

REPORT



11주차 결과 보고서

수강과목 : 임베디드 시스템 설계 및 실험

담당교수 : 백윤주 교수님

학 과 : 전기컴퓨터공학부 정보컴퓨터공학전공

조 원 : 201724648 김경호 201724651 김장환

201624481 박윤형 201524588 조창흠

제출일자 : 2021 / 11 / 17

11주차 결과보고서

1. 실험 목표

- 1.1. 임베디드 시스템의 기본 원리 습득
- 1.2. Timer 이해 및 실습
- 1.3 PWM 이해 및 서보모터를 이용한 실습

2. 배경 지식

2.1. Timer

- 주기적 시간 처리에 사용되는 디지털 카운터 회로 모듈
- 타이머 모듈을 사용하여 일정한 주기적 시간 처리를 하는 시스템 요구에 대응하여 사용할 수 있다.
- 주로 펄스폭 계측, interrupt 등에 사용된다.
- 일련의 사건이나 process를 제어하는데 사용된다.
- 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮추고, 나아진 주파수를 8,16 비트 등의 카운터 회로를 사용하여 주기를 얻는다.
- 기계적, 전자기계적, 전기적, S/W적 방식을 취하기도 한다.

2.2. Timer 종류와 특징

2.2.1. SysTick timer

- 24bit down counter이다.
- Cortex-M3 core에 내장되어 있다.
- 일반 Timer와 달리 오직 정주기를 만드는 용도로 사용된다.
- counter가 0에 도달하면 설정에 따라 인터럽트가 발생하고, counter값은 reload값으로 자동으로 설정된다.
- Real-time operating system 전용이지만, standard down counter로 사용할 수도 있다.

2.2.2 Watchdog(WDG) Timer

- Watchdog(WDG)는 특수 상황에서 CPU가 올바르게 작동하지 않을 경우를 대비해 강제로 reset 시키는 기능을 의미한다.
- 컴퓨터의 오작동을 감지하고 복구하기 위해 사용된다.
- 정상작동 중인 컴퓨터는 시간이 경과하거나 Time out 되는 것을 막기 위해, 정기적으로 Watchdog Timer를 재가동 시킨다.
- IWDG는 LSI로부터 clock를 제공받는 타이머로 HSI/HSE clock에 문제가 발생해도 독립적으로 동작할 수 있다.
- WWDG는 IWDG 와 달리 기본적으로 Window 라고 호칭하는 시간 내에서만 refresh 가 가능하고, 그 외의 구역에서는 Reset 이 발생한다.

< IWDG >

- 타이밍 정확도 제약이 낮은 애플리케이션에 적합하다.
- 카운터가 0이 되면 Reset

< WWDG >

- 7-bit down counter
- WWDG의 clock은 APB1 clock를 prescale해서 정의가 가능하다.
- 비정상적 어플리케이션 동작감지를위해 설정가능한 time-window가 존재한다.
- Time-window 내에서 반응하도록 요구하는 애플리케이션에 적합하다.
- 카운터가 0x40보다 작을 경우 혹은 카운터가 Time-window밖에 Reload 되었을 경우 Reset 가능하다.
- Early wakeup interrupt (EWI) : 카운터가 0x40과 같을 경우, EWI 인터럽트를 발생하도록 설정이 가능하다.

2.2.3. Advanced-control timers (TIM1 & TIM8)

- 16bit auto-reload counter를 포함하고 있다.
- General-purpose Timer와 거의 유사하다. 그러나 complementary output 기능을 지원하고 PWM(Pulse Width Modulation : 펄스 폭 변조)의 기능을 좀 더 보완한 것이다.
- The advanced-control (TIM1&TIM8) and general-purpose (TIMx)는 자원을 공유하지 않는 독립적인 구조이고 동기화도 가능하다

