임베디드시스템 설계 및 실험

11 주차 결과보고서

2023.11.20

003 (수) 분반

8 조

201924437 김윤하

202055604 조찬우

202155565 성가빈

202012142 이은진

목차

1. 실험 목표

1. 임베디드 시스템의 기본 원리 습득
2. Timer, PWM 이해 및 실습

2. 배경 지식

1. Timer

- 임의의 주기를 갖는 신호를 측정하거나 생성할 때 사용하는 디지털 카운터 회로.
- 주기적인 interrupt 를 발생시키거나, PWM 과 같은 파형을 생성 및 출력할 때에 사용.
- 주파수가 높기 때문에 우선 prescaler 를 사용하여 주파수를 낮춘 후 낮아진 주파수로 8, 16 비트 등의 카운터 회로를 사용하여 주기를 얻음.

2. STM32 Timer 종류

1. Systick Timer

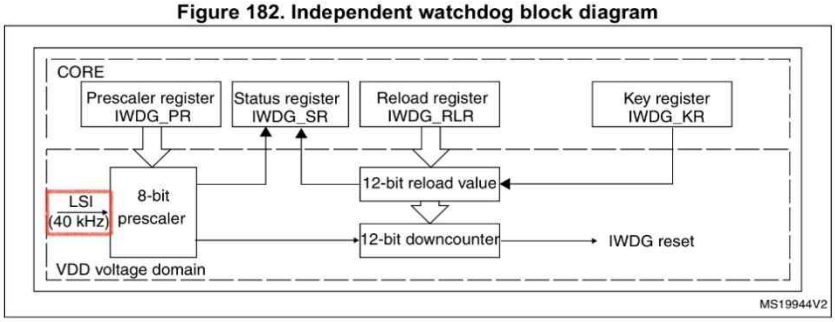
- Real-time operating system 전용이지만, standard down counter 로도 사용 가능.
- 24bit down counter, auto reload capability, programmable clock source.
- Counter 값이 0 에 도달하면 설정에 따라 interrupt 를 발생시킴.

2. Watchdog Timer

- 특수 상황에서 CPU 가 올바르게 작동하지 않을 시 강제 리셋시키는 기능을 수행함.
- IWDG 와 WWDG 모두 소프트웨어 고장으로 인한 오작동을 감지하고 이를 해결함.
- 카운터가 주어진 시간 초과 값에 도달했을 때, 시스템 재설정 또는 인터럽트(only WWDG)를 트리거함.

- **IWDG(Independent Watchdog):**

Figure 182. Independent watchdog block diagram

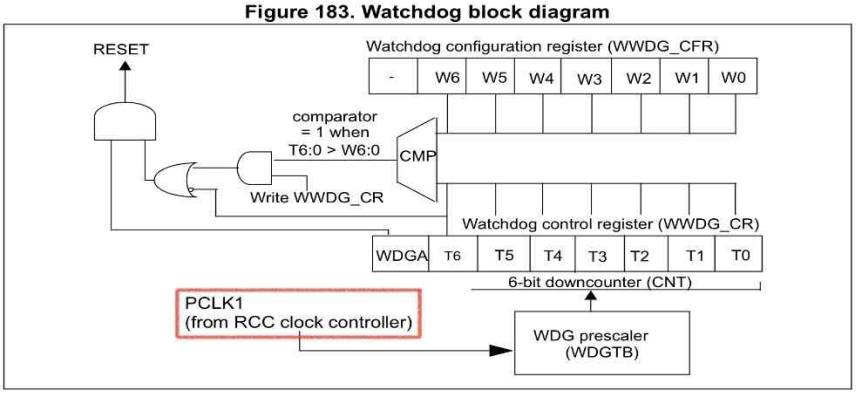


[그림 1] IWDG(Independent)

- LSI 클럭 기반으로 메인 클럭이 고장나도 활성 상태 유지 가능.
- 타이밍 정확도 제약이 낮은 어플리케이션에 사용 적합함.
- Counter 값이 0 에 도달하면 reset 함.

