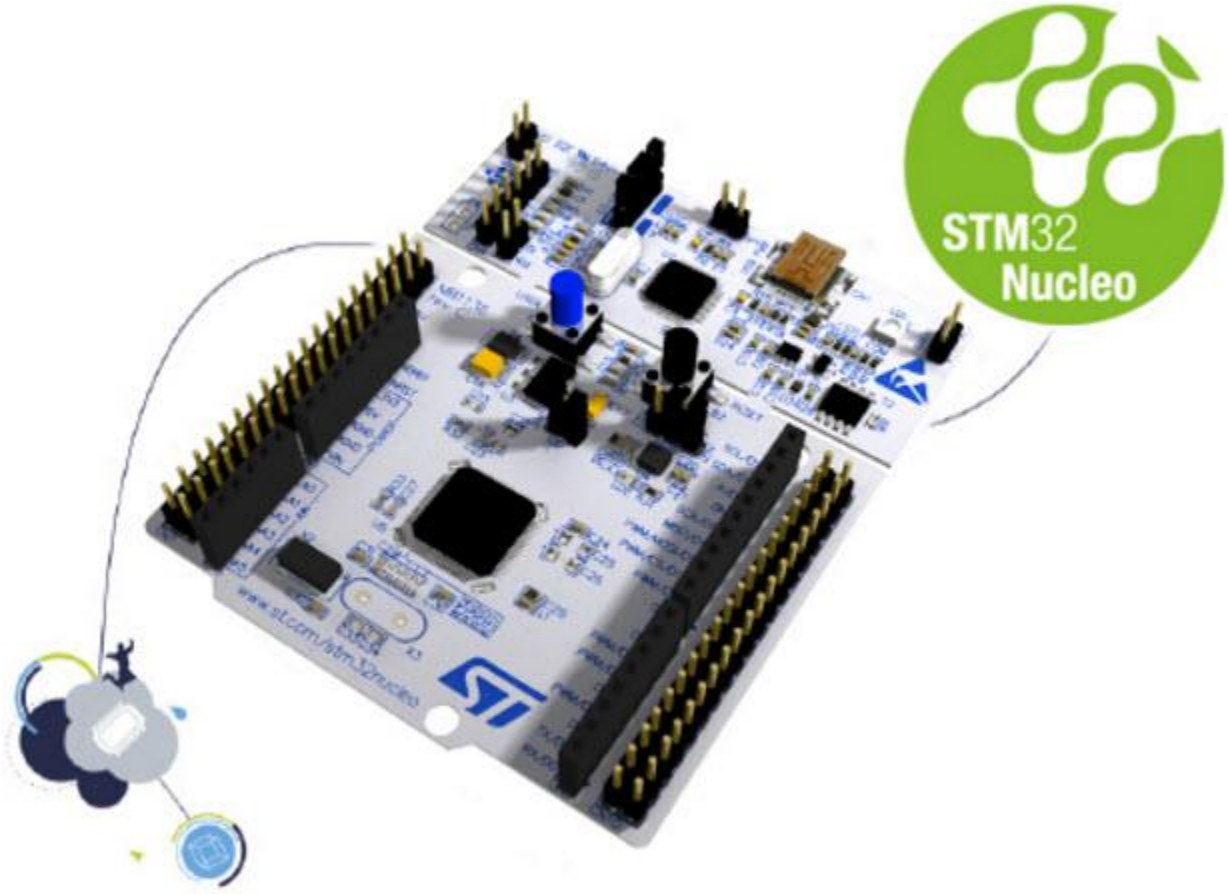
Technical report

Assignment 4:Timers output

By Veselin Atanasov



Teachers: Rene Bakx, Robert Verhijen

Date:10-10-2024

Contents

Introduction

The goal of this assignment is to create PWM output with the timers on the STM32 board in order to be able to control devices that need PWM input. The final goal is to create an application that slowly turns up a LED to full brightness and fades it back to off and to control the angles of a servo. In this document, the whole process of setting up the proper registers and the thought process of the decisions I have taken will be documented.