2.2.4. General-purpose timers (TIM2 to TIM5)

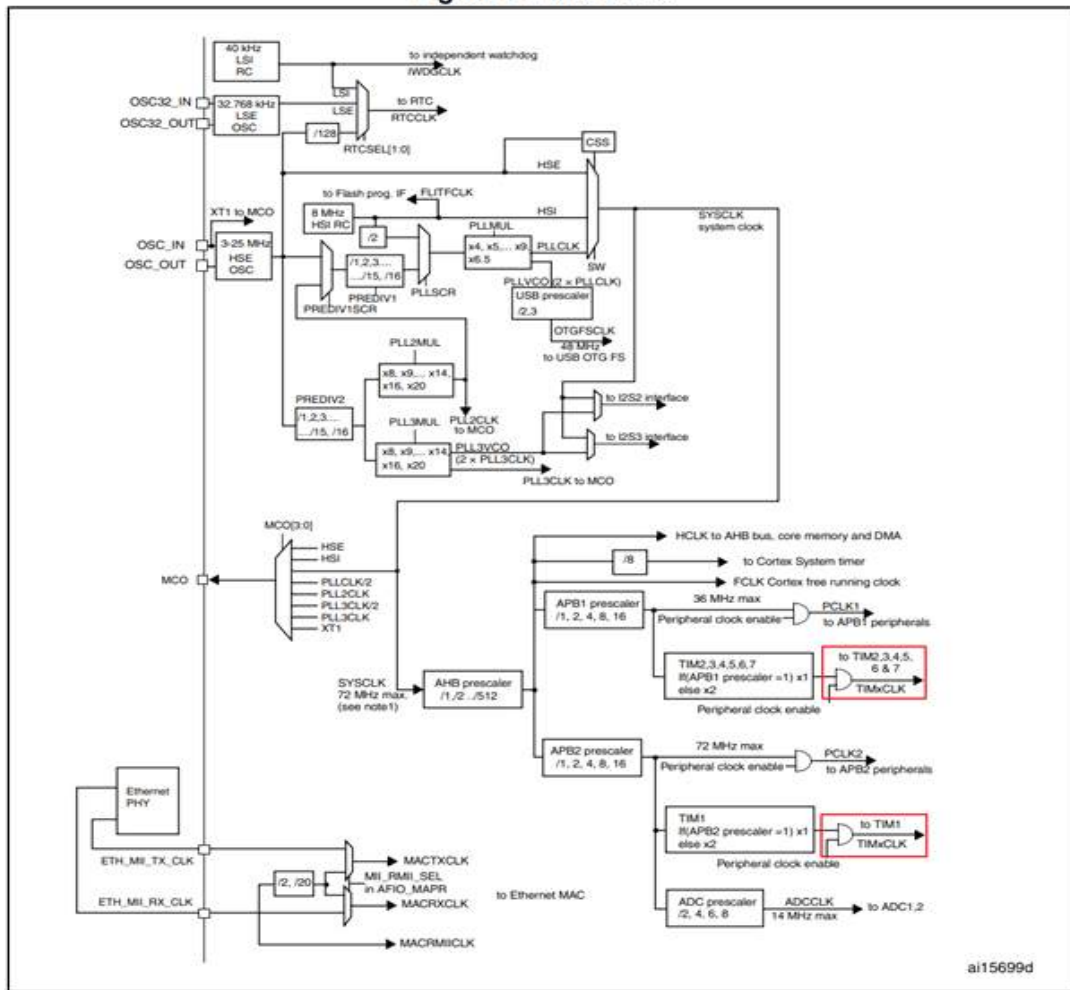
- 16bits auto-reload up/down 카운터로 16bit prescaler를 기반으로 둔다.
- prescaler를 통해 설정 가능한 16-bit up, down, up/down, auto-reload counter를 포함하고 있다.
- 외부 event를 카운트하고 시간을 측정하기 위해 사용된다.
- input capture, output compare, PWM, one pulse module output을 위한 4개의 독립 채널을 가지고 있다.
- 펄스 길이와 파형 주기는 timer prescaler 와 RCC clock controller prescaler를 사용하여 몇 μs 에서 몇 ms 까지 변조 가능하다.
- 타이머들은 완전히 독립적이고 어떤 자원도 공유하지 않는다. 하지만 동기화가 가능하다.