- **WWDG(Window Watchdog):**

Figure 183. Watchdog block diagram

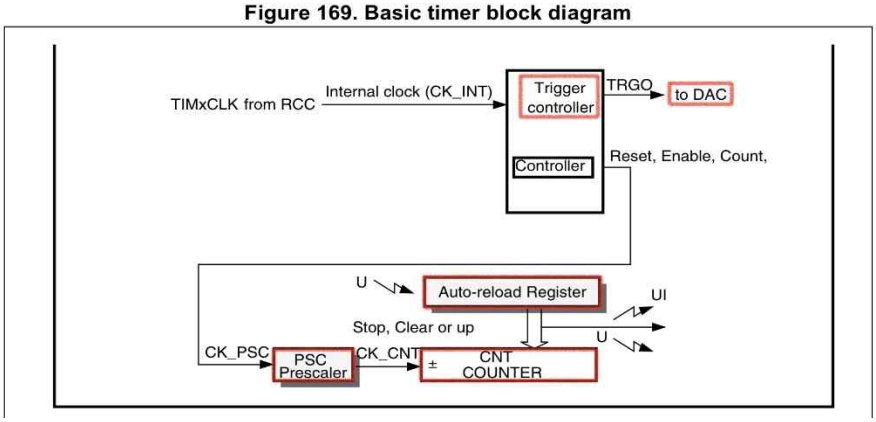


[그림 2] WWDG(Window)

- 7bit douwn counter 로 APB1 클럭을 프리스케일하여 정의 가능함.
- 비정상적 어플리케이션 동작 감지를 위해 설정 가능한 time window 가 있음.
- time window 내에서 반응하도록 요구하는 어플리케이션에 사용하기 적합함.
- counter 가 0X40 보다 작거나, time window 밖에서 reload 될 경우에 reset 가능.
- Early Wakeup Interrupt(EWI): counter 가 0X40 일 때, interrupt 발생 설정 가능.

3. Basic Timer(TIM6 & TIM7)

Figure 169. Basic timer block diagram

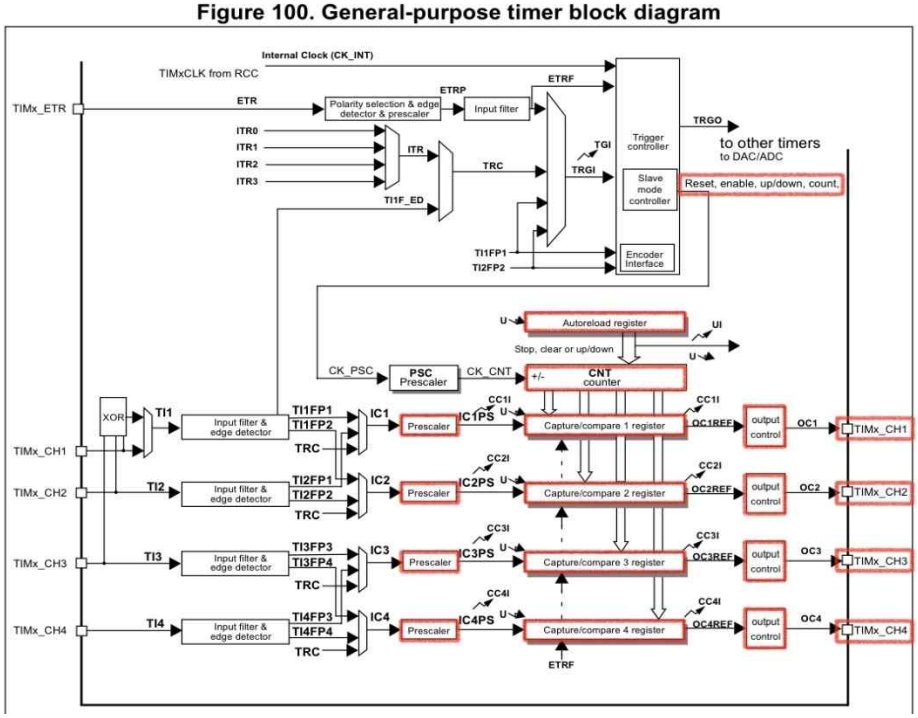


[그림 3] Basic Timer

- 16bit auto reload up counter 로 DAC 트리거에 사용.
- 설정 가능한 16bit prescaler 를 이용해 counter clock 주파수를 나눠 설정 가능.
- 카운터 오버플로우 발생 시에 interrupt 또는 DMA 생성 가능.

