11주차 결과보고서

10조 (001분반)
과목명:임베디드시스템설계및실험
담당 교수 : 정상화교수님
담당 조교 : 최호진 조교님
조원 : 201924603 하규승(조장)
201727102 강준혁
201924525 이광훈
202023139 박지원
제출 날짜 : 2023.11.19

목차

1. 실험목표

Timer, PWM 이해 및 실습

1.1. Timer 종류

● 범용(General-purpose)타이머

- 출력비교, 원펄스 입력캡쳐, 센서 인터페이스(엔코더, 홀 센서 등) 등의 용도로 사용할 수 있는 범용 기능을 가지는 타이머

● 고급(Advanced)타이머

- 범용 타이머보다 더 많은 기능을 가지는 타이머, 주로 모터 제어와 디지털 파워 변환(Power Conversion)

● 기본(Basic)타이머

- 입출력 기능은 없고 시간기반 타이머나 DAC의 트리거 용도로 만 사용되는 타이머

● 채널(Chanel)타이머

- 범용 타이머와 기능은 동일하나 채널을 1~2개만 가지는 타이머

1.2. 분주 계산(General-purpose timers)

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

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

그림 1 분주 계산법1

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

그림 2 분주 계산법 2

분주란 MCU에서 제공하는 Frequency를 우리가 사용하기 쉬운 값으로 바꾸

어 주는 것을 말합니다. Counter clock frequency를 1~65536의 값으로 나누기 위해 16-bit programmable prescaler을 사용합니다. 그리고 period로 몇 번 count하는지 설정합니다.

1.3. PWM(Pulse Width Modulation)

일정한 주기 내에서 Duty ratio를 변화 시켜서 평균 전압을 제어하는방법. ex) 0~5V 전력 범위에서 2.5V 전압을 가하고 싶다면, 50% 듀티 사이클 적용.

대부분의 서보모터는 50Hz ~ 1000Hz 의 주파수를 요구하고 데이터 시트를 반드시 확인해서 사용해야함.

그림 3 서보모터

그림 4 펄스값에 따른 서보모터 각도

2. 세부 목표

2.1. TFT LCD에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼 생성

2.2. LED ON 버튼 터치 시 TIM2 interrupt, TIM3 PWM을 활용하여 LED 2개와 서보모터 제어

a. LED : 1초 마다 LED1 TOGGLE, 5초마다 LED2 TOGGLE

b. 서보모터 : 1초 마다 한쪽 방향으로 조금씩(100) 이동

2.3. LED OFF 버튼 터치 시 LED Toggle 동작 해제 및 서보모터 동작 반전

a. 서보모터: 1초 마다 반대쪽 방향으로 조금씩(100) 이동

3. 배경 지식

- STM32F107 보드 사용방법
- TFT 사용방법
- Interrupt 사용방법
- IAR 프로그램 사용방법

4. 실험 방법

먼저 프로그램을 작동시킨 후 필요한 파일을 업로드하고 초기설정을 진행하줍니다. 그리고 이번 실습을 진행하기 위해 지난주에 사용하였던 헤더파일인 lcd.h와 touch.h을 사용하기 위해 파일에 마찬가지로 업로드해줍니다. 그 후 main.c 코드를 작성해보았습니다.

```
#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"
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;
```

그림 6 헤더파일 코드

먼저 헤더파일을 선언해주어 필요한 함수들을 사용할 수 있게 해줍니다.

```
uint16_t prescale;
void TIM2_Configure(void){
   prescale = (uint16_t) (SystemCoreClock / 10000);

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

그림 7 TIM2_Configure 코드

그리고 LED on/off를 위한 타이머를 설정해주기 위해 TIM2_Configure을 선

언하여 TIM2 interrup를 설정해줍니다. 이때 이전 주차의 실험과 동일하게 구조체 선언을 통해 값을 설정해줍니다.
1초로 설정해주기 위해 1/fclk * prescaler * period = 1[s]을 만들어주어야 합니다. SystemCoreClock(fclk)이 72Mhz이므로, 이를 10000로 나누어준 값인 7200로 prescaler를 설정해주고 period를 10000로 설정해주면 1s를 설정해줄 수 있습니다. ClockDivision은 기본값인 TIM_CKD_DIV1로 설정해주고, CounterMode는 TIM_CounterMode_Up로 설정해주었습니다.
이렇게 값을 설정해주고 TIM_Cmd, TIM_ITConfig를 통해 Enable을 시켜주어 사용가능하게 만들어줍니다.