2.2.5. Basic timer (TIM6 & TIM7)

- 16bit auto-reload up counter를 포함하고 있다.
- 16bit prescaler를 이용해 the counter clock 주파수를 나눠서 설정 가능하다.
- input/output 핀 없이 순수 time base generator 동작을 수행한다.
- 주로 DAC trigger generation에 사용된다.
- Counter/Timer Overflow 발생 시 인터럽트/DMA를 생성한다.

2.3 clock tree

Figure 11. Clock tree



2.4 분주

$$\frac{1}{f_{clk}} \times prescaler \times period$$

- 분주란 MCU에서 제공하는 Frequency를 우리가 사용하기 쉬운 값으로 바꾸어 주는 것을 말합니다.

$$\frac{1}{72Mhz} \times 7200 \times 10000 = 1[s]$$

- Counter clock frequency 를 1~65536의 값으로 나누기 위해 16-bit programmable prescaler 사용

- period 로 몇 번 count하는지 설정

$$f_{clk} * \frac{1}{prescaler} * \frac{1}{period} = \text{주파수}[Hz]$$

- period는 분주비를 몇 번 count하는지 설정하는 값을 의미한다.
- prescaler는 분주비를 의미하며 주파수를 어떤 간격으로 나눌지 설정하기 위한 값이다
- MCU에서 제공하는 frequency를 사용하기 쉬운 값으로 바꾸어 주는 것을 말한다.

2.5 PWM(Pulse Width Modulation)

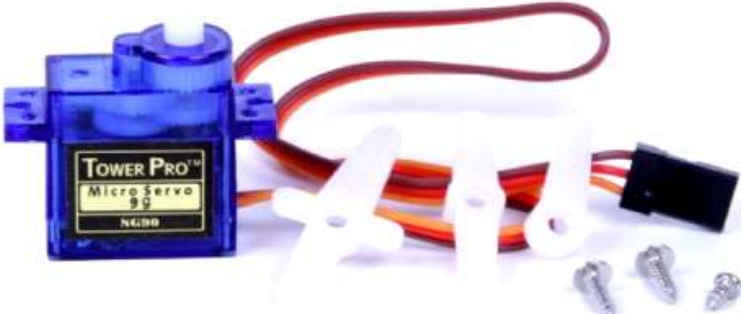
- 일정한 주기 내에서 Duty ratio를 변화시켜서 평균 전압을 제어하는 방법이다.
- 대부분의 서보모터는 50Hz ~ 1000Hz의 주파수를 요구한다.

2.6 서보모터(Servo Motor)

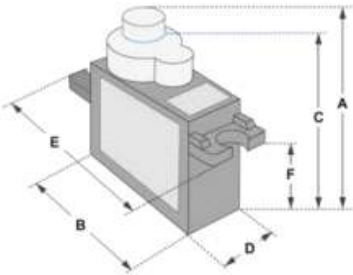
- 일반적인 모터와는 달리 빈번하게 변화하는 위치나 속도의 명령치에 대응하여 신속하고 정확하게 따를 수 있도록 설계된 모터

