임베디드 시스템 설계 및 실험 보고서
8 주차 실험_TFT LCD 와 ADC(조도센서)

분반 : 001 분반

교수님 : 정 상화 교수님

조교님 : 유 동화 조교님

실험일 : 2019-10-21

제출일 : 2019-11-04

00. 목차

7 조

장 수현

박 창조

임 다영

이 힘찬

01. 실험 목적

- TFT LCD 의 이해 및 제어
- ADC(조도센서)의 이해 및 제어
- 오실로스코프를 사용하여 조도센서의 값 변화 출력

02. 실험 과제

2.1 주 과제

TFT LCD 에 ADC(조도 센서)의 변화 값 출력

- LCD 를 터치할 때마다 좌표 값 전달
- ADC(조도 센서)의 값을 받아서 LCD 를 터치할 때마다 출력
- 구현되어 있는 함수를 사용하여 터치할 때마다 동그라미 생성

2.2 세부 과제

- 오실로스코프를 이용하여 조도 센서의 값이 바뀌는 것을 확인
- LCD 상에 LED 버튼을 만들고 해당 버튼 클릭 시 연결된 LED on
- DB 파일, 라이브러리, scatter 파일, flashclear 파일을 프로젝트 폴더 안으로 복사

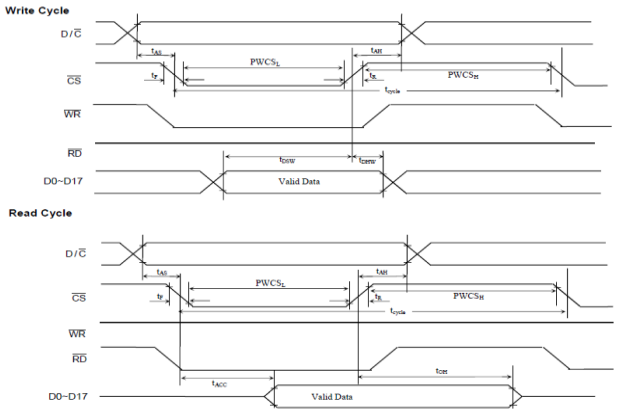
03. 실험 준비

3.1 실험에 필요한 기초지식

3.1.1 TFT-LCD

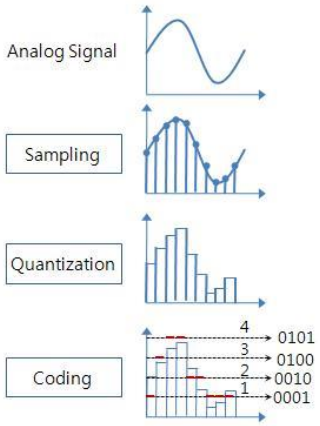
- 초박막 액정표시장치(Thin Film Transistor Liquid Crystal Display)의 약자
- 액체와 고체의 중간 특성을 가진 액정의 상태 변화와 편광판의 편광 성질 이용
- 통과하는 빛의 양을 조절함으로써 정보를 표시하는 첨단 디지털 디스플레이
- TFT-LCD 는 Color Filter 와 TFT 가 형성된 두 장의 유리기판과 그 사이에 주입된 액정(Liquid Crystal), 광원(Back Light Unit)으로 구성됨.

3.1.2 Timing Diagram

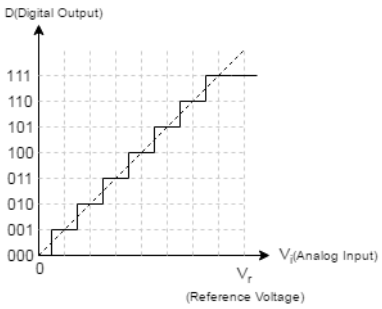


- CS(Chip Select) : low 일 때 chip 이 동작
- RS = D/C(Data/Command) : low 일 때 command 전송/ high 일 때 data 전송
- WR(Write) : low->high , data 를 RAM 에 쓴다.
- RD(Read) : low->high, data 를 읽는다.

3.1.3 ADC



- 아날로그 신호를 디지털 신호로 변환한다.
- 아날로그 신호가 입력되어 들어오면 이를 샘플링하여 양자화를 거쳐 부호화 한다.
- 실험에서는 ADC 의 한 종류인 조도센서를 사용한다.



- 부호화(coding) : 신호처리가 용이한 디지털 코드 형태로 변환
- 샘플링(sampling) : 시간 축 방향으로 연속된 아날로그 신호를 특정 시간 간격으로 나누고, 나눈 시간 간격의 값을 추출해서 추출한 표본으로 샘플 생성
- 양자화(Quantization) : 일반적으로 ADC 의 아날로그 입력(Vi)과 디지털 출력(D)의 상관관계는
  D = Vi * Vr * 2n 으로 표현할 수 있다.(Vr 은 레퍼런스 전압, n 은 디지털 출력 비트 수)

3.1.4 DAC

- 디지털 신호를 아날로그 신호로 변환한다.

3.1.5 조도 센서(Photo Resistor)

- 주변의 밝기를 측정하는 센서
- 빛의 양이 많아질수록 전도율이 높아져 저항이 낮아진다.(반비례)
- 조도 센서를 가리면 저항이 높아져 TFT LCD 에 출력되는 숫자가 높아진다.

3.2 실험 진행 시 주의사항

- 4 주차 스캐터 파일을 수정해서 메모리 범위를 늘린다.
- 조도 센서의 +는 5V 에, -는 GND 에, 저항이랑 연결된 곳은 디지털 핀에 연결한다.
- 최적화 level 을 high 로 설정한다.
- LCD 핀 개수가 똑같지 않기 때문에 주의한다.
- LCD 터치 영역을 정확하게 잡아서 글자가 깨지지 않도록 한다.
- cycle 을 볼 때는 high 는 1, low 면 0 으로 생각하고 본다.
- ADC 선은 저항 가운데에 꽂아야 한다.
- Touch_GetXT()함수의 마지막 인자 값으로 1 을 넣어주어야 한다(터치 시 한번만 동그라미 출력).

