# 임베디드시스템 설계 및 실험

## 11주차 결과 보고서

2023.11.14

4분반 02조

202155592 이지수

202155553 박은재

201924515 유승훈

201524410 고상현

201924402 강민수

# 1. 실험 목표

- 임베디드 시스템의 기본 원리 습득
- Timer, PKM 이해 및 실습

# 2. 배경 지식

1. 타이머

- 주기적 시간 처리에 사용하는 디지털 카운터 회로 모듈이다.
- 펄스폭 계측, 주기적인 interrupt 발생 등에 사용한다.
- 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮춘 후 낮아진 주파수로 8,16 비트 등의 카운터 회로를 사용하여 주기를 얻는다.

2. STM32 타이머 종류

- Systick timer
- Watchdog Timer
- Advanced-control Timer (TIM1, TIM8)
- General-purpose Timer (TIM2 ~ TIM5): General-purpose timer은 prescaler를 이용해 설정 가능한 16-bit up, down, up/down auto-reload counter를 포함하고 있다. 입력 신호의 펄스 길이 측정(input capture) 또는 출력 파형 발생(output compare and PWM) 등 다양한 용도로 사용할 수 있다. 펄스 길이와 파형 주기는 timer prescaler와 the RCC clock controller prescaler를 사용하여 몇 $\mu s$에서 몇 $ms$까지 변조할 수 있다. 타이머들은 완전히 독립적이며, 어떤 자원도 공유하지 않으나 동기화 가능하다.
- Basic Timer (TIM6, TIM7)

3. 분주 계산

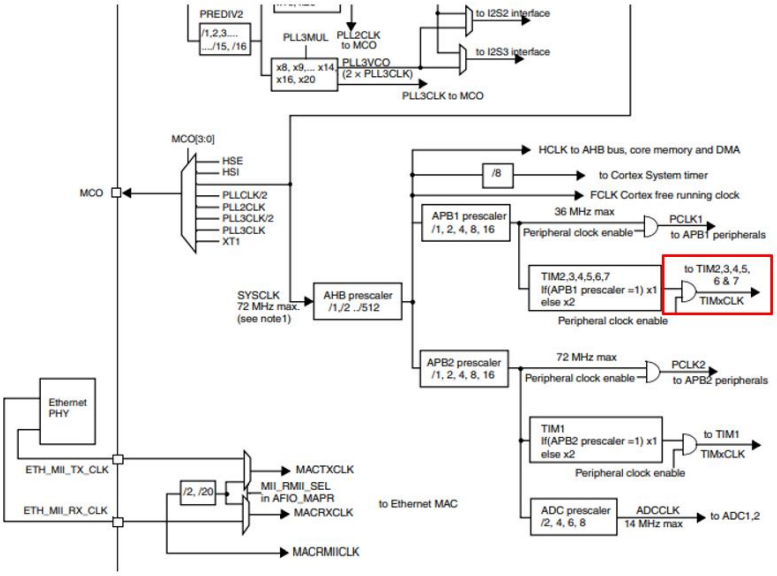
$$\frac{1}{f_{clk}} \times prescaler \times period$$

- 분주란 MCU에서 제공하는 frequency를 우리가 사용하기 쉬운 값으로 바꾸어 주는 것을 말한다.

- Counter clock frequency를 1~65536의 값으로 나누기 위해 16 bit programmable prescaler 사용한다

- period로 몇 번 count 하는지 설정한다.

4. Clock frequency




라이브러리에서 설정된 timer clock frequency를 확인한다.

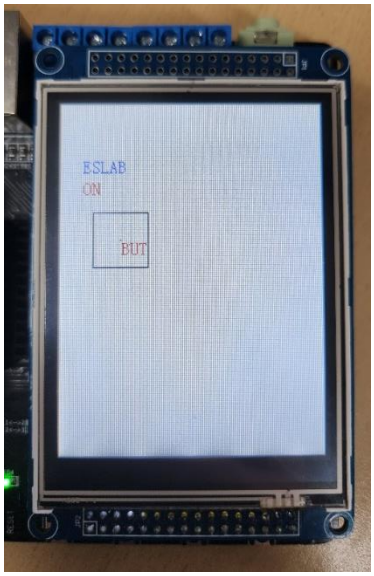
5. PWM(pulse width modulation)

- 일정한 주기 내에서 duty ratio를 변화시켜서 평균 전압을 제어하는 방법이다.

