LAB: Timer & PWM

Date : 2023.10.22

Author/Partner: Duwon Yang

Github : LINK

Demo Video : P1 LINK / P2 LINK

Introduction

In this lab, We create a simple program using TIMER that control a servo motor and a DC motor with PWM output.

We used Nucleo-F411RE to implement this program, and the library by creating a HAL driver for GPIO digital input and output control.

Requirement

Hardware

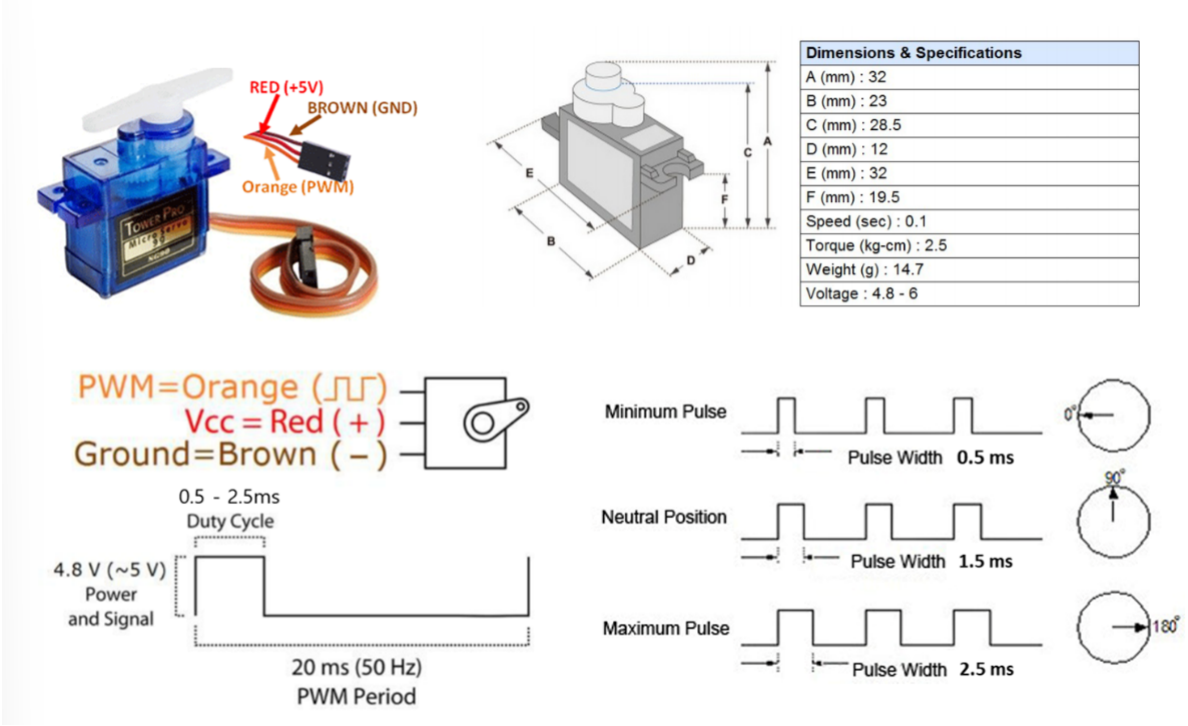
- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 3 LEDs and load resistance
  - RC Servo Motor (SG90)
  - DC motor (5V)
  - DC motor driver(LS9110s)
  - breadboard

Software

- Keli uVision, CMSIS, EC_HAL library

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 (90degrees in each direciton) and commonly applied in RC car, and Small-scaled robots. The angle of 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.