4. General Purpose Timer(TIM2 to TIM5)

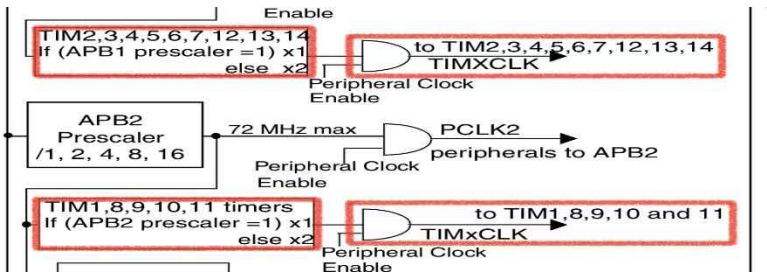
Figure 100. General-purpose timer block diagram



[그림 4] General Purpose Timer

- prescaler 를 이용해 설정 가능한 16bit up, down, up/down auto reload counter 를 포함함.
- 입력 신호의 철스 길이 측정(input caputer) 또는 출력 파형 발생(output compare and PWM) 등 다양한 용도로 사용 가능.
- 펄스 길이와 파형 주기는 timer prescaler 와 RCC clock controller prescaler 를 사용하여 μs~ms 까지 변조할 수 있음.
- 타이머들은 서로 완전히 독립적이고 자원을 공유하지 않지만, 동기화도 가능.

5. Advanced Control Timer(TIM1 & TIM8)



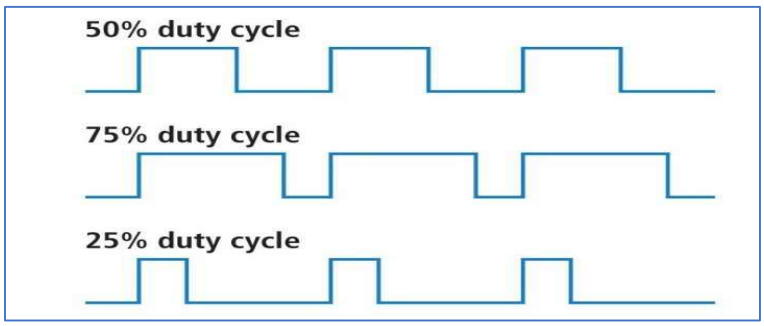
[그림 5] Advanced Control Timer

- 프리스케일러를 이용해 설정 가능한 16bit auto reload counter 를 포함.

- 입력 신호 펄스 길이 측정(input capture) 또는 출력 파형 생성(output compare, PWM, complementary PWM with dead time insertion) 등에 사용 가능.
- Advanced control 은 general purpose 보다 좀 더 많은 기능을 가짐.
- Advanced control 과 general purpose 는 자원을 공유하지 않는 독립적 구조이며, 동기화 시켜 사용하는 것도 가능.

3. PWM(Pulse Width Modulation)

- PWM(Pulse Width Modulation) 방식이란 일정한 주기 내에서 Duty Cycle 을 변화시켜서 평균 전압을 제어하는 방법.
- 예) 0~5V 전력 범위에서 2.5V 의 전압을 가하고 싶다면, 50%의 Duty Cycle 을 적용함.



[그림 6] Duty Cycle

- Duty Cycle 이란 위의 그림처럼 HIGH 신호와 LOW 신호의 비율을 나타내기 위한 용어.
- 50%가 넘는 Duty Cycle 을 가진다면 한 주기에서 HIGH 인 신호가 LOW 인 신호보다 더 많은 비율을 차지하며, 50% 이하라면 HIGH 가 LOW 보다 더 적은 비율을 차지함.
- 100% Duty Cycle 은 단순히 HIGH 신호로, 0% Duty Cycle 은 GND 로 생각할 수 있음.
- 서보모터는 PWM 신호를 이용하는 대표적인 예시임.
- 서보모터를 특정 각도에 위치시키기 위해서는 그에 대응되는 PWM 신호를 입력.

