LAB: Timer & PWM

Date: 2022-10-30

Author/Partner: Dongmin Kim / Seongjun Park

Github: https://github.com/DongminKim21800064/EC_dmkim-064

Demo Video: https://www.youtube.com/watch?v=JqZM9r-GVFc

PDF version:

Introduction

In this lab, you are required to create a simple program that control a servo motor with PWM output.

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator
  - RC Servo Motor (SG90)
  - DC motor
- Others
  - breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Create HAL library

Create HAL library

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* timerx, uint32_t unit, uint32_t time);

void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec);

void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec);

// Timer Interrupt setup
```

```
void TIM_INT_init(TIM_TypeDef* timerx, uint32_t unit, uint32_t time);

void TIM_INT_enable(TIM_TypeDef* TIMx);

void TIM_INT_disable(TIM_TypeDef* TIMx);



// Timer Interrupt Flag

uint32_t is_UIF(TIM_TypeDef *TIMx);

void clear_UIF(TIM_TypeDef *TIMx);
```

ecPWM.h

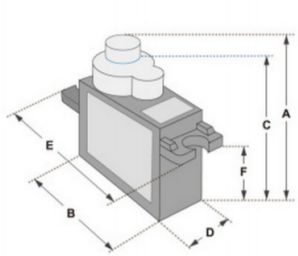
```
/* PWM STRUCTURE */

typedef struct {

•   GPIO_TypeDef *port;

•   int pin;

•   TIM_TypeDef *timer;

•   int ch;

} PWM_t;



/* PWM initialization */

// Default: 84MHz PLL, 1MHz CK_CNT, 50% duty ratio, 1msec period

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



/* PWM PERIOD SETUP */

// allowable range for msec:  1~2,000

void PWM_period_ms(PWM_t *pwm,  uint32_t msec);

// allowable range for usec:  1~1,000

void PWM_period_us(PWM_t *pwm, uint32_t usec);



