thread and when an interrupt occurs, the interrupt modifies the “pwm” variable and then the thread sets the position of the servo which actually sets the duty cycle.





Here the function calculates the correct value for the PWM. If we want to rotate the servo to 90°, the duty cycle must be 1.5ms. and in this case it will be around 14.5. The in the CCR1 register we set the value and when the counter reaches this value the value goes to zero until CNT > ARR. After that the generation starts again. Here we have 10kHz clock so 14.5 will be reached in 1.5ms.

When we have the PWM analyzed with Logic Analyzer we get the following output:

Timeline

Description automatically generated with medium confidence

On the current setting we have a cycle of 20ms where the duty cycle is 2.1 ms which means the servo is set to +90°.

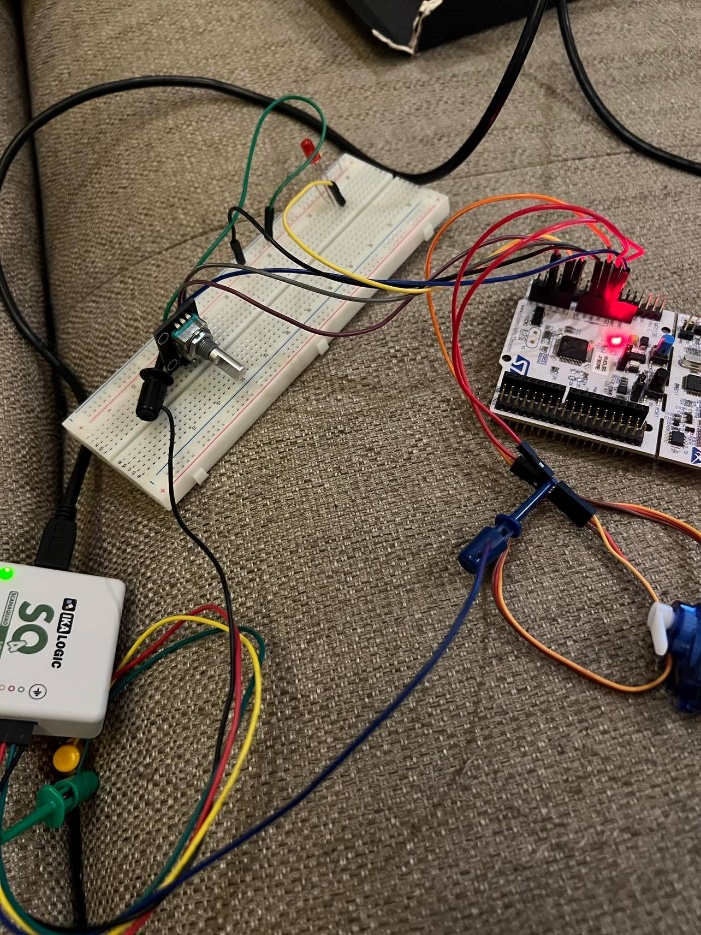
If we try it with different values like:

A screenshot of a computer

Description automatically generated

The duty cycle this time is 20.5 – 19.295 which gives us 1.250ms which is around -45°. In this application the duty cycle is a little longer but doesn’t affect the servo.

The hardware setup for the logic analyzer is as follows:





We use one PIN configured for PWM input in the Logic analyzer which is connected to the PWM pin and the common ground for the analyzer which is connected to ground on the board.

Graphical user interface, application

Description automatically generated

This is CH1 on the analyzer which is configured as PWM output.

# Conclusion

In this challenge I have learned to control timers and use PWM generation to control actuators. Further, I have learned to use FreeRTOS to have multiple threads and synchronize when necessary. Furthermore, I have learned to use a logic analyzer to test my board.