3.3 환경 설정

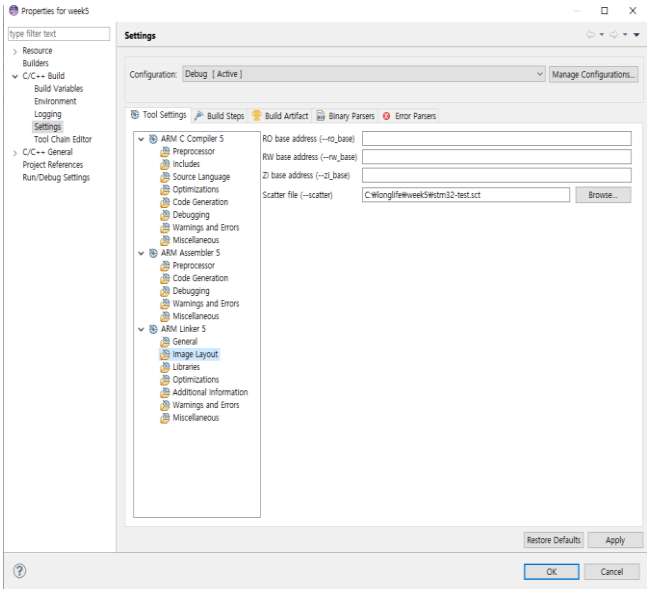
실험을 진행하기 전 아래의 프로젝트 생성 및 기본 환경 설정을 해주어야 한다.

3.3.1 프로젝트 생성

C 언어로 소스코드를 작성하기 때문에 C++프로젝트가 아닌 C 프로젝트를 생성해준다. Project type 은 Bare-metal Executable -〉 Empty Project 로, Toolchains 은 Arm Compiler 5 로 설정한다.

3.3.2 프로젝트 Properties-Settings

4 주차 실험과 동일하게 스캐터 파일을 이용한다.



3.3.3 보드 연결

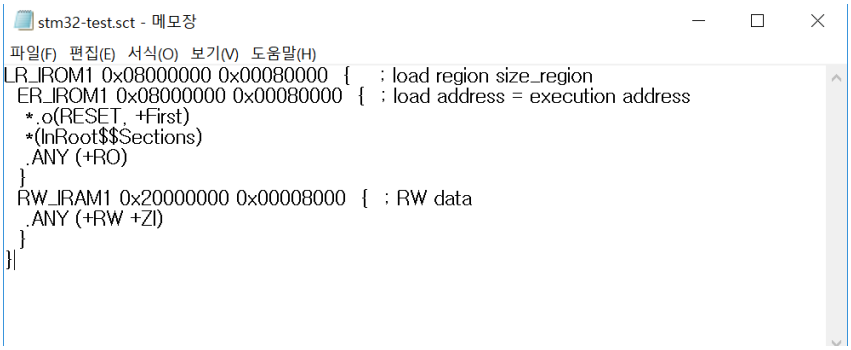
보드 연결 시에는 반드시 연결 순서를 지키고, 규격에 맞는 전원선을 사용해야한다.

연결 순서 : 보드와 Dstream JTAG 연결 -〉 보드전원선 연결(보드 전원은 OFF) -〉 Dstream 전원 연결 및 ON -〉 Dstream Status LED 점등 확인 후 보드 전원 ON -〉 Dstream Target LED 점등 확인 후 DS-5 에서 'connect target'

3.3.4 데이터 베이스 설정

Dstream 를 USB 로 컴퓨터에 연결하고 Debug Hardware config 에서 보드를 connect 한다. 보드 연결 후에는 Auto Configure 를 클릭하여 설정을 진행하고, rvc 파일로 저장한다. rvc 파일 저장 경로에 한글이 들어가지 않도록 주의한다.

3.3.5 Scatter file 수정



스캐터 파일의 메모리 범위를 위와 같이 수정한다.

3.3.6 Optimization level 설정

실험에서 사용하는 라이브러리의 용량이 크기 때문에 최적화 레벨을 high 로 설정한다. (O2 로 설정하되 필요시 O3 설정 가능)

Project -〉 Option -〉 Settings -〉 Optimization level high

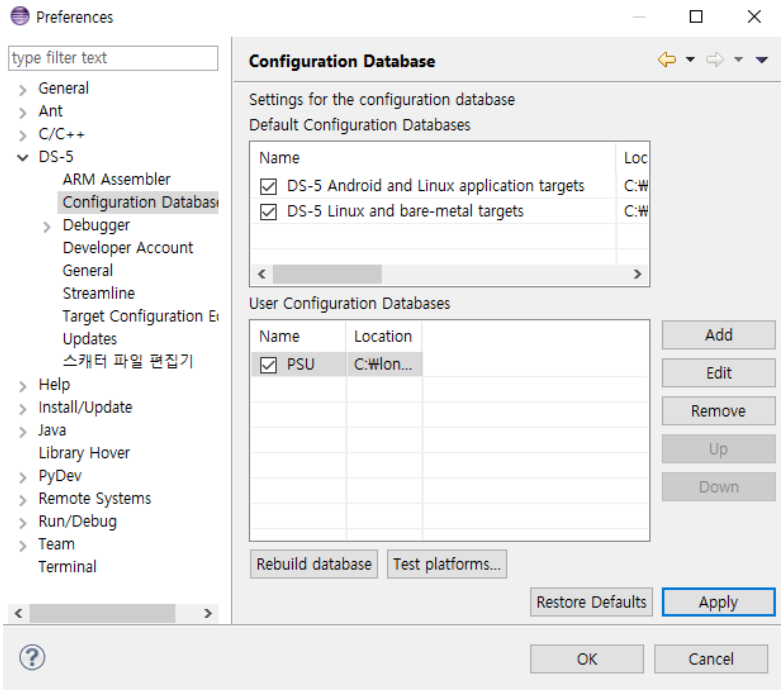
3.3.6 cdbimporter

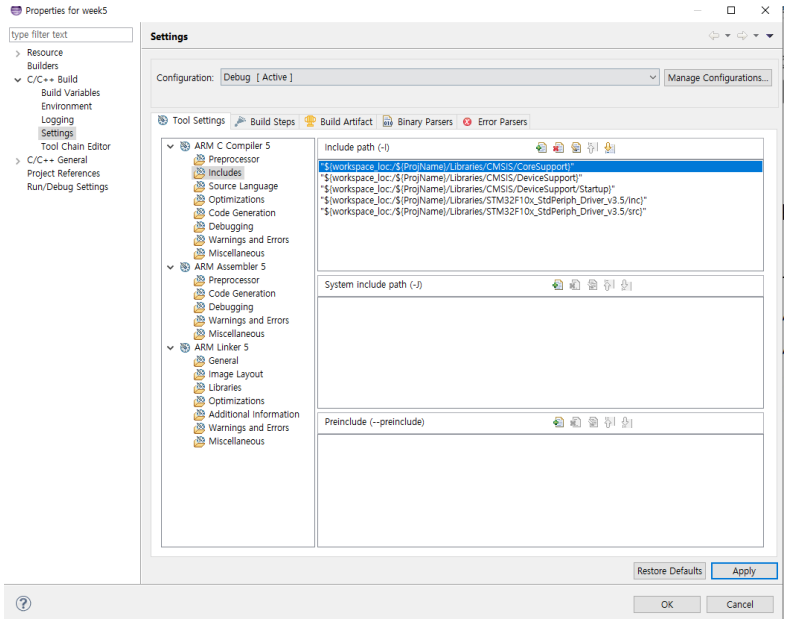
DS-5 Command Prompt 에서 RVC 파일이 있는 폴더로 이동 후 아래와 같이 명령문을 작성한다.

