REPORT

Title: PWM Out – Servo Motor

Submission Date	2021.10.29	Major	Mechanical and control engineering
Subject	Embedded Controller	Professor	Young-Keun Kim
Name	Yeong-Won Song	Student Number	21700375
Partner Name	Sang-Hyeon-Kim	Partner Number	21700102

Contents

I. Introdution

1.1 Purpose

In this lab, we are required to create a simple program that control a sevo motor with PWM output. Create HAL drivers for Timer and PWM control and use these APIs for the lab.

1.2 Parts List

Parts	Specification	Quantity
RC Servo-Motor	SG90	1
breadboard	-	1
NUCLEO-F411RE	Arm®(a) Cortex®-M4 with FPU	1

Table 1. Part List

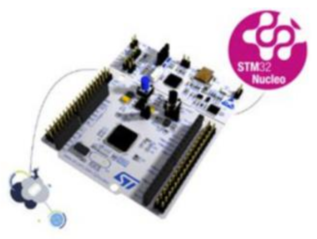
		
NUCLEO-F411RE	RC Servo-Motor	Breadboard

Table 2. experiment equipment

II. Procedure

2.1 Create EC_HAL functions

Include File	Function	Description
ecGPIO.h,c	//modify functions or add new functions to allow AF mode for TIMx	

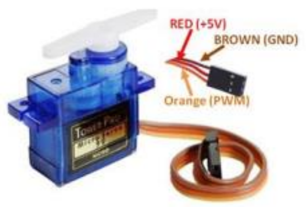
ecTIM.h,c	voidTIM_init(TIM_TypeDef *timerx, uint32_t msec);
	void TIM_period_us(TIM_TypeDef* timx, uint32_t usec);
	voidTIM_period_ms(TIM_TypeDef* timx, uint32_t msec);
	voidTIM_INT_init(TIM_TypeDef* timerx, uint32_t msec);
	void TIM_INT_enable(TIM_TypeDef* timx);
	void TIM_INT_disable(TIM_TypeDef* timx);
	uint32_t is_UIF(TIM_TypeDef *TIMx);
	void clear_UIF(TIM_TypeDef *TIMx);
		Initialize timer counter period of usec. For Timerx=TIM1, TIM2,...
		
		Update Interrupt

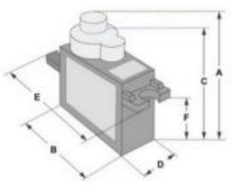
ecPWM.h,c	typedef struct{
	    GPIO_TypeDef *port;
	    int pin;
	    TIM_TypeDef *timer;
	    int ch;
	} PWM_t;
	voidPWM_init(PWM_t *pwm, GPIO_TypeDef *port, int pin);
	voidPWM_period_ms(PWM_t *pwm, uint32_t msec);
	voidPWM_period_us(PWM_t *PWM_pin, uint32_t usec);
	voidPWM_pulsewidth_ms(PWM_t*pwm,floatpulse_width_ms);
	void PWM_duty(PWM_t *pwm, float duty);
		PWM_t is a structure type for initializing GPIO port, pin and the nubmer of timer, channel. You can use this variable as a handler of PWM signal.

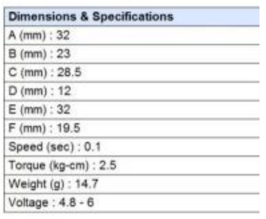
2.2 RC Servo motor: RC Servo Motor (SG90)

An RC servo motor is a tiny and light weight motor with high output power. It is used to control rotation angles, approximately 180 degrees (90 degrees in each direction) and commonly applied in RC car, and Small-scaled robots.

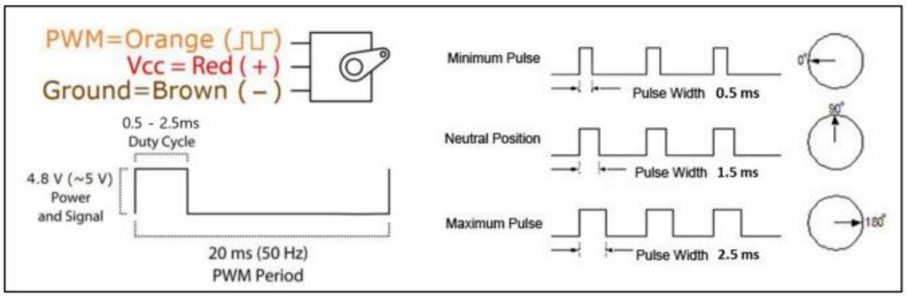
The angle of the motor can be controlled by the pulse width (duty ratio) of PWM signal. The PWM period should be set at 20ms or 50Hz. Refer to the data sheet of the RC servo motor for detailed specifications.







Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6



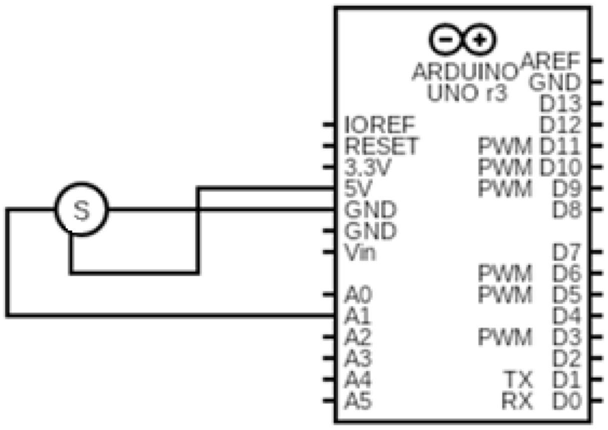


Figure 1. External Circuit

2.3 Configuration

Create a new project named as “**LAB_PWM_Servo**”.

Name the source file as “**LAB_PWM_Servo.c**”

You MUST write your name in the top of the source file, inside the comment section.

- **Configure Input and Output pins**

Digital In: Button	**Digital Out:**
GPIOC, Pin 13 Digital Input Set PULL-UP	PA1 AF output Push-Pull No Pull-up Pull-down Fast
TIMER	**PWM**
TIM2: Counter Period 1kHz	TIM2_CH2: GPIO A, Pin 1 PWM period: 20ms PWM duty ratio: 0.5ms to 2.5ms

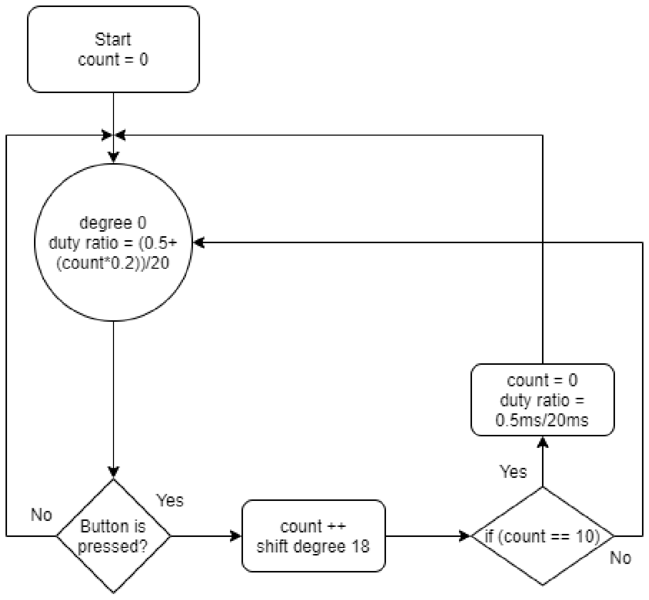


Figure 2. Flow chart

2.4 RC Servo Motor control

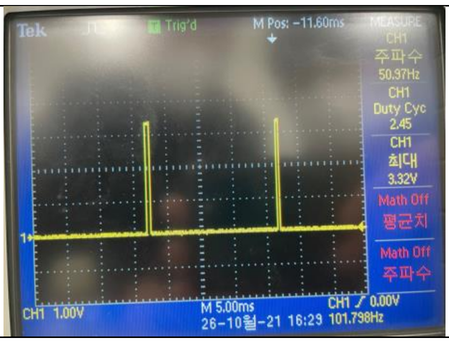
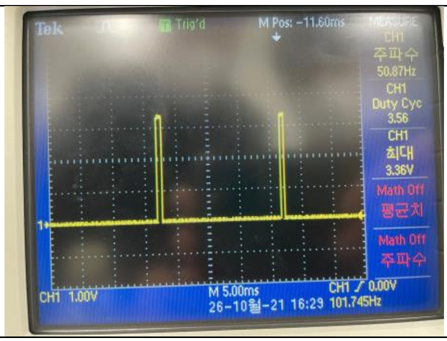
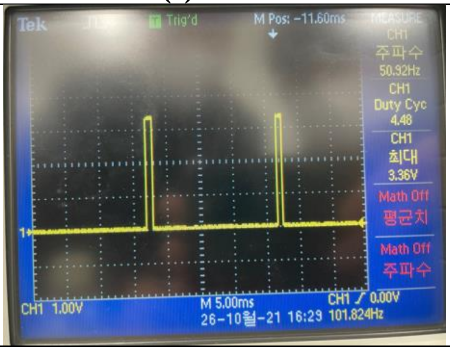
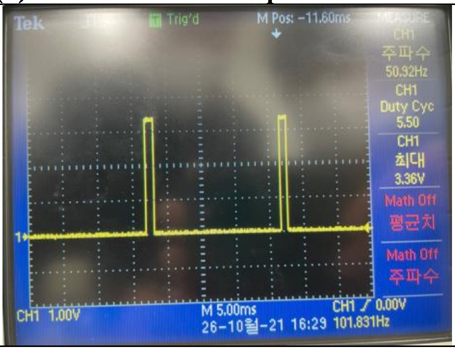
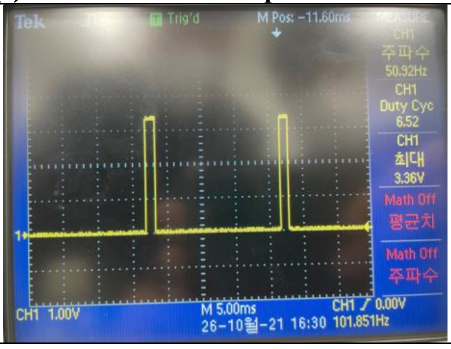
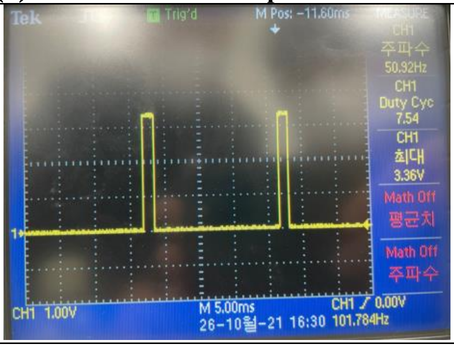
Make a simple program that changes the angle of the RC servo motor by pressing the push button (PC13).