/* DUTY RATIO SETUP */

// High Pulse width in msec
```

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

// Duty ratio 0~1.0

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

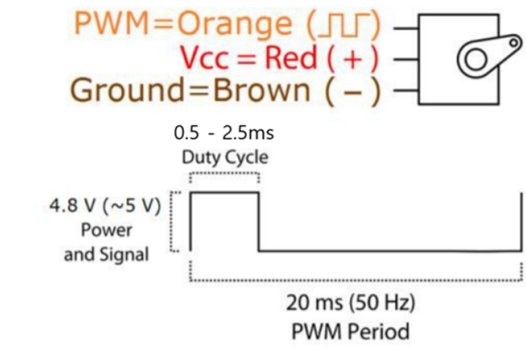
Problem 2: RC Servo motor

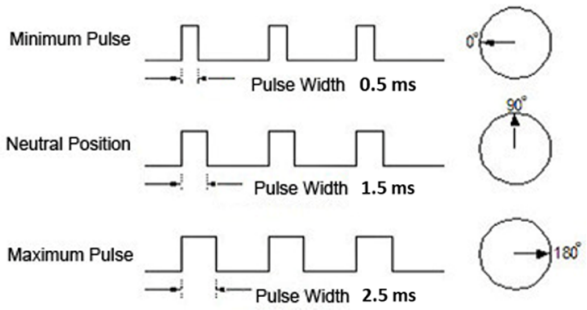
An RC servo motor is a tiny and light weight motor with high output power. It is used to control rotation angles, approximately 180 degrees (90 degrees in each direction) and commonly applied in RC car, and Small-scaled robots.

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 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





Make a simple program that changes the angle of the RC servo motor that rotates with a given period of time and reset by pressing the push button (PC13).

- The button input has to be External Interrupt
- Use Port A Pin 1 as PWM output pin, for TIM2_Ch2.
- Use Timer interrupt of period 500msec.
- The 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 input button is pushed, using an oscilloscope. You need to capture the Oscilloscope output in the report.

Procedure

1. Create a new project under the directory `\repos\EC\LAB\LAB_PWM_RCmotor`

- The project name is "**LAB_PWM_RCmotor".**
- Create a new source file named as "**LAB_PWM_RCmotor.c"**

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

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

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

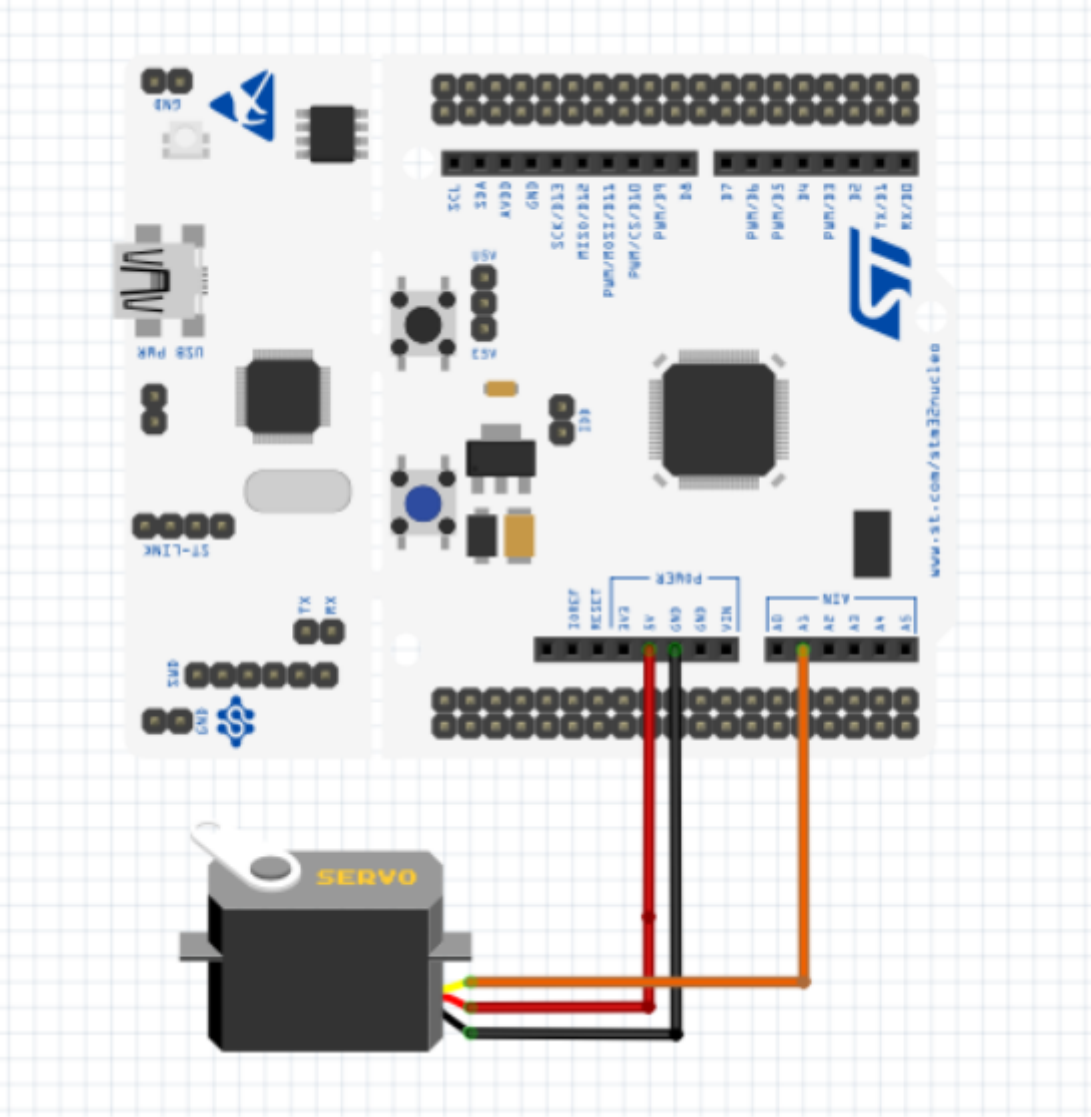
1. Connect the RC servo motor to MCU pin (PA1) , VCC and GND
2. Increase the angle of RC servo motor from 0° to 180° with a step of 10° every 500msec. After reaching 180°, decrease the angle back to 0°. Use timer interrupt IRQ.
3. When the button is pressed, it should reset to the angle 0° and start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

Configuration

Button	**PWM Pin**	**Timer**	**PWM**
Digital In (PC13)	AF (PA1)	TIM3	TIM2_CH2 (PA1)
Pull-Up	Push-Pull	Timer Period: 100usec	PWM period: 20msec
	Pull-up, Fast	Timer Interrupt of 500msec	duty ratio: 0.5~2.5msec

Circuit Diagram

> You need to include the circuit diagram



Discussion

1. Derive a simple logic to calculate for CRR and ARR values to generate xHz and y% duty ratio of PWM. How can you read the values of input clock frequency and PSC?

> First, the system frequency must be set. The system frequency includes PLL, HSI, and HSE, and is selected from them. Update event frequency can be set by setting PSC (prescaler value) and ARR (auto-reload value).
>
> Second, the duty ratio is determined by CCR (Compare Capture value) and CNT (Counter value). The counter goes up from 0 to ARR. CCR gives a reference signal among them. If CCR<CNT, PWM output becomes HIGH, and if CCR>CNT, it becomes LOW.

$$f_{cntCLK} = \frac{f_{sysCLK}}{PSC + 1}$$

$$f_{updatefreq} = \frac{f_{cntCLK}}{ARR + 1}$$

$$f_{updateEvent} = \frac{f_{sysCLK}}{(ARR + 1)(PSC + 1)}$$

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

