# 11주차 실험 결과 보고서

## **0.** 목차



## **1.** 타이머의 이해

타이머는 주기적 시간 처리에 사용하는 디지털 카운터 회로 모듈로 펄스폭 계측, 주기적인 인터럽트 발생 등에 사용될 수 있다.

- 타이머의 종류와 특징
    1. SysTick Timer
       : 24bit 시스템 타이머로 count가 0에 도달 시 설정에 따라 인터럽트가 발생할 수 있다.
    2. Watchdog Timer
       : 소프트웨어 고장으로 인한 오작동을 감지하고 CPU가 올바르게 작동하지 않을 시 강제로 리셋시켜주는 타이머로 IWDG와 WWDG로 분류된다.
       - IWDG : LSI 클럭 기반으로 메인 클럭 고장에도 활성 상태 유지가 가능하며 카운터가 0이 되면 reset이 가능하다.
       - WWDG : APB1 클럭을 프리스케일해서 clock을 정의할 수 있으며 카운터가 0x40보다 작거나 Time-window 밖에 Reload 됐을 경우 Reset이 가능하다.카운터가 0x40과 같을 때에는 EWI 인터럽트가 발생하도록 설정도 가능하다.
    3. Advanced-control Timer(TIM1, TIM8)
       : prescaler를 이용해 설정 가능한 16-bit auto-reload counter를 포함하고 있다. 자원을 공유하지 않는 독립적인 구조이며, 동기화시키는 것도 가능하다.입력 신호 펄스 길이 측정 또는 출력 파형 생성 등에 사용될 수 있다.
    4. General-purpose Timer(TIM2~TIM5)
       : prescaler를 이용해 설정 가능한 16-bit up,down, up/down auto-reload counter를 포함하고 있다. 자원을 공유하지 않는 독립적인 구조이며, 동기화시키는 것도 가능하다.입력 신호 펄스 길이 측정 또는 출력 파형 생성 등에 사용될 수 있다. time prescaler와 RCC clock controller prescaler를 이용하면 펄스 길이와 파형 주기의 변조가 가능하다.
    5. Basic Timer(TIM6, TIM7)
       : 16-bit auto-reload 업카운터를 가지며, 16-bit prescaler를 통해 counter clock 주파수를 나눠서 설정 가능하다. DAC 트리거에 사용되고 카운터 오버플로우 발생 시 인터럽트/DMA를 생성한다.

## **2.** 분주 계산 방법의 이해

타이머는 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮추어 사용하는데, MCU에서 제공하는 Frequency를 사용하기 쉬운 값으로 바꾸어 주는 것을 분주라 한다.

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

timer clock frequency를 설정해둔 prescaler 값과 주기로 나누어 주게 되면, 분주된 주파수 값을 구할 수 있다. 이때 timer clock frequency는 라이브러리에서 설정된 값을 확인해 사용하면 된다.

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| $t_{res(TIM)}$ | Timer resolution time | - | 1 | - | $t_{TIMxCLK}$ |
| | | $f_{TIMxCLK}$ = 72 MHz | 13.9 | - | ns |
| $f_{EXT}$ | Timer external clock frequency on CH1 to CH4 | - | 0 | $f_{TIMxCLK}/2$ | MHz |
| | | $f_{TIMxCLK}$ = 72 MHz | 0 | 36 | MHz |
| $Res_{TIM}$ | Timer resolution | - | - | 16 | bit |
| $t_{COUNTER}$ | 16-bit counter clock period when internal clock is selected | - | 1 | 65536 | $t_{TIMxCLK}$ |
| | | $f_{TIMxCLK}$ = 72 MHz | 0.0139 | 910 | µs |
| $t_{MAX\_COUNT}$ | Maximum possible count | - | - | 65536 × 65536 | $t_{TIMxCLK}$ |
| | | $f_{TIMxCLK}$ = 72 MHz | - | 59.6 | s |

1. TIMx is used as a general term to refer to the TIM1, TIM2, TIM3, TIM4 and TIM5 timers.

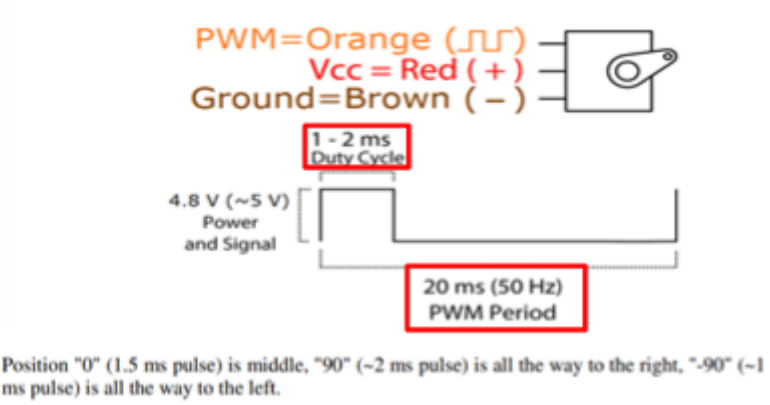
<그림 1> timer clock frequency 확인하기[1]

이번 실험에서는 그림과 같이 timer clock frequency로 72MHz를 사용한다.