- The button input must be External Interrupt
- Use Port A Pin 1 as PWM output pin, for TIM2_Ch2.

Increase the angle of RC servor motor from 0° to 180° each time you push the button. After reaching 180°, decrease the angle back to 0°.

- Divide 180° into 10 intervals.

You need to observe how the PWM signal output is generated as input button is pushed, using an oscilloscope. You need to captute the Oscilloscope output in the report.

	
(a) Reset mode	**(b) When the button is pressed 1 time**
	
(c) When the button is pressed 2 times	**(d) When the button is pressed 3 times**
	
(e) When the button is pressed 4 times	**(f) When the button is pressed 5 times**

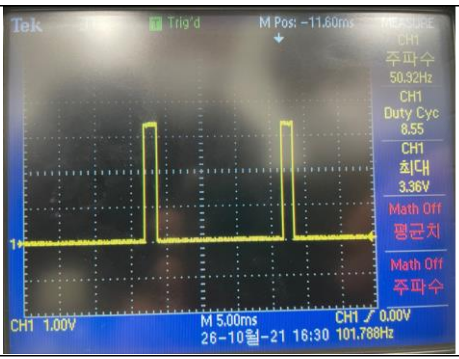
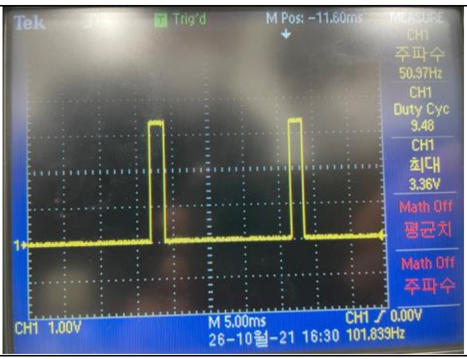
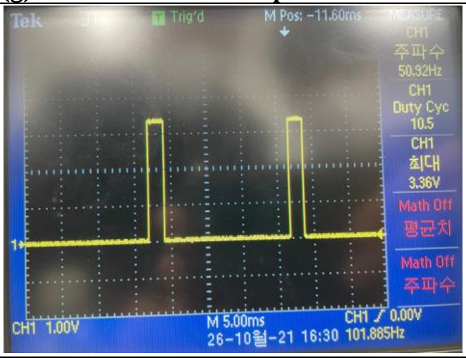
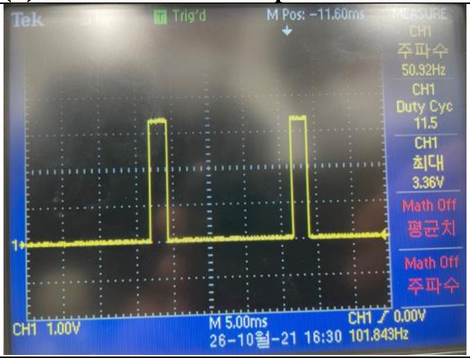
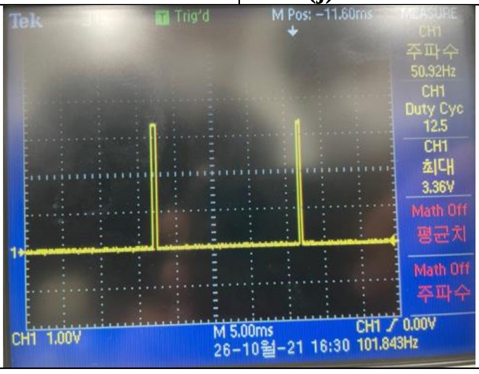
 Tek Trig'd M Pos: -11.60ms CH1 주파수 50.92Hz CH1 Duty Cyc 8.55 CH1 최대 3.36V Math Off 평균치 Math Off 주파수 CH1 1.00V M 5.00ms CH1 ／ 0.00V 26-10월-21 16:30 101.788Hz	 Tek Trig'd M Pos: -11.60ms CH1 주파수 50.97Hz CH1 Duty Cyc 9.48 CH1 최대 3.36V Math Off 평균치 Math Off 주파수 CH1 1.00V M 5.00ms CH1 ／ 0.00V 26-10월-21 16:30 101.839Hz
(g) When the button is pressed 6 times	**(h) When the button is pressed 7 times**
 Tek Trig'd M Pos: -11.60ms CH1 주파수 50.92Hz CH1 Duty Cyc 10.5 CH1 최대 3.36V Math Off 평균치 Math Off 주파수 CH1 1.00V M 5.00ms CH1 ／ 0.00V 26-10월-21 16:30 101.885Hz	 Tek Trig'd M Pos: -11.60ms CH1 주파수 50.92Hz CH1 Duty Cyc 11.5 CH1 최대 3.36V Math Off 평균치 Math Off 주파수 CH1 1.00V M 5.00ms CH1 ／ 0.00V 26-10월-21 16:30 101.843Hz
(i) When the button is pressed 8 times	**(j) When the button is pressed 9 times**
 Tek Trig'd M Pos: -11.60ms CH1 주파수 50.92Hz CH1 Duty Cyc 12.5 CH1 최대 3.36V Math Off 평균치 Math Off 주파수 CH1 1.00V M 5.00ms CH1 ／ 0.00V 26-10월-21 16:30 101.843Hz	
(k) When the button is pressed 10 times	