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

> Prescaler register is 16bit. So that the value cna be 0 to 65,535.
>
> ARR is 16bit or 32bit.
>
> ARR 16bit : 0 to 65,535
>
> ARR 32bit : 0 to 4,294,967,295
>
> In 16bit

$$f_{PWM\_min} = \frac{f_{sysCLK}}{(ARR+1)(PSC+1)} = \frac{84[MHz]}{65{,}536 * 65{,}536} = 19.5578[mHz]$$

$$f_{PWM\_max} = \frac{f_{sysCLK}}{(ARR+1)(PSC+1)} = \frac{84[MHz]}{1} = 84[MHz]$$

> In 32bit

$$f_{PWM\_min} = \frac{f_{sysCLK}}{(ARR+1)(PSC+1)} = \frac{84[MHz]}{65{,}536 * 4{,}294{,}967{,}296} = 0.2984[\mu Hz]$$

$$f_{PWM\_max} = \frac{f_{sysCLK}}{(ARR+1)(PSC+1)} = \frac{84MHz}{1} = 84[MHz]$$

Code

Your code goes here: https://github.com/DongminKim21800064/EC_dmkim-064/tree/main/lab/EC_LAB_PWM_RCmotor_21800064_%EA%B9%80%EB%8F%99%EB%AF%BC

Explain your source code with necessary comments.

LAB_PWM_RCmotor.c

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

#define RCMotor_PIN 1

PWM_t pwm;
static uint32_t pwm_period = 20;
static uint32_t EXTI_flag = 0;
static uint32_t TIM_flag = 0;

uint32_t count = 0;
uint32_t state = 0;

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

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

    // Inifinite Loop ---------------------------------------------------------
    while(1){
            if(EXTI_flag==1){                       // If the Button Pushed ->
reset
                state =0 ;
                PWM_duty(&pwm, (0.5 /pwm_period)); // Set 0 degree
                EXTI_flag = 0;
            }
        }
    }


// Initialiization
void setup(void){
    RCC_PLL_init();                         // System Clock = 84MHz
    TIM_INT_init(TIM3,u_sec,100);           // Timer Period 100usec


    // Digital In: Button Pin to use interrupt
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);

    // Digital Out: ServoMotor
    GPIO_init(GPIOA, RCMotor_PIN, AF);    // calls RCC_GPIOA_enable()
    GPIO_output (GPIOA, RCMotor_PIN, PUSH_PULL, EC_PU, FAST_SPEED);

    // PWM
    PWM_init(&pwm, GPIOA, RCMotor_PIN);
    PWM_period_ms(&pwm, pwm_period);
```

```
}

void TIM3_IRQHandler(void){                 // Timer Interrupt
    if (is_UIF(TIM3) == TIM_SR_UIF) {       // update interrupt flag

        count++;
        if(count >5000){                    // delay 500ms = 100us x 5000

        if(state >= 18) TIM_flag = 1;       // If PWM Restart flag

        PWM_duty(&pwm, (0.5 + ((2.5-0.5)/18)*state)/pwm_period);    // PWM
operate

            if(TIM_flag==1){                // To prevent the state from
operating at 1 to 18
                TIM_flag = 0;               // It will operate at 0 to 18
                state = 0;
            }else if(TIM_flag==0)   state++;

            count = 0;
        }

    clear_UIF (TIM3);                       // clear by writing 0
    }
}

void EXTI15_10_IRQHandler(void) {           // EXTI Button interrupt
    if (is_pending_EXTI(BUTTON_PIN)){
        EXTI_flag = 1;
        clear_pending_EXTI(BUTTON_PIN);     // cleared by writing '1'
    }
}
```