- 대부분의 서보모터는 50Hz~1000Hz의 주파수를 요구하고 데이터 시트를 반드시 확인해서 사용해야 한다.

# 3. 실험 세부 내용

<Timer, PWM 서보모터 이해 및 실습>




1. TFT LCD 에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼 생성
2. LED ON 버튼 터치 시 TIM2 interrupt, TIM3 PWM 을 활용하여 LED 2 개와 서보모터 제어
   - LED : 1 초 마다 LED1 TOGGLE, 5 초 마다 LED2 TOGGLE
   - 서보모터 : 1 초 마다 한쪽 방향으로 조금씩(100) 이동
3. LED OFF 버튼 터치 시 LED Toggle 동작 해제 및 서보모터 동작 반전
   - 서보모터 : 1 초 마다 반대쪽 방향으로 조금씩(100) 이동

# 4. 실험 방법

## 4-1 전처리 및 RCC/GPIO Configure

```
void RCCInit() {
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);


}

void GpioInit() {
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure_LED;

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //PIN B0 = ADC_Channel_8
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //or OD
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_InitStructure_LED.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure_LED.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure_LED.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure_LED);
}
```

stm보드의 Timer 2 번 3 번을 사용하기 위해 RCCInit에서 클럭 인가를 한다.

또한 LED 및 Timer3의 채널 3 번을 사용하기 위해 GPIOB 및 D에 클럭 인가를 한다.

1,2 번 LED 사용을 위해 GPIOInit 함수를 활용하여 D 2,3 번 Port를 초기화 한다.

Timer3의 채널 3 번 사용을 위해 B0 Port를 Alternative Mode 로 초기화 한다.

## 4-2 Timer Configure

```
void TIM_configure() {
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    TIM_TimeBaseStructure.TIM_Period = 10000;
    TIM_TimeBaseStructure.TIM_Prescaler = 7200;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_ARRPreloadConfig(TIM2, ENABLE);
    TIM_Cmd(TIM2, ENABLE);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    ////////////////////////////////////////////

    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure_2;
    TIM_OCInitTypeDef TIM_OCInitStructure_2;

    TIM_TimeBaseStructure_2.TIM_Period = 20000;
    TIM_TimeBaseStructure_2.TIM_Prescaler = 72;              //set 50Hz
    TIM_TimeBaseStructure_2.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure_2.TIM_CounterMode = TIM_CounterMode_Down;

    TIM_OCInitStructure_2.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure_2.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure_2.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure_2.TIM_Pulse = 1500;                  // set move 0
    TIM_OC3Init(TIM3, &TIM_OCInitStructure_2);

    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure_2);
    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
    TIM_Cmd(TIM3, ENABLE);

}
```

Timer Configure 함수를 이용하여 Timer2 와 3의 설정을 조율한다. MCU에서 제공하는 72MHz의 주파수를 적당히 분주하여 원하는 대로 조정하는데 방법은 다음과 같다.

Period 와 Prescaler의 값만큼 -물론 범위가 존재하지만- 주파수를 나누어 줄 수 있다. 따라서

72MHz/10000*7200으로 Timer2의 경우 1Hz의 주파수를 갖고 Timer3의 경우 72MHz/20000*72로 50Hz의 주파수를 갖게 되는데 이는 서보모터의 권장 주파수가 50~1000Hz 이기 때문이다.

Timer2의 경우 분주 된 주파수를 그대로 사용하기 때문에 TODO의 Period 와 Prescaler 값만

조절해주면 되지만, Timer3의 경우 추가적으로 Pulse 값을 조정해주어야 한다.

Pulse 값은 분주한 주파수의 Duty 값을 정해주기 위해 조정하는 값이다.

Pulse 값에 따라 서보모터의 날개 각도가 달라지게 된다.

이번 실험에서 초기 값을 1500 으로 두었다.