Figure 4. The duty cycle of the servo motor output to the Oscilloscope

- Discussion

1) Derive a simple logic to calculate for CCR and ARR values to generate $x[Hz]$ and y% duty ratio of PWM. How can you read the values of input clock frequency and PSC?

In this LAB, PLL clock was used. First, f_{sys_CLK} is lowered to enter the time counter through a PSC

$$\frac{f_{sys\_CLK}}{(psc+1)} = f_{cnt\_CLK}$$

Since we must set the period, we want to generate an event, we set the value to the desired interrupt frequency through the ARR value.

$$\frac{f_{sys\_CLK}}{(psc+1)(ARR+1)} = f_{update}$$

In this way, we have determined the frequency to generate the event and can determine the duty ratio in PWM through CCR values.

We will explain using the TIM_period_ms() function we created in this lab.

If an msec value is added to the TIM_period_ms() function, an event should be generated every 1 ms. We made 10[kHz] through PSC, and 10[kHz] has a cycle of 0.1[ms], so a total of 10 steps must pass to be 1 ms. Therefore, the ARR value is set to 9.

If you want to set the duty ratio to 50%, you must receive only 5 steps out of 10 steps. If so, the CCR value must be 5 so that we can set the duty ratio we want as 50%. If this is implemented with simple logic, it is as follows.

$$Duty\ ratio = \frac{CCR}{ARR+1} \times 100\ [\%]$$

To use the system clock, HSI, HSE, and PLLON are first turned on. Then, the unlocked bit is automatically assigned to HSIRDY, HSERDY, and PLLRDY. Therefore, check each bit of HSIRDY, HSERDY, and PLLRDY in the RCC_CR register to find out which clock is used. After that, the corresponding clock frequency is returned. The psc value is obtained through the mathematical correlation expression between the system input clock frequency and the count clock frequency value.

2) What is the smallest and highest PWM frequency that can be generated for Q1?

Previously, we set the values of PSC, ARR, and CCR to create the frequency and duty ratio of the desired PWM.

$$\frac{f_{sys\_CLK}}{(psc+1)(ARR+1)} = f_{update}$$

The PSC register and the ARR register may allocate a value of up to 65535 at 16-bits. That is, it would be convenient to give a PSC value of 84000 to make 84MHz a cycle of 1 ms, but the actual PSC value is maximum of 65535, so the PSC value cannot be set to 84000. Therefore, a desired PWM frequency may be set through an appropriate combination of PSC and ARR values. Therefore, if there is a frequency $\frac{f_{sys\_CLK}}{(psc+1)} = f_{cnt\_CLK}$ that is prescaled, f_{pwm} is minimized when the ARR value is 65535, and f_{pwm} is maximized when the ARR value is 0.

$$\frac{f_{cnt\_CLK}}{ARR_{min}+1} = f_{pwm\_max}$$

$$\frac{f_{cnt\_CLK}}{ARR_{max}+1} = f_{pwm\_min}$$

3) What is the major difference of advanced timer and general-purpose timer?

