LAB: Timer & PWM

LAB: PWM – Servo motor and DC motor

Date: 2023-10-25

Author: Han TaeGeon

Demo Video: Problem1 : https://youtu.be/GMjwl4zIrc8

Problem2 : https://youtu.be/1A320E_q1kY

Introduction

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

You must submit

- LAB Report (*.md & *.pdf)
- Zip source files(main*.c, ecRCC.h, ecGPIO.h, ecSysTick.c etc...).
  - Only the source files. Do not submit project files

Requirement

Hardware

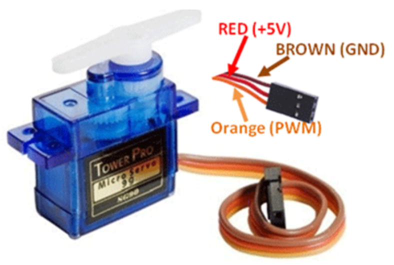
- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 3 LEDs and load resistance
  - RC Servo Motor (SG90)
  - DC motor (5V)
  - DC motor driver(LS9110s)
  - breadboard

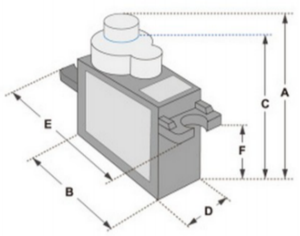
Software

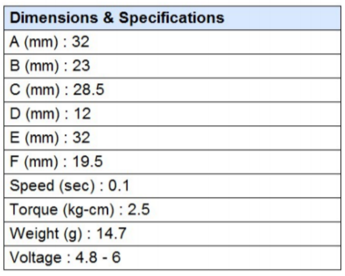
- Keil uVision, CMSIS, EC_HAL library

Problem 1: RC servo motor

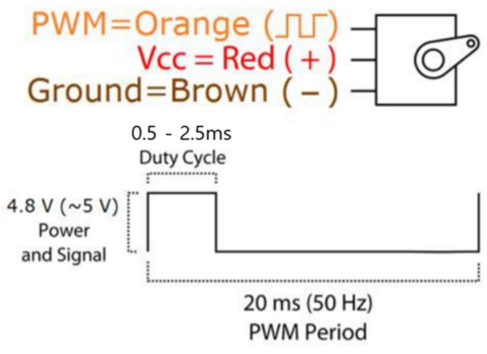
An RC servo motor is a tiny and light weight motor with high output power. It is used to control rotation angles, approximately 180 degrees (90 degrees in each direction) and commonly applied in RC car, and Small-scaled robots. The angle of the motor can be controlled by the pulse width (duty ratio) 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.

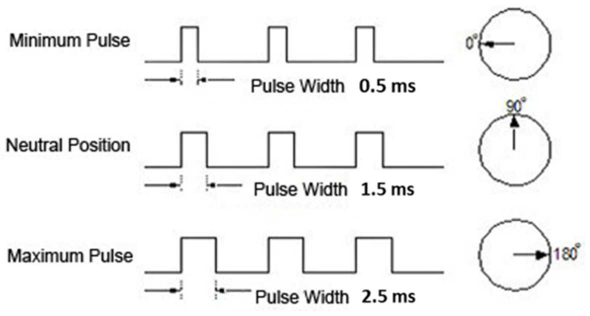






Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6





1-1. Create HAL library

Download files:

- ecPinNames.h ecPinNames.c
- ecTIM_student.h, ecTIM_student.c
- ecPWM_student.h, ecPWM_student.c

Then, change the library files as

- ecTIM.h, ecTIM.c
- ecPWM.h, ecPWM.c

Declare and define the following functions in your library. You must update your header files located in the directory `EC \lib\`.