Digital Input

> Button Pin is enable to use interrupt. Button Pin is PC13, input mode, Pull-up mode. EXTI initializes the Button pin when it is falling edge, and priority is 0.

Digital Ouput

> Digital output is Servomotor which is PA1 AF mode. Servomotor set Push-pull, Pullup, and fast speed.

PWM

> PWM period is 20ms.

TIM3_IRQHANDLER

> TIM3 sets timer period 100us. Count is up to 5000 and it means that 5000* 100us = 500ms.
>
> PWM state is from 0° to 180° with a step of 10°. If the state reach 180°, reset to 0°.
>
> The code below is for preventing the state from operating at 1 to 18. Because of I use the TIM_flag, It will operate at 0 to 18

```
if(TIM_flag==1){        // To prevent the state from operating at 1 to 18
                TIM_flag = 0;           // It will operate at 0 to 18
                state = 0;
            }else if(TIM_flag==0)   state++;
```

EXTI15_10_IRQHandler

> If the Button pin is pressed, EXTI interrupt operate.

main

> If EXTI interrupt operates, PWM reset to 0 ° and restart.

Sample Code : Timer Interrupt IRQ

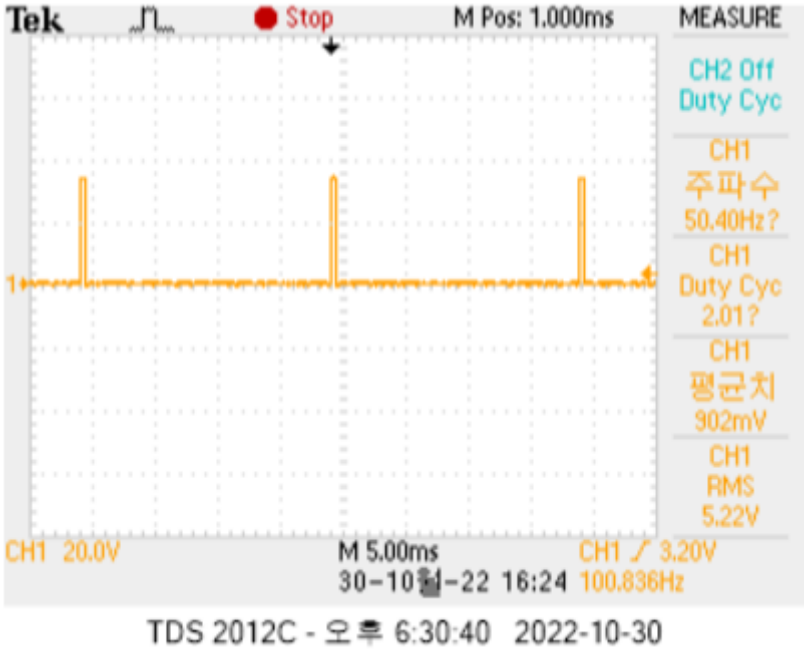
```
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"

uint32_t _count=0;

#define LED_PIN   5

void setup(void);

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

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

// Initialiization
void setup(void)
{
  RCC_PLL_init();                          // System Clock = 84MHz
  GPIO_init(GPIOA, LED_PIN, OUTPUT);     // calls RCC_GPIOA_enable()
  TIM_INT_init(TIM2,1000);   // usec >=100
  TIM_INT_enable(TIM2);
}

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

}
```