3.3.7 데이터 베이스 등록 및 라이브러리 추가

아래와 같이 데이터 베이스 등록 후 프로젝트 폴더에 라이브러리를 추가한다.

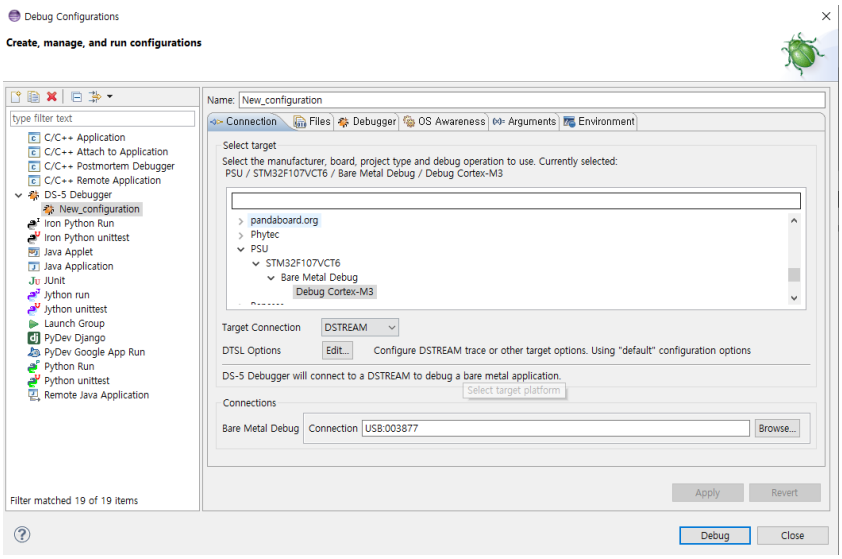
프로젝트 파일이 있는 경로에 font.h, lcd.c, lcd.h, touch.c, touch.h 를 추가한다.