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);
```

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. 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 your updated library in `\repos\EC\lib\` to your project.

- **ecPinNames.h ecPinNames.c**
- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecEXTI.h, ecEXTI.c**
- **ecTIM.h**, **ecTIM.c**
- **ecPWM.h ecPWM.h**

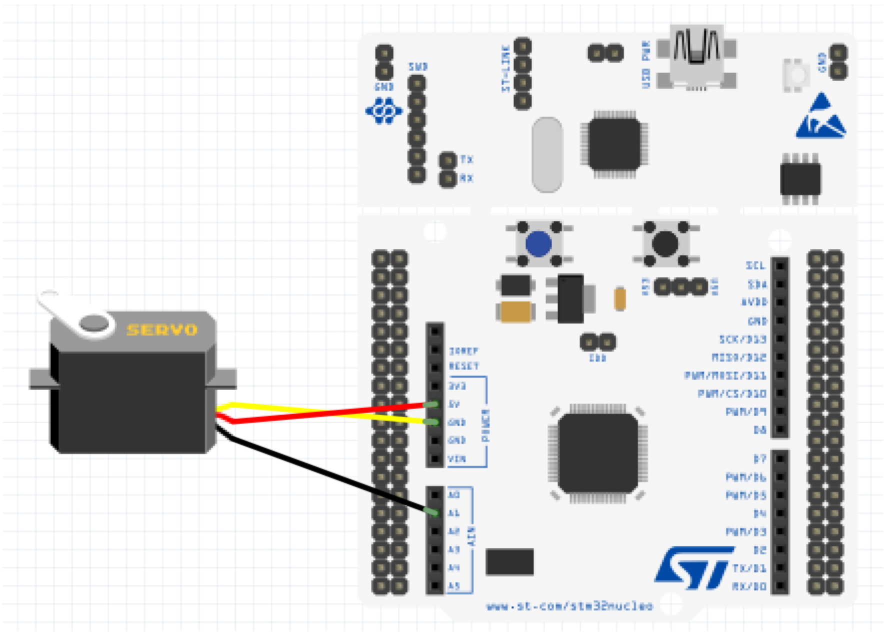
3. Connect the RC servo motor to MCU pin (PA1) , VCC and GND

4. 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.
5. When the button is pressed, it should reset to the angle 0° 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 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?

   There is a simple logic to calculate the CRR and ARR values to generate the x[Hz] and y[%] duty ratios of PWM. You must first define the frequency before the PSC is entered. The PSC value must then be set and the cycle must be determined using the following equation.

$$f_{CLCNT} = \frac{f_{CLPSC}}{PSC + 1}$$

After that, the Count Period(Event Frequency) is changed according to the ARR value. Here, if the ARR value is set too small, it is not good because the PWM level becomes sparse. Therefore, it should be set to an appropriate value that is not too small.

$$f_{eventf} = \frac{f_{CKCNT}}{ARR+1}$$
$$f_{eventf} = \frac{f_{CLPSC}}{(PSC+1)(ARR+1)}$$

Substituting the first equation into the second equation, as in the above equation, allows us to obtain the frequency of the counter by one equation. In addition, the event frequency can be updated according to the PSC and ARR. That is, we can control the PWM period by adjusting these two values.

In case of Up-Counting mode, modes are divided into 1 and 2. In Mode 1, if CCR > CNT, PWM is 'high'. In Mode 2, PWM becomes 'LOW' if CCR > CNT. Therefore, the equation for the Dutycycle is as follows.

$$Mode1 : DutyCycle = \frac{CCR}{ARR+1}$$
$$Mode2 : DutyCycle = 1 - \frac{CCR}{ARR+1}$$

In case of Center-aligned mode, there is only one mode. CNT > CCR is 'High' and CNT < CCR is 'Low'. The equation for the Dutycycle is as follows.

$$DutyCycle = 1 - \frac{CCR}{ARR}$$

Therefore, you cannot read input clock frequency or PSC unless you know any value in any mode. If ARR and CCR are known, input clock frequency and PSC can be calculated. In addition, even if you know the value of either the input clock frequency or the PSC, you can know the value of the other.

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

The 84 MHz system clock was used as an example to solve the problem. And we can find the PWM frequency through the following equation.

$$f_{PWM} = \frac{f_{CLPSC}}{(PSC+1)(ARR+1)}$$
$$f_{CLPSC} = 84MHz$$

ARR can be represented from 0 to 65535. Since ARR is in the denominator, PWM frequency is minimized if ARR is at its maximum, and PWM frequency is at its maximum if it is at its minimum. The PSC is also 16-bit, so it's the same as ARR.

$$f_{PWMmax} = \frac{84M}{(0+1)(0+1)} = 84MHz$$
$$f_{PWMmin} = \frac{84M}{(65535+1)(65535+1)} = 0.02Hz$$

Code

Your code goes here: ADD Code LINK such as github

There are all other header files in the gather.h. And the codes below set up what is needed to solve the problem

```
#include "stm32f411xe.h"
#include "math.h"
#include "gather.h"
#define BUTTON_PIN 13
#define PWM_PIN PA_1
void setup(void);
void EXTI15_10_IRQHandler(void);

static volatile int dir = 1;                // direction of motor
static volatile float duty = 0.5;           // PWM duty
static volatile uint32_t count = 0;         // Count for 0.5s
static volatile uint32_t State = 0;         // Button_State
static volatile uint32_t period = 20;       // PWM Period
```

Set the setup and execute the duty value in the while statement.

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

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

It's a code that timer interrupt and adjust duty values according to the situation here.

```
void TIM3_IRQHandler(void){
    if(is_UIF(TIM3)){            // Check UIF(update interrupt flag)
        count++;
        if (count > 500) {
            if(State == 0){
            if(duty >= 2.5 || duty < 0.5) {dir *= -1;}
            duty += (float)(dir * 0.1111);
        }
            else if(State == 1) {duty = 0.5;}
            count = 0;
            State = 0;
    }
        clear_UIF(TIM3);         // Clear UI flag by writing 0
    }
}
```

The codes below are external interrupt for Pin 13.

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

The codes below are set according to the conditions given in the question.

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

    // PWM of 20 msec:  TIM2_CH2 (PA_1 AFmode)
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    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);
}
```

Example Code

Sample Code : Timer Interrupt IRQ

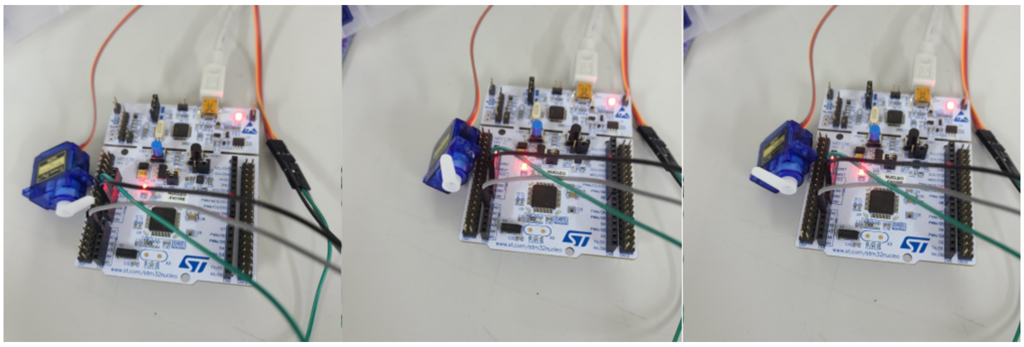
```
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"


#define LED_PIN 5
uint32_t _count = 0;
void setup(void);


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

    // Infinite Loop ---------------------------------------------------
    while(1){}
}


// Initialization
void setup(void){
    RCC_PLL_init();             // System Clock = 84MHz
    GPIO_init(GPIOA, LED_PIN, OUTPUT);  // calls RCC_GPIOA_enable()
    TIM_UI_init(TIM2, 1);           // TIM2 Update-Event Interrupt every 1 msec
    TIM_UI_enable(TIM2);
}

void TIM2_IRQHandler(void){
    if(is_UIF(TIM2)){            // Check UIF(update interrupt flag)
        _count++;
        if (_count > 1000) {
            LED_toggle();        // LED toggle every 1 sec
            _count = 0;
        }
        clear_UIF(TIM2);         // Clear UI flag by writing 0
    }
}
```

