LAB: Timer & PWM – Servo motor and DC motor

Date: 2023-10-25

Author: Gyeonheal An

Github: https://github.com/AnGyeonheal/Embedded_Control_GH

Demo Video: https://www.youtube.com/shorts/B1C2QEXgeFs

I. Introduction

Create a simple program that control a servo motor and a DC motor with PWM output.

i. Requirement

Hardware

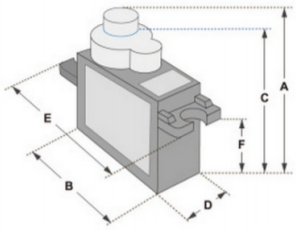
- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 3 LEDs and load resistance
  - RC Servo Motor (SG90)
  - DC motor (5V)
  - DC motor driver(LS9110s)
  - breadboard

Software

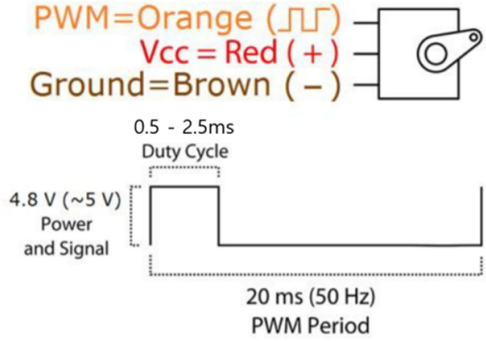
- Keil uVision, CMSIS, EC_HAL library
- Clion(with PlatformIO core plugin)
  Library: STM32Cube library package(Official), EC_HAL library
  Compiler: GNU Arm Embedded Toolchain
  Debugger: ST-Link

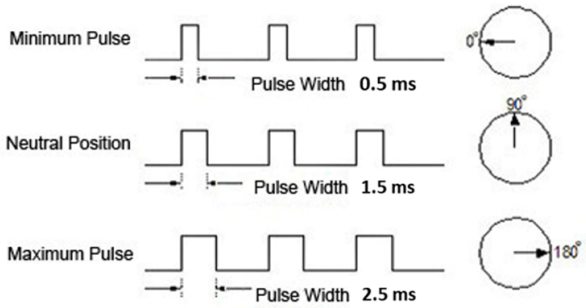
II. Problem 1: RC servo motor

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 datasheet 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





i. Create HAL library

You can read my header code in my Github link

ecTIM.h

```
// Timer Period setup
void TIM_init(TIM_TypeDef *TIMx, uint32_t msec);
void TIM_period(TIM_TypeDef* TIMx, uint32_t msec);
void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec);
void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec);
```

```
// Timer Interrupt setup
void TIM_UI_init(TIM_TypeDef* TIMx, uint32_t msec);
void TIM_UI_enable(TIM_TypeDef* TIMx);
void TIM_UI_disable(TIM_TypeDef* TIMx);
```

```
// Timer Interrupt Flag
uint32_t is_UIF(TIM_TypeDef *TIMx);
void clear_UIF(TIM_TypeDef *TIMx);
```

ecPWM.h

```
/* PWM Configuration using PinName_t Structure */
```

```
/* PWM initialization */
// Default: 84MHz PLL, 1MHz CK_CNT, 50% duty ratio, 1msec period
void PWM_init(PinName_t pinName);
void PWM_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN);
```

```
/* PWM PERIOD SETUP */
// allowable range for msec:  1~2,000
void PWM_period(PinName_t pinName,  uint32_t msec);
void PWM_period_ms(PinName_t pinName,  uint32_t msec);  // same as PWM_period()
```

```
// allowable range for usec:  1~1,000
void PWM_period_us(PinName_t pinName, uint32_t usec);
```

```
/* DUTY RATIO SETUP */
// High Pulse width in msec
void PWM_pulsewidth(PinName_t pinName, uint32_t pulse_width_ms);
void PWM_pulsewidth_ms(PinName_t pinName, uint32_t pulse_width_ms);  // same as
void PWM_pulsewidth
```

```
// Duty ratio 0~1.0
void PWM_duty(PinName_t pinName, float duty);
```

ii. Procedure

Make a simple program that changes the angle of the RC servo motor that rotates back and forth from 0 deg to 180 degree within a given period of time.

Reset to '0' degree by pressing the push button (PC13).

- Button input has to be an External Interrupt
- Use Port A Pin 1 as PWM output pin for TIM2_CH2.
- Use Timer interrupt of period 500msec.
- Angle of RC servo motor should rotate from 0° to 180° and back 0° at a step of 10° at the rate of 500msec.