## **3. PWM**의 이해

PWM(펄스 폭 변조)는 일정한 주기 내에서 Duty cycle을 변화 시켜서 평균 전압을 제어하는 방법으로 디지털신호를 아날로그 신호로 표현하기 위해 사용된다. Duty Cycle은 펄스폭을 주기로 나누어 구할 수 있으며 백분율로 표시한다.

[1] stm32_Datasheet, 63 page




<그림 2> 서보모터(SG90)의 PWM[2]

위의 그림을 통해 실험에서 사용하는 SG90 서보모터는 50Hz의 주파수를 요구함을 확인해 볼 수 있다.

STM32보드에서 사용하는 PWM 신호는 데이터시트의 Pin definitions 표를 통해 알 수 있다. 자세한 설명은 [4. 구현 내용]에서 확인할 수 있다.

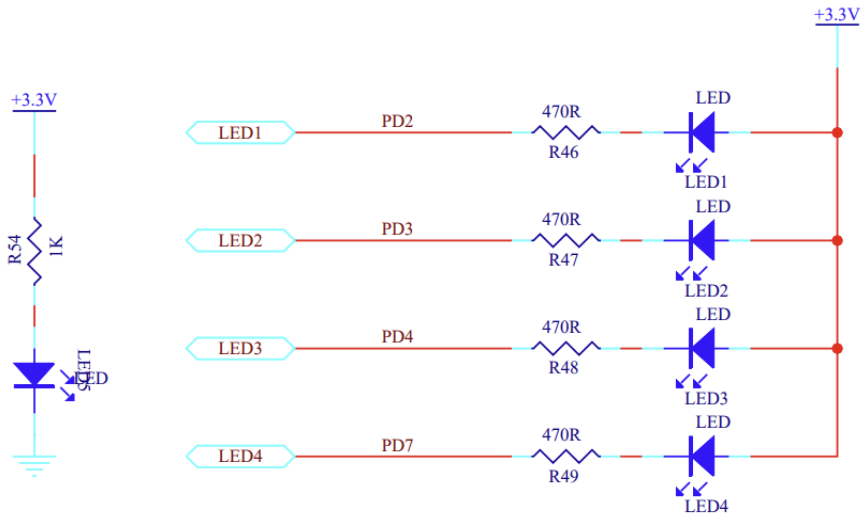
## 4. 구현 내용

### 4.1 RCC Init

LED ON 버튼 터치 시 TIM2 interrupt, TIM3 PWM을 활용하여 LED 2개와 서보모터 제어 동작이 가능하도록 하기 위해 TIM2, TIM3, AFIO에 전원을 인가한다.

```
RCC_APB1PeriphClockCmd(RCC_APB1ENR_TIM2EN, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
```

[2] PPT- 11주차 강의자료, 14 page




<그림 3> LED의 Port Configuration Register[3]

그림에서 볼 수 있듯이, LED1과 LED2에 연결된 핀이 PD2, PD3이므로 GPIOD를 enable시킨다.

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| H4 | 25 | 34 | PC5 | I/O | - | PC5 | ADC12_IN15/ ETH_MII_RXD1(8)/ ETH_RMII_RXD1 | - |
| J4 | 26 | 35 | PB0 | I/O | - | PB0 | ADC12_IN8/TIM3_CH3/ ETH_MII_RXD2(8) | TIM1_CH2N |
| K4 | 27 | 36 | PB1 | I/O | - | PB1 | ADC12_IN9/TIM3_CH4(7)/ ETH_MII_RXD3(8) | TIM1_CH3N |
| G5 | 28 | 37 | PB2 | I/O | FT | PB2/BOOT1 | - | - |

<그림 4> PIN Definitions[4]

TIM3 PWM을 이용할 것이고, 위 표에서 TIM3의 핀이 PB0이므로 GPIOB를 enable시킨다.

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
```

따라서 RccInit 함수를 정리해보면 다음과 같다.