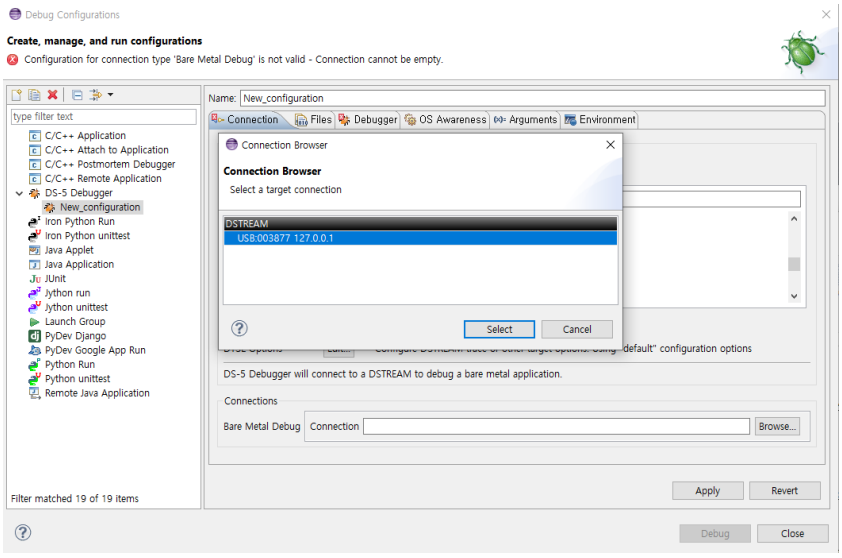


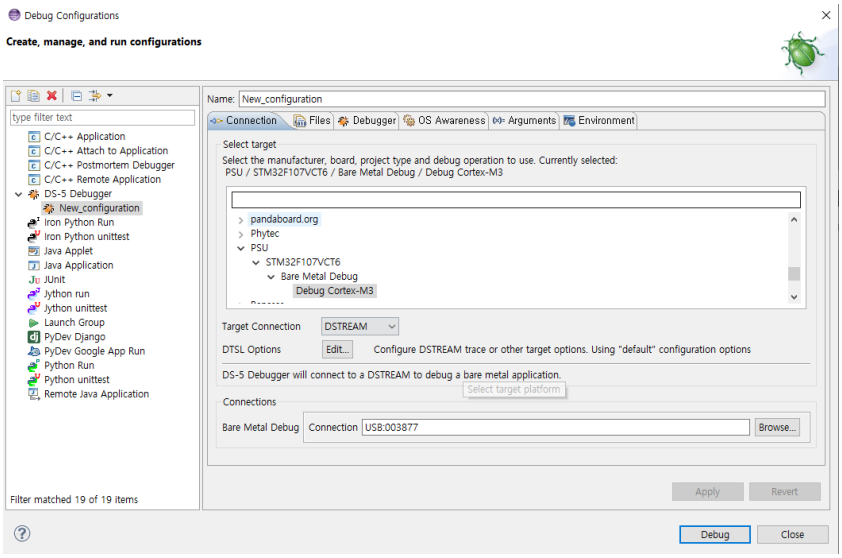


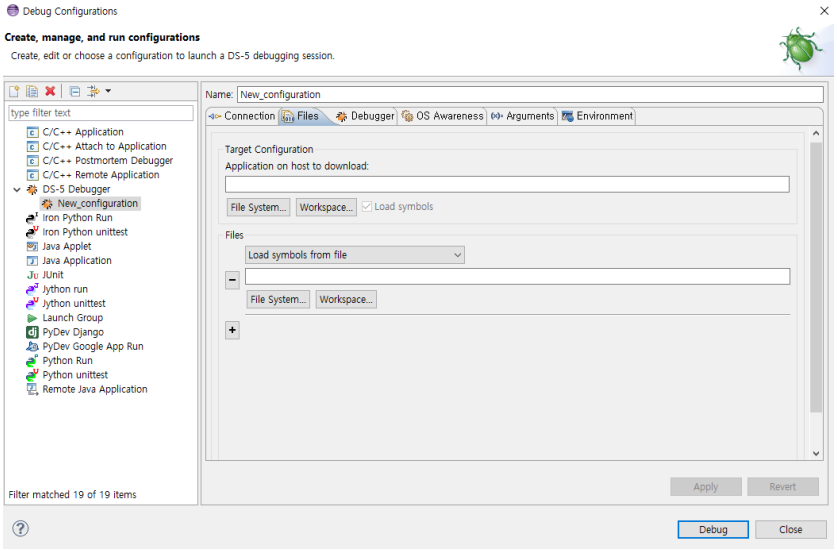
3.3.8 디버그 설정

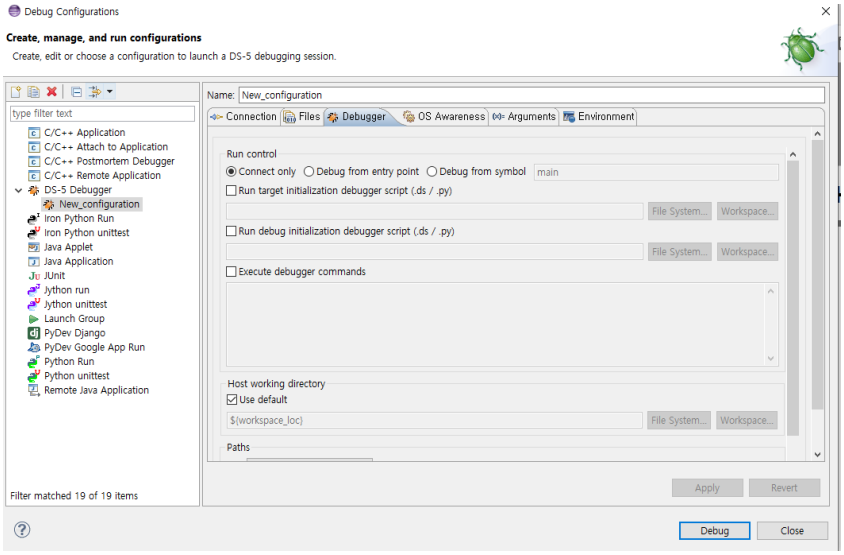
C 언어 소스파일을 만들어 빌드하고 디버그 설정을 한다. 4 주차와 동일한 환경으로 진행한다.





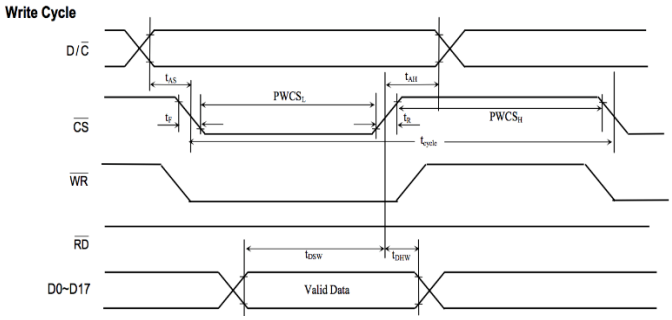






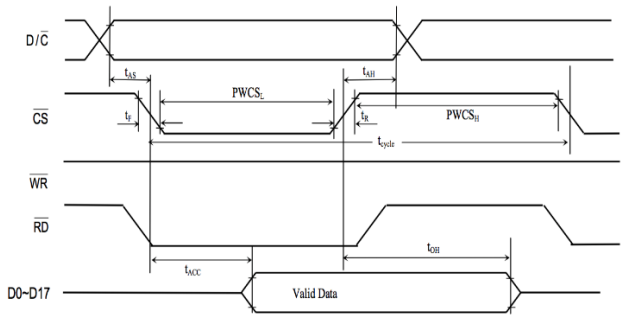
04. 실험 및 과제 해결

4.1 LCD Library 수정



1) CS 를 low 로 하여 chip 동작시킨다.
2) RS(=D/C)가 low 라면 command, high 라면 display data 가 register 에 write 된다.
3) Write cycle 이므로 RD 는 항상 high 이다.
4) WR 이 low->high 일 때 display RAM 에 data/ register 에 command 를 write 한다.

Read Cycle

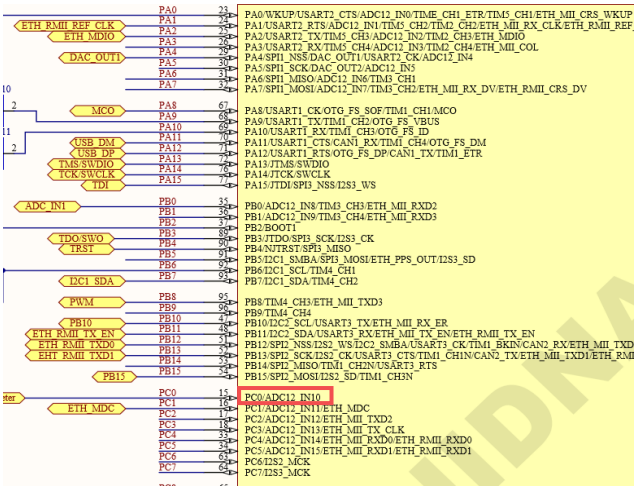


1) CS 를 low 로 하여 chip 동작시킨다.
2) RS 를 high 로 하여 display RAM 에 저장된 display data 를 읽어오도록 한다.
3) Read cycle 이므로 WR 는 high 를 유지한다.
4) RD 가 low-〉high 일 때 display data 를 읽는다.

위의 Timing diagram 을 참고하여 lcd.c 파일을 아래와 같이 수정한다.

```
static void LCD_WR_REG(uint16_t LCD_Reg)
{
    LCD_CS(0);
    LCD_RS(0);
    LCD_WR(0);
    GPIO_Write(GPIOE, LCD_Reg);
    LCD_WR(1);
    LCD_CS(1);
}
static void LCD_WR_DATA(uint16_t LCD_Data)
{
    LCD_CS(0);
    LCD_RS(1);
    LCD_WR(0);
    GPIO_Write(GPIOE, LCD_Data);
    LCD_WR(1);
    LCD_CS(1);
}
static uint16_t LCD_ReadReg(uint16_t LCD_Reg)
{
      uint16_t temp;
      GPIO_InitTypeDef GPIO_InitStructure;
      LCD_WR_REG(LCD_Reg);
      // To Read from Data Line
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
      GPIO_Init(GPIOE, &GPIO_InitStructure);
      LCD_CS(0);
      LCD_RS(1);
      LCD_RD(0);
      temp = GPIO_ReadInputData(GPIOE);
      LCD_RD(1);
      LCD_CS(1);
      // Read Done, Reset
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
      GPIO_Init(GPIOE, &GPIO_InitStructure);
      return temp;
}
```