1-1. Create HAL library

save **ecTIM.h, ecTIM.c, ecPWM.h, ecPWM.c** in directory `EC \lib\`

Declare the following functions in library : **ecTIM.h, ecPWM.h**

ecEXTI.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);
```

1-2. 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 $\degree$ to 180 $\degree$ and back 0 $\degree$ at a step of 10 at the rate of 500msec

1. Create a new project under the directory `\repos\EC\LAB\LAB_PWM`
    - The project name is **"LAB_PWM"**
    - Create a new source file named as **"LAB_PWM_RCmotor.c"**
2. Include library in `\repos\EC\lib\`
    - **"ecPinNames.h, ecPinNames.c"**
    - **"ecGPIO.h, ecGPIO.c"**
    - **"ecRCC.h, ecRCC.c"**
    - **"ecEXTI.h, ecEXTI.c"**
    - **"ecTIM.h, ecTIM.c"**
    - **"ecPWM.h, ecPWM.c"**
3. Connect the RC servo motor to MCU pin (PA1) ,VCC and GND
4. Increase the angle of RC servo motor from 0 $\degree$ to 180 $\degree$ with a step of 10 $\degree$ every 500msec. After reaching 180 $\degree$, decrease the angle back to 0 $\degree$. Use time

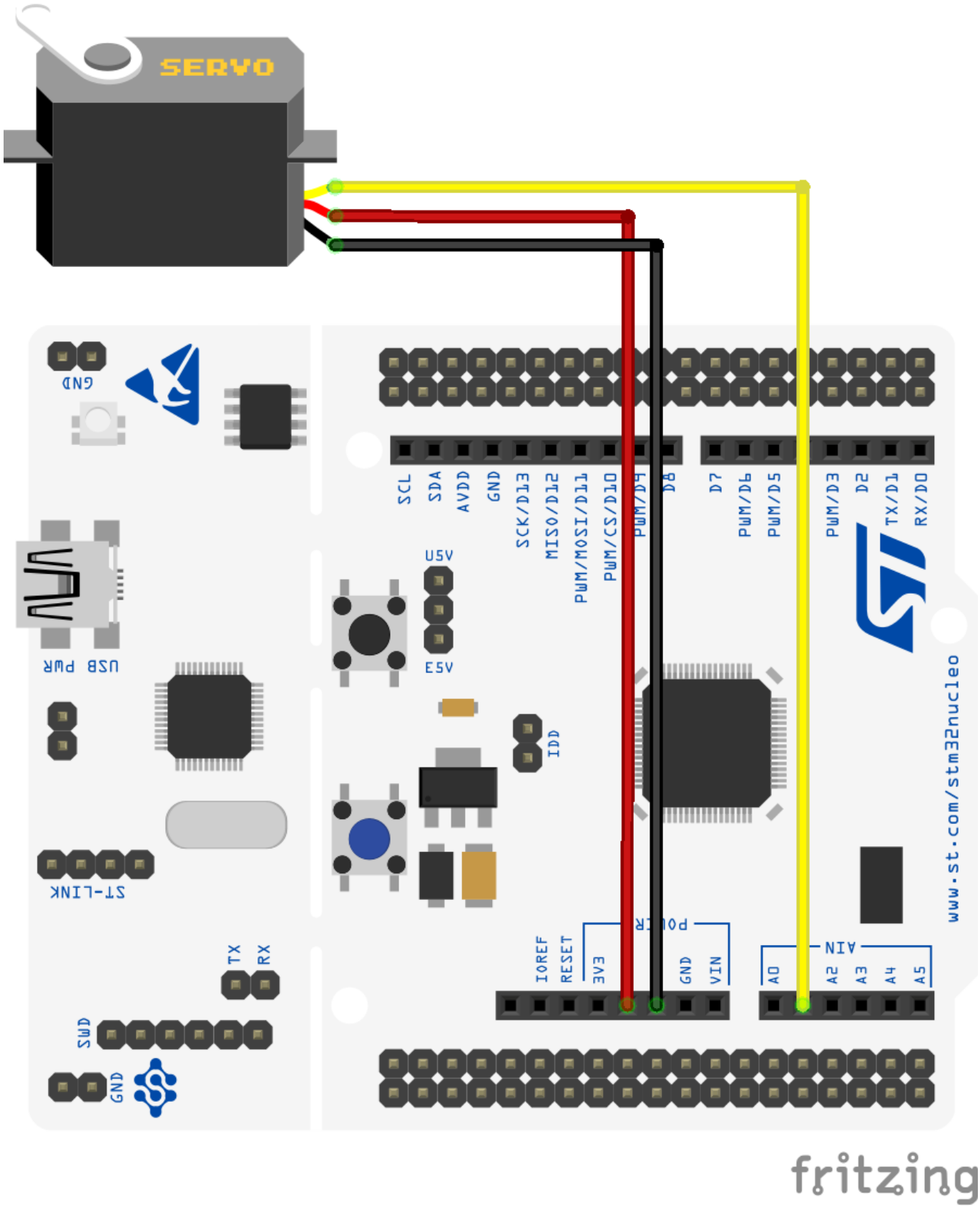
interrupt IRQ.

5. When the button is pressed, it should reset to the angle 0 $\degree$ and start over. Use EXT interrupt.

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

Circuit Diagram



Discussion

1. Derive a smiple logic to calculate 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?

1. select the type of Clock(HSI/HSE/PLL)

   determine the input clock frequency (sys_clk)

2. Calculate the prescaler value

$$PSC = \frac{Sys\ CLK}{Setting\ CLK} - 1$$

3. Calculate ARR value based on the desired frequency x[Hz] of PWM

$$ARR = \frac{Setting\ CLK}{x} - 1$$

4. Calculate CCR value based on the desired y[%] duty ratio of PWM

$$CCR = y * (ARR + 1)$$

How can you read the values of input clock frequency and PSC?

access to RCC_CFGR register and use the bitwise operator '&' to verify the desired clock configuration