SERVO MOTOR SG90

DATA SHEET





Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

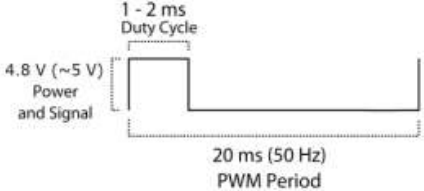


Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

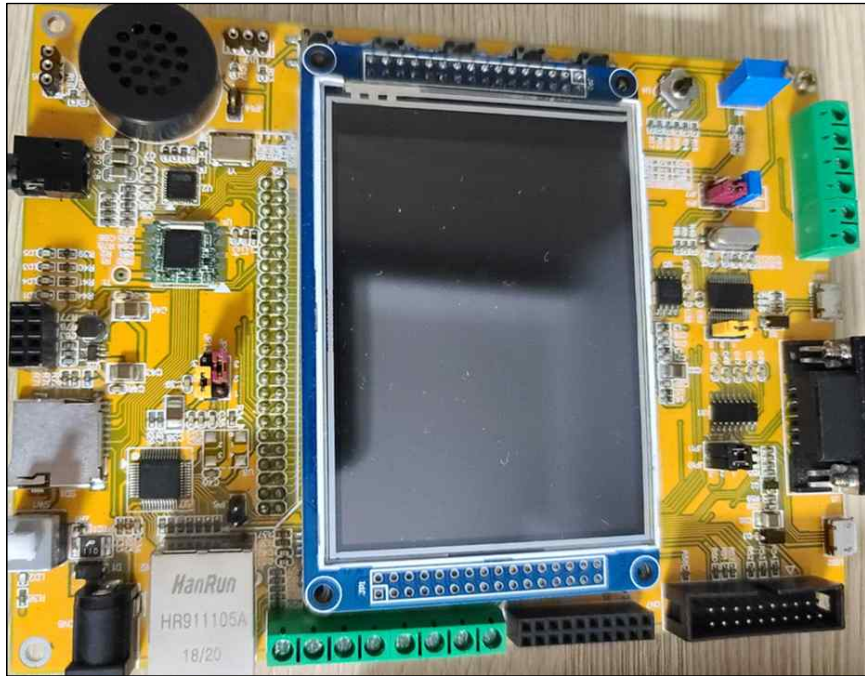
PWM=Orange ()
Vcc = Red (+)
Ground=Brown (-)



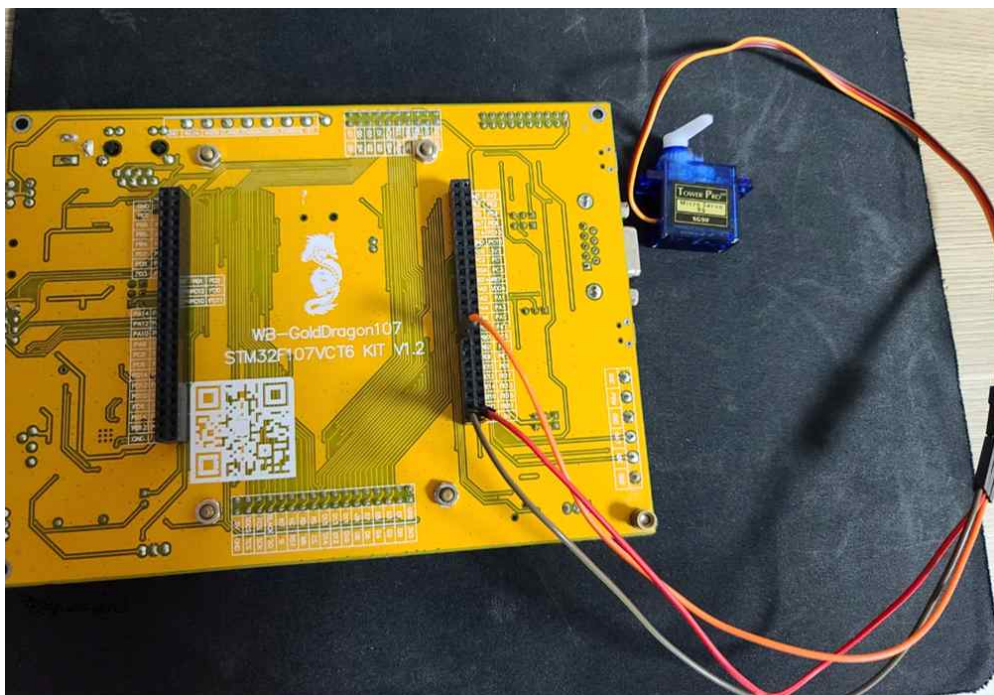


3. 실험 방법

(1) 10주차 실험과 동일하게 TFT-LCD와 STM32 보드를 연결한다.