Sample Code : PWM output

```
#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 "ecPWM.h"   // ecPWM2.h


// Definition Button Pin & PWM Port, Pin
#define BUTTON_PIN 13
#define PWM_PIN PA_5
void setup(void);


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

    // Infinite Loop ---------------------------------------------------
    while(1){
        LED_toggle();
        for (int i=0; i<5; i++) {
            PWM_duty(PWM_PIN, (float)0.2*i);
            delay_ms(1000);
        }
    }
}


// Initialiization
void setup(void) {
    RCC_PLL_init();
    SysTick_init();

    // PWM of 20 msec:  TIM2_CH1 (PA_5 AFmode)
    GPIO_init(GPIOA, 5, EC_AF);
    PWM_init(PWM_PIN);
    PWM_period(PWM_PIN, 20);   // 20 msec PWM period
}
```

Results

Images



Results

The servomotor was connected to the ground, 5v, and PA_1. After that, when the code was executed, it moved by 10 degrees every 0.5 seconds. It moved from 0 degrees to 180 degrees, and when it reaches 180 degrees, it moves again to 0 degrees by 10 degrees every 0.5 seconds. If you press the button in the middle of moving, it goes back to 0 degrees and moves back 10 degrees.

Demo Video

Link : https://youtu.be/GMjwl4zIrc8

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

First, you MUST read Tutorial: DC motor driver connection

1. Use the same project.

- Create a new source file named "**LAB_PWM_DCmotor.c**"
- You need to eliminate the other source file that contains `main()` from the project
  - e.g. Eliminate ""**LAB_PWM_RCmotor.c**" from the project

> You MUST write your name on the source file inside the comment section.

2. Connect DC motor and DC motor driver.

- PA_1 for the DC motor PWM

- PC_2 for Direction Pin

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

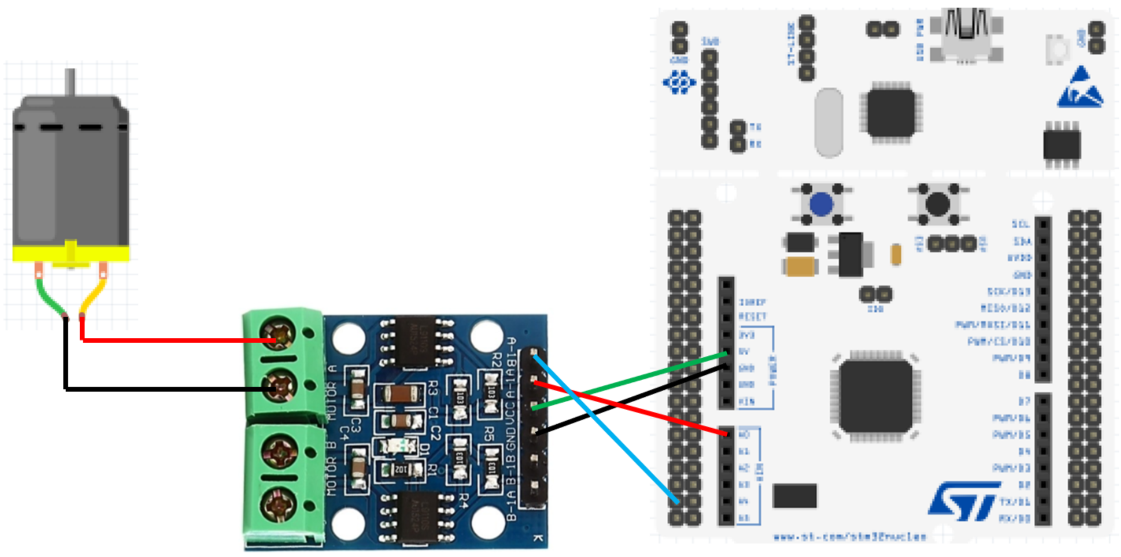
- e.g. 25% -->75%--> 25% --> and so on.

4. When 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

Your code goes here: [ADD Code LINK such as github]()

The codes below set up what is needed to solve the problem.

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

```
#include "gather.h"


// Definition Button Pin & PWM Port, Pin
#define BUTTON_PIN 13
#define Direction_PIN 2
#define PWM_PIN PA_0

void setup(void);
void EXTI15_10_IRQHandler(void);

static volatile int flag = 1;                         // variables to change duty
static volatile int dir = -1;                         // stop or continue
static volatile float duty = 0.25;            // PWM duty
static volatile uint32_t count = 0;           // Count for 0.5s
static volatile int State = 1;                    // Button_State
```

The PWM duty value is given differently depending on the situation within the main function.

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

    // Inifinite Loop ----------------------------------------------------------
    while(1){
        if(State == 1){
        PWM_duty(PWM_PIN,(float)duty);
        }
        else if(State == -1){
            PWM_duty(PWM_PIN,(float)0);
        }
    }
}
```

It's a code that timer interrupt and adjust duty values according to the situation here.