4. 분주 계산

- 분주란 MCU 에서 제공하는 주파수를 우리가 사용하는 쉬운 값으로 바꾸는 것.
- 주파수를 1~65536 의 값으로 나누기 위해서 16bit programmable prescaler 를 사용.
- Period 값으로 몇 번 count 하는지 설정함.

$$\frac{1}{f_{clk}} * prescaler \quad * period$$

$$\frac{1}{72\,(MHz)} * 7200 * 10000 = 1(s)$$

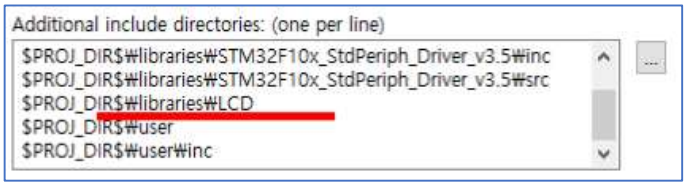
3. 세부 실험 내용

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
   - LED : LED Toggle 동작 해제 (LED 꺼짐)
   - 서보모터 : 1 초 마다 반대쪽 방향으로 조금씩(100) 이동

4. 실험 과정 및 결과

1. 프로젝트 생성 및 설정

이전 실험과 마찬가지로 프로젝트를 생성하고 설정한다. 또한, 10 주차 실험과 마찬가지로 LCD 를 사용하므로 LCD 관련 라이브러리도 프로젝트에 추가해 주어야 한다.



[그림 7] LCD 라이브러리 추가

2. 함수 구현 및 main 문 작성

1. 라이브러리 및 전역 변수

```
#include "stm32f10x.h"
#include "core_cm3.h"
#include "misc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_adc.h"
#include "stm32f10x_tim.h"
#include "lcd.h"
#include "touch.h"

int color[12] = {WHITE,CYAN,BLUE,RED,MAGENTA,LGRAY,GREEN,YELLOW,BROWN,BRRED,GRAY};
uint16_t x, y;

int toggle = 0;
int pulse = 1500;
int ledCount = 0;
```

[그림 8] 라이브버리 및 전역 변수

[그림 8]은 필요한 라이브러리들을 include 하고, 전역 변수를 선언한 것이다. color 배열과 x, y 변수는 10 주차 실험과 동일하게 LCD 화면의 글자 색상과 좌표를 지정하기 위한

변수이다. toggle 은 LED1 과 LED2 의 토글 상태 즉 LED ON 또는 LED OFF 를 나타내기 위한 변수이다. pulse 은 서보모터의 위치 각을 조절하기 위한 변수이다. 최소값 1000 과 최대값 2000 의 중간값인 1500 을 초기값으로 두었다. ledCount 은 LED1 과 LED2 의 토글을 위해 1 초, 5 초를 세기 위한 변수이다.

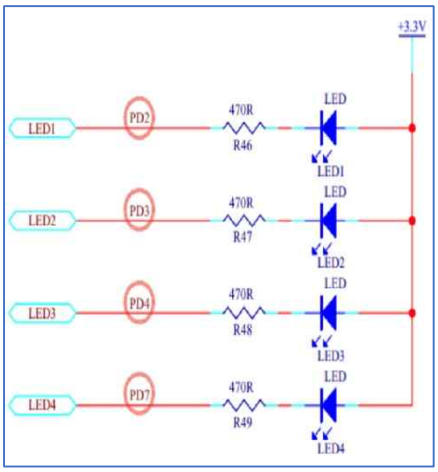
2. RCC_Configure() 함수

```
void RCC_Configure(){
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //서 보 모 터
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); //LED
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); //TIM2
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //TIM3
}
```