Description

Setting up alternate function for GPIO pins

The very first step I to decide which pins are going to be used and to which output channel of the timers can they be assigned to. This can be seen in the alternate function table in the datasheet. For this application I have chosen PA1 for the LED and PA6 for the servo motor. I chose them, because they both belong to port A and can be configured to timer 2 output channel 2 and timer 3 output channel 1(as can be seen in the alternate function table) respectively. I chose two different timers for the two pins because different prescaler and auto reload value settings are needed to serve the need for 50Hz of the servo motor. To prepare the pins to receive input from the output channels of the timers, they both need to be in alternate function mode. That is done by writing b10 to the MODER register, afterwards the specific alternate function has to be chosen. This is done by setting the proper bits in the AFR register. This register is separated into two sub registers AFRL(AFR low) and AFRH(AFR high). AFRL is responsible for setting the alternate functions of pins 0 to 7 and AFRH from 8 to 15. In this application, the pins are 1 and 6 so they both fall under AFRL control. To set up PA1 to receive signals from timer 2 channel 2, a setting of 0001 has to be loaded in the first position of the AFR register. This setting corresponds to alternate function 1(can be seen in the alternate function table) which connects the output channel 2 of timer 2 to PA1. The same has to be done for PA6 the difference being that the sixth position in AFR is modified and that the alternate function setting is 0010 for alternate function 2 which connects the first output channel of timer 3 to the PA6 pin.

Configuring timers to PWM mode

The first step of setting up the timers is to enable the clock source. Both timer 2 and 3 use APB1 as their clock source and run at two times its frequency(72Mhz). For timer 2 a prescaler of 1 is used in the PSC register as this is a 32 bit timer and can handle very large values. For the led a period of 1ms is more than sufficient so I will be reusing the formula from the last assignment(arr=(desired period*clock frequency)+1) to set the value of the auto reload register. This register resets the value of the counter whenever the counter reaches the arr value. Then the PWM mode on the second output has to be

configured by setting the appropriate bits in the CCMR register. The CCMR register is also in charge of enabling the preload on the channels. After this the output compare mode is selected in the CCER register to compare the value of the counter to the ARR value. Timer two setup is finished by enabling the auto reload on CR1 to automatically reset the counter when ARR value is reached and finally enabling the counter, again using CR1.

The process of setting up timer 3 is virtually the same, the difference being that a frequency of 50Hz has to be achieved from the 72Mhz clock. To achieve this frequency, the proper prescaler setting and auto reload value have to be calculated. I am using the following formula: output of timer= (clock frequency)/(prescaler+1)*(auto-reload+1). To make calculations easier, I decided to choose a prescaler of 719. By replacing with actual values the formula looks like this: 50Hz=(72Mhz)/(719+1)*(auto-reload+1). The result of this calculation is that the value of the ARR(auto-reload register) with a prescaler of 719, to achieve 50Hz, should be 1999.

After PWM configuration is done, the final step to control the width of the pulse is to manipulate the CCR(capture and compare register) to achieve desired pulse width. Timer 2 and 3 are set up to work in PWM mode 1 and output compare mode meaning that the output will be high if the value of the counter is lower than the value of the compare register(CCR). This means that the higher the value of CCR, the higher the pulse width and duty cycle will be.

The application

For the PWM signal for the led, I created a function which accepts the duty cycle as a percentage as an argument. For the value of the compare register, a calculation is done which multiplies the value of the ARR+1 to the percentage that has been selected by the user. The formula looks like this: compare_value = (TIM2->ARR + 1) * duty_cycle / 100. To light up the led smoothly I use a duty cycle variable and a constant step variable with a value of 1, that work together in a for loop. To make the led go to full brightness, the for loop increments the value of duty cycle by step(1) each iteration. To make transition smoother, a delay of 10ms is administered after each iteration. 100 steps*10ms = 1000ms(1 second). This means that the led goes from off to on in 1 second precisely. The same process is used for turning of the led smoothly, the difference being that the duty cycle is decremented by step.

The servo is a bit more specific and reacts to signals in the range of 0.8ms to 2.2ms to reach positions from 0 to 180 degrees. For that reason, I created a function that takes an angle from 0 to 180 degrees as an arguments and converts it to a compare value for the CCR register. By observations, I noticed that the CCR values 55, 155 and 255 correspond to 0, 90 and 180 degrees. I used those values as a starting and reference point for my formula. This is the result: compare_value = (angle * 1.07) + 55 + (angle * 0.05). I have reached that formula based on trial and calibration. I adjusted multiplications and aditions based, on observations and calculation. For example if we use this formula to calculate the compare value to achieve an angle of 90 degrees is: compare_value=(90*1.07)+55+(90*0.05). The result is 155.8 which is off by 0.8 which is a negligible amount for applications using the sg90 servo. To make the servo sway from side to side, I am using the same approach as dimming the led. The difference her is that there is a 1 second delay when the servo reaches 0, 90 and 180 degrees.

Conclusion

Timers are a great tool if we need to control PWM devices. They are also a good option to control devices, without consuming processor power. This assignment helped me achieve a better understanding of the concept of PWM and how timers work.

References

STM32 F303RE and AM-Cortex documentation

SG90 datasheet

https://www.youtube.com/watch?v=zkrVHIcLGww&t=326s