```
void TIM3_IRQHandler(void){
    if(is_UIF(TIM3)){           // Check UIF(update interrupt flag)
        count++;
        if (count > 2000){
            if(State == 1){
                flag *= -1;
                if(flag == 1)   {duty = 0.25;}
                else if(flag == -1) {duty = 0.75;}
            count = 0;
            }
        }
        clear_UIF(TIM3);        // Clear UI flag by writing 0
    }
}
```

The codes below are external interrupt for BUTTON_PIN.

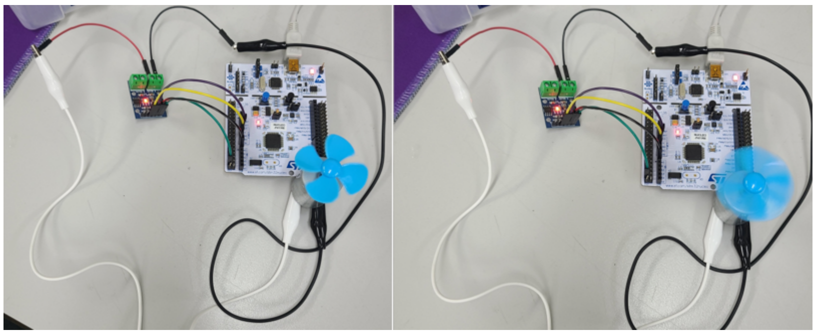
```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        State *= -1;
        dir *= -1;
        for(int i = 0; i < 50000; i++){};
        clear_pending_EXTI(BUTTON_PIN);
    }
}
```

The codes below are set according to the conditions given in the question.

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

    // PWM of 1 msec:  TIM2_CH1 (PA_0 AFmode)
    GPIO_init(GPIOC, Direction_PIN, OUTPUT);
    GPIO_pupd(GPIOC, Direction_PIN, EC_PU);


    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);

    PWM_init(PWM_PIN);
    PWM_period(PWM_PIN, 1);    // 1 msec PWM period

    TIM_UI_init(TIM3, 1);
}
```

Results

images



Results

The DC motor was connected to the DC driver and connected to 5V, ground, PWM, and PC2 related to the Direction Pin. Repeat 25% and 75% to adjust the duty ratio to adjust the speed every two seconds. Press the button to stop the motor, and press it again to return the motor to the original speed. We can check the speed of turning by plugging a pinwheel into the DC motor.

Demo Video

Link : https://youtu.be/1A320E_q1kY

Reference

Complete list of all references used (github, blog, paper, etc)

```
Young-Keun Kim (2023). https://ykkim.gitbook.io/ec/
```

Troubleshooting

Delay was given in the external interruption function. As before, I thought it was a bouncing problem because the switch button did not press well. Therefore, the problem was solved by giving delay.

The output varies depending on how the motor and clamp wire are connected. Even if it appears to be connected, it must be completely in contact to produce a proper output. In addition, it is necessary to firmly connect the jumper line and the motor driver. If you look at the driver, there is a place where it is tightened with the drive, but if it is not tightened properly, the connection becomes unstable. Therefore, the experiment was conducted by firmly fixing it with a screwdriver.