Sample Code : PWM output

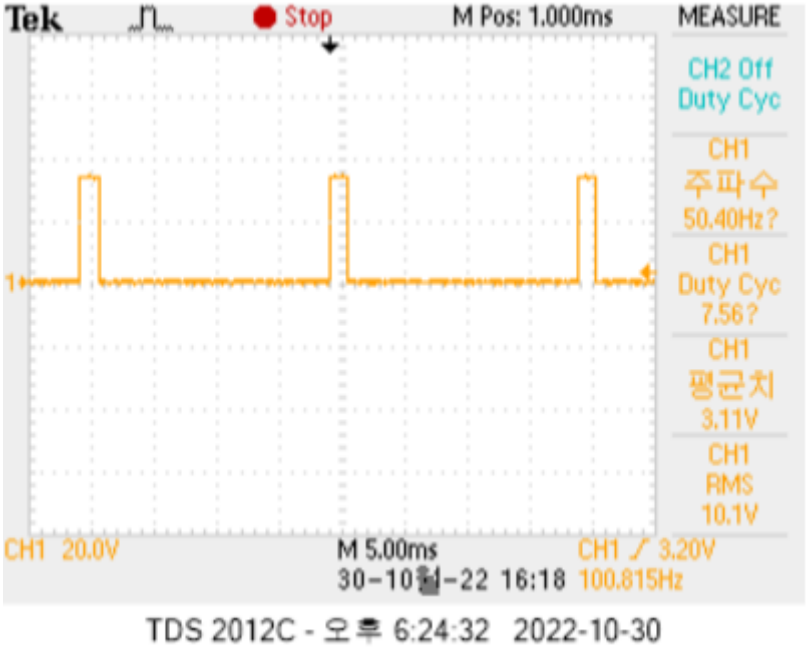
```c
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecSysTick.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecPWM.h"

#define LED_PIN   5

PWM_t pwm;
// Initialiization
void setup(void)
{
  RCC_PLL_init();                          // System Clock = 84MHz
  SysTick_init();

  GPIO_init(GPIOA, LED_PIN, EC_ALTE);     // calls RCC_GPIOA_enable()
  GPIO_ospeed(GPIOA, 1, EC_HIGH);
  GPIO_pudr(GPIOA, 1, EC_NONE);

  PWM_init(&pwm, GPIOA, 5);
  PWM_period_ms(&pwm, 1);

}

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

  // Inifinite Loop ---------------------------------------------------------
  while(1){
    for(int i =0; i<3;i++){
    PWM_duty(&pwm,0.5*i);
    delay_ms(100);
    }
  }
}