```
void TIM3_Configure(void) {
   prescale = (uint16_t) (SystemCoreClock / 1000000);

   TIM_TimeBaseStructure.TIM_Period = 20000-1;
   TIM_TimeBaseStructure.TIM_Prescaler = prescale-1;
   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 = 1500; // 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 8 TIM3_Configure 코드

마찬가지로 서보모터를 제어하기 위한 PWM 및 티이머를 설정해주기 위해 TIM3_Configure을 선언해줍니다.
먼저 TimeBaseStructure을 이용하여 타이머에 대한 설정부터 해줍니다. PWM Period는 20ms이기 때문에 TIM_Period는 20000에서 1을 뺀값으로 설정해줍니다. 그리고 fclk가 72mHz이고 Period가 50Hz이므로 prescale을 SystemCoreClock / 1000000로 설정해줍니다. 그리고 Prescaler는 prescale-1로 설정해줍니다. ClockDivision은 0으로 설정해주고, CounterMode는 TIM_CounterMode_Down로 설정해주었습니다. 그다음으로 OCInitStructure 구조체를 통해 PWM 설정해줍니다. OCMode는 TIM_OCMode_PWM1, OCPolarity는 TIM_OCPolarity_High, OutputState는 TIM_OUTputState_Enable로 설정해줍니다. PWM는 1.5ms pulse일 때 서보모터가 middle에 위치하므로 Pulse는 1500으로 설정해줍니다.

이렇게 값을 설정해준 후 OC3PreloadConfig, ARRPreloadConfig, Cmd를 통해 사용값을 적용해주어 enable시켜줍니다.

```
void RCC_Configure(void)
{
    // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    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);

}
```

그림 9 RCC_Configure 코드

RCC_Configure을 선언해주어 LED, TFT LCD, 서보모터, 타이머 등을 사용하기 위해 관련된 각 핀들을 활성화해줍니다.

```
void GPIO_Configure(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    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);
  // TODO: Initialize the GPIO pins using the structure 'GPIO_InitTypeDef' and the function 'GPIO_Init'
}
```

그림 10 GPIO_Configure

각각의 핀에 대한 값을 설정해주기 위한 GPIO_Cofigure을 선언해줍니다. LED를 제어하기 위해 LED1, LED2와 연결된 핀인 PD2, PD3를 사용하기 위해 구조체를 이용하여 값을 설정해주는데, Speed는 GPIO_Speed_50MHz, Mode는 Mode_Out_PP로 설정해줍니다. 마찬가지로 TIM3의 channel3에 연결된 핀인 PB0을 사용하기 위해 Speed는 GPIO_Speed_50MHz, Mode는 GPIO_Mode_AF_PP로 설정해줍니다.

```c
void NVIC_Configure(void) {

    NVIC_InitTypeDef NVIC_InitStructure;

    // TODO: fill the arg you want


    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

그림 11 NVIC_Configure 코드

Interrupt를 사용하기 위해 VNIC_Configure도 선언해줍니다. LED 토글을 TIM2 interrupt를 활용하여 제어하므로 구조체를 사용하여 TIM2_IRQn를 설정하여 PreemptionPriority를 0, SubPriority를 0으로 설정하여 Cmd를 Enable시켜줍니다.

```c
void LED1_ON(void){
  GPIO_SetBits(GPIOD, GPIO_Pin_2);
}

void LED1_OFF(void){
  GPIO_ResetBits(GPIOD, GPIO_Pin_2);
}

void LED2_ON(void){
  GPIO_SetBits(GPIOD, GPIO_Pin_3);
}

void LED2_OFF(void){
  GPIO_ResetBits(GPIOD, GPIO_Pin_3);
}
```

그림 12 LED ON/OFF 코드

LED1, LED2를 On, Off하기 위한 함수도 SetBit, ResetBit를 이용하여 정의해줍니다.