Appendix

- ecGPIO.c

```
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecGPIO.h"

void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode){
    // mode  : Input(0), Output(1), AlterFunc(2), Analog(3)
    if (Port == GPIOA)
        RCC_GPIOA_enable();
    if (Port == GPIOC)
        RCC_GPIOC_enable();
    if (Port == GPIOB)
        RCC_GPIOB_enable();
    if (Port == GPIOD)
        RCC_GPIOD_enable();

    // Make it for GPIOB, GPIOD..GPIOH

    // You can also make a more general function of
    // void RCC_GPIO_enable(GPIO_TypeDef *Port);

    GPIO_mode(Port, pin, mode);

}


// GPIO Mode          : Input(00), Output(01), AlterFunc(10), Analog(11)
void GPIO_mode(GPIO_TypeDef *Port, int pin, unsigned int mode){
   Port->MODER &= ~(3UL<<(2*pin));
   Port->MODER |= mode<<(2*pin);
}


// GPIO Speed          : Low speed (00), Medium speed (01), Fast speed (10), High
speed (11)
void GPIO_ospeed(GPIO_TypeDef *Port, int pin, unsigned int speed){
    Port->OSPEEDR &= ~(3UL<<(2*pin));
    Port->OSPEEDR |= speed<<(2*pin);
}

// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
void GPIO_otype(GPIO_TypeDef *Port, int pin, unsigned int type){
    Port->OTYPER &= ~(1UL)<< pin;
    Port->OTYPER |= (type<< pin);

}

// GPIO Push-Pull    : No pull-up, pull-down (00), Pull-up (01), Pull-down (10),
Reserved (11)
void GPIO_pupd(GPIO_TypeDef *Port, int pin, unsigned int pupd){
    Port->PUPDR &= ~(3UL<<2*pin);
    Port->PUPDR |= pupd<<(2*pin);   //write
}

int GPIO_read(GPIO_TypeDef *Port, int pin){
    unsigned int BVal = (Port->IDR)>>pin & (1);

    return BVal;
}

void GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output){
    Port->ODR &= ~(1<<pin);
    Port->ODR |= (Output << pin);
}


void sevensegment_init(void){
        // Calls RCC_GPIO_enable()
        GPIO_init(GPIOA, LED_PA5, OUTPUT);
        GPIO_init(GPIOA, LED_PA6, OUTPUT);
        GPIO_init(GPIOA, LED_PA7, OUTPUT);
        GPIO_init(GPIOB, LED_PB6, OUTPUT);
        GPIO_init(GPIOC, LED_PC7, OUTPUT);
        GPIO_init(GPIOA, LED_PA9, OUTPUT);
        GPIO_init(GPIOA, LED_PA8, OUTPUT);
        GPIO_init(GPIOB, LED_PB10, OUTPUT);
}


void sevensegment_decoder(uint8_t  num){
      //pins are sorted from upper left corner of the display to the lower right
corner
    //the display has a common cathode
    //the display actally has 8 led's, the last one is a dot
        unsigned int led[8]=
{LED_PA5,LED_PA6,LED_PA7,LED_PB6,LED_PC7,LED_PA9,LED_PA8,LED_PB10};
```

```
        //each led that has to light up gets a 1, every other led gets a 0
        //its in order of the DigitalOut Pins above
        unsigned int number[10][8]={
                                        {0,0,0,0,0,0,1,1},    //zero
                                        {1,0,0,1,1,1,1,1},    //one
                                        {0,0,1,0,0,1,0,1},    //two
                                        {0,0,0,0,1,1,0,1},    //three
                                        {1,0,0,1,1,0,0,1},    //four
                                        {0,1,0,0,1,0,0,1},    //five
                                        {0,1,0,0,0,0,0,1},    //six
                                        {0,0,0,1,1,0,1,1},    //seven
                                        {0,0,0,0,0,0,0,1},    //eight
                                        {0,0,0,0,1,0,0,1},    //nine
        };
            //all led's off
        for(int i = 0; i<8;i++){led[i] = 0;}

            //display shows the number in this case 6
        for (int i=0; i<8; i++){led[i] = number[num][i];}         //the digit
after "number" is displayed

        GPIO_write(GPIOA, LED_PA5,  led[0]);
        GPIO_write(GPIOA, LED_PA6,  led[1]);
        GPIO_write(GPIOA, LED_PA7,  led[2]);
        GPIO_write(GPIOB, LED_PB6,  led[3]);
        GPIO_write(GPIOC, LED_PC7,  led[4]);
        GPIO_write(GPIOA, LED_PA9,  led[5]);
        GPIO_write(GPIOA, LED_PA8,  led[6]);
        GPIO_write(GPIOB, LED_PB10, led[7]);
}


void sevensegment_display_init(void){
            // Calls RCC_GPIO_enable()
        GPIO_init(GPIOA, LED_PA7, OUTPUT);