```
void RccInit(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1ENR_TIM2EN, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
}
```

[3] STM32107VCT6_Schematic.pdf, LED, 3 page
[4] stm32_Datasheet.pdf, Table5. Pin definitions , 28 page

### 4.2. GPIO Init

```
GPIO_InitTypeDef GPIO_InitStructure;
uint16_t prescale = 0;

// LED 1
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOD, &GPIO_InitStructure);
// LED 2
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOD, &GPIO_InitStructure);

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

PD2와 PD3을 LED의 출력으로 사용하므로 2번핀과 3번핀을 GPIO_PIN으로 설정한다. 최대 출력 speed는 50MHz로 설정하고, General Purpose output Push-Pull 모드로 설정한다.

Port B의 0번 Pin을 PWM을 위해 사용하고, 출력 속도는 50MHz, Alternative Function Push-Pull 모드로 설정한다.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| J4 | 26 | 35 | PB0 | I/O | - | PB0 | ADC12_IN8/TIM3_CH3/ ETH_MII_RXD2(8) | TIM1_CH2N |

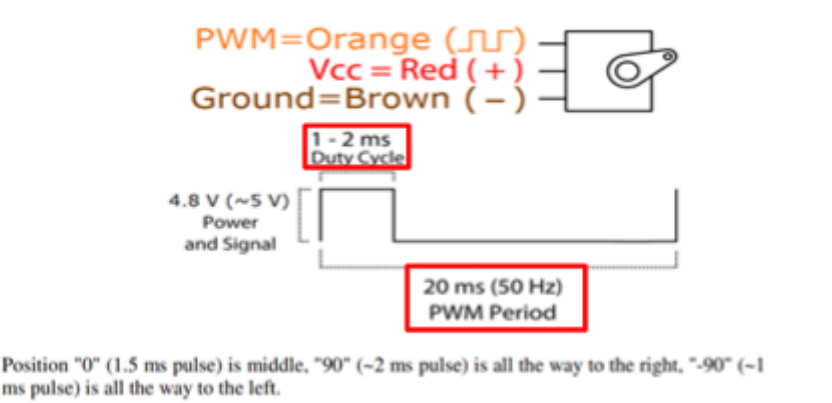
<그림 5> Pin Definitions[5]

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

TIM_TimeBaseStructure.TIM_Period = 20000 - 1;
TIM_TimeBaseStructure.TIM_Prescaler = prescale;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
```

[5] PPT- 11주차 강의자료, 15 page

```
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 1500;
TIM_OC3Init(TIM3, &TIM_OCInitStructure);

TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
TIM_ARRPreloadConfig(TIM3, ENABLE);
TIM_Cmd(TIM3, ENABLE);
```

<그림 6> 서보모터(SG90)의 PWM[6]




서보모터를 50Hz로 동작시키기 위해 SystemCoreClock인 72Mhz를 세분화하는 과정이 필요하다. 계산의 편의성을 위해 prescaler 값을 지정할 수 있는데, 이번 실험에서는 prescaler의 값을 (72-1)로 지정하였다. SystemCoreClock을 prescaler로 나누어주면 1Mhz가 되는데, 목표로 하는 PWM 주파수가 50Hz이므로 (20000-1)을 period로 설정하여 1Mhz / period를 통해 50Hz를 달성할 수 있다.

서보모터의 Duty Cycle이 1-2ms이므로 1.5ms로 설정하면 1500μs와 같다. 따라서 TIM_Pulse 에 1500을 대입해준다.

### 4.3 TIM configure

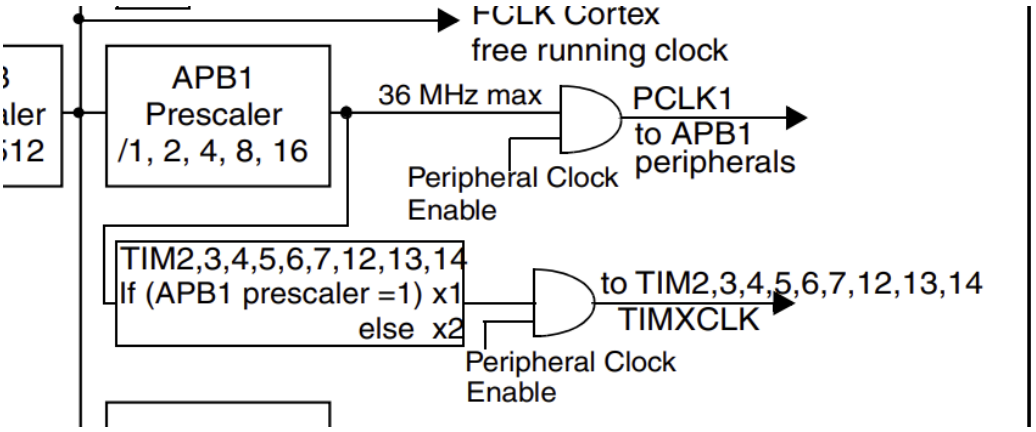
$$f_{clk} * \frac{1}{prescaler} * \frac{1}{period} = 주파수[Hz]$$

위 식을 참고해보면 주파수는 fIck * (1/prescaler) * (1/period)를 통해 구할 수 있다. 우선 fclk의 경우, 앞선 [2. 분주 계산 방법의 이해] 부분을 참고하면 72MHz임을 알 수 있다.

[6] PPT- 11주차 강의자료, 14 page

분주가 주파수를 사용하기 쉬운 값으로 바꾸어 주는 것을 의미하기 때문에, prescaler를 7200으로 설정하였다. 따라서 period = 72000000 / 7200 = 10000 이 된다. 이 값을 attribute에 넣어주면 다음과 같다.