4.2 RCC & GPIO Configure.




```
void RCC_configure() {
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOC |
RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE | RCC_APB2Periph_GPIOB | RCC_APB2Periph_ADC1,
ENABLE);
}

void GPIO_configure() {
	GPIO_InitTypeDef GPIO_InitStructure;

	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_7);
	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_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

아래의 reference 에서 알 수 있듯이 ADC 포트로는 ADC 가 표기된 모든 포트를 사용 가능하다. 하나의 포트에 / 를 기준으로 나열되어 있는 기능들은 해당 포트에서 사용할 수 있는 모든 기능들을 의미한다. 예를 들어 PB0 의 경우 ADC 포트로도 사용할 수 있고, TIMER 로도 사용할 수 있다. 실험에서는 ADC 포트 외에 다른 기능을 사용하지 않아 포트를 임의로 선택할 수 있었지만 기능의 중복이 없는 PC0 포트를 ADC 포트로 설정하였다. 실험에서는 ADC, LED(2,3,4,7), TFT LCD 를 사용했고 필요한 포트에 클럭을 인가하고 아래와 같이 GPIO 를 설정하였다.

4.3 ADC configure

```
typedef struct
{
  uint32_t ADC_Mode;                      /*!< Configures the ADC to operate in independent or
                                               dual mode.
                                               This parameter can be a value of @ref ADC_mode */

  FunctionalState ADC_ScanConvMode;       /*!< Specifies whether the conversion is performed in
                                               Scan (multichannels) or Single (one channel) mode.
                                               This parameter can be set to ENABLE or DISABLE */

  FunctionalState ADC_ContinuousConvMode; /*!< Specifies whether the conversion is performed in
                                               Continuous or Single mode.
                                               This parameter can be set to ENABLE or DISABLE. */

  uint32_t ADC_ExternalTrigConv;          /*!< Defines the external trigger used to start the analog
                                               to digital conversion of regular channels. This parameter
                                               can be a value of @ref ADC_external_trigger_sources_for_regular_channels_conversion */

  uint32_t ADC_DataAlign;                 /*!< Specifies whether the ADC data alignment is left or right.
                                               This parameter can be a value of @ref ADC_data_align */

  uint8_t ADC_NbrOfChannel;               /*!< Specifies the number of ADC channels that will be converted
                                               using the sequencer for regular channel group.
                                               This parameter must range from 1 to 16. */
}ADC_InitTypeDef;
```

```
void ADC_configure() {
        ADC_InitTypeDef ADC_InitStructure;

        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        ADC_InitStructure.ADC_ScanConvMode = ENABLE;
        ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
        ADC_InitStructure.ADC_NbrOfChannel = 1;
        ADC_Init(ADC1, &ADC_InitStructure);

        ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
        ADC_Cmd(ADC1, ENABLE);
        ADC_ResetCalibration(ADC1);
        while(ADC_GetResetCalibrationStatus(ADC1));
        ADC_StartCalibration(ADC1);
        while(ADC_GetCalibrationStatus(ADC1));
}
```

ADC channel 을 사용하여 센서 값을 LCD 에 출력하기 위한 ADC configure 은 아래와 같다. 실험에서는 ADC1 을 사용하므로 Cmd, ChannelConfig, Calibration(Reset/Start)에 ADC1 을 사용한다. 각각 ResetCalibration, StartCalibration 이후에 while 문을 사용해 상태를 확인해서 제대로 세팅 되었는지 확인한다.

4.4 main

```
/*touch.c Touch_GetXY 함수의 원형*/
void Touch_GetXY(uint16_t *x, uint16_t *y, uint8_t ext)
{
        if (ext)
        {
                while (T_INT)
                        ;
        }
        Touch_GexX(x, 0);
        Touch_GexY(y, 0);
        if (ext)
        {
                while (!(T_INT))
                        ;
        }
}
```