(2) 서보모터를 STM32 보드와 연결한다. (Vcc, PWM, Ground)



4. 코드 설명

< 변수 선언 >

```
// number of IRQ calls
u32 IRQ_counter = 0;

// Button flag
int btn=0;

// LED1/ LED2 flag
int LED1=0, LED2=0;

// location of coordinate values
uint16_t cur_x,cur_y,pixel_x,pixel_y;
```

- IRQ handler를 호출한 횟수, button에 대한 flag, LED를 켜기위한 flag 및 lcd의 좌표값을 계산하기 위한 변수를 선언한다.

< RCC Configure >

```
/*
    Enable TIM2 to use timer
    Enable LED to use Port D
    Enable AF IO Clock
*/
void RCC_Configure(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);

    //LED1,2(PD2,3) , PWM(PB0)
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

- TIM2, LED 사용을 위한 Port D, Alternate Function IO Clock을 활성화 시킨다.

< GPIO Configure >

```
/*
Since it is used as the output of LED in Port D 2,3,
it is set as GPIO_PIN, the maximum output speed is 10MHz,
and the general purpose output push-pull mode is set.
```

Finally, set GPIO Pin to control PWM.

```
*/

void GPIO_Configure(void)
{
```

```

GPIO_InitTypeDef GPIO_InitStructure;

// LED
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOD, &GPIO_InitStructure);

// PWM
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOB, &GPIO_InitStructure);

}

```

- Port D의 2,3번 Pin을 LED의 출력으로 사용하므로 GPIO_PIN으로 설정하고, 최대 출력 속도는 10MHz, General Purpose output Push-Pull 모드로 설정한다.
- Port B의 0번 Pin을 PWM을 위해 사용하고, 출력 속도는 50MHz, Alternative Function Push-Pull 모드로 설정한다.

< TIM2 Configure >

```

void TIM2_Configure(void){
    TIM_TimeBaseInitTypeDef Timer;

    // input clock is used as it is.
    Timer.TIM_ClockDivision = TIM_CKD_DIV1;

    // Set up counting mode in which count is incremented by 1 for every 1clk.
    Timer.TIM_CounterMode = TIM_CounterMode_Up;

    // Since the period count starts from 0, set it to a value obtained by subtracting 1 from 7200.
    Timer.TIM_Prescaler = 7200 - 1;

    // Since the count of the prescaler starts from 0, set it to a value minus 1 from 10000.
    //==1s
    Timer.TIM_Period = 10000 - 1;

    TIM_TimeBaseInit(TIM2, &Timer);

    // Enable interrupt of TIM2.
    TIM_ARRPreloadConfig(TIM2,ENABLE);
    TIM_Cmd(TIM2,ENABLE);
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);//interrupt enable
}

```

- Timer 2에 대한 설정으로 TIM2의 주파수는 72MHz, prescaler는 7200, period는 10000이므로 $1/72\text{Mhz} * 7200 * 10000 = 1/72000000 * 7200 * 10000 = 1\text{sec}$ 가 나오게 되고, 해당 값으로 Timer를 설정하였다.

< TIM3 Configure >

```
void TIM3_Configure(void){

    // declare the variable about TimeBase
    TIM_TimeBaseInitTypeDef Timer;

    uint16_t prescale= (uint16_t) (SystemCoreClock/ 1000000)-1;

    Timer.TIM_Period= 20000-1;
    Timer.TIM_Prescaler= prescale;
    Timer.TIM_ClockDivision= 0;
    Timer.TIM_CounterMode= TIM_CounterMode_Down;
    TIM_TimeBaseInit(TIM3, &Timer);