4-3 NVIC_Configure / TIM2_IRQHandler / changeServoM

NvicInit 함수로 인터럽트 설정을 한다.

```
void NvicInit() {
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 01;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}
```

NVIC를 활용하여 인터럽트 설정을 해주고 Timer2의 인터럽트 Handler를 조정하여 시간을 계산한다. 또한 changeServoM 함수를 이용해 서보모터의 날개 각도를 조정하는데 이 때에 Handler의 계산한 시간을 활용하여 1 초에 1 회씩 날개를 돌아가게 한다. 이번 실험에서 Timer2 번의 인터럽트 설정을 해 주되 우선순위는 고려하지 않는다.

```
void changeServoM(int percent) {

    int myPulse = percent;

    TIM_OCInitTypeDef TIM_OCInitStructure_2;

    TIM_OCInitStructure_2.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure_2.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure_2.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure_2.TIM_Pulse = myPulse;
    TIM_OC3Init(TIM3, &TIM_OCInitStructure_2);


}

int t1 = 0;
int index = 0;
int arr[17] = { 700,800,900,1000,1100,1200,1300,1400,1500,1600,1700,1800,1900,2000,2100,2200,2300 };
void TIM2_IRQHandler(void) {
    t1++;
    if (t1 == 0xffff) t1 = 1;
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    if (flag == 1) {
        index++;
        changeServoM(arr[index % 17]);
    }
    else {
        index--;
        if (index == -1) {
            index = 16;
        }
        changeServoM(arr[index % 17]);
    }
}
```

changeServoM 함수에서 Timer3의 펄스값을 조정하여 서보모터의 날개 각도를 변하게 한다. 정수형의 t1 변수를 0으로 초기화 하고 전역으로 두어 1초에 한번씩 호출되는 Timer2의 인터럽트 함수를 이용해 시간을 계산하고 거기에 더해 서보모터 또한 1초에 한번 씩 돌아가게 한다. 그리고 LED ON 버튼 터치 시에는 서브모터가 1초 마다 한쪽 방향으로 100씩 이동하게 하고 LED OFF 버튼 터치 시에는 서브모터가 1초 마다 반대쪽 방향으로 100씩 이동하게 하였다. 그리고 TIM_ClearITPendingBit 함수를 활용해 인터럽트를 초기화 하여 핸들러가 다시 호출되게 한다.

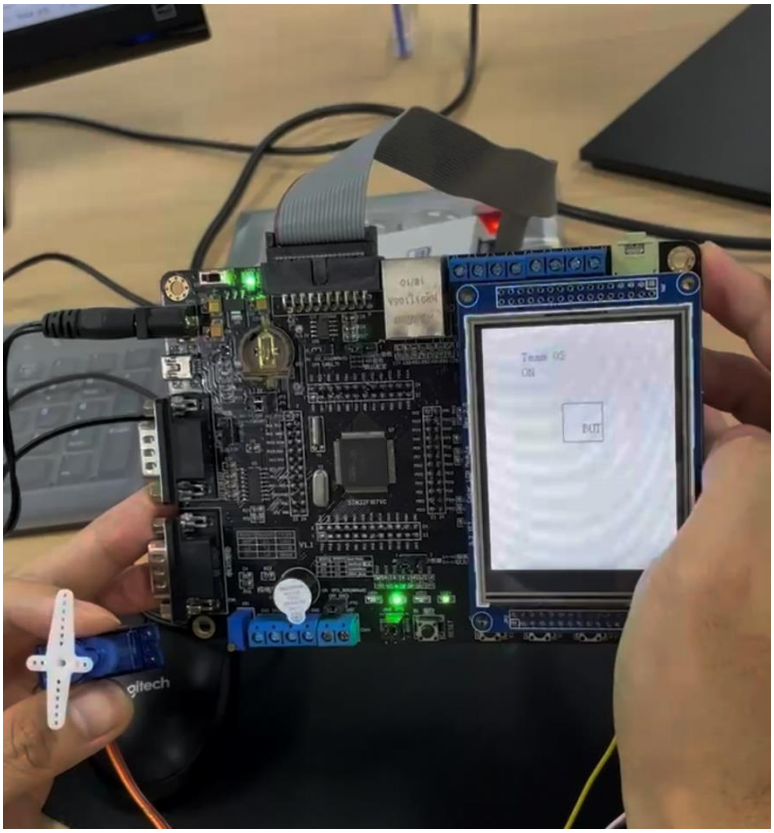
## 4-4 main 함수

앞서 작성한 함수를 통해 클럭 인가 및 초기화 작업을 하고 while 문을 활용하여 요구 동작을 구현한다. 먼저 LCD_ShowString 함수를 이용해 좌상단에 Team02를 출력한다. LCD_DrawRectangle 함수를 이용해 버튼의 역할을 할 사각형을 출력하고 버튼임을 표시하기 위해 LCD_ShowString 함수를 활용, But 라는 문자열을 사각형 내부에 출력한다. Touch_GetXY 함수를 활용해 사용자가 클릭한 스크린의 위치를 알아내고 Convert_Pos 함수를 이용해 스크린의 좌표값을 직관적인 값으로 전환한다. 조건문을 활용해 버튼을 클릭할 때 마다 토글 방식으로 ON OF 가 전환되게 하고 ON 상태일 시 ON 을 좌상단에 출력하고 LED1 번은 1초마다 2 번은 5 초마다 토글되게 한다. main 함수의 코드는 다음 페이지에 첨부하였다.