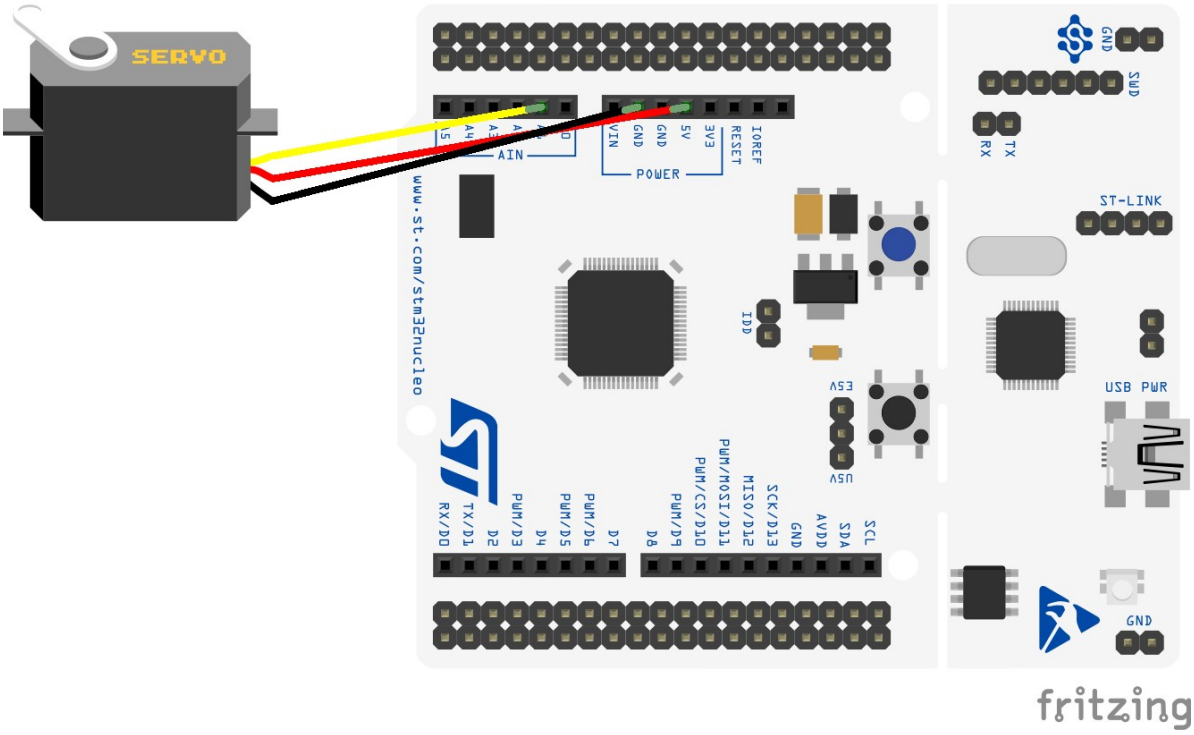
You need to observe how the PWM signal output is generated as the input button is pushed, using an oscilloscope. You need to capture the Oscilloscope output in the report.

1. Connect the RC servo motor to MCU pin (PA1) , VCC and GND
2. Increase the angle of RC servo motor from 0° to 180° with a step of 10° every 500msec. After reaching 180°, decrease the angle back to 0°. Use timer interrupt IRQ.
3. When the button is pressed, it should reset to the angle 0° and start over. Use EXT interrupt.

iii. Configuration

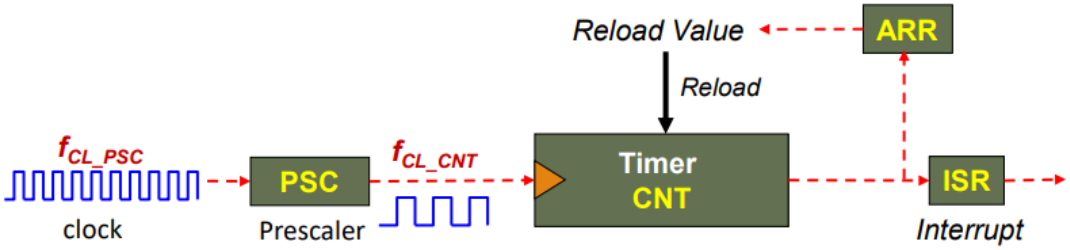
Type	Port - Pin	Configuration
Button	Digital In (PC13)	Pull-Up
PWM Pin	AF (PA1)	Push-Pull, Pull-Up, Fast
PWM Timer	TIM2_CH2 (PA1)	TIM2 (PWM) period: 20msec, Duty ratio: 0.5~2.5msec
Timer Interrupt	TIM3	TIM3 Period: 1msec, Timer Interrupt of 500 msec

iv. Circuit Diagram



v. Discussion

1. **Derive a simple logic to calculate CRR 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?**



(1). PSC

First, adjust the pulse of the system clock with the PSC value. In this case, the PSC value may be calculated as follows.