    // declare the variable about OCinit
    TIM_OCInitTypeDef TIM_OCInitStructure;
    TIM_OCInitStructure.TIM_OCMode= TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OCPolarity= TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OutputState= TIM_OutputState_Enable;

    TIM_OCInitStructure.TIM_Pulse= 1500; // us 1.5ms
    TIM_OC3Init(TIM3, &TIM_OCInitStructure);

    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
    TIM_Cmd(TIM3, ENABLE);
}
```

- Timer 3에 대한 설정으로 TIM3의 주파수는 72MHz, prescaler는 72, period는 20000이므로 $1/72\text{Mhz} * 72 * 20000 = 1/20000000 * 72 * 20000 = 50$ 이 나오게 되고, 해당 값으로 Timer를 설정하였다.
- 0도에 해당하는 1.5ms을 초기값으로 하여 PWM mode를 설정한다.

< NVIC Configure >

```
/*
NVIC_PriorityGroup_0 is set because no other pre-emption priority group is needed.
Also, pre-emption priority and subpriority were given the same priority as 0.
*/
void NVIC_Configure(void) {
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    // Initialize the NVIC using the structure 'NVIC_InitTypeDef' and the function 'NVIC_Init'

    // UART1
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

```

    NVIC_Init(&NVIC_InitStructure);
}

```

- 다른 pre-emption priority group이 필요 없기 때문에 NVIC_PriorityGroup_0으로 설정하였다.
- pre-emption priority와 subpriority도 0으로 동일한 우선순위를 부여하였다.

< TIM2 IRQHandler >

```
void TIM2_IRQHandler(void) {
```

```

    /*
    When the button on the LCD is pressed, the LED toggle becomes ON,
    and the number of times TIM2 IRQ handler is called increases and is stored in IRQ_counter.
    */
    if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
        if(btn==1){
            IRQ_counter++;
            LED1=!LED1;

            //Whenever TIM2 IRQ handler is called 5 times, the flag is switched to the opposite state of
current LED2.
            if(IRQ_counter%5==0){
                LED2=!LED2;
            }
        }
        // Based on the current LED1,2 flag, the function to turn on and off LED1,2 is called.
        LED_on();
        TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
    }
}

```

- TFT-LCD의 Button을 누르면 LED1 flag 변수의 상태가 바뀌게 되고, IRQ_counter 변수가 증가하게 된다.
- IRQ_counter가 5의 배수가 될 때마다 LED2 flag 변수의 값을 변화시킨다.
- LED1,2 flag 변수를 변화시킨 후 LED_on()함수를 호출하여 LED의 점등 상태를 변화시킨다.

< LED_on >

```
void LED_on(void){
```

```

    /*
    If the flag of LED1 is 1, the LED of Port D2 is turned on,
    and the character string ??ON?? is output in red letters on a white background at (90, 260).
    In the opposite case, the LED is turned off and the string ??OFF?? is output in the same way.
    */
    if(LED1==1){
        GPIO_SetBits(GPIOD,GPIO_Pin_2);
    }
    else{
        GPIO_ResetBits(GPIOD,GPIO_Pin_2);
    }
    if(LED2==1){
        GPIO_SetBits(GPIOD,GPIO_Pin_3);
    }
}

```

```

    }
    else{
        GPIO_ResetBits(GPIOD,GPIO_Pin_3);
    }
}

```

- LED1,2 flag 변수에 따라 LED1,2의 점등 상태를 변화시킨다.

< PWM duty cycle 변경 >

```

void change_pwm_duty_cycle(int percentx10){
    int pwm_pulse;
    pwm_pulse = percentx10 * 20000 / 100;

    // declare the variable about OCinit
    TIM_OCInitTypeDef TIM_OCInitStructure;
    // Determines the duty level of PWM.
    TIM_OCInitStructure.TIM_OCMode= TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OCPolarity= TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OutputState= TIM_OutputState_Enable;

    TIM_OCInitStructure.TIM_Pulse = pwm_pulse;
    TIM_OC3Init(TIM3, &TIM_OCInitStructure);
}

```

- 매개변수 percentx10에 따라 PWM의 duty cycle을 변화시킨다.

