부산대학교
PUSAN NATIONAL UNIVERSITY

# [2조] 10주차 결과보고서


| Subject | 임베디드 시스템 설계 및 실험 |
|---|---|
| Professor | 김원석 |
| Major | 정보컴퓨터공학부 |
| Date | 2022.11.05 |
| Team Member | 202055600 정홍빈 |
| | 201924617 끼앗띠퐁 유옌 |
| | 201824534 윤상호 |
| | 201824636 이강우 |

# 1. 실험 목적

TFT-LCD 라이브러리를 작성하고 이해하여 TFT-LCD의 Touch 동작을 제어하고, ADC를 이해하여 조도 센서값을 읽을 수 있다.

# 2. 배경 지식

## 가. TFT-LCD(초박막 액정표시장치)

### 1) LCD(Liquid Crystal Display) : 액정 디스플레이

LCD는 분극화된 재료의 두 층 사이에 있는 쐐기로 된 액체 크리스털을 사용하는 평면 디스플레이다. 뒤의 백라이트에서 쏜 빛을 편광 성질을 활용하여 액정의 배열에 따라 조절하면 영상을 표시하는 디스플레이로써, 작은 크기의 제품에서도 해상도가 높은 것이 장점이다.

### 2) TFT(Thin Film Transistor) : 박막 트랜지스터

디스플레이를 구성하는 픽셀 안에서 빛의 밝기를 조정하는 전기적 스위치 역할을 하고, 일정 전압을 가하면 스위치가 ON 되고, 일정 전압보다 낮은 전압을 가하면 OFF가 되어 화소에 필요한 데이터를 화소로 입력한다.

### 3) TFT-LCD(Thin Film Transistor Liquid Crystal Display)

AMLCD(Active matrix LCD)의 종류 중 하나이다. AMLCD는 각 화소 하나당 트랜지스터가 장착되어 있어 동작을 화소마다 독립적으로 제어할 수 있는 디스플레이다. 화소(pixel) 하나에 sub-pixel이 3개(R, G, B)로 이루어져 있으며 sub-pixel 하나마다 TFT가 하나씩 장착되어 있다.

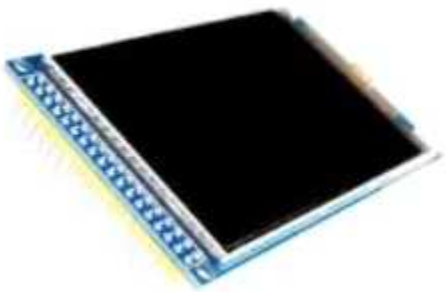

그림 2 TFT-LCD

4) TFT LCD Timing 특징과 Datagram

$\overline{CS}$ (Chip Select) : LOW일 때, Chip이 동작한다.
$\overline{WR}$ (Write) : Falling 할 때, Display RAM에 Data/Command를 Write 한다.
$\overline{RD}$ (Read) : Falling 할 때, Display RAM에서 Data를 Read 한다.
$D/\overline{C}$ (RS) : LOW일 때, Command를 전송, HIGH일 때, Data를 전송한다.

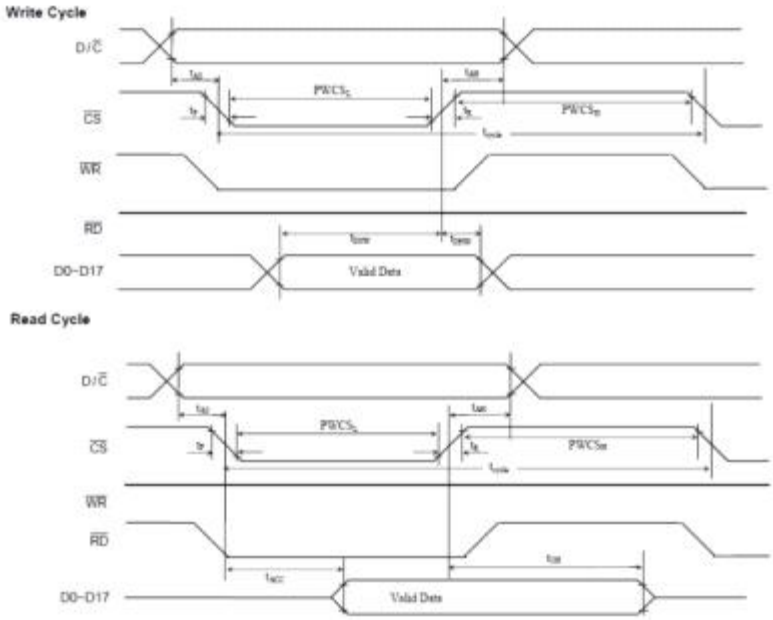


그림 3 LCD-timing-diagram

해당 symbol의 Min time 내에 Falling / Rising 해야 한다.