[그림 9] RCC_Configure() 함수

APB1	UART5	0x4000 5000 - 0x4000 53FF
	UART4	0x4000 4C00 - 0x4000 4FFF
	USART3	0x4000 4800 - 0x4000 4BFF
	USART2	0x4000 4400 - 0x4000 47FF
	Reserved	0x4000 4000 - 0x4000 43FF
	SPI3/I2S3	0x4000 3C00 - 0x4000 3FFF
	SPI2/I2S2	0x4000 3800 - 0x4000 3BFF
	Reserved	0x4000 3400 - 0x4000 37FF
	IWDG	0x4000 3000 - 0x4000 33FF
	WWDG	0x4000 2C00 - 0x4000 2FFF
	RTC	0x4000 2800 - 0x4000 2BFF
	Reserved	0x4000 1800 - 0x4000 27FF
	TIM7	0x4000 1400 - 0x4000 17FF
	TIM6	0x4000 1000 - 0x4000 13FF
	TIM5	0x4000 0C00 - 0x4000 0FFF
	TIM4	0x4000 0800 - 0x4000 0BFF
	TIM3	0x4000 0400 - 0x4000 07FF
	TIM2	0x4000 0000 - 0x4000 03FF

[그림 10] APB1(TIM2, TIM3)



[그림 11] LED pin

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	-	-
H5	-	38	PE7	I/O	FT	PE7	-	TIM1_ETR

[그림 12] TIM3_CH3 연결 pin

[그림 9]은 타이머를 위한 TIM2 과 TIM3, 입출력을 위한 GPIOB 과 GPIOD 를 사용하기 위해 RCC 클럭 인가를 ENABLE 시켜주는 것이다. **[그림 10]**을 보면 TIM2 와 TIM3 는 APB2 에 속한 GPIO 와 달리 APB1 에 속하므로 RCC_APB1Periph_TIM2 와 RCC_APB1Periph_TIM3 를 ENABLE 시켜야 한다. LED1 과 LED2 는 **[그림 11]**에서 확인할 수 있듯이 PD2 와 PD3 에 연결되어 있으므로 RCC_APB2Periph_GPIOD 를 ENABLE 시켜야 한다. **[그림 12]**에서 볼 수 있듯이 TIM3 의 채널 3 을 이용하여 서보모터에 PWM 신호를 주려면 PB0 pin 을 이용해야 하므로 RCC_APB2Periph_GPIOB 를 ENABLE 시켜야 한다.

3. GPIO_Configure() 함수

```
void GPIO_Configure(){
  GPIO_InitTypeDef GPIO_InitStructure;

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

  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);
}
```

[그림 13] GPIO_Configure() 함수

[그림 13]은 GPIO pin 모드를 설정해주는 코드이다. GPIOD2 와 GPIOD3 는 LED 출력을 위해 GPIO_Mode_Out_PP 모드로 설정해주고, GPIOB0 은 서보모터의 PWM 신호 출력을 위해서 GPIO_Mode_AF_PP 모드로 설정해준다.

4. NVIC_Configure() 함수

```
void NVIC_Configure(){
  NVIC_InitTypeDef NVIC_InitStructure;

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
  NVIC_EnableIRQ(TIM2_IRQn);

  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xf;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
  NVIC_Init(&NVIC_InitStructure);
}
```

[그림 14] NVIC_Configure() 함수

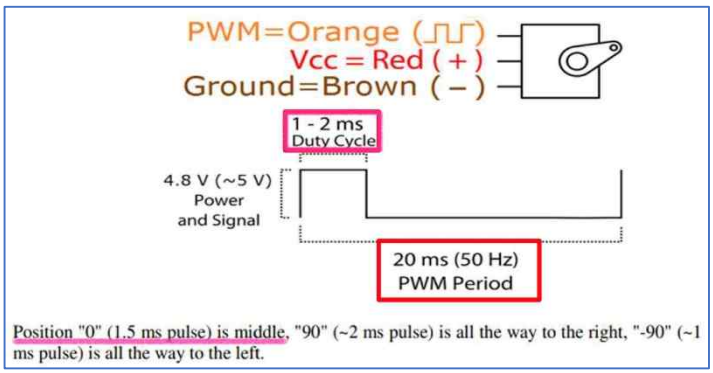
[그림 14]은 TIM2_IPQn 인터럽트 설정을 위한 코드이다. 이전 실험들과 마찬가지로 하나의 인터럽트만 사용하므로 우선순위 그룹과 우선순위가 중요하지 않다. 따라서, 이전 실험들과 똑같이 임의로 설정하였다.