```
TIM_TimeBaseStructure.TIM_Period = 10000;
TIM_TimeBaseStructure.TIM_Prescaler = 7200;
```




<그림 7 > Clock Tree의 일부[7]

그림과 같이 TIM2는 APBI Prescaler를 사용하므로, ClockDivision으로 TIM_CKD_DIV1을 이용한다. up counter로 모드를 설정해서 TIM_Configure 함수를 작성하면 다음과 같다.

```
void TIM_Configure(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_TimeBaseStructure.TIM_Period = 10000;
    TIM_TimeBaseStructure.TIM_Prescaler = 7200;
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_Cmd(TIM2, ENABLE);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
}
```

### 4.4 Nvic Init

```
void NvicInit(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    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);
}
```

[7] stm32_ReferenceManual.pdf, Clock tree, 93 page

인터럽트 스케쥴링을 위해 우선순위를 설정해주어야 한다. 이번 실험에서는 인터럽트가 짧은 시간 안에 완료된다. 따라서 인터럽트 간 우선순위 설정에 따른 스케줄링이 큰 영향을 미치지 않고 인터럽트들의 실행 순서의 변경을 원하지 않으므로 인터럽트가 대기하지 않고 즉시 실행되도록 우선순위들을 0으로 설정하였다.

### 4.5 LED Toggle

```
void ledToggle(int num)
{
    uint16_t pin;

    switch (num)
    {
    case 1:
        pin = GPIO_Pin_2;
        break;
    case 2:
        pin = GPIO_Pin_3;
        break;
    default:
        return;
    }

    if (GPIO_ReadOutputDataBit(GPIOD, pin) == Bit_RESET)
    {
        GPIO_SetBits(GPIOD, pin);
    }
    else
    {
        GPIO_ResetBits(GPIOD, pin);
    }
}
```

`ledToggle()`를 통하여 led를 제어한다. 함수의 입력으로는 제어할 led 번호가 필요하다. led가 꺼져있는 경우, led를 켜준다. led가 켜져있는 경우, led를 꺼준다.

### 4.6 Move Motor

```
void moveMotor()
{
    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 = motorAngle + 700;
    if (motorDir == 0)
    {
        motorAngle = motorAngle + 100;
```

```
        if (motorAngle == 1500)
            motorAngle = 0;
    }
    else
    {
        motorAngle = motorAngle - 100;
        if (motorAngle == 0)
            motorAngle = 1500;
    }

    TIM_OC3Init(TIM3, &TIM_OCInitStructure);
}
```

moveMotor()를 이용하여 모터의 동작을 제어한다. motorDir 변수가 0인 경우 모터의 각도가 10도씩 증가한다. motorDir 변수가 1인 경우 모터의 각도가 10도씩 감소한다. 모터의 구조체를 계속 초기화하는 방법으로 동작한다.

### 4.7 TIM2 IRQHandler

```
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
    {
        // 1초마다 count
        t1++;
        t2++;
        moveMotor();

        if (ledOn == 1)
        {
            // led 1 toggle
            ledToggle(1);
            if (t1 % 5 == 0)
            {
                // led 2 toggle
                ledToggle(2);
            }
        }

        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

매 초마다 인터럽트가 발생한다. ledOn 변수가 1일 때, 매 초마다 LED1의 상태를 변경하고, 모터를 동작시킨다. 매 5초마다 LED2의 상태를 변경한다.