| Symbol | Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $t_{cycle}$ | Clock Cycle Time (write cycle) | 100 | - | - | ns |
| $t_{cycle}$ | Clock Cycle Time (read cycle) | 1000 | - | - | ns |
| $t_{AS}$ | Address Setup Time | 0 | - | - | ns |
| $t_{AH}$ | Address Hold Time | 0 | - | - | ns |
| $t_{DSW}$ | Data Setup Time | 5 | - | - | ns |
| $t_{DHW}$ | Data Hold Time | 5 | - | - | ns |
| $t_{ACC}$ | Data Access Time | 250 | - | - | ns |
| $t_{OH}$ | Output Hold time | 100 | - | - | ns |
| $PWCS_L$ | Pulse Width /CS low (write cycle) | 50 | - | - | ns |
| $PWCS_H$ | Pulse Width /CS high (write cycle) | 50 | - | - | ns |
| $PWCS_L$ | Pulse Width /CS low (read cycle) | 500 | - | - | ns |
| $PWCS_H$ | Pulse Width /CS high (read cycle) | 500 | - | - | ns |
| $t_R$ | Rise time | - | - | 4 | ns |
| $t_F$ | Fall time | - | - | 4 | ns |

그림 4 symbol의 Min time

## 나. ADC와 DAC

### 1) ADC(Analog-to-Digital Converter)

입력으로 들어오는 아날로그 신호를 디지털 신호로 변환하는 역할을 한다. 현실의 아날로그 신호를 컴퓨터의 디지털 신호로 변환하기 위해서 사용된다. 표본화, 양자화, 부호화의 과정을 거쳐서 디지털 신호로 변환된다.

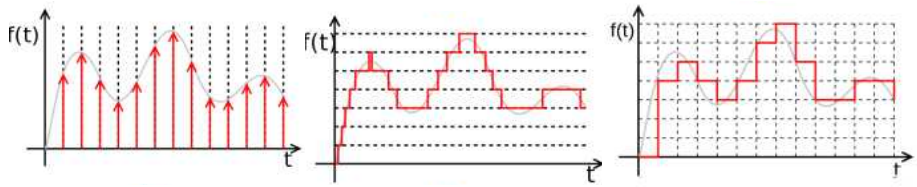


그림 5 ADC 처리 과정 (표본화, 양자화, 부호화)

### 2) DAC(Digital-to-Analog Converter)

출력으로 나가는 디지털 신호를 아날로그 신호로 변환한다. 컴퓨터의 디지털 신호를 현실에서 사용하기 위해서 변환을 시행한다. ADC의 역순으로 진행되며 암호화, 양자화, 표본화 순으로 변환을 진행한다.

### 3) 변환 과정

가) 표본화(Sampling)

입력 아날로그 신호를 일정한 주기의 표본으로 변환하는 과정을 의미한다. 샘플링 신호라는 주기적인 신호와 본래 신호를 비교하여 불규칙한 아날로그 신호를 주기마다 전압값을 직선 값으로 표현한 것이다. 본래 신호를 재현하기 위해서는 Nyquist 표본화 주파수 이상, 즉 본래 신호의 최고 주파수 2배 이상의 빈도로 표본화하여야 한다.

나) 양자화(Quantization)

표본화된 데이터를 일정 구간을 나눠 비트로 표현하기 어려운 값들을 비트로 표현할 수 있는 값으로 근사시킨다. ADC의 bit 수가 클수록 할당할 수 있는 영역이 커져 잡음이 줄어든다.

다) 부호화(Coding)

양자화된 데이터를 디지털 데이터 0과 1로 표현하는 것을 부호화라고 한다. 디지털 출력으로 값을 받기 위해 부호화 과정을 거치는 것이다. 이 과정을 거치면 비트마다 0이나 1 신호를 나타내긴 하는데, 펄스가 같지 않아 디지털 전달이 어려워 일정한 펄스를 주어서 인식시킨다.

# 3. 실험 과정

## 가. LCD 라이브러리 등록 및 lcd.c 완성

라이브러리에 미리 주어진 코드 font.h, lcd.c, lcd.h, touch.c, touch.h를 추가한다. Libraries 폴더 밑에 LCD 폴더 생성 후 위 5개 라이브러리 파일 추가한다.