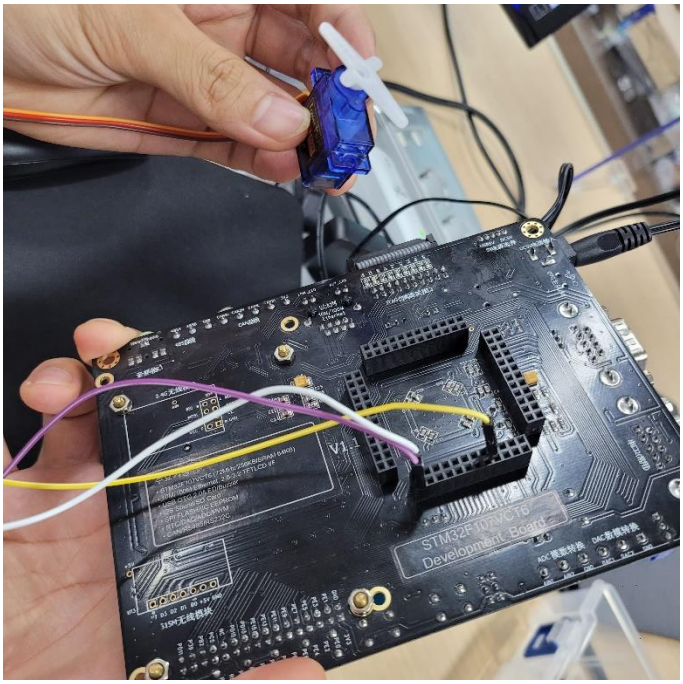
```c
uint16_t x, y;
uint16_t lcd_x, lcd_y;
int main() {
    SystemInit();
    RCCInit();
    GpioInit();
    TIM_configure();
    NvicInit();
    //--------------------------
    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();
    LCD_Clear(WHITE);

    while (1) {
        LCD_ShowString(50, 30, "Team 02", BLACK, WHITE);

        LCD_DrawRectangle(100, 100, 150, 150);
        LCD_ShowString(125, 125, "BUT", BLACK, WHITE);

        Touch_GetXY(&x, &y, 0);
        Convert_Pos(x, y, &lcd_x, &lcd_y);


        if (flag ^ (100 < lcd_x && lcd_x < 150 && 100 < lcd_y && lcd_y < 150)) {
            flag = 1;
        }
        else { flag = 0; }

        if (flag == 1) {
            LCD_ShowString(50, 50, "ON  ", BLACK, WHITE);
            if (t1 % 2) {
                GPIO_SetBits(GPIOD, GPIO_Pin_2);
            }
            else {
                GPIO_ResetBits(GPIOD, GPIO_Pin_2);
            }
            if (t1 % 10 < 5) {
                GPIO_SetBits(GPIOD, GPIO_Pin_3);
            }
            else {
                GPIO_ResetBits(GPIOD, GPIO_Pin_3);
            }
        }
        else {
            GPIO_ResetBits(GPIOD, GPIO_Pin_2);
            GPIO_ResetBits(GPIOD, GPIO_Pin_3);
            LCD_ShowString(50, 50, "OFF", BLACK, WHITE);
        }

    }
}
```

## 5. 실험 결과

ON 일 때 LED가 점등되는 모습




서보모터와 보드 연결



# 6. 결론

이번 실험에서는 타이머와 분주 계산, PWM 제어를 통해 서보모터를 제어하는 방법을 배웠다. 타이머 종류에는 Systick timer, IWDG/WWDG Timer-Watching timer, Advanced-control timer, Basic timer, General-purpose timer 가 있으며 이번 실험에서는 General-purpose timer 두 개를 사용했다. 한 타이머는 PWM 제어를 하는데 사용했고 나머지 한 개는 interrupt를 통해 시간을 측정하는 용도로 사용했다.