```c
int main(void) {
	char str[10];
	uint16_t temp;
	SystemInit();
	RCC_configure();
	GPIO_configure();
	ADC_configure();
	LCD_Init();
	Touch_Configuration();
	Touch_Adjust();
	LCD_Clear(WHITE);
	LCD_ShowString(1,1,"Mon_Team07", BLACK, WHITE);
	LCD_ShowString(20,210,"LED1", BLACK, WHITE);
	LCD_ShowString(130,210,"LED2", BLACK, WHITE);
	LCD_ShowString(20,260,"LED3", BLACK, WHITE);
	LCD_ShowString(130,260,"LED4", BLACK, WHITE);
	LCD_DrawRectangle(10, 200, 100, 230);
	LCD_DrawRectangle(120, 200, 210, 230);
	LCD_DrawRectangle(10, 250, 100, 280);
	LCD_DrawRectangle(120, 250, 210, 280);
	while(1) {
		Touch_GetXY(&pos_x, &pos_y, 1);
		Convert_Pos(pos_x, pos_y, &con_x, &con_y);
		ADC_SoftwareStartConvCmd(ADC1, ENABLE);
		ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
		temp = ADC_GetConversionValue(ADC1);
		if( con_x > 10 && con_x < 100 && con_y > 200 && con_y < 230 ) {
			GPIOD->BSRR = 0x04;
		}
		else if( con_x > 120 && con_x < 210 && con_y > 200 && con_y < 230 ) {
			GPIOD->BSRR = 0x08;
		}
		else if( con_x > 10 && con_x < 100 && con_y > 250 && con_y < 280 ) {
			GPIOD->BSRR = 0x10;
		}
		else if( con_x > 120 && con_x < 210 && con_y > 250 && con_y < 280 ) {
			GPIOD->BSRR = 0x80;
		}
		if(T_INT == 0) {
			LCD_DrawCircle(pos_x, pos_y, 3);
			sprintf(str, "(%d, %d)", con_x, con_y);
			LCD_ShowString(100, 80, str, BLACK, WHITE);
			LCD_ShowNum(10, 40, temp, 10, BLACK, WHITE);
			delay();
		}
	}
	return 0;
}
```

- ADC(조도센서)의 값을 받아서 TFT LCD 터치 시마다 화면에 출력한다. 조도센서 값이 변화되었다면 갱신된 값이 다음 터치 때 출력되도록 한다.

- 터치 시 동그라미가 출력되도록 한다. 동그라미는 DrawCircle 함수를 사용하여 생성한다.

- DrawRectangle 함수를 사용하여 LED 1,2,3,4 버튼을 생성한다. ShowString 함수를 사용하여 팀 명 출력 및 LED 버튼 제목(LED1,2,3,4)을 출력한다.

- Touch_GetXY 함수를 사용하여 좌표를 받아올 수 있다. 이 때, Touch_GetXY 함수의 마지막 인자로는 1 이 들어가야 한다. 마지막 인자 값으로 1 이 들어가야 인터럽트가 발생(T_INT)해서 터치 시 동그라미가 두 번 출력되지 않는다. 마지막 인자 값으로 0 을 넣을 경우에 터치하고 뗄 때 한 번 더 값이 들어가 동그라미가 두 번 출력된다.

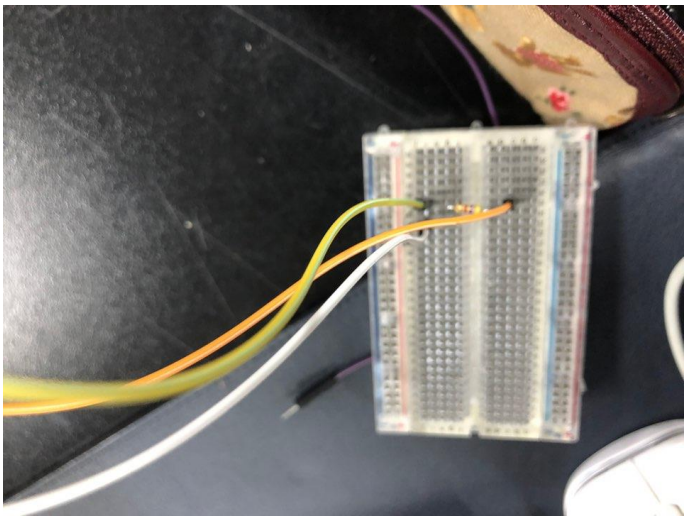
- LED 버튼으로 설정한 영역 안에 터치(입력)가 발생하면 해당 영역에 매핑된 LED 가 점등된다.

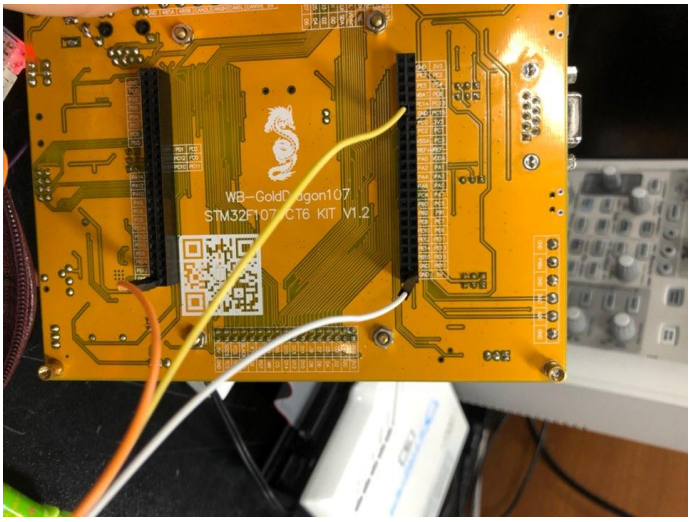
(LED1 – PD2 / LED2 – PD3 / LED3 – PD4 / LED4 – PD7)

4.5 조도 센서와 오실로스코프 setting

4.5.1 조도 센서

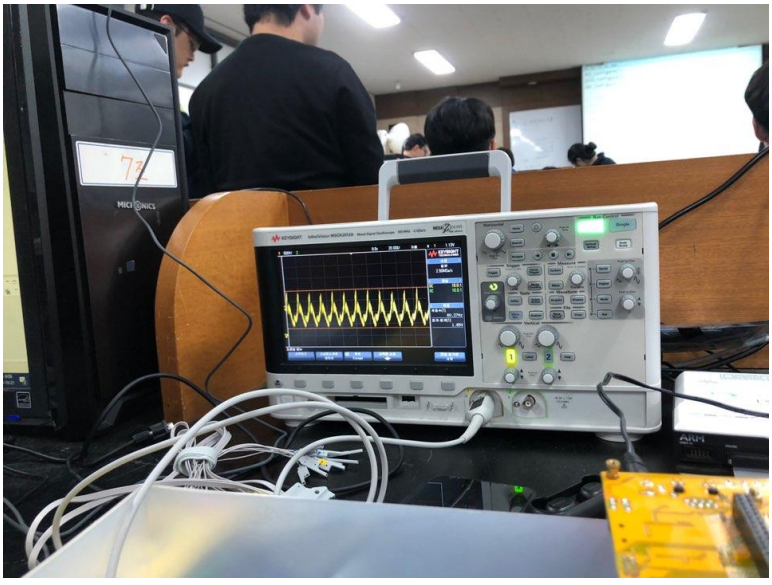
조도 센서를 아래와 같이 연결한다.





4.5.2 오실로스코프

오실로스코프를 사용하여 확인한 조도센서 값의 변화는 아래와 같다.



조도센서와 같은 아날로그 센서들은 저항을 이용해서 값을 변경하는 원리로 작동한다. 조도센서의 경우 빛의 양이 많아질수록 전도율이 높아져 저항이 낮아진다.(반비례 관계) 오실로스코프의 파형에서 밝기에 따른 조도센서 값의 변화를 확인할 수 있었다.

05. 실험 결과

5.1 소스코드

위의 내용을 바탕으로 작성한 전체 소스코드는 아래와 같다.

```
#include <misc.h>
#include <stm32f10x.h>
#include <stm32f10x_exti.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_usart.h>
#include <stm32f10x_adc.h>
#include "lcd.h"
#include "touch.h"

uint16_t pos_x, pos_y, con_x, con_y;

void delay(void) {
	int i;
	for(i=0; i<100000; ++i) ;
}

void RCC_configure() {
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOC |
RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE | RCC_APB2Periph_GPIOB | RCC_APB2Periph_ADC1,
ENABLE);
}

void GPIO_configure() {
	GPIO_InitTypeDef GPIO_InitStructure;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_7);
	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_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
```

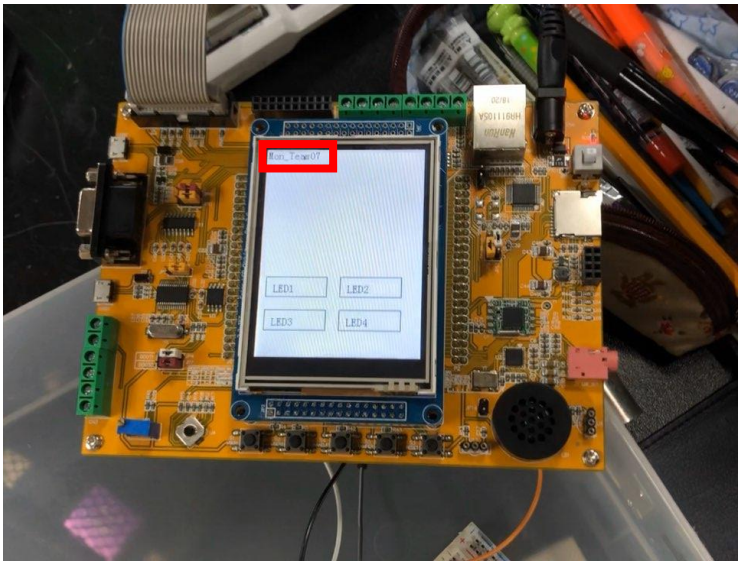
```
	GPIO_Init(GPIOC, &GPIO_InitStructure);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOC, &GPIO_InitStructure);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
	GPIO_Init(GPIOD, &GPIO_InitStructure);
}

