Embedded systems 3

Assignment 4

Abstract: This assignment is about the implementation of PWM signal on a Servo and an LED using the external timers on the STM32 Nucleo board

Made by: Stefan Andonov Date: 27/09/22

Introduction

In this assignment we are going to investigate the usage of general-purpose timers and their application in terms of outputting a PWM signal. This assignment is split into three parts – lighting up an LED at 75% brightness, making an LED pulse and controlling a servo motor with PWM. Everything runs on the same circuit at the same time using 2 different timers.

Procedure

The first step in our solution is to enable the peripheral clock for the timers we're using – in our case we're using TIM2 and TIM3. We are going to configure them in 2 separate functions.

TIM3

We're only using this timer to control the servo. To achieve that, we're first going to set up the following registers:

TIMx_PSC

21.4.11 TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in "reset mode").

This is the prescaler register and we assign such a value to it, so that we get a convenient frequency later on for our PWM signal. The way it works it 'slows down' the core clock. It tells the microcontroller: 'get a tick every *prescaler value* number of times.' This is the formula for it:

FREQUENCY = CLOCK FREQ / PRESCALER

Since we want to end up with 50hZ frequency for our servo's PWM signal, we are going to set the prescaler as follows:

TIM3->PSC = 160 – 1;

We subtract one from 160, since we start counting from 0, and not one. Since the frequency of APB1ENR clock is 16mhZ, we end up with the following frequency of our timer:

*FREQUENCY = (16 * 10 ^ 6) / (16 * 10) = 10 ^ 5 = 100khZ*

TIMx_ARR

21.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2)

Bits 15:0 **ARR[15:0]**: Low Auto-reload Prescaler value
ARR is the value to be loaded in the actual auto-reload register.
Refer to the *Section 21.3.1: Time-base unit on page 603* for more details about ARR update and behavior.
The counter is blocked while the auto-reload value is null.

This is the auto-reload value. It is the ‘ceiling’ for our counter. This is the number our counter counts up to in up-counting mode. The formula then becomes

FREQUENCY = CLOCK FREQUENCY / PRESCALER / AUTO-RELOAD VALUE =
*= CLOCK FREQUENCY / (PRESCALER * AUTO-RELOAD VALUE)*

From our previous calculation, our current clock frequency is 10khZ. To match the 50hZ required frequency of the servo PWM signal, we’re going to use 2000 as an auto-reload value. Now let’s calculate the frequency again while considering the auto-reload value:

*FREQUENCY = (10 ^ 5) / (2 * 10 ^ 3) = 50hZ*

TIMx_CCR2

21.4.14 TIMx capture/compare register 2 (TIMx_CCR2)

Address offset: 0x38

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR2[31:16] (depending on timers)															
rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r

Bits 31:16 **CCR2[31:16]**: High Capture/Compare 2 value (on TIM2 and TIM5)

Bits 15:0 **CCR2[15:0]**: Low Capture/Compare 2 value

If channel CC2 is configured as output:
CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).
It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.
The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC2 output.
If channel CC2 is configured as input:
CCR2 is the counter value transferred by the last input capture 2 event (IC2). The TIMx_CCR2 register is read-only and cannot be programmed.

This register stores the compare value for our timer. When the counter reaches our compare value, something happens. In this case, the signal goes from HIGH to LOW, since we're in PWM mode 1, which is explained later in the document.

We want to start the servo in a centered position, which requires a duty cycle of 7.5%. This means that we need 7.5% of the reload value to be stored in the compare value register.

*75/100 * 2000 = 150*

Then we store the value in the register as follows:

TIM3->CCR2 = 150 – 1;

We subtract –1 again, since we start counting from 0.

TIMx_CNT

21.4.10 TIMx counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31] or UIFCPY	CNT[30:16] (depending on timers)														
rw or r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This is the value that our timer increments. We reset the counter by assigning ‘0’ to this register, like this:

TIM3->CNT = 0;

TIMx_CR1

21.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIF RE-MAP	Res.	CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This register is for enabling the counter. We need to set the first bit in the register to start counting, like this:

TIM3->CR1 |= 0b1;

TIMx_CCMR1

21.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.
0000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).
0001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
0010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
0011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.
0100: Force inactive level - OC1REF is forced low.
0101: Force active level - OC1REF is forced high.
0110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).
0111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

**OC2M works analogically to OC1M*

This register is for configuring different I/O modes of the timers. Since we're working with the servo, we need to configure the output mode to PWM mode 1. PWM mode 1 tells the pin to go 'HIGH' until the counter reaches the compare value mentioned earlier in the document. The servo is connected to pin PA4, which is channel 2 of the timer we're using. This means that we need to write the following in the register, according to the datasheet:

TIM3->CCMR1 |= 0b110 << 12;

TIMx_CCER

21.4.9 TIMx capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

Finally, we are going to enable the output compare mode for channel 3 and the timer is now set up.

TIM3->CCER |= 0b1 << 4;

TIM2

I am using this timer to output a constant 75% duty cycle PWM signal to an LED, pulse an LED between 20% and 100% duty cycle and as a basic timer using interrupts. Everything works analogically to TIM3, except that I enable output compare for 2 separate channels with the TIMx_CCER register.

I am also using PWM mode 2 to set the constant 75% duty cycle, so the signal starts low and only goes high when the counter reaches the compare value. This means that the compare value is selected as 25% of the auto-reload value so that we get 75% 'HIGH'.

As mentioned previously, I am also using this timer as a basic timer. I achieve this by enabling interrupts and in the interrupt handler I only increment the 'time' variable. The frequency of the timer is 1khZ, which means we get a tick every 1 millisecond.

To enable interrupts on channel one we use the following line register:

TIMx_DIER

21.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

To enable interrupts on channel 1 of timer 2 we do it like this:

TIM2->DIER |= TIM_DIER_CC1E;

TIMx_SR

21.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

This is the status register for the timer and we need to reset it each time an interrupt is generated. We do it like this:

TIM2->SR &= ~TIM_SR_CC1IF;

Proof of concept

I decided to make the following video as a proof of concept.

[ES3 timers PWM output]

References

es/Timers/Lab_04_PWM_Output

Stm32 Intro To timers

Stm32 Timers in PWM mode

es/Timers