The PSC calculation method is shown in the SPAC sheet of the STM board.

$$PSC = \frac{System\ Clock}{Setting\ CLK} - 1$$

For example, if you want to use PLL (84Mhz) for System CLK and set Setting CLK to 100 kHz, the PSC value can enter 839, creating a Pulse waveform with 100,000 cycles per second.

(2). ARR

If you set the Setting CLK by setting the PSC value, set the ARR value to determine how many counts to generate Overflow or Underflow.

$$Clock\ Period = \frac{(1 + ARR)}{Setting\ CLK}$$

Therefore, when it is Up/DownCounting, the ARR value can be obtained through the following equation.

$$ARR = Setting\ CLK \times Clock\ Period - 1$$

For example, if you want to create a counting period of 1ms

Substitute Setting CLK 100kHz and Count Period 1msec obtained by setting the above PSC into the equation.

$$ARR = 100 \times 10^3 \times 1 \times 10^{-3} - 1 = 99$$

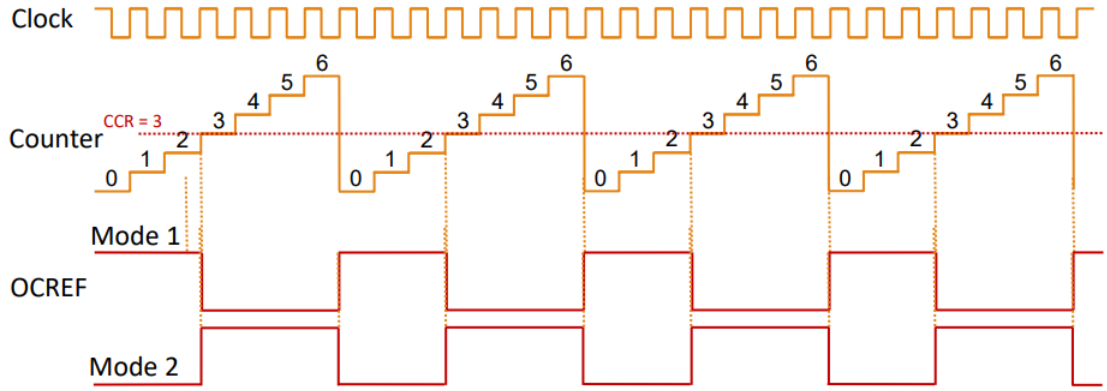
Therefore, when the PSC value is set to 839 and the ARR value to 99, a reference counter period of 1 ms may be created.

After that, if the reference value is repeated 1000 times in the main statement, an event occurrence interval of 1 second can be created.

※ Note that in TIM2 and TIM5, the ARR value may have up to 32 bits and the rest may have up to 16 bits.

(3). CCR

CCR is a reference value and the duty cycle value before exceeding the CCR value is regarded as the duty cycle value, and the duty ratio can be obtained by dividing the duty cycle value obtained through this by pwm period.



$$Duty\ Cycle = \frac{CCR}{ARR + 1}$$

$$PWM\ period = (1 + ARR) * CLK\ period$$

$$DutyRatio = \frac{Duty\ Cycle}{PWM\ period}$$

```
TIMx -> CCRy = (ARR + 1) * duty
```

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

The smallest PWM frequency has max ARR, PSC value (16bits).

$$PWM\ period = \frac{(1+ARR)^2 \times (PSC+1)}{System\ CLK}$$

$$PWM\ frequency_{min} = \frac{1}{PWM\ period} = \frac{84 \times 10^6}{(1+2^{16})^2 \times (1+2^{16})} = 2.98 \times 10^{-7} == 0$$

The highest PWM frequency has mimimum ARR, PSC value.

$$PWM\ frequency_{max} = \frac{1}{PWM\ period} = \frac{84 \times 10^6}{(1)^2 \times (1)} = 84 \times 10^6$$

vi. Code

```
#include "stm32f411xe.h"
#include "math.h"

// #include "ecSTM32F411.h"
#include "ecPinNames.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecPWM.h"
#include "ecEXTI.h"


// Definition Button Pin & PWM Port, Pin
#define BUTTON_PIN 13
PinName_t PWM_PIN = PA_1;

void setup(void);

void TIM3_IRQHandler(void);
void EXTI15_10_IRQHandler(void);

uint32_t count = 0;
float period = 20;
float n = 0;
float duty = 0;
int dir = 0;                                        // dir = 0 , 1
```