TiM1 and TIM8 is advanced-control timers. TIM2~5, TIM9~14 is general purpose timers. The advanced-control timers consist of a 16-bit auto-reload counter and the general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler. Advanced-control timers and general-purpose timers have a common role. They may be used for a variety of purposes which is measuring the pulse lengths of input signals or generating output waveforms. They have independent 4 channels for input capture, Output capture, PWM generation and one-pulse mode output. However, there are three major differences between advanced timer and general-purpose timer. Advanced timer can generate complementary PWM with programmable dead-time. The dead time is necessary to prevent the short circuit. Second, it can break input to put the timer's output signals in reset state or in a known state. Also, it is repetition counter that update the timer registers only after a given number of cycles of the counter.

III. Conclusion

The functions used for Timer and PWM were directly created, and the servo motor was operated using the created functions. In addition, the duty cycle was checked using oscilloscope. Through this process, we were able to improve our understanding by reviewing what we learned in class, and to learn about the operation of the servo motor.

In the process of conducting this experiment, Oscilloscope was used for the first time in a long time, so it was difficult to measure the duty cycle with inexperienced manipulation. Fortunately, however, we were able to get the data we wanted safely through the help of our friends around us and our continuous efforts.

In the remaining LAB, we will try to experiment with familiarity with the equipment to be used so that there is no problem like above.

- Troubleshooting

Q. How did you solve the problem that the servo motor did not rotate as much as the value of the duty ratio?

A. The CCR value on the pwm_duty() function should be $(\mathrm{ARR}+1) \times \text{duty ratio}$, but the CCR value was $\mathrm{CCR} = (\mathrm{ARR}+1) \times \text{duty ratio} - 1$. Recognizing that it was an error in the code, it was confirmed that it worked accurately after correcting the code.

Q. What is the solution to operating the servo motor without using the pwm_duty() function?

A. Instead of the pwm_duty() function, the pwm_pulsewidth_ms() function is used.
In the pwm_pulsewidth_ms() function, given the pulse_width_ms variable, the CCR value is calculated, so we can easily implement it by adding the pulse width without calculating the duty ratio.

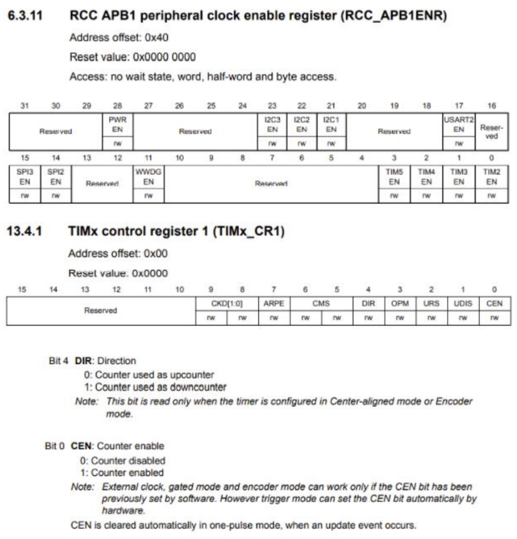
Appendix

- **Video demo Link**

  Click here watch video.

- **Documentation**

TIM_init()



6.3.11 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PWR EN	Reserved				I2C3 EN	I2C2 EN	I2C1 EN	Reserved			USART2 EN	Reser-ved
			rw					rw	rw	rw				rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved							TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw								rw	rw	rw	rw

13.4.1 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
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 4 **DIR**: Direction
  0: Counter used as upcounter
  1: Counter used as downcounter
  Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 0 **CEN**: Counter enable
  0: Counter disabled
  1: Counter enabled
  Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.
  CEN is cleared automatically in one-pulse mode, when an update event occurs.

```c
void TIM_init(TIM_TypeDef *timerx, uint32_t msec);
```

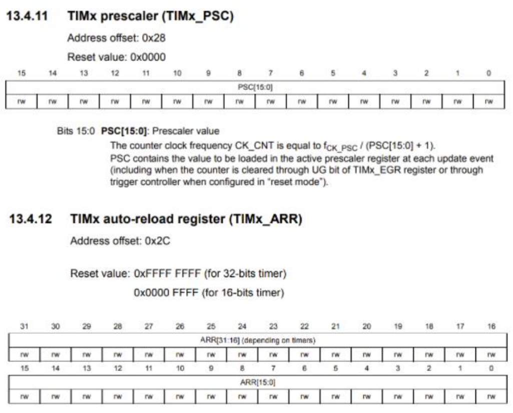
Parameters

- timex : Timer number
- integer : msec

Example code

```
TIM_init(TIM2, 20); //Initialize Timer2 20ms
```

TIM_period_us()



13.4.11 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").