5. TIM_Configure() 함수

```
void TIM_Configure() {
  //TIM3
  int prescale = (uint16_t) (SystemCoreClock / 1000000);
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  TIM_OCInitTypeDef TIM_OCInitStructure;

  TIM_TimeBaseStructure.TIM_Period = 20000;
  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;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 1000; // us
  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 15] TIM3 의 Tim_Configure() 함수

[그림 15]은 서보모터 구동에 사용하는 TIM3 의 설정 코드이다. TIM_TimBaseInit 함수를 통해 분주비를 설정하고, TIM_OC3Init 함수를 통해 PWM 신호의 duty cycle(pulse)을 설정함을 확인할 수 있다.



[그림 16] 서보모터 구동 주파수 및 duty cycle

[그림 16]에서 확인할 수 있듯이 우리가 실험에서 사용하는 서보모터 구동에는 50Hz 주파수의 신호가 필요하다. 앞서 배운 분주 계산 식을 이용하면, prescaler 값으로 72 를, period 값으로 20000 을 설정해주면, 72MHz 의 SystemCoreClock 으로부터 50Hz 주파수의 신호를 얻을 수 있다. int prescale = (uint16_t) (SystemCoreClock / 1000000); 코드와 TIM_TimeBaseStructure.TIM_Prescaler = prescale; 코드를 통해 prescaler 값을 72 로 설정하고, TIM_TimeBaseStructure.TIM_Period = 20000; 코드를 통해 period 값을 20000 으로 설정하고 있다. 또한, 1.0ms~2.0ms 의 pulse 값을 설정해 서보모터의 위치 각을 설정함도 알 수 있다. TIM_OCInitStructure.TIM_Pulse = 1000; 코드를 통해 설정할 수 있는 가장 작은 pulse 값인 1000 μs(1.0ms)로 초기 설정해 주었다.

```
  //TIM2
  prescale = (uint16_t) (SystemCoreClock / 10000);

  TIM_TimeBaseStructure.TIM_Period = 10000;
  TIM_TimeBaseStructure.TIM_Prescaler = prescale;
  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);
}
```

[그림 17] TIM2 의 Tim_Configure() 함수

[그림 17]은 LED 의 토글을 위해 1 초 간격 인터럽트를 발생을 위한 TIM2 의 설정 코드이다. TIM3 와 마찬가지로 TIM_TimBaseInit 함수를 통해 분주비를 설정하고 있다. prescaler 값으로 7200 을, period 값으로 10000 을 설정해 72MHz 의 SystemCoreClock 으로부터 1Hz(1ms) 주파수의 신호를 얻을 수 있다. int prescale = (uint16_t) (SystemCoreClock / 10000); 코드와 TIM_TimeBaseStructure.TIM_Prescaler = prescale; 코드를 통해 prescaler 값을 7200 으로

설정하고, TIM_TimeBaseStructure.TIM_Period = 10000; 코드를 통해 period 값을 10000 으로 설정하고 있다.

6. changePulse()함수

```
void changePulse() {
  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 = pulse;
  TIM_OC3Init(TIM3, &TIM_OCInitStructure);
}
```

[그림 18] changePulse() 함수

[그림 18]은 1 초마다 인터럽트 함수 내부에서 서보모터의 각을 변경하기 위해서 호출할 함수이다. TIM_OCInitStructure.TIM_Pulse = pulse; 코드를 통해 PWM 신호의 전역 변수인 pulse 값으로 변경하는 코드이다.

7. TIM2_IRQHandler()함수