```
int PWM = 1500;
void motor_left(void){
 PWM = PWM + 100;
 if(PWM > 1500)
   PWM = 0;
 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 +700 ; // us

 TIM_OC3Init(TIM3, &TIM_OCInitStructure);
}

void motor_right(void){
 PWM = PWM - 100;
 if(PWM <500)
   PWM = 1500;

 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+700 ; // us

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

그림 13 motor 제어 함수

TIM3의 Pulse값을 바꾸어주기 위해 PWM를 전역변수로 선언해주고 기본값을 middle 위치의 값인 1500으로 설정해줍니다. 마찬가지로 서보모터를 제어하기 위한 함수도 정의해줍니다. motor_left 함수를 통해 TIM3을 1초로 설정해주었기 때문에 1초마다 서보모터가 100씩 이동하도록 설정해줍니다. 마찬가지로 motor_right 함수를 통해 1초마다 서보모터가 반대방향으로 100씩 이동하도록 설정해줍니다. 이 때 Pulse값을 PWM + 700으로 설정하여 값을 보정해주었습니다.

```
int count = 0;
int led_status = -1;
void TIM2_IRQHandler(void){
  if(TIM_GetITStatus(TIM2,TIM_IT_Update)!=RESET){
    if(led_status == 1) {
      if (count%2==0){
        LED1_ON();
        motor_right();
      }
      else {
        LED1_OFF();
        motor_right();
      }
      if(count==0) LED2_ON();
      if(count==5) LED2_OFF();

      count++;
      count = count % 10;
    }
    else {
      motor_left();
    }
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
  }
}
```

그림 14 TIM2_IRQHandler

시간을 계산하기 위해 전역변수로 count를, LED 활성화 여부를 뜻하는 led_status를 선언해줍니다. 이 때 led_status가 1일 때는 LED ON 상태, -1일때는 LED OFF 상태를 뜻하도록 설정해줍니다. 그리고 LED를 제어하기 위한 TIM2_IRQHandler를 선언해주어서 앞서 1초로 설정해준 타이머인 TIM2를 TIM_IT_Update를 통해 상태를 받아오며 LED ON 상태일때는 count를 1씩 증가시켜 1초마다 LED1이 toggle될 수 있게하고, LED2가 5초마다 toggle될 수 있도록 설정해줍니다. 마찬가지로 서보모터가 1초마다 움직이는 함수 motor_right를 호출합니다. 이때 count값이 너무 커지지 않도록 count % 10 연산을 해줍니다. LED OFF 상태일 때는 서보모터가 반대방향으로 1초마다 움직이도록 motor_left를 호출합니다. 그리고 ClearITPendingBit를 설정해줍니다.

```c
int main() {
    // LCD ◆◆◆◆ ◆◆◆◆◆◆ LCD_Init◆◆ ◆◆◆◆◆Ø◆ ◆◆◆◆◆p◆ ◆◆◆⊔ ◆◆ ◆'◆ ◆◆◆◆
    SystemInit();
    RCC_Configure();
    GPIO_Configure();
    TIM2_Configure();
    TIM3_Configure();
    NVIC_Configure();
    // ---------------------------------
    LCD_Init();
    Touch_Configuration();
    LED1_ON();
    LED2_ON();
    Touch_Adjust();
    LCD_Clear(WHITE);


    vint16_t posX, posY;
    LCD_ShowString(40, 40, "TEAM10", BLACK, WHITE); //◆◆◆◆ ◆◆◆

    LCD_DrawRectangle(40,80,80,120);
    LCD_ShowString(40, 100, "  BUT", RED, WHITE); //◆◆U ◆Ø◆◆◆
    LCD_ShowString(40, 60, "OFF", RED, WHITE);
```