13.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF (for 32-bits timer)
0x0000 FFFF (for 16-bits timer)

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

```
void TIM_period_us(TIM_TypeDef* timx, uint32_t usec);
```

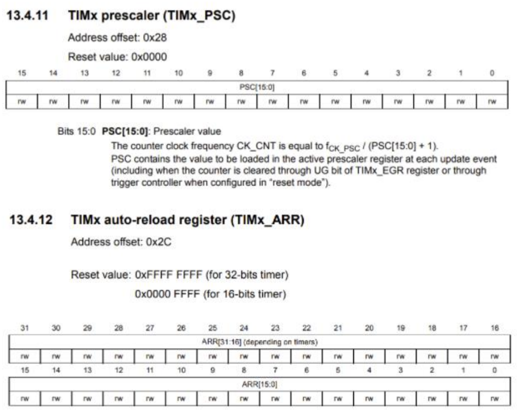
Parameters

- timex : Timer number
- integer : usec

Example code

```
TIM_TIM_period_us(TIM2, 1); //Initialize Timer2 1us
```

TIM_period_ms()



13.4.11 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").

13.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF (for 32-bits timer)
0x0000 FFFF (for 16-bits timer)

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

```
void TIM_period_ms(TIM_TypeDef* timx, uint32_t msec);
```

Parameters

- timex : Timer number
- integer : msec

Example code

```
TIM_period_ms(TIM2, 20); //Initialize Timer2 20ms
```

TIM_INT_init()

Initialize timerx and enable Update Event Interrupt.

```
void TIM_INT_init(TIM_TypeDef* timerx, uint32_t msec);
```

Parameters

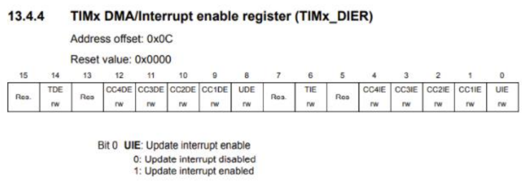
- timex : Timer number
- integer : msec

Example code

```
TIM_INT_init(TIM2, 20); //Initialize Timer2 20ms
```

TIM_INT_enable()

Enable Timer Update Interrupt



13.4.4 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

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled
1: Update interrupt enabled

```
void TIM_INT_enable(TIM_TypeDef* timx);
```

Parameters

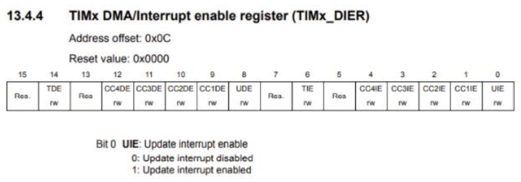
- timex : Timer number

Example code

```
TIM_INT_init(TIM2); //enable Timer2
```

TIM_INT_disable()

disable Timer Update Interrupt



13.4.4 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

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled
1: Update interrupt enabled

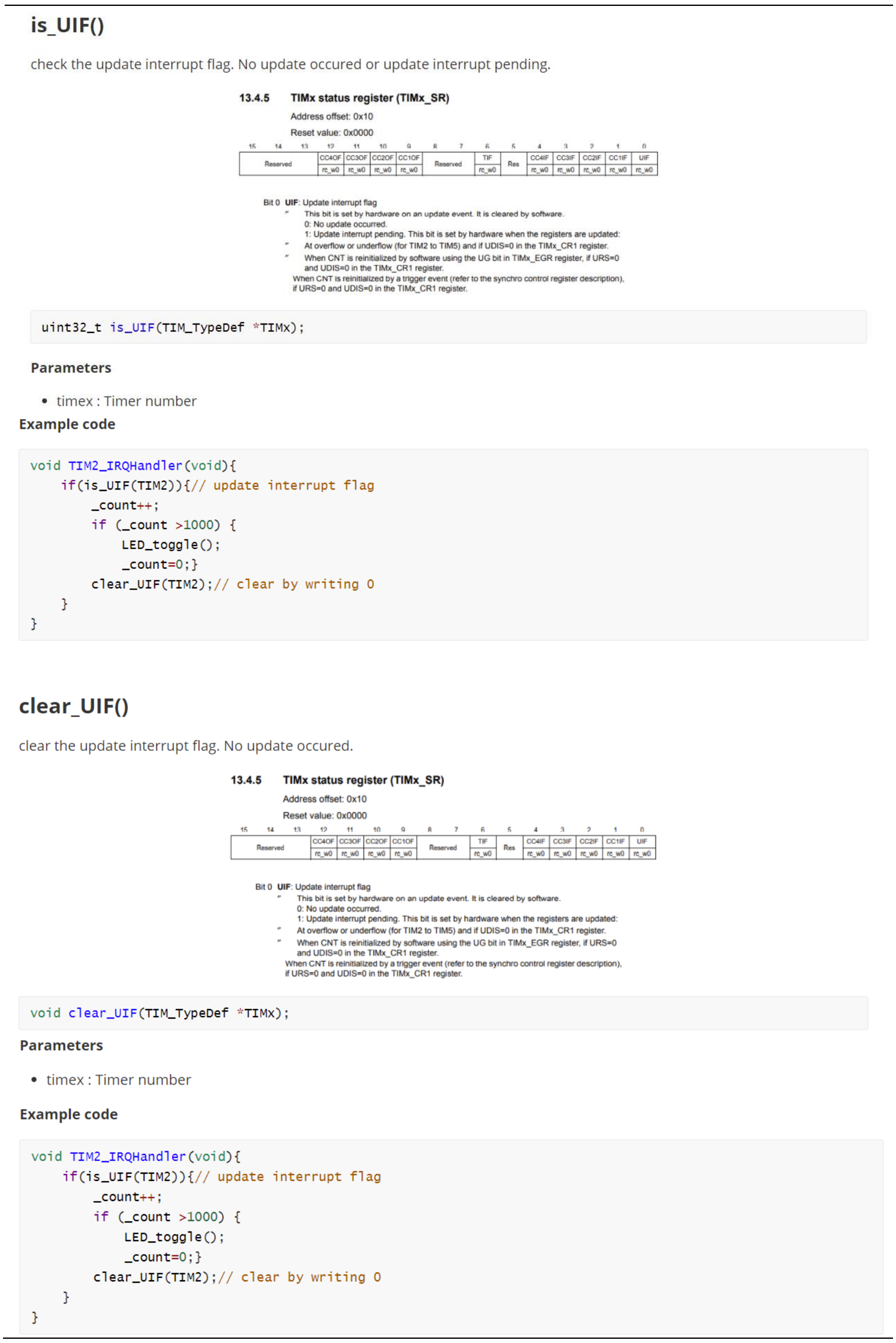
```
void TIM_INT_disable(TIM_TypeDef* timx);
```