        GPIO_init(GPIOB, LED_PB6, OUTPUT);
        GPIO_init(GPIOC, LED_PC7, OUTPUT);
        GPIO_init(GPIOA, LED_PA9, OUTPUT);
}

void sevensegment_display(uint8_t  num){
    //pins are sorted from upper left corner of the display to the lower right
corner
    //the display has a common cathode
    //the display actally has 8 led's, the last one is a dot
        unsigned int led[4]={LED_PA7,LED_PB6,LED_PC7,LED_PA9}; // A,B,C,D

        //each led that has to light up gets a 1, every other led gets a 0
        //its in order of the DigitalOut Pins above
        unsigned int number[10][4]={
                                        {0,0,0,0},      //zero
                                        {0,0,0,1},    //one
                                        {0,0,1,0},    //two
                                        {0,0,1,1},    //three
                                        {0,1,0,0},    //four
                                        {0,1,0,1},    //five
                                        {0,1,1,0},    //six
```

```
                                                {0,1,1,1},     //seven
                                                {1,0,0,0},     //eight
                                                {1,0,0,1},     //nine
        };
            //all led's off
        for(int i = 0; i<4;i++){led[i] = 0;}

            //display shows the number in this case 6
        for (int i=0; i<4; i++){led[i] = number[num][i];}         //the digit
after "number" is displayed

        GPIO_write(GPIOA, LED_PA7,  led[3]);
        GPIO_write(GPIOB, LED_PB6,  led[2]);
        GPIO_write(GPIOC, LED_PC7,  led[1]);
        GPIO_write(GPIOA, LED_PA9,  led[0]);

}

void LED_toggle(){
    int led_state = GPIO_read(GPIOA, LED_PIN);
    int time = 0;
    while(time < 1000)
        time++;

    GPIO_write(GPIOA, LED_PIN, !led_state);
}
```

- ecEXTI.c

```
#include "ecGPIO.h"
#include "ecSysTick.h"
#include "ecEXTI.h"


void EXTI_init(GPIO_TypeDef *Port, int Pin, int trig_type,int priority){

    // SYSCFG peripheral clock enable
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Connect External Line to the GPIO
    int EXTICR_port;
    if          (Port == GPIOA) EXTICR_port = 0;
    else if (Port == GPIOB) EXTICR_port = 1;
    else if (Port == GPIOC) EXTICR_port = 2;
    else if (Port == GPIOD) EXTICR_port = 3;
    else                                        EXTICR_port = 4;

    SYSCFG->EXTICR[Pin >> 2] &= ~(15 << 4*(Pin & 0x03));            // clear 4
bits
    SYSCFG->EXTICR[Pin >> 2] |= EXTICR_port << 4*(Pin & 0x03);           //
connect port number

    // Configure Trigger edge
    if (trig_type == FALL) EXTI->FTSR |= 1UL << Pin;   // Falling trigger enable
    else if (trig_type == RISE) EXTI->RTSR |= 1UL << Pin;   // Rising trigger
enable
```

```
    else if (trig_type == BOTH) {           // Both falling/rising trigger
enable
        EXTI->RTSR |= 1UL << Pin;
        EXTI->FTSR |= 1UL << Pin;
    }

    // Configure Interrupt Mask (Interrupt enabled)
    EXTI->IMR  |= 1 << Pin;     // not masked


    // NVIC(IRQ) Setting
    int EXTI_IRQn = 0;

    if (Pin == 0) EXTI_IRQn = EXTI0_IRQn;
    else if (Pin == 1) EXTI_IRQn = EXTI1_IRQn;
    else if (Pin == 2) EXTI_IRQn = EXTI2_IRQn;
    else if (Pin == 3) EXTI_IRQn = EXTI3_IRQn;
    else if (Pin == 4) EXTI_IRQn = EXTI4_IRQn;
    else if (Pin > 4 && Pin < 10)   EXTI_IRQn = EXTI9_5_IRQn;
    else            EXTI_IRQn = EXTI15_10_IRQn;

    NVIC_SetPriority(EXTI_IRQn, priority);  // EXTI priority
    NVIC_EnableIRQ(EXTI_IRQn);  // EXTI IRQ enable
}


void EXTI_enable(uint32_t pin) {
    EXTI->IMR |= 1UL << pin;     // not masked (i.e., Interrupt enabled)
}
void EXTI_disable(uint32_t pin) {
    EXTI->IMR |= ~(1UL << pin);     // masked (i.e., Interrupt disabled)
}

uint32_t is_pending_EXTI(uint32_t pin){
    uint32_t EXTI_PRx = pin;        // check  EXTI pending
    return ((EXTI->PR & 1 << pin) == 1 << pin);
}


void clear_pending_EXTI(uint32_t pin){
    EXTI->PR |= 1 << pin;     // clear EXTI pending
}
```