그림 15 main 함수 코드

main()에서 각각의 기능을 활성화하기 위해 사용할 함수들을 불러와 값을 설정해줍니다. 그리고 위치를 받아오기 위한 posX, posY값을 설정합니다. 화면에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼을 보이게 하기 위해 ShowString을 통해 명시해주며, DrawRectangle 함수를 통해 버튼을 만들어줍니다.

```c
    while(1){
        Touch_GetXY(&posX, &posY, 1); //◆◆ğ ◆◆ᵧ ◆◆◆
        Convert_Pos(posX, posY, &posX, &posY); // ◆◆ₒ

        if(led_status==1) {
          LCD_ShowString(40, 60, "OFF", RED, WHITE);
          LCD_ShowNum(40,180,count,2,RED, WHITE);
          LED1_ON();
          LED2_ON();
        }
        else {
          LCD_ShowString(40, 60, "ON ", RED, WHITE);
          LCD_ShowNum(40,180,count,2,RED, WHITE);
        }

        if(40<posX&&80>posX&&80<posY&&120>posY){
          led_status = -1* led_status;
        }
```

그림 16 while 문 코드

while 함수에서 Touch_GetXY를 통해 touch가 입력된 위치를 받아옵니다. 이 위치가 버튼 위치안에 위치할 때 led_status가 바뀌게 하여 LED ON/OFF 상태를 변화시킵니다. led_status의 상태에 따라 화면의 글씨를 OFF로 바꾸고, -1일때는 ON으로 바뀌게 설정해줍니다.

5. 실험 결과

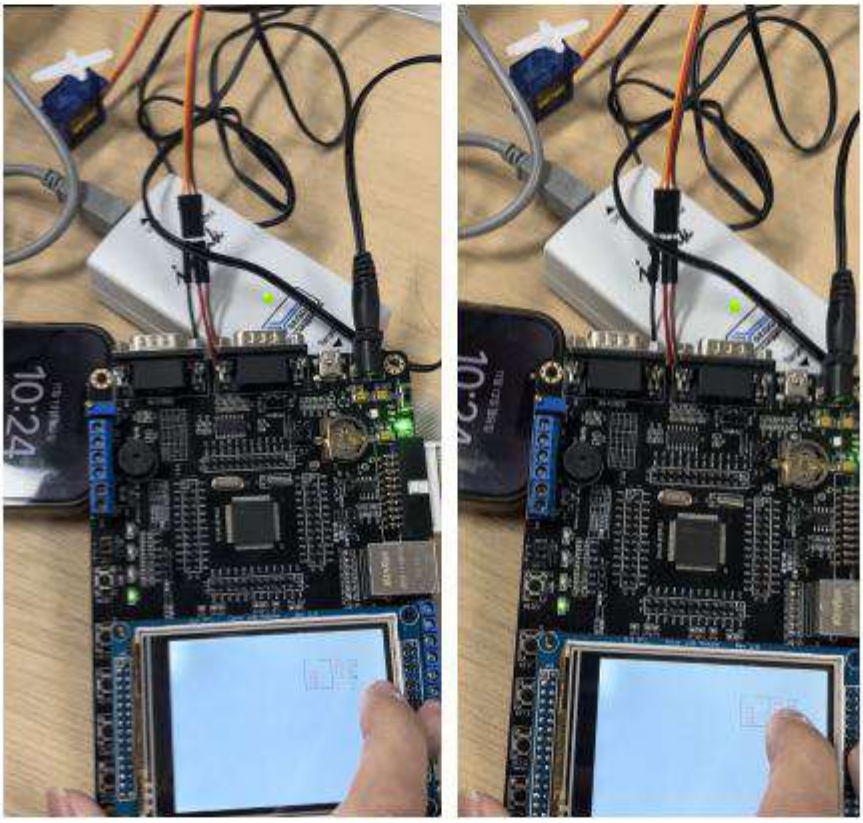

그림 17 LED OFF 인 경우

화면을 켰을 경우 TEAM 10, LED OFF, 버튼을 확인할 수 있었습니다. 또한 LED1, LED2가 꺼져있으며, 서보모터가 1초마다 왼쪽으로 조금씩 회전하는 것을 확인할 수 있었습니다. 버튼 바깥의 화면을 터치하였을 겨우 아무일도 발생하지 않는 것을 알 수 있었습니다.