```
if((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)
    Sys_CLK = 84000000;

else if((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI)
    Sys_CLK = 16000000;
```

6.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
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
PPRE2[2:0]			PPRE1[2:0]			Reserved		HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

Bits 1:0 **SW:** System clock switch

Set and cleared by software to select the system clock source.
Set by hardware to force the HSI selection when leaving the Stop or Standby mode or in case of failure of the HSE oscillator used directly or indirectly as the system clock.
00: HSI oscillator selected as system clock
01: HSE oscillator selected as system clock
10: PLL selected as system clock
11: not allowed

access to TIMx->PSC register and use the bitwise operator

```
unsigned int bit;
bit = TIMx->PSC & (1<<x);
```

12.4.11 TIM1 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”).

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

$$Set\_clk = \frac{Sys\_clk}{(PSC+1)} \qquad Period_{PWM} = \frac{(1+ARR)}{Set\_clk} = \frac{(1+ARR)\times(1+PSC)}{Sys\_clk} \qquad freq_{PWM} = \frac{Sys\_clk}{(1+ARR)\times(1+PSC)}$$

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

$$PWM_{freq_{min}} = \frac{84\times10^6}{(1+2^{16})\times(1+2^{16})} \approx 0.64$$

$$PWM\_freq_{max} = \frac{84\times10^6}{(1+2^0)\times(1+2^0)} = 21\times10^6$$

Code

this code includes those header files, and we set PWM_PIN as PA1

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

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


// Definition Button Pin & PWM Port, Pin
#define BUTTON_PIN 13
#define PWM_PIN PA_1
void setup(void);
void TIM3_IRQHandler(void);
void EXTI15_10_IRQHandler(void);


uint32_t count = 0;
unsigned int cnt = 0;
float tick = 0, duty = 0;
int dir = 0;
```

In this main code, PWM pulse is changed with duty ratio

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

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

Using Timer3, the servo motor works 10 degree from 0 to 180 degrees per 500milisecs. when it becomes to 180 degree, it works 10 degree in opposite direction.

```
void TIM3_IRQHandler(void){
    if((TIM3->SR & TIM_SR_UIF) == TIM_SR_UIF){ // update interrupt flag

        if(count > (99+1) * 5){

            if(dir == 0){
                duty = (0.5+(2.0/18.0)*tick)/20;
                if((int)tick==18){ tick=0; dir = 1;}
            }
            else if(dir == 1){
                duty = (2.5 - (2.0/18.0)*tick)/20;
                if((int)tick==18){ tick=0; dir = 0;}
            }
            tick++;
            count = 0;
        }
        count++;
        TIM3->SR &= ~TIM_SR_UIF;            // clear by writing 0
    }
}
```

as the button is pressed, EXTI operator works. According to it, tick, direction, and duty value initialize to 0.

```
void EXTI15_10_IRQHandler(void){
    if (is_pending_EXTI(BUTTON_PIN)){
            tick = 0; dir = 0; duty = 0;
            clear_pending_EXTI(BUTTON_PIN);
        }
}
```