void ADC_configure() {
	ADC_InitTypeDef ADC_InitStructure;
	ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
	ADC_InitStructure.ADC_ScanConvMode = ENABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
	ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_NbrOfChannel = 1;
	ADC_Init(ADC1, &ADC_InitStructure);
	ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
	ADC_Cmd(ADC1, ENABLE);
	ADC_ResetCalibration(ADC1);
	while(ADC_GetResetCalibrationStatus(ADC1));
	ADC_StartCalibration(ADC1);
	while(ADC_GetCalibrationStatus(ADC1));
}

int main(void) {
	char str[10];
	uint16_t temp;
	SystemInit();
	RCC_configure();
	GPIO_configure();
	ADC_configure();
	LCD_Init();
	Touch_Configuration();
	Touch_Adjust();
	LCD_Clear(WHITE);
	LCD_ShowString(1,1,"Mon_Team07", BLACK, WHITE);
	LCD_ShowString(20,210,"LED1", BLACK, WHITE);
	LCD_ShowString(130,210,"LED2", BLACK, WHITE);
	LCD_ShowString(20,260,"LED3", BLACK, WHITE);
	LCD_ShowString(130,260,"LED4", BLACK, WHITE);
	LCD_DrawRectangle(10, 200, 100, 230);
	LCD_DrawRectangle(120, 200, 210, 230);
	LCD_DrawRectangle(10, 250, 100, 280);
	LCD_DrawRectangle(120, 250, 210, 280);

	while(1) {
		Touch_GetXY(&pos_x, &pos_y, 1);
		Convert_Pos(pos_x, pos_y, &con_x, &con_y);
		ADC_SoftwareStartConvCmd(ADC1, ENABLE);
		ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
		temp = ADC_GetConversionValue(ADC1);
		if( con_x > 10 && con_x < 100 && con_y > 200 && con_y < 230 ) {
			GPIOD->BSRR = 0x04;
		}
		else if( con_x > 120 && con_x < 210 && con_y > 200 && con_y < 230 ) {
			GPIOD->BSRR = 0x08;
```