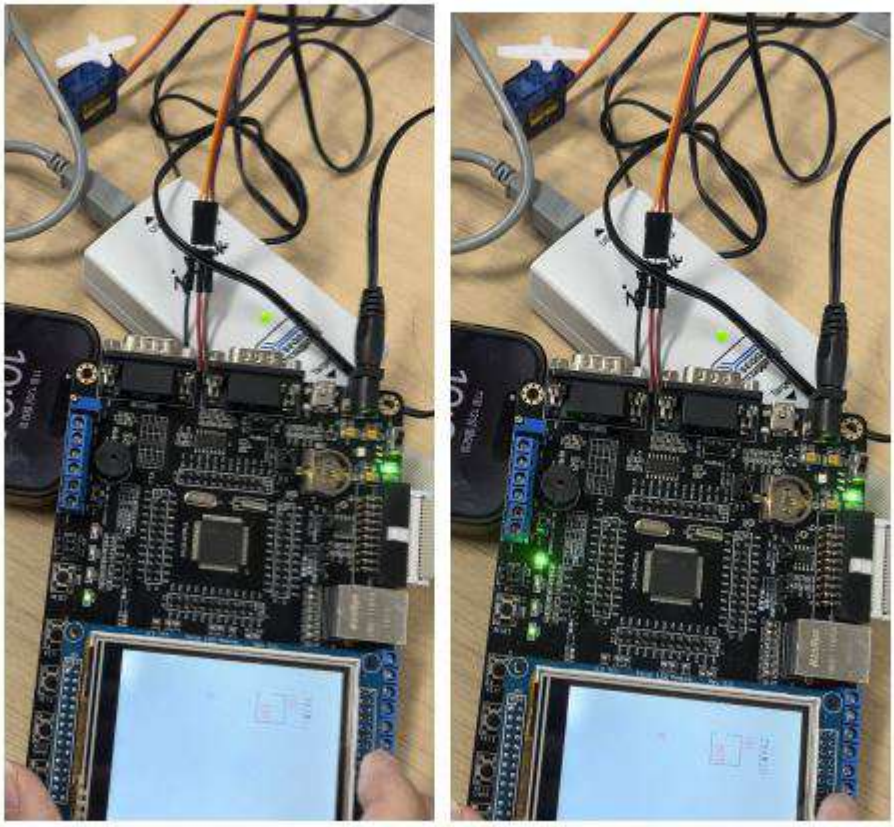

그림 18 LED ON 인 경우

버튼을 눌렀을 경우 OFF라는 글자가 ON으로 바뀌며 LED1, LED2에 불이 들어오는 것을 확인할 수 있었습니다. LED1은 1초마다 toggle되며, LED2는 5초마다 toggle되고 서보모터가 OFF일때와는 반대방향으로 1초마다 조금씩 회전하는 것 또한 확인할 수 있었습니다.

6. 결과에 대한 논의

먼저 전원을 킨 후 TFT LCD에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼이 생성되어 화면에 표시되는 것을 확인하여 초기값이 잘 설정된 것을 확인할 수 있었습니다. 그리고 LED OFF일 때는 서보모터가 1초마다 왼쪽으로 조금씩 이동하며 LED1, LED2에 둘다 불이 들어오지 않는 것으며 버튼을 터치하여 LED ON으로 상태가 변경되었을 경우 LED1는 1초마다 toggle, LED2는 5초마다 toggle, 서보모터는 1초마다 이전과 반대방향인 오른쪽으로 조금씩 이동하는 것을 보아 프로그램이 정상작동하며 원래 실험 세부 목표를 잘 만족한 것을 알 수 있었습니다.

7. 결론

이번 실험에서 TFT LCD를 이용하여 화면에 버튼 및 글씨를 출력하며 타이머와 관련된 TIM2 interrupt, TIM3 PWM를 위해 period 및 prescale을 직접 계산하여 값을 집어넣어 1초마다 count되게 하였으며 이를 통해 LED 2개와 서보모터를 제어하는 코드를 직접 작성해보고 컴파일을 해보았습니다. 프로그램이 STM32 보드에서 정상작동함을 확인해보면서 원래 목표였던 임베디드 시스템의 기본 원리를 습득할 수 있었고 Timer, PWM 이해 및 실습을 진행해볼 수 있었습니다.