< Delay >

```

void Delay(void) {
    int i;

    for (i = 0; i < 2000000; i++);
}

```

- 반복문을 통해 프로그램 동작을 Delay 시킨다.

< main 문 >

```

int main(void) {

    SystemInit();

    RCC_Configure();
    GPIO_Configure();
    TIM2_Configure();
    TIM3_Configure();
    NVIC_Configure();

    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();
    LCD_Clear(WHITE);
}

```

```

LCD_ShowString(LCD_TEAM_NAME_X, LCD_TEAM_NAME_Y, "THU_TEAM08", BLUE, WHITE);
LCD_ShowString(LCD_ON_X, LCD_ON_Y, "OFF", RED, WHITE);
LCD_ShowString(LCD_BUT_X, LCD_BUT_Y, "BTN", RED, WHITE);

LCD_DrawRectangle(LCD_REC_X, LCD_REC_Y, LCD_REC_X+LCD_REC_LEN,LCD_REC_Y+LCD_REC_LEN);

while (1) {
    // While waiting for touch to be recognized in the while loop, if it is recognized,
    // location is stored in cur_x and cur_y.
    // It controls the servo motor by changing the duty cycle of PWM.
    change_pwm_duty_cycle(3);
    Delay();
    change_pwm_duty_cycle(12);

    Touch_GetXY(&cur_x, &cur_y, 1);
    // locations where the touch is recognized, cur_x and cur_y, are stored as pixel_x and pixel_y
    Convert_Pos(cur_x, cur_y, &pixel_x, &pixel_y);

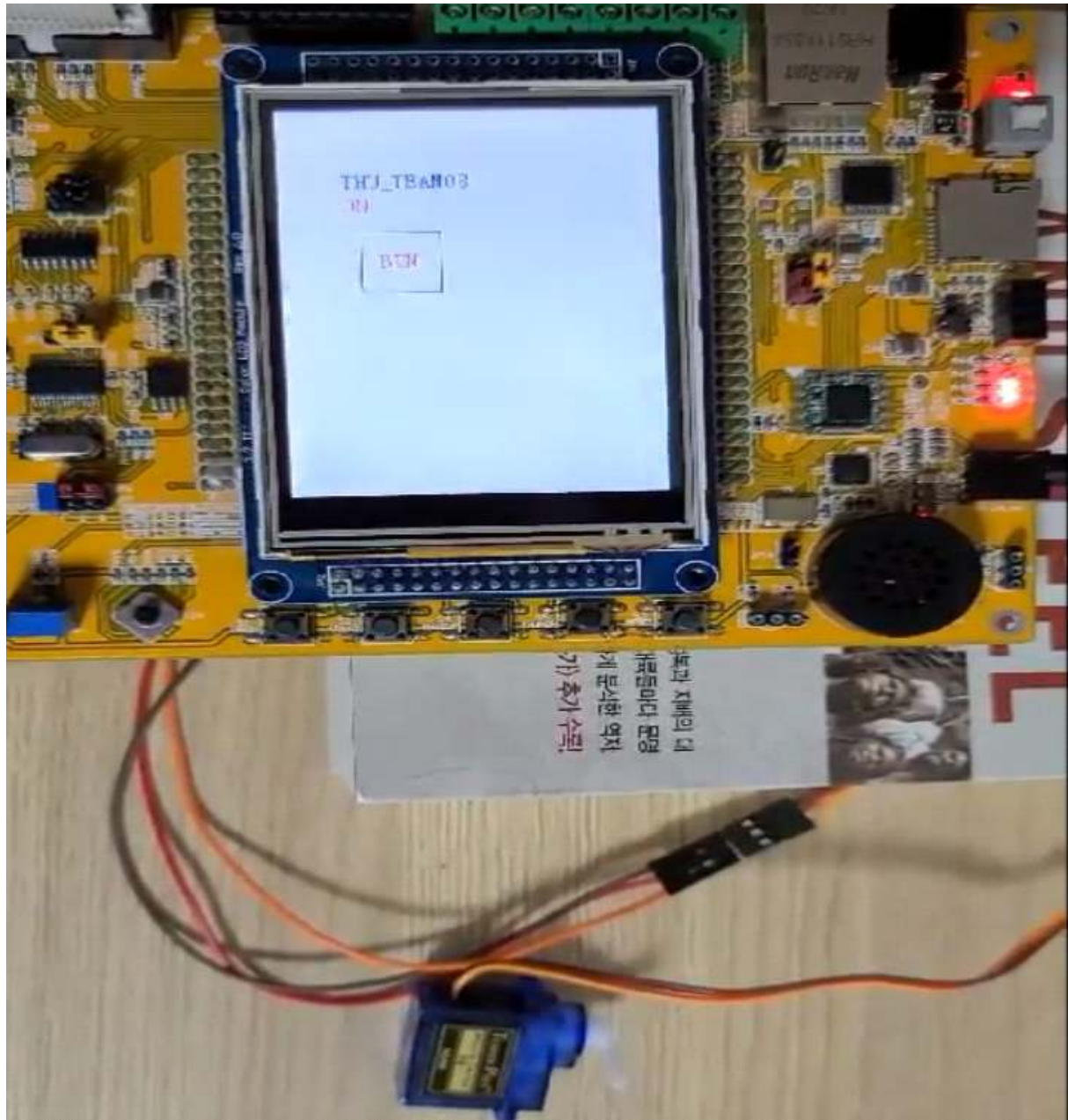
    // When a touch is recognized within the position of the rectangle representing the button,
    // the flag is switched to the opposite state of the current button.
    if(pixel_x>LCD_REC_X&&pixel_x<LCD_REC_X+LCD_REC_LEN){
        if(pixel_y>LCD_REC_Y&&pixel_y<LCD_REC_Y+LCD_REC_LEN){
            btn=!btn;
        }
    }
    if(btn){
        LCD_ShowString(LCD_ON_X, LCD_ON_Y, "ON ", RED, WHITE);
    }
    else{
        LCD_ShowString(LCD_ON_X, LCD_ON_Y, "OFF", RED, WHITE);
    }
    Delay();

}
return 0;
}