- ecSysTick

```
#include "ecSysTick.h"

#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

volatile uint32_t msTicks=0;

//EC_SYSTEM_CLK

void SysTick_init(void){
    //  SysTick Control and Status Register
    SysTick->CTRL = 0;                                          // Disable
SysTick IRQ and SysTick Counter

    // Select processor clock
    // 1 = processor clock;  0 = external clock
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
    SysTick->LOAD = MCU_CLK_PLL / 1000 - 1;                     // 1ms, for HSI
PLL = 84MHz.

    // SysTick Current Value Register
    SysTick->VAL = 0;

    // Enables SysTick exception request
    // 1 = counting down to zero asserts the SysTick exception request
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    // Enable SysTick IRQ and SysTick Timer
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

    NVIC_SetPriority(SysTick_IRQn, 16);     // Set Priority to 1
    NVIC_EnableIRQ(SysTick_IRQn);           // Enable interrupt in NVIC
}



void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(){
    msTicks++;
}


void delay_ms (uint32_t mesc){
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < mesc);

    msTicks = 0;
}

//void delay_ms(uint32_t msec){
//  uint32_t now=SysTick_val();
//  if (msec>5000) msec=5000;
//  if (msec<1) msec=1;
//  while ((now - SysTick_val()) < msec);
//}


void SysTick_reset(void)
{
    // SysTick Current Value Register
    SysTick->VAL = 0;
```

```
}

uint32_t SysTick_val(void) {
    return SysTick->VAL;
}

//void SysTick_counter(){
//  msTicks++;
//  if(msTicks%1000 == 0) count++;
//}

void SysTick_enable(void) {
    NVIC_EnableIRQ(SysTick_IRQn);
}

void SysTick_disable(void) {
    NVIC_DisableIRQ(SysTick_IRQn);
}
```

- ecPinNames.c

```
#include "ecPinNames.h"

void ecPinmap(PinName_t pinName, GPIO_TypeDef **GPIOx, unsigned int *pin)
{

    unsigned int pinNum= pinName & (0x000F);
    *pin=pinNum;

    unsigned int portNum=(pinName>>4);


    if (portNum==0)
        *GPIOx=GPIOA;
    else if (portNum==1)
        *GPIOx=GPIOB;
    else if (portNum==2)
        *GPIOx=GPIOC;
    else if (portNum==3)
        *GPIOx=GPIOD;
    else if (portNum==7)
        *GPIOx=GPIOH;
    else
        *GPIOx=GPIOA;
}
```

- ecTIM.c

```
#include "ecTIM.h"
#include "ecGPIO.h"

/* Timer Configuration */

void TIM_init(TIM_TypeDef* TIMx, uint32_t msec) {

    // 1. Enable Timer CLOCK
    if (TIMx == TIM1) RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
    else if (TIMx == TIM2) RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    else if (TIMx == TIM3) RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    // repeat for TIM4, TIM5, TIM9, TIM11
  else if (TIMx == TIM4) RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;
    else if (TIMx == TIM5) RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;
    else if (TIMx == TIM9) RCC->APB2ENR |= RCC_APB2ENR_TIM9EN;
    else if (TIMx == TIM11) RCC->APB2ENR |= RCC_APB2ENR_TIM11EN;


// 2. Set CNT period
    TIM_period_ms(TIMx, msec);


    // 3. CNT Direction
    TIMx->CR1 &= ~(1 << 4);                    // Upcounter

    // 4. Enable Timer Counter
    TIMx->CR1 |= TIM_CR1_CEN;
}


//  Q. Which combination of PSC and ARR for msec unit?
//  Q. What are the possible range (in sec ?)
void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec) {
    // Period usec = 1 to 1000

    // 1us(1MHz, ARR=1) to 65msec (ARR=0xFFFF)
    uint16_t PSCval;
    uint32_t Sys_CLK;

    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;


    if (TIMx == TIM2 || TIMx == TIM5) {
        uint32_t ARRval;

        PSCval = Sys_CLK / 1000000;                             // 84 or 16
--> f_cnt = 1MHz
        ARRval = Sys_CLK / PSCval / 1000000 * usec;     // 1MHz*usec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
    else {
        uint16_t ARRval;

        PSCval = Sys_CLK / 1000000;                                 // 84 or
16 --> f_cnt = 1MHz
        ARRval = Sys_CLK / PSCval / 1000000 * usec;     // 1MHz*usec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
}
```

```
void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec) {
    // Period msec = 1 to 6000

    // 0.1ms(10kHz, ARR = 1) to 6.5sec (ARR = 0xFFFF)
    // uint16_t PSCval = 8400;
    // uint16_t ARRval = ______________;            // 84MHz/1000ms
    //
    // TIMx->PSC = PSCval - 1;
    // TIMx->ARR = ARRval;
    uint16_t PSCval;
    uint32_t Sys_CLK;

    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;


    if (TIMx == TIM2 || TIMx == TIM5) {
        uint32_t ARRval;

        PSCval = Sys_CLK / 100000;                              // 840 or
160   --> f_cnt = 100kHz
        ARRval = Sys_CLK / PSCval / 1000 * msec;        // 100kHz*msec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
    else {
        uint16_t ARRval;

        PSCval = Sys_CLK / 10000;                               // 8400 or
1600 --> f_cnt = 10kHz
        ARRval = Sys_CLK / PSCval / 1000 * msec;        // 10kHz*msec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
}

// msec = 1 to 6000
void TIM_period(TIM_TypeDef* TIMx, uint32_t msec){
    TIM_period_ms(TIMx, msec);
}

// Update Event Interrupt
void TIM_UI_init(TIM_TypeDef* TIMx, uint32_t msec) {
    // 1. Initialize Timer
    TIM_init(TIMx, msec);

    // 2. Enable Update Interrupt
    TIM_UI_enable(TIMx);

    // 3. NVIC Setting
    uint32_t IRQn_reg = 0;
    if (TIMx == TIM1)         IRQn_reg = TIM1_UP_TIM10_IRQn;
    else if (TIMx == TIM2)    IRQn_reg = TIM2_IRQn;
    // repeat for TIM3, TIM4, TIM5, TIM9, TIM10, TIM11
```

```
  else if (TIMx == TIM3)    IRQn_reg = TIM3_IRQn;
    else if (TIMx == TIM4)    IRQn_reg = TIM4_IRQn;
    else if (TIMx == TIM5)    IRQn_reg = TIM5_IRQn;
    else if (TIMx == TIM9)    IRQn_reg = TIM1_BRK_TIM9_IRQn;
    else if (TIMx == TIM10)  IRQn_reg = TIM1_UP_TIM10_IRQn;
    else if (TIMx == TIM11)  IRQn_reg = TIM1_TRG_COM_TIM11_IRQn;

    NVIC_EnableIRQ(IRQn_reg);
    NVIC_SetPriority(IRQn_reg, 2);
}



void TIM_UI_enable(TIM_TypeDef* TIMx) {
    TIMx->DIER |= TIM_DIER_UIE;         // Enable Timer Update Interrupt
}

void TIM_UI_disable(TIM_TypeDef* TIMx) {
    TIMx->DIER &= ~TIM_DIER_UIE;                // Disable Timer Update
Interrupt
}

uint32_t is_UIF(TIM_TypeDef* TIMx) {
    return TIMx->SR & TIM_SR_UIF;
}

void clear_UIF(TIM_TypeDef* TIMx) {
    TIMx->SR &= ~TIM_SR_UIF;
}
```