```

Results

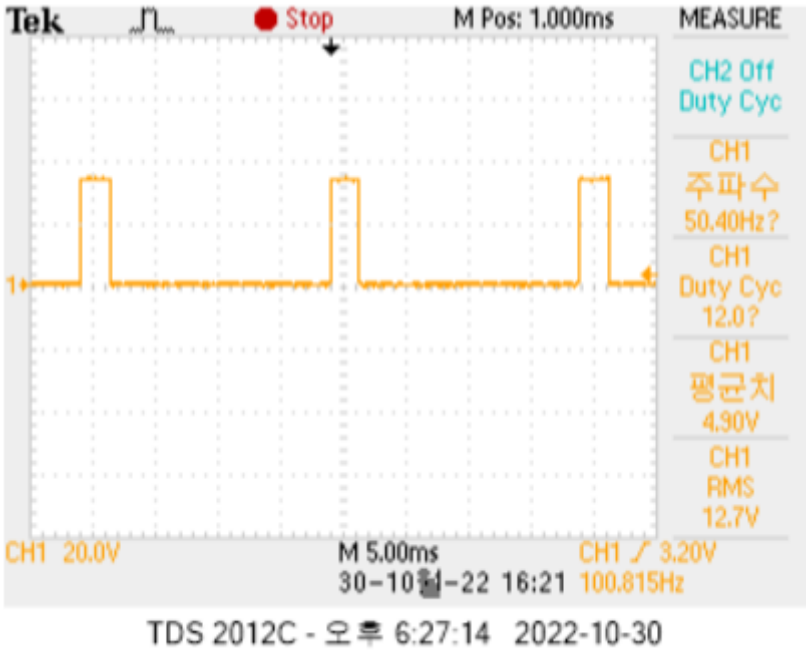
PWM 0° oscilloscope figure

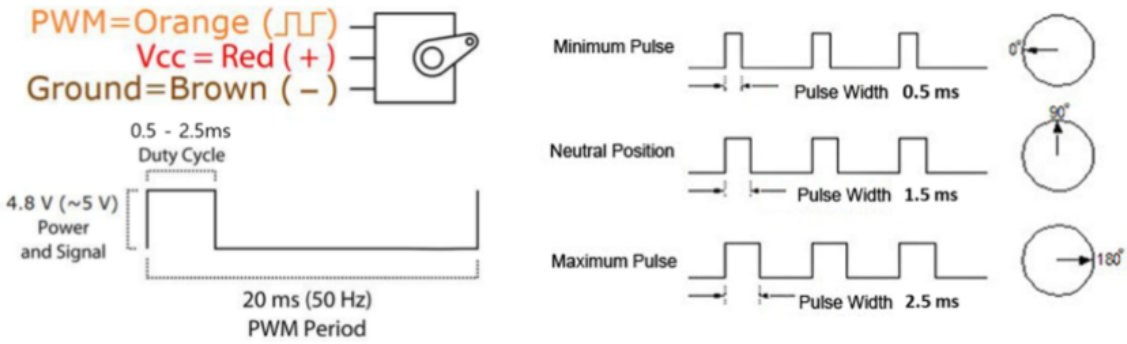


PWM 90° oscilloscope figure



PWM 180° oscilloscope figure

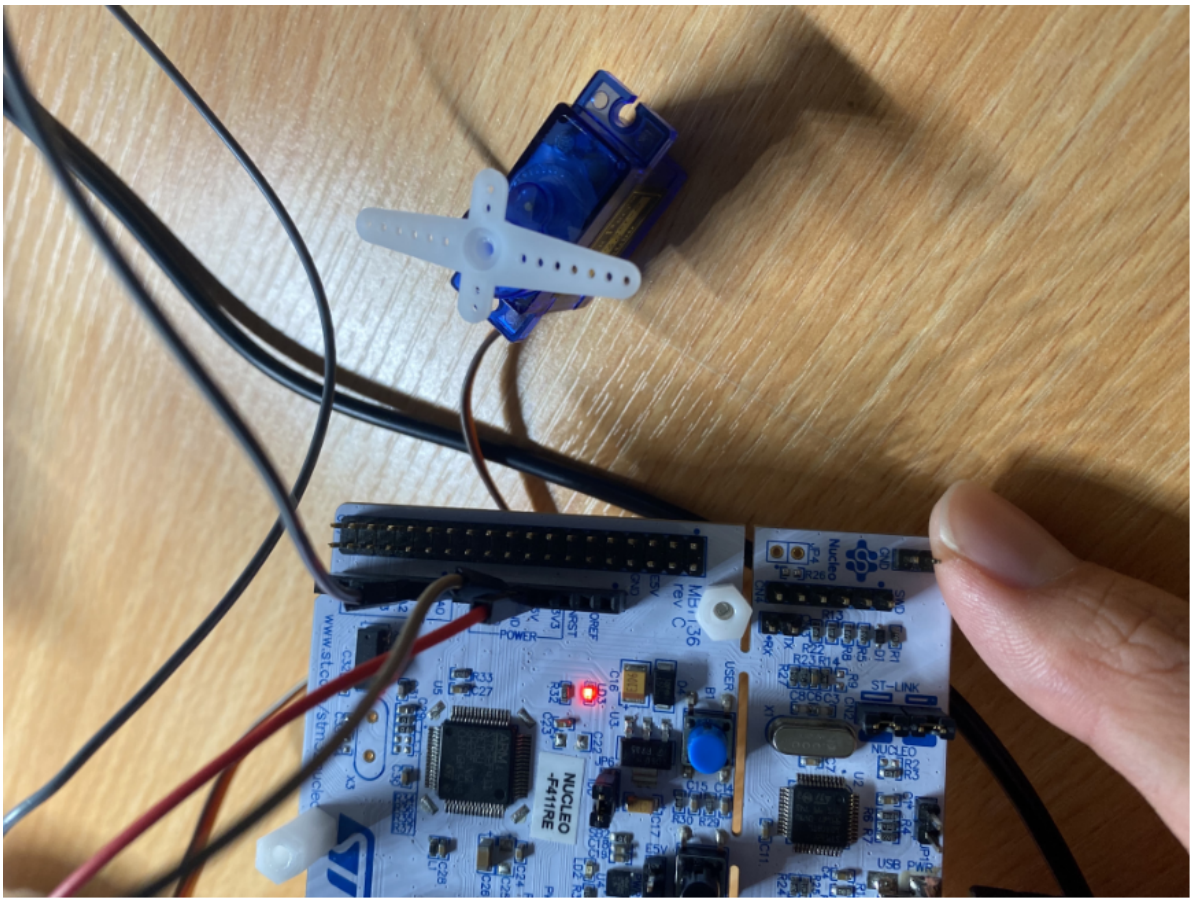




In this experiment, the PWM period was set to 20 ms (50 Hz). Therefore, Duty ratio will be 2.5% , 7.5%, and 12.5% each at 0°, 90°, and 180°.

As you can see, the frequency of PWM observe 50.40Hz , and the duty cycle calculate 2.01%, 7.56%, 12.0% which is similar to specification.

> Show experiment images /results



Demo video : https://www.youtube.com/watch?v=JqZM9r-GVFc

Conclusion

Through this experiment, I succeed to control the servo motor PWM output duty ratio. In order to control the desired angle, the pulse width was changed over time according to the given conditions to operate the motor. The timer interrupt gave a delay of 500 ms each step, and the EXTI interrupt allows the motor to be restarted with the button.

Reference

Given information by

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

Troubleshooting

RC servo motor Deadzone

- Increase the angle of RC servo motor from 0° to 180° with a step of 10° every 500msec. In this case I must observe 19 steps which is 0° to 180° with a step of 10°. But only 18steps observed. I tested like below

```
void TIM3_IRQHandler(void){
  if ((TIM3->SR & TIM_SR_UIF) == TIM_SR_UIF) { // update interrupt flag
    }
      count++;
    if(count >5000){       // delay 500ms = 100us x 5000