프로젝트 옵션 - C/C++ Compiler - Preprocessor에서 생성한 LCD 라이브러리 폴더 경로를 추가한다.

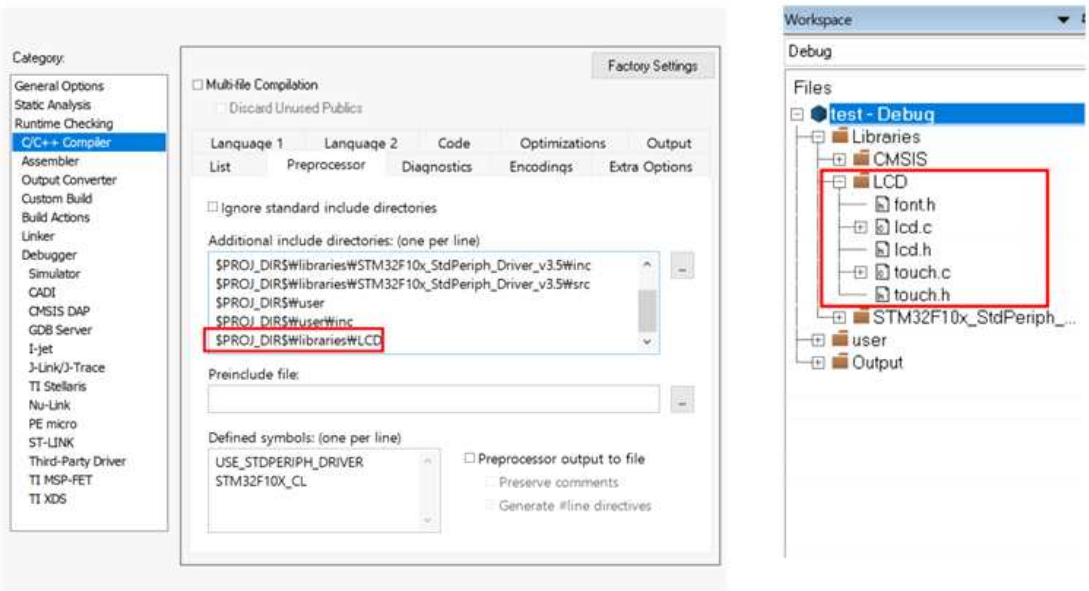


그림 6 LCD 라이브러리 추가 및 경로 설정

그림 3의 LCD timing diagram을 보면서 lcd.c의 todo를 다음과 같이 수정하여 완성한다.

```
static void LCD_WR_REG(uint16_t LCD_Reg)
{
  //TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_ResetBits(GPIOD, GPIO_Pin_13); //LCD_RS
  GPIO_ResetBits(GPIOC, GPIO_Pin_6);  //LCD_CS
  GPIO_ResetBits(GPIOD, GPIO_Pin_14); //LCD_WR

  GPIO_Write(GPIOE, LCD_Reg);
  //TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOC, GPIO_Pin_6);    //LCD_CS
  GPIO_SetBits(GPIOD, GPIO_Pin_14);   //LCD_WR

}
```

```
static void LCD_WR_DATA(uint16_t LCD_Data)
{
  //TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOD, GPIO_Pin_13);     //LCD_RS
  GPIO_ResetBits(GPIOC, GPIO_Pin_6);    //LCD_CS
  GPIO_ResetBits(GPIOD, GPIO_Pin_14);   //LCD_WR

  GPIO_Write(GPIOE, LCD_Data);
  //TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOC, GPIO_Pin_6);      //LCD_CS
  GPIO_SetBits(GPIOD, GPIO_Pin_14);     //LCD_WR

}
```

그림 7 LCD_WR_REG와 LCD_WR_DATA

**나. 조도 센서와 TFT-LCD 연결**

외부에서 들어온 신호를 ADC를 통해 ADC interrupt를 발생시킬 수 있다. 조도 센서에서 Analog 신호를 받기 위해서는 조도 센서와의 연결이 필요하다.

그림 8의 회로도와 같이 연결한 모습이다. 빛의 양이 많아질수록 저항이 낮아져 값이 작아진다. 노란색 선이 3v3(Vcc), 파란색 선이 GND, 초록색 선이 PB0(Analog 2) 핀과 연결되어 있다.