### 4.8 main

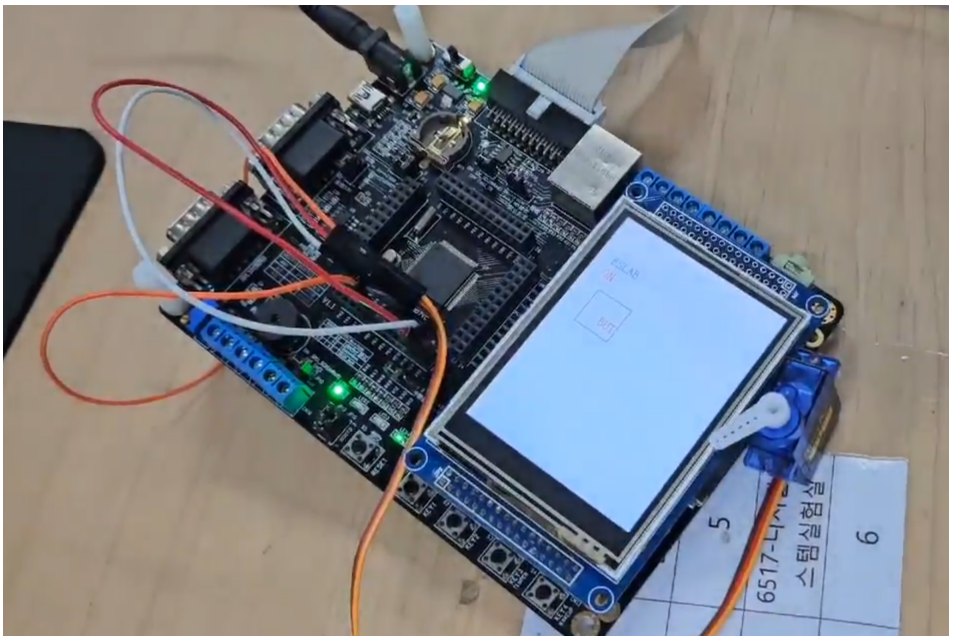
```
int main()
{
    SystemInit();
    RccInit();
    GpioInit();
    TIM_Configure();
    NvicInit();
    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();

    uint16_t pos_x, pos_y;
    uint16_t pix_x, pix_y;

    t1 = 0;
    t2 = 0;
    ledOn = 0;

    LCD_Clear(WHITE);

    // team name
     LCD_ShowString(LCD_TEAM_NAME_X, LCD_TEAM_NAME_Y, "THU_Team03",
BLUE, WHITE);
    // button
      LCD_DrawRectangle(LCD_BUTTON_X, LCD_BUTTON_Y, LCD_BUTTON_X +
LCD_BUTTON_W, LCD_BUTTON_Y + LCD_BUTTON_H);
     LCD_ShowString(LCD_BUTTON_X + (LCD_BUTTON_W / 2), LCD_BUTTON_Y
+ (LCD_BUTTON_H / 2), "BUT", RED, WHITE);

    while (1)
    {
        if (ledOn == 0)
        {
             LCD_ShowString(LCD_STATUS_X, LCD_STATUS_Y, "OFF", RED,
WHITE);
            motorDir = 0;
        }
        else
        {
             LCD_ShowString(LCD_STATUS_X, LCD_STATUS_Y, "ON ", RED,
WHITE);
            motorDir = 1;
        }
        // get touch coordinate
        Touch_GetXY(&pos_x, &pos_y, 1);
        Convert_Pos(pos_x, pos_y, &pix_x, &pix_y);
```

```
            if (
                pix_x >= LCD_BUTTON_X &&
                pix_x <= LCD_BUTTON_X + LCD_BUTTON_W &&
                pix_y >= LCD_BUTTON_Y &&
                pix_x <= LCD_BUTTON_Y + LCD_BUTTON_H)
            {
                // button 눌림
                ledOn = !ledOn;
            }
        }
}
```

위에서 정의한 초기화 설정 함수들을 실행한다. 그 후 팀 이름인 "THU_Team03"과 직사각형 모양의 버튼을 출력한다.

그 후, 폴링 방식을 통해 LCD 화면에 터치로 입력된 좌표를 구한다. 좌표가 버튼의 경계 안에 있을 경우, LED 상태와 모터의 동작 방향을 반대로 전환한다.

## **5.** 결과



<그림 8> LCD와 서보모터의 동작

LED ON 버튼 터치 시 1초마다 LED1이 TOGGLE되고, 5초마다 LED2가 TOGGLE 되는 모습을 확인할 수 있다. 또한, 서보모터는 1초마다 한쪽 방향으로 조금씩(100) 이동한다.

LED OFF 버튼 터치 시 LED Toggle 동작 해제되고 서보모터가 1초마다 반대쪽 방향으로 조금씩(100) 이동하는 모습을 확인할 수 있다.