This main code include those header files and we set PWM_PIN as PA_0 from structure in Pinnames.h.

```
int main(void) {
    // Initialization --------------------------------------------------
    setup();

    // Infinite Loop ---------------------------------------------------
    while (1) {
        PWM_duty(PWM_PIN, (duty / period));
    }
}
```

In the main sentence, the PWM pulse is changed according to duty.

```
void TIM3_IRQHandler(void) {
    if (is_UIF(TIM3)) {            // Check UIF(update interrupt flag)
        count++;
        if(count > 500){
            if(dir == 0){
                n++;
                duty = 0.5 + (2 * n / 18);
                if(n == 18) dir = 1;
            }
            else if ( dir == 1){
                n--;
                duty = 0.5 + (2 * n / 18);
                if(n == 0) dir = 0;
            }
            count = 0;
        }
        clear_UIF(TIM3);            // Clear UI flag by writing 0
    }
}
```

In Timer Interrupt, the servo motor rotates from 0 degrees to 180 degrees as the duty increases every time 500 msec passes. Then, when the variable n becomes 18, the direction changes and rotates in the opposite direction from 180 degrees to 0 degrees. And when it reaches zero degrees again, the direction changes and rotates.

```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)){
        duty = 0;
        dir = 0;
        n = 0;
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}
```

When the button is pressed, the EXTI is operated, and the duty, direction, and angle values all become zero. And clear pending to receive the next Interrupt.

```
// Initialiization
void setup(void) {
    // CLK SETUP
    RCC_PLL_init();
    // EXTI: BUTTON SETUP
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
```

```
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
    // TIMER SETUP
    TIM_UI_init(TIM3, 1);
    // PWM SETUP
    PWM_init(PWM_PIN);
    PWM_period_ms(PWM_PIN, period);
}
```

The setup function sets System Clock, Button, Timer, and PWM. Timer used TIM3 and PWM_period was set to 20 msec.

vii. Results

demo video link

III. Problem 2: DC motor

i. Procedure

Make a simple program that rotates a DC motor that changes the duty ratio from 25% -->75%--> 25% --> and so on.

The rotating speed level changes every 2 seconds.

By pressing the push button (PC13), toggle from Running and stopping the DC motor

1. Connect DC motor and DC motor driver.

- PA_0 for the DC motor PWM
- PC_2 for Direction Pin

2. Change DC motor from LOW Speed to HIGH Speed for every 2 seconds

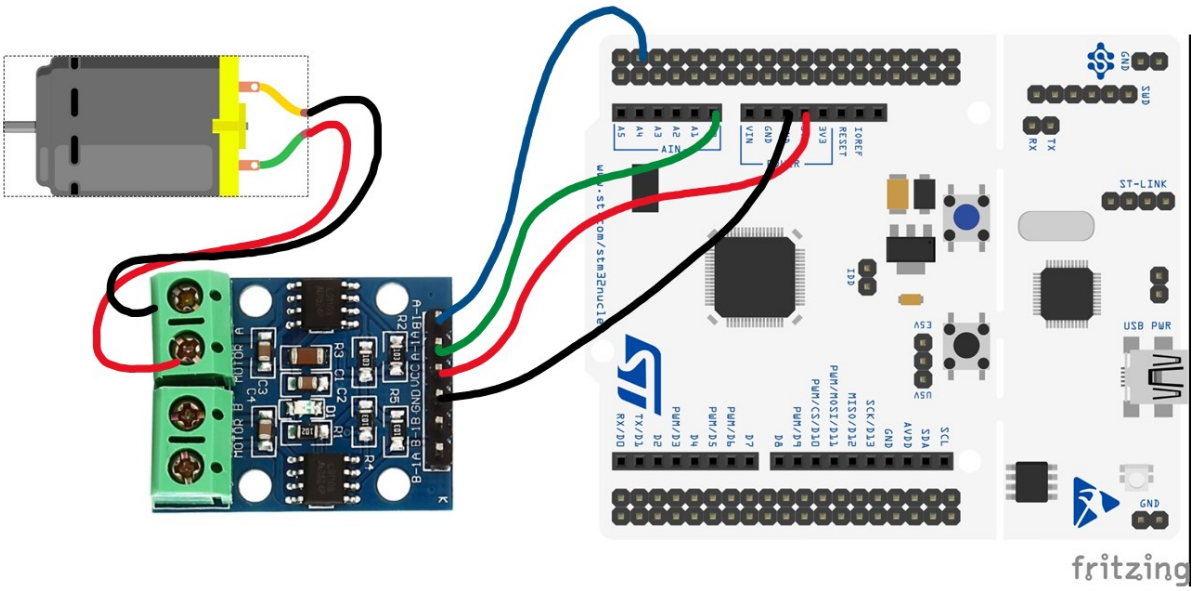
- e.g. 25% -->75%--> 25% --> and so on.