Parameters

- timex : Timer number

Example code

```
TIM_INT_disablet(TIM2, 20); //disable Timer2
```



is_UIF()

check the update interrupt flag. No update occured or update interrupt pending.

13.4.5 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
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		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

Bit 0 **UIF**: Update interrupt flag

" This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

" At overflow or underflow (for TIM2 to TIM5) and if UDIS=0 in the TIMx_CR1 register.

" When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

```
uint32_t is_UIF(TIM_TypeDef *TIMx);
```

Parameters

- timex : Timer number

Example code

```
void TIM2_IRQHandler(void){
    if(is_UIF(TIM2)){// update interrupt flag
        _count++;
        if (_count >1000) {
            LED_toggle();
            _count=0;}
        clear_UIF(TIM2);// clear by writing 0
    }
}
```

clear_UIF()

clear the update interrupt flag. No update occured.

13.4.5 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
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		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

Bit 0 **UIF**: Update interrupt flag

" This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

" At overflow or underflow (for TIM2 to TIM5) and if UDIS=0 in the TIMx_CR1 register.

" When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

```
void clear_UIF(TIM_TypeDef *TIMx);
```

Parameters

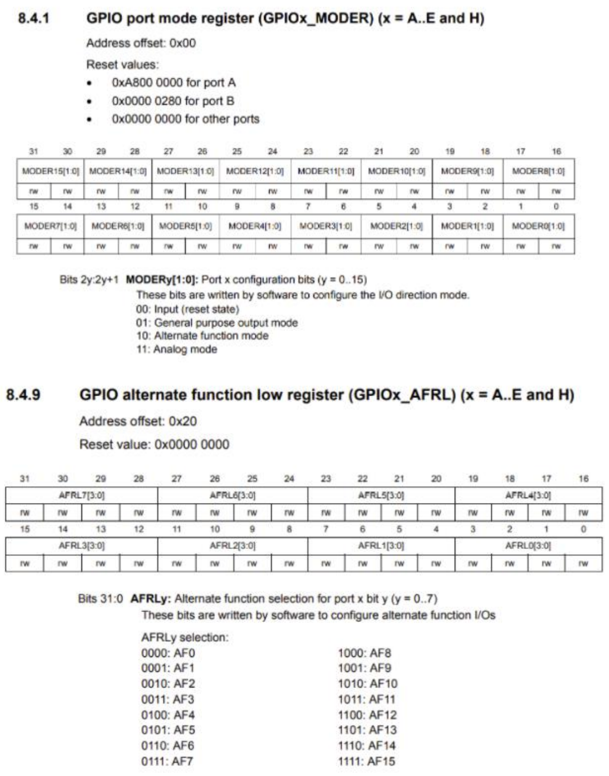
- timex : Timer number

Example code

```
void TIM2_IRQHandler(void){
    if(is_UIF(TIM2)){// update interrupt flag
        _count++;
        if (_count >1000) {
            LED_toggle();
            _count=0;}
        clear_UIF(TIM2);// clear by writing 0
    }
}
```

Figure 5. documentation of TIMER

PWM_init()



8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

8.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..E and H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
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
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRLy:** Alternate function selection for port x bit y (y = 0..7)
These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

```
void PWM_init(PWM_t *pwm, GPIO_TypeDef *port, int pin);
```

Parameters

- pwm : typedef pwm structure
- port : GPIOx port
- pin : Output pin number

Example code

```
PWM_init(&pwm,GPIOA,PA1);
```

PWM_period_ms()