```
void TIM2_IRQHandler() {
  if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
    changePulse();

    if (toggle == 1) {
      if (ledCount%2 == 0)
        GPIO_SetBits(GPIOD, GPIO_Pin_2);
      else
        GPIO_ResetBits(GPIOD, GPIO_Pin_2);
      if (ledCount == 0)
        GPIO_SetBits(GPIOD, GPIO_Pin_3);
      else if (ledCount == 5)
        GPIO_ResetBits(GPIOD, GPIO_Pin_3);

      ledCount++;
      ledCount %= 10;

      pulse += 100;
      if (pulse > 2000)
        pulse = 1000;
    }

    else {
      GPIO_SetBits(GPIOD, GPIO_Pin_2);
      GPIO_SetBits(GPIOD, GPIO_Pin_3);

      pulse -= 100;
      if (pulse < 1000)
        pulse = 2000;
    }

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

[그림 19] TIM2_IRQHandler() 함수

[그림 19]은 TIM2 인터럽트 함수 코드이다. 1 초마다 인터럽트가 발생하면 changePulse() 함수를 호출하여 서보모터의 각을 변경한다. 또한, 조건문을 통해 toggle 값이 1 즉 LED On

상태인지 확인한다. toggle 값이 1 이라면, ledCount 변수 값을 확인하여, LED1 은 1 초마다, LED2 는 5 초마다 토글하도록 구현하였다. 그리고 pulse 값을 100 씩 증가시켜 서보모터 각이 정방향으로 증가하도록 구현하였다. 반대로 toggle 값이 0 이라면, LED1 과 LED2 를 OFF 상태로 만들고, pulse 값을 100 씩 감소시켜 서보모터 각이 역방향으로 증가하도록 구현하였다. Pulse 값을 1000~2000 의 값만을 가질 수 있으므로 이에 대한 조건문도 추가하였다. TIM_ClearITPendingBit(TIM2, TIM_IT_Update); 코드는 다시 Interrupt 를 발생시킬 수 있도록 초기화 해주는 역할을 한다.

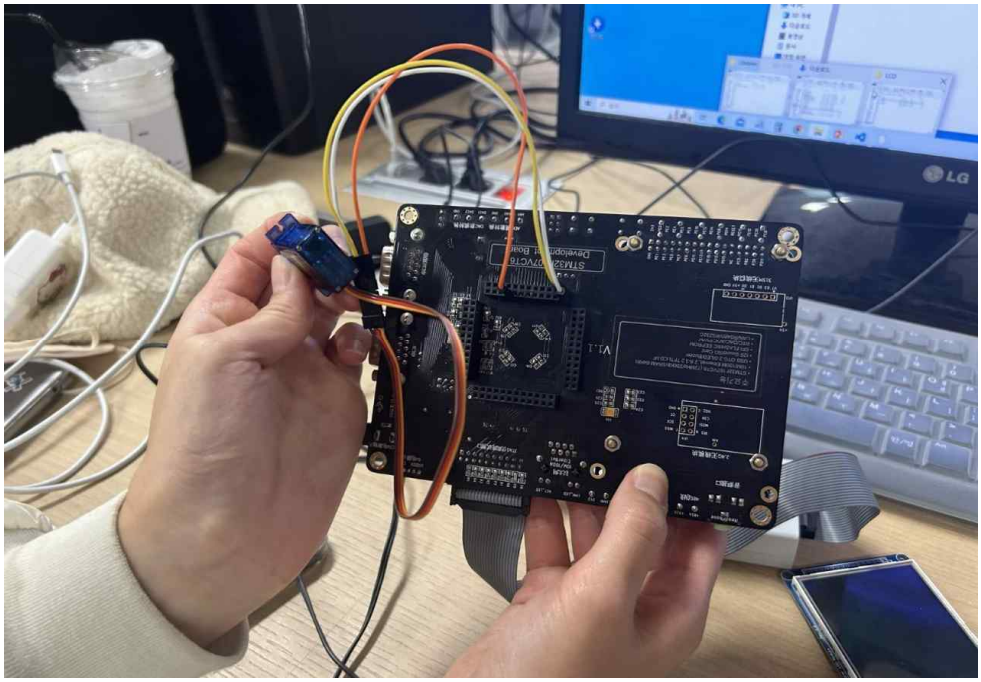
8. main() 함수

```
int main() {
  SystemInit();
  RCC_Configure();
  GPIO_Configure();
  NVIC_Configure();
  TIM_Configure();

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

  LCD_ShowString(20, 50, "WED_TEAM08", BLUE, WHITE);
  LCD_ShowString(20, 70, "OFF", RED, WHITE);
  LCD_DrawRectangle(40, 100, 90, 150);
  LCD_ShowString(40, 115, "BUTTON", RED, WHITE);

  while(1) {
    Touch_GetXY(&x, &y, 1);
    Convert_Pos(x, y, &x, &y);

    if ((x>40)&&(x<90)&&(y>100)&&(y<150)) {
      toggle = 1 - toggle;
      if (toggle == 1) {
        LCD_ShowString(20, 70, "OFF", WHITE, WHITE);
        LCD_ShowString(20, 70, "ON", RED, WHITE);
      }
      else {
        LCD_ShowString(20, 70, "ON", WHITE, WHITE);
        LCD_ShowString(20, 70, "OFF", RED, WHITE);
      }
    }
  }
  return 0;
}
```