The setup function sets System clock, Button, Timer, and PWM. We used Timer3 and PWM_PIN period was setted as 20milisec.

```
void setup(void) {
    RCC_PLL_init();
    SysTick_init();

    // PWM of 20 msec:  TIM2_CH1 (PA_5 AFmode)
    GPIO_init(GPIOA, LED_PIN, OUTPUT);
        // Initialize GPIOC_13 for Input Button
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);


    // EXTI Initialization ------------------------------------------------------
    EXTI_init(GPIOC, BUTTON_PIN, FALL,0);


    PWM_init(PWM_PIN);
    PWM_period(PWM_PIN, 20);   // 20 msec PWM period

    TIM_UI_init(TIM3, 1);         //1msec / Time Interrupt of 500msec

}
```

Results

demo video : LINK

In this experiment, We implemented a program on the Nucleo board that can control the RC servo motor. The angle of the motor was controlled by the pulse width(duty ratio) of PWM signal. The RC servo motor works 10 degree from 0 to 180 degrees per 500milisecs. when it becomes to 180 degree, it works 10 degree in opposite direction.

Problem 2: DC motor

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. Create HAL library

save **ecSysTick.h, ecSysTick.c** in directory `EC \lib\`

Declare the following functions in library : **ecSysTick.h**

ecEXTI.h

```
void SysTick_init(uint32_t msec);
void delay_ms(uint32_t msec);
uint32_t SysTick_val(void);
void SysTick_reset (void);
void SysTick_enable(void);
void SysTick_disable (void)
```