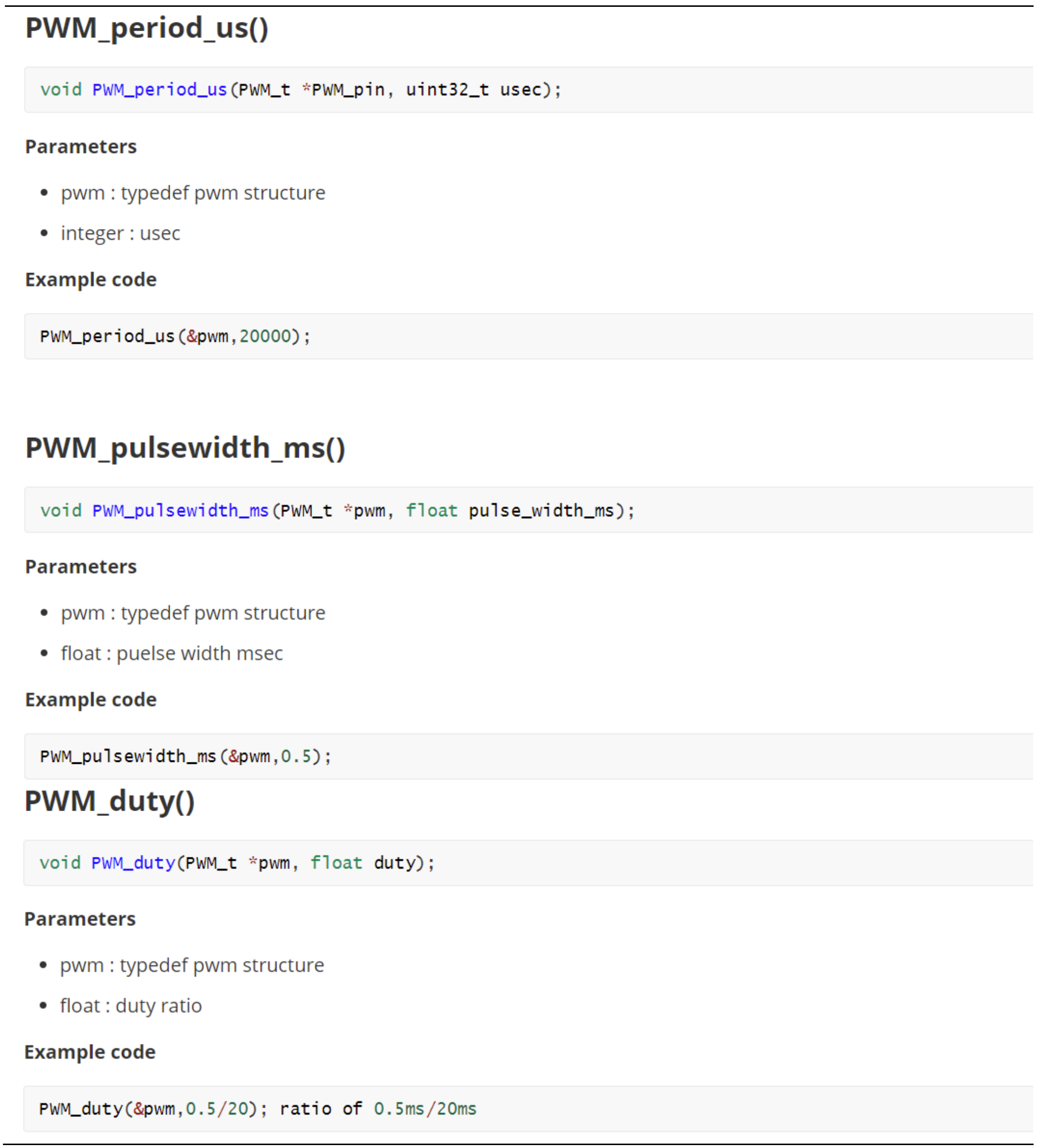
```
void PWM_period_ms(PWM_t *pwm,  uint32_t msec);
```

Parameters

- pwm : typedef pwm structure
- integer : msec

Example code

```
PWM_period_ms(&pwm,20);
```



PWM_period_us()

```
void PWM_period_us(PWM_t *PWM_pin, uint32_t usec);
```

Parameters

- pwm : typedef pwm structure
- integer : usec

Example code

```
PWM_period_us(&pwm,20000);
```

PWM_pulsewidth_ms()

```
void PWM_pulsewidth_ms(PWM_t *pwm, float pulse_width_ms);
```

Parameters

- pwm : typedef pwm structure
- float : puelse width msec

Example code

```
PWM_pulsewidth_ms(&pwm,0.5);
```

PWM_duty()

```
void PWM_duty(PWM_t *pwm, float duty);
```

Parameters

- pwm : typedef pwm structure
- float : duty ratio

Example code

```
PWM_duty(&pwm,0.5/20); ratio of 0.5ms/20ms
```

Figure 6. documentation of PWM