```

- 메인함수에서 주요 동작은 Button 클릭 시 LED btn flag를 변화시켜 LED의 상태를 변화시키는 것, change_pwm_duty_cycle 함수를 통해 duty cycle을 변화시키고 이를 통해 서보모터를 제어하는 것이다.

5. 실험 결과



- Button 클릭으로 인해 LED ON 상태가 되면 LED1과 LED2가 각각 1초, 5초마다 점멸되고, 서보모터가 -90도에서 90도로 움직이게 된다.

6. 결론 및 느낀점

지난 주차 실험과 동일하게 TFT LCD와 STM32 보드를 사용하여 실험을 진행하였고, 추가적으로 PWM을 통해 서보모터를 제어하였다. Timer에 의해서 LED1,2를 1초, 5초간 점멸시키고 PWM을 통해 서보모터를 동작시키는 실험을 진행하였으며 이번 실험을 통해 Timer에 대한 기본적인 개념 및 종류, 분주, PWM에 대해 익힐 수 있었다. STM32 보드에서의 period 및 prescaler의 동작 방식을 통해 timer의 동작에 대해 실제로 알아볼 수 있었으며 duty cycle에 따라 달라지는 서보모터의 움직임을 파악할 수 있었다. 실험 진행 중 어려웠던 점은 초기에 서보모터를 제어하기 위해서 duty cycle변경 방법에 대해 정확히 파악하지 못한 것이다. 하지만 강의 자료를 통해 해당 부분을 정확하게 이해한 후에는 원하는 대로 서보모터의 동작을 제어할 수 있었다.