      if(state >= 18){
          PWM_duty(&pwm, (0.5 + ((2.5-0.5)/18)*state)/pwm_period);
          TIM_flag = 1;
        } else{
          PWM_duty(&pwm, (0.5 + ((2.5-0.5)/18)*state)/pwm_period);
          if(state==1)delay_ms(3000);
        }
      if(TIM_flag==1){     // To prevent the state from operating at 1 to 18
        TIM_flag = 0;      // IT will operate at 0 to 18
        state = 0;
      }else if(TIM_flag==0){
        state++;
      }
      count = 0;
    }
```

As a result, it was found that the servomotor was in the same position when it was 0° and 10°. I thought the cause of this phenomenon was 'DEAD ZONE', and I raised the initial value of pulse width by 0.1 ms

```
void TIM3_IRQHandler(void){
  if ((TIM3->SR & TIM_SR_UIF) == TIM_SR_UIF) { // update interrupt flag
    }
      count++;
    if(count >5000){       // delay 500ms = 100us x 5000

      if(state >= 18){
          PWM_duty(&pwm, (0.6 + ((2.5-0.5)/18)*state)/pwm_period);
          TIM_flag = 1;
        } else{
          PWM_duty(&pwm, (0.6 + ((2.5-0.5)/18)*state)/pwm_period);
          //if(state==1)delay_ms(3000);
        }
      if(TIM_flag==1){     // To prevent the state from operating at 1 to 18
        TIM_flag = 0;      // IT will operate at 0 to 18
        state = 0;
      }else if(TIM_flag==0){
        state++;
      }
      count = 0;
    }
```

As I expected, when the initial value of pulse width was raised, all 19 steps were observed. For more information, please refer to the YouTube.