- ecPWM.c

```
#include "stm32f4xx.h"
#include "ecPWM.h"
#include "math.h"
#include "ecPinNames.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){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Initialize GPIO port and pin as AF
    GPIO_init(port, pin, AF);  // AF=2
    GPIO_otype(port, pin, EC_PUSH_PULL);
    GPIO_pupd(port, pin, EC_PU);
    GPIO_ospeed(port, pin, EC_FAST);
```

```
// 2. Configure GPIO AFR by Pin num.
    //  AFR[0] for pin: 0~7,     AFR[1] for pin 8~15
    //  AFR=1 for TIM1,TIM2 AFR=2 for TIM3 etc
         if(pin >= 0 && pin <8){
             if(TIMx == TIM1 || TIMx == TIM2)       port->AFR[0] |= 1 <<
(4*pin);
             else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)      port-
>AFR[0] |= 2 << (4*pin);
             else if(TIMx == TIM9 || TIMx == TIM10 || TIMx == TIM11)    port-
>AFR[0] |= 3 << (4*pin);
         }
         else if(pin >= 8 && pin <=15){
             if(TIMx == TIM1 || TIMx == TIM2)       port->AFR[1] |= 1 << (pin-
8);
             else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)      port-
>AFR[1] |= 2 << (pin-8);
             else if(TIMx == TIM9 || TIMx == TIM10 || TIMx == TIM11)    port-
>AFR[1] |= 3 << (pin-8);
         }


// 3. Initialize Timer
    TIM_init(TIMx, 1);  // with default msec=1msec value.
    TIMx->CR1 &= ~TIM_CR1_CEN;

// 3-2. Direction of Counter
    TIMx->CR1 &= ~TIM_CR1_DIR;                          // Counting direction: 0
= up-counting, 1 = down-counting


// 4. Configure Timer Output mode as PWM
    uint32_t ccVal = TIMx->ARR/2;  // default value  CC=ARR/2
    if(chN == 1){
        TIMx->CCMR1 &= ~TIM_CCMR1_OC1M;                     // Clear ouput
compare mode bits for channel 1
        TIMx->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // OC1M = 110 for
PWM Mode 1 output on ch1. #define TIM_CCMR1_OC1M_1          (0x2UL <<
TIM_CCMR1_OC1M_Pos)
        TIMx->CCMR1 |= TIM_CCMR1_OC1PE;                     // Output 1 preload
enable (make CCR1 value changable)
        TIMx->CCR1  = ccVal;
// Output Compare Register for channel 1 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC1P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC1E;
// Enable output for ch1
    }
    else if(chN == 2){
        TIMx->CCMR1 &= ~TIM_CCMR1_OC2M;                     // Clear ouput
compare mode bits for channel 2
        TIMx->CCMR1 |= TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2; // OC1M = 110 for
PWM Mode 1 output on ch2
        TIMx->CCMR1 |= TIM_CCMR1_OC2PE;                     // Output 1 preload
enable (make CCR2 value changable)
```

```
        TIMx->CCR2   = ccVal;
// Output Compare Register for channel 2 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC2P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC2E;
// Enable output for ch2
    }
    else if(chN == 3){
        TIMx->CCMR2 &= ~TIM_CCMR2_OC3M;                     // Clear ouput
compare mode bits for channel 3
        TIMx->CCMR2 |= TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2; // OC1M = 110 for
PWM Mode 1 output on ch3
        TIMx->CCMR2 |= TIM_CCMR1_OC1PE;                     // Output 1 preload
enable (make CCR3 value changable)
        TIMx->CCR3   = ccVal;
// Output Compare Register for channel 3 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC3P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC3E;
// Enable output for ch3
    }
    else if(chN == 4){
        TIMx->CCMR2 &= ~TIM_CCMR2_OC4M;                     // Clear ouput
compare mode bits for channel 4
        TIMx->CCMR2 |= TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2; // OC1M = 110 for
PWM Mode 1 output on ch4
        TIMx->CCMR2 |= TIM_CCMR1_OC2PE;                     // Output 1 preload
enable (make CCR3 value changable)
        TIMx->CCR4   = ccVal;
// Output Compare Register for channel 4 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC4P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC4E;
// Enable output for ch4
    }


// 5. Enable Timer Counter
    // For TIM1 ONLY
    if(TIMx == TIM1) TIMx->BDTR |= TIM_BDTR_MOE;                    // Main
output enable (MOE): 0 = Disable, 1 = Enable
    // Enable timers
    TIMx->CR1  |= TIM_CR1_CEN;
// Enable counter

}

/* PWM PERIOD SETUP */
// allowable range for msec:  1~2,000
void PWM_period_ms(PinName_t pinName,  uint32_t msec){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);
```

```c
// 1. Set Counter Period in msec
    TIM_period_ms(TIMx, msec);

}


// allowable range for msec:  1~2,000
void PWM_period(PinName_t pinName,  uint32_t msec){
    PWM_period_ms(pinName,  msec);
}


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

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Set Counter Period in usec
    TIM_period_us(TIMx, usec);  //YOUR CODE GOES HERE

}

/* DUTY RATIO SETUP */
// High Pulse width in msec
void PWM_pulsewidth(PinName_t pinName, uint32_t pulse_width_ms){
// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Declaration System Frequency and Prescaler
    uint32_t fsys = 0;
    uint32_t psc = TIMx->PSC;


// 2. Check System CLK: PLL or HSI
    if((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)        fsys = 84000;
 // for msec 84MHz/1000 [msec]
    else if((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI) fsys = 16000;


// 3. Configure prescaler PSC
    float fclk = fsys/(psc+1);                                      //
fclk=fsys/(psc+1);
    uint32_t value = pulse_width_ms *fclk - 1;      // pulse_width_ms *fclk - 1;
    switch(chN){
        case 1: TIMx->CCR1 = value; break;
        case 2: TIMx->CCR2 = value; break;
        case 3: TIMx->CCR3 = value; break;
        case 4: TIMx->CCR4 = value; break;
        default: break;
    }
}

// High Pulse width in msec
void PWM_pulsewidth_ms(PinName_t pinName, uint32_t pulse_width_ms){
    PWM_pulsewidth(pinName, pulse_width_ms);
}

// High Pulse width in usec
void PWM_pulsewidth_us(PinName_t pinName, uint32_t pulse_width_us){
// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);

// 1. Declaration system frequency and prescaler
    uint32_t fsys = 0;
    uint32_t psc = TIMx->PSC;


// 2. Check System CLK: PLL or HSI
    if((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)        fsys = 84;  //
for msec 84MHz/1000000 [usec]
    else if((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI) fsys = 16;


// 3. Configure prescaler PSC
    float fclk = fclk=fsys/(psc+1);                                //
fclk=fsys/(psc+1);
    uint32_t value = pulse_width_us *fclk - 1;      // pulse_width_us *fclk - 1;

    switch(chN){
        case 1: TIMx->CCR1 = value; break;
        case 2: TIMx->CCR2 = value; break;
        case 3: TIMx->CCR3 = value; break;
        case 4: TIMx->CCR4 = value; break;
        default: break;
    }
}

// Dutry Ratio from 0 to 1
void PWM_duty(PinName_t pinName, float duty){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Configure prescaler PSC
    float value = TIMx->ARR + 1;                        //
(ARR+1)*dutyRatio + 1
        value = value*duty - 1;

    if(chN == 1)      { TIMx->CCR1 = value; }          //set channel
        else if(chN == 2)      { TIMx->CCR2 = value; }
        else if(chN == 3)      { TIMx->CCR3 = value; }
        else if(chN == 4)      { TIMx->CCR4 = value; }

}

// DO NOT MODIFY HERE
void PWM_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN)
{
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);


  if(port == GPIOA) {
      switch(pin){
         case 0 : *TIMx = TIM2; *chN    = 1; break;
         case 1 : *TIMx = TIM2; *chN = 2; break;
         case 5 : *TIMx = TIM2; *chN = 1; break;
         case 6 : *TIMx = TIM3; *chN = 1; break;
         //case 7: TIMx = TIM1; *chN = 1N; break;
         case 8 : *TIMx = TIM1; *chN = 1; break;
         case 9 : *TIMx = TIM1; *chN = 2; break;
         case 10: *TIMx = TIM1; *chN = 3; break;
         case 15: *TIMx = TIM2; *chN = 1; break;
         default: break;
      }
   }
   else if(port == GPIOB) {
      switch(pin){
         //case 0: TIMx = TIM1; *chN = 2N; break;
         //case 1: TIMx = TIM1; *chN = 3N; break;
         case 3 : *TIMx = TIM2; *chN = 2; break;
         case 4 : *TIMx = TIM3; *chN = 1; break;
         case 5 : *TIMx = TIM3; *chN = 2; break;
         case 6 : *TIMx = TIM4; *chN = 1; break;
         case 7 : *TIMx = TIM4; *chN = 2; break;
         case 8 : *TIMx = TIM4; *chN = 3; break;
         case 9 : *TIMx = TIM4; *chN = 4; break;
         case 10: *TIMx = TIM2; *chN = 3; break;
         default: break;
      }
   }
   else if(port == GPIOC) {
      switch(pin){
         case 6 : *TIMx = TIM3; *chN = 1; break;
         case 7 : *TIMx = TIM3; *chN = 2; break;
         case 8 : *TIMx = TIM3; *chN = 3; break;
```

```
            case 9 : *TIMx = TIM3; *chN = 4; break;
            default: break;
         }
      }
        // TIM5 needs to be added, if used.
  }
```