```
                    }
                    else if( con_x > 10 && con_x < 100 && con_y > 250 && con_y < 280 ) {
                              GPIOD->BSRR = 0x10;
                    }
                    else if( con_x > 120 && con_x < 210 && con_y > 250 && con_y < 280 ) {
                              GPIOD->BSRR = 0x80;
                    }
                    if(T_INT == 0) {
                              LCD_DrawCircle(pos_x, pos_y, 3);
                              sprintf(str, "(%d, %d)", con_x, con_y);
                              LCD_ShowString(100, 80, str, BLACK, WHITE);
                              LCD_ShowNum(10, 40, temp, 10, BLACK, WHITE);
                              delay();
                    }
          }
          return 0;
}
```

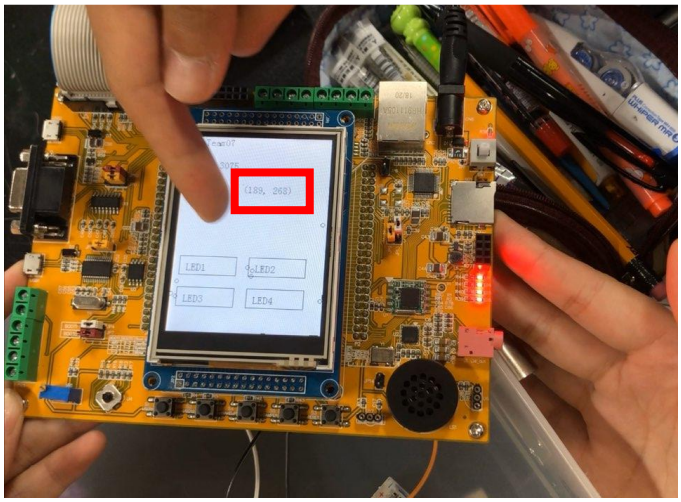
5.2 결과 확인 및 사진

소스 코드의 동작을 다음과 같이 확인했다.

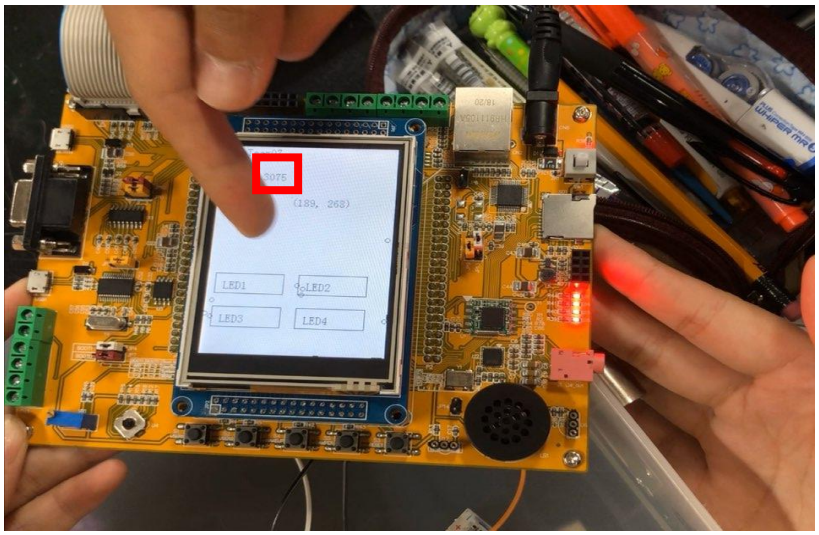
1. 팀 명 출력



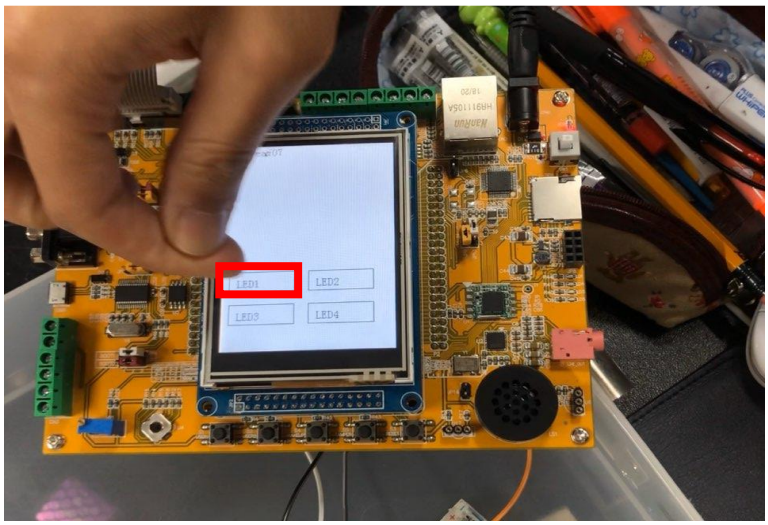
2. 터치 시 해당 위치에 작은 원과 좌표(x,y)출력

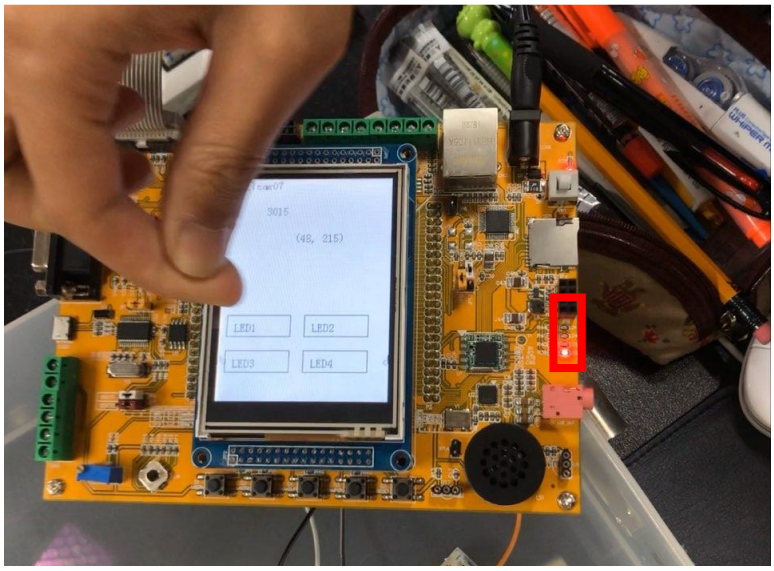


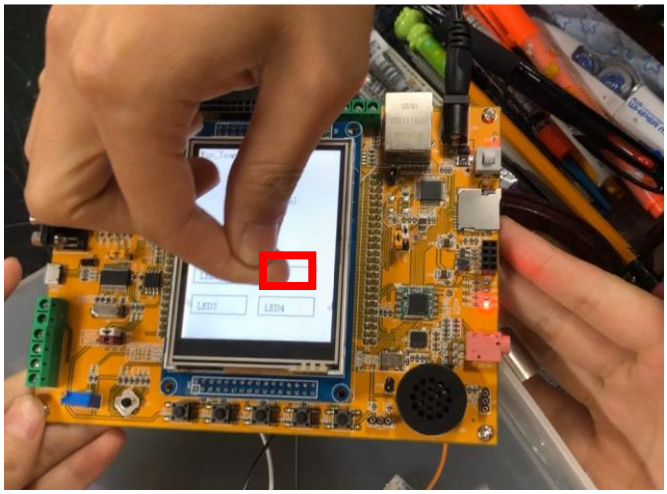
3. 조도센서(ADC) 값의 변화를 출력

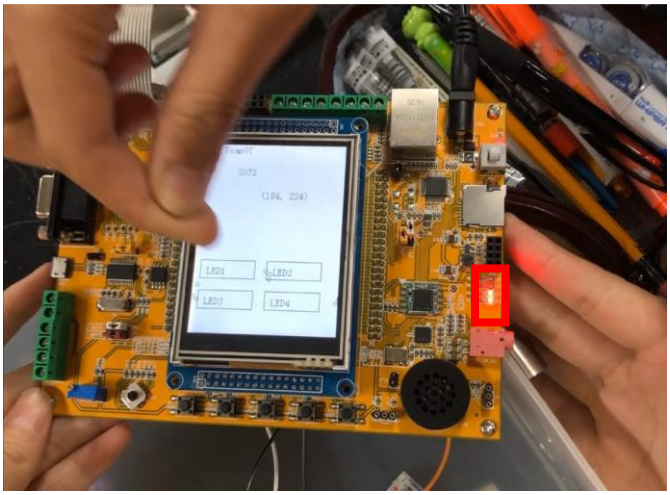


4. LED 버튼 터치 시 해당 LED 점등









06. 결론

TFT LCD 와 조도 센서를 처음 사용해보았다. Timing diagram 을 참고하여 직접 write 함수를 수정하는 것이 신기했다. 또한 조도 센서를 포함한 모든 아날로그 센서의 동작 원리가 저항 값의 변화라는 사실을 알고 나서 오실로스코프를 사용하여 파형이 변화하는 것을 눈으로 확인하니 막연했던 개념이 쉽게 이해되었다. 또한 조도 센서(ADC) 값을 받아와서 TFT LCD 에 출력하는 과정을 통해서 확인하고 싶은 센서의 값이나 표현하고 싶은 정보들을 LCD 모니터에 어떻게 나타내야 하는지 연습할 수 있었다.

8 주차 실험을 통해 앞으로의 실험이나 텀 프로젝트를 진행할 때에 어떤 센서에서 어떻게 값을 받아와서 어떤 형식으로 출력할 것인지, 또 해당 센서를 위한 포트를 설정할 때에 기능이 중복되지 않도록 어떻게 배분해야 할 것인지 충분히 고민해야 하는 이유에 대해 깨닫게 되었다.

또한 라이브러리를 사용할 때에 함수의 원형과 파라미터에 대해 충분히 숙지하고 구현을 진행한다면 예기치 않은 상황의 발생을 줄일 수 있을 것이라는 생각이 들었다. 조교님이 주신 라이브러리에 많은 것들이 이미 구현되어 있어서 직접 짜야 할 코드의 양은 적었지만, 라이브러리를 활용하기 위해 그것을 분석하고 조합하는 데 꽤 많은 시간이 소요되었다.

처음 해보는 실험이라 우여곡절이 많았지만 중간중간 질문을 통해 어려움을 해결할 수 있었고, 이해되지 않는 결과에 대해서는 설명을 듣고 코드를 다시 점검하는 과정에서 많은 것들을 배울 수 있었다.