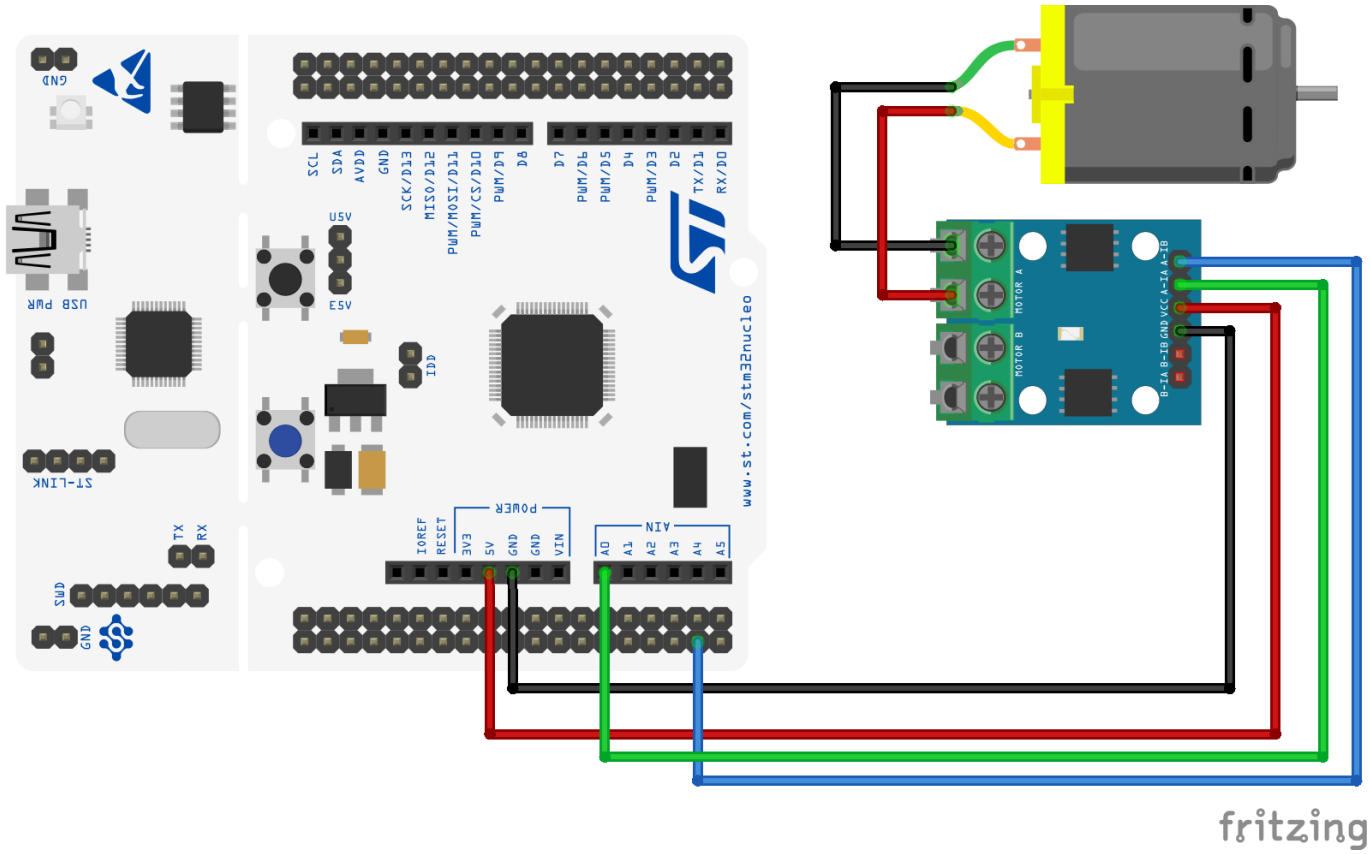
2-2. Procedure

1. Use the sam project `\repos\EC\LAB\LAB_SysTick`
    - Create a new source file named **"LAB_PWM_DCmotor.c"**
2. Connect DC motor and DC motor driver.
    - PA_0 for the DC motor PWM
    - PC_2 for Direction Pin
3. Change DC motor from LOW Speed to HIGH Speed for every 2 seconds
    - e.g. 25%-->75%-->25%--> and so on.
4. When the Button is pressed, it should PAUSE or CONTINUE motor run

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 500 msec

Circuit Diagram



Code

CODE : LINK

a DC motor changes the duty ratio from 25% > 75% > 25% >... every 2seconds. By pressing the push button (PC13), toggle from Running and stopping the DC motor

```
#define BUTTON_PIN 13
#define PWM_PIN PA_0
#define DIR_PIN PC_2
void setup(void);
void TIM3_IRQHandler(void);
void EXTI15_10_IRQHandler(void);


uint32_t count = 0;
unsigned int cnt = 0;
float duty = 0;
float period = 1;
int mode = 0;
static int stop  = 0;
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));  // HIGH -> STOP
        }
    }

    void TIM3_IRQHandler(void){
        if((TIM3->SR & TIM_SR_UIF) == TIM_SR_UIF){ // update interrupt flag
            //Create the code to toggle LED by

            if(count > (99+1) * 20){
                if(mode == 0)           {duty = 0.75; mode = 1;}
                else if(mode == 1){duty = 0.25; mode = 0;}
                count = 0;
            }
            count++;
            clear_UIF(TIM3);            // clear by writing 0
        }
    }


    void EXTI15_10_IRQHandler(void){
        if (is_pending_EXTI(BUTTON_PIN)){
                stop ^= 1 ;
                clear_pending_EXTI(BUTTON_PIN);
            }
    }
```

Results

In this experiment, We implemented a program on the Nucleo board that can work DC motor with duty ratio 25% -> 75% -> 25% -> ... per 2 seconds. as the button is pressed, DC motor mode toggles from Running and stopping mode.

demo video : LINK

Reference

https://ykkim.gitbook.io/ec/ec-course/lab/lab-report-template

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

https://ykkim.gitbook.io/ec/ec-course/tutorial/tutorial-dcmotor-motor-driver-connection