[그림 20] main() 함수

[그림 20]은 main() 함수 코드이다. 우선, 앞서 정의했던 설정 함수들을 모두 호출한다. 이전 주차와 마찬가지로 LCD 사용을 위한 시작 함수들도 호출한다. LCD 에 team 이름과 LED 토글 ON/OFF 상태를 표시하고, LED ON/OFF 버튼을 생성하기 위해 LCD_ShowString 함수를 이용하고 있다. Touch_GetXY 함수와 Convert_Pos 함수를 이용하여 전역 변수 x 와 y 에 LCD 화면 터치 좌표를 얻어온 후, 조건문을 통해 좌표가 LED ON/OFF 버튼에 해당하는지 판단한다. 만약 버튼을 터치했다고 판단된다면, toggle 값을 변경하고 LCD 에 출력되는 LED 상태를 변경하도록 구현하였다.

3. 회로 구성 및 동작 확인



[그림 21] 서보 모터 연결 모습

[그림 21]은 회로에 서보모터를 연결한 모습이다. **[그림 16]**처럼 중간 빨간색 선을 VCC 에, 갈색 선을 GND 에, 주황색 선을 PWM 신호가 출력되는 B0 에 연결하였다.

[그림 22] LED ON 상태

[그림 23] LED OFF 상태

5. 결론 및 느낀점

이번 실험에서는 10 주차 실험과 마찬가지로 TFT LCD 를 이용하여 원하는 위치에 문자열을 출력하고, ON/OFF 버튼을 생성하여 LED 를 제어해보았다. 또한, 타이머를 이용하여 1 초마다 LED 를 토글하고, PWM 신호 출력을 통해 서보모터도 제어하였다. 문자열을 출력하거나, 버튼을 생성하는 LCD 관련된 부분은 10 주차 실험과 거의 유사하여 어려움 없이 해결할 수 있었다. 다만, 타이머의 분주 계산에서 약간의 어려움을 겪었다. 1 초 간격으로 인터럽트를 발생시키기 위한 TIM2 분주 계산은 수업 자료를 보며 쉽게 prescaler 값과 period 값을 구할 수 있었다. 하지만, TIM3 의 경우 50Hz 의 신호를 만들기 위한 prescler 값과 period 값을 계산하는 부분과 적당한 pulse 값을 설정하는 부분에서 일차적으로 어려움을 겪었다. 적절한 prescaler 값과 period 값을 계산하여 대입하였음에도 불구하고 서보모터가 작동하지 않아 이를 해결하는 부분에서 이차적으로 어려움을 겪었다. 다행히 조교님께 도움을 받고 조원들이 함께 고민하여 다양한 값을 시도한 후에 이를 해결할 수 있었다.