3. When Button is pressed, it should PAUSE or CONTINUE motor run

ii. Configuration

Function	Port - Pin	Configuration
Button	Digital In (PC13)	Pull-Up
Direction Pin	Digital Out (PC2)	Push-Pull
PWM Pin	AF (PA0)	Push-Pull, Pull-Up, Fast
PWM Timer	TIM2_CH1 (PA0)	TIM2 (PWM) period: **1msec (1kHz)**
Timer Interrupt	TIM3	TIM3 Period: 1msec, Timer Interrupt of 2000 msec

iii. Circuit Diagram



iv. Code

LAB_PWM_DCmotor.c

```
#include "stm32f411xe.h"
#include "math.h"

// #include "ecSTM32F411.h"
#include "ecPinNames.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecPWM.h"
#include "ecEXTI.h"


// Definition Button Pin & PWM Port, Pin
#define BUTTON_PIN 13
PinName_t PWM_PIN = PA_0;

void setup(void);

void TIM3_IRQHandler(void);
void EXTI15_10_IRQHandler(void);

uint32_t count = 0;
float period = 1;
float duty = 1;
float n = 1;
static int stop = 0;
```

This main code include those header files and we set PWM_PIN as PA_0 from structure in Pinnames.h.

```
int main(void) {
    // Initialization --------------------------------------------------
    setup();
```

```
    // Infinite Loop ------------------------------------------------------
    while (1) {
        if(stop == 0){
            PWM_duty(PWM_PIN, (duty / period));
        }
        else if(stop == 1){
            PWM_duty(PWM_PIN, (1 / period));
        }
    }
}
```

If button EXTI signal is recognized, it turns on or off.

The motor we used stops when it comes duty 1.

```
void TIM3_IRQHandler(void) {
    if (is_UIF(TIM3)) {            // Check UIF(update interrupt flag)
        count++;
        if(count > 2000 && count < 4000){
            duty = 0.75;
        }
        else if(count > 4000){
            duty = 0.25;
            count = 0;
        }
    }
    clear_UIF(TIM3);            // Clear UI flag by writing 0
}
```

If update interrupt flag is recognized, first 2 sec, duty is 0.75 which is 25% output speed.

After 2 sec, duty becomes 0.25 which is 75% output speed and clear the timer flag

```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)){
        stop ^= 1;
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}
```

If external interrupt has occurred by the button pin, it makes stop variable 1, and if EXTI occurred again, it toggles.

```
// Initialiization
void setup(void) {
    // CLK SETUP
    RCC_PLL_init();
    // EXTI: BUTTON SETUP
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
    // TIMER SETUP
    TIM_UI_init(TIM3, 1);
    // PWM SETUP
    PWM_init(PWM_PIN);
```

```
    PWM_period_ms(PWM_PIN, period);

}
```

This is the setting of System clock, button, timer, pwm.

We initialized timer TIM3, 1ms and PWM period 1 as we can see in configuration.

v. Results

demo video link

IV. Reference

Advanced Timer

Timer	Channel	Port	Pin
1	1	A	8
	1N	A	7
		B	13
	2	A	9
	2N	B	0
		B	14
	3	A	10
	3N	B	1
		B	15
	4		
	4N		

General Purpose Timer

Timer	Channel	Port	Pin
2	1	A	0
		A	5
		A	15
	2	A	1
		B	3
	3	B	10
	4		
3	1	A	6
		B	4
		C	6
	2	B	5
		C	7
	3	C	8
	4	C	9
4	1	B	6
	2	B	7
	3	B	8
	4	B	9

https://ykkim.gitbook.io/ec/ec-course/lab/lab-timer-and-pwm

V. Troubleshooting

In Problem 2, the reference value of the duty ratio is different for each DC motor, so the value at which the motor stops is set to duty ratio 1. In addition, because the computer did not provide enough power to USB, the motor received a signal but did not rotate when the motor was at 25% speed due to weak output.