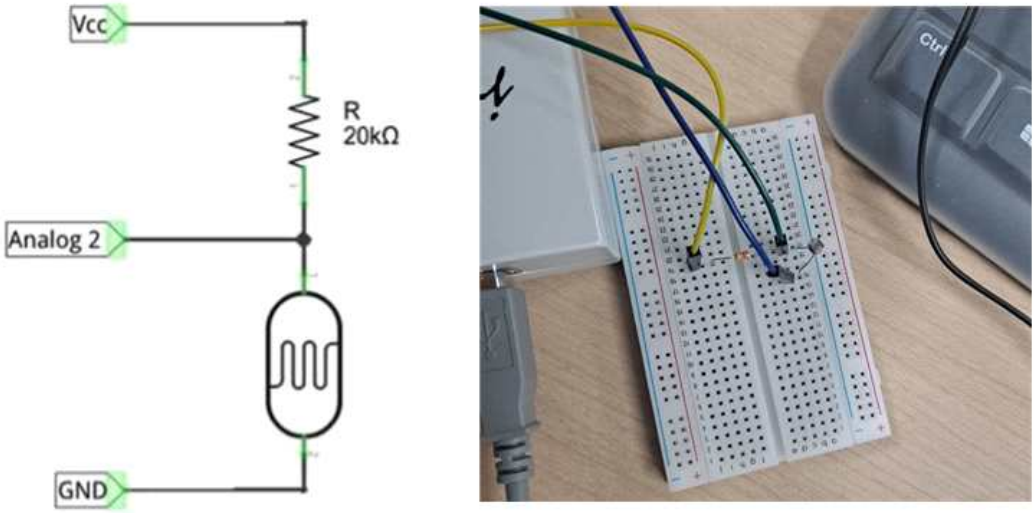


그림 8 조도 센서 회로도와 실제 연결 모습.

메인보드에 제일 오른쪽 핀만 남기고 연결한다.
TFT-LCD를 알맞게 연결하면 그림 9와 같이 LCD의 백라이트가 켜진다.

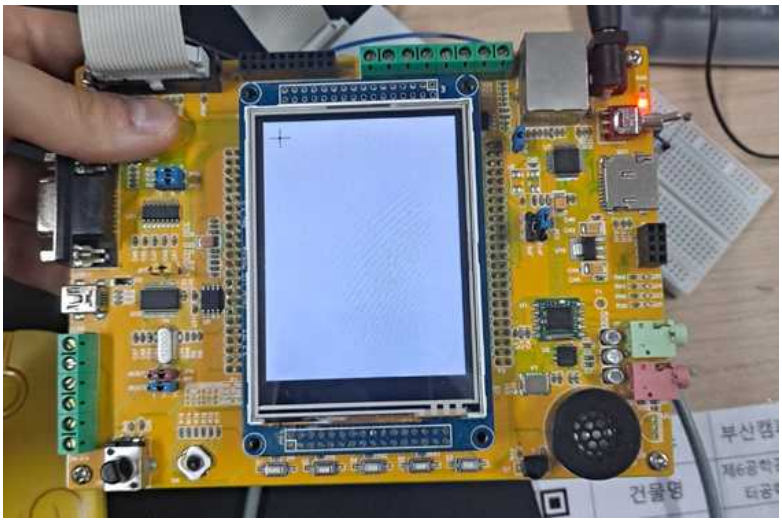

그림 9 연결된 TFT-LCD의 모습.

다. 소스코드

```
#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 "lcd.h"
#include "touch.h"
int color[12] =
{WHITE,CYAN,BLUE,RED,MAGENTA,LGRAY,GREEN,YELLOW,BROWN,BRRED,GRAY};

uint16_t value;

void RCC_Configure(void){
    // ADC port clock Enable
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    /* Alternate Function IO clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}

void GPIO_Configure(void){
    GPIO_InitTypeDef  GPIO_InitStructure;

   // TODO: Initialize the GPIO pins using the structure 'GPIO_InitTypeDef' and the function 'GPIO_Init'

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

그림 10 Source code 1

```
void ADC_Configure(void){
    ADC_InitTypeDef  ADC_InitStructure;

    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_Mode =  ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_239Cycles5);
    ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
    ADC_Cmd(ADC1, ENABLE);
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

void NVIC_Configure(void){
    NVIC_InitTypeDef NVIC_InitStructure;

    // TODO: Initialize the NVIC using the structure 'NVIC_InitTypeDef' and the function 'NVIC_Init'

    NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

그림 11 Source code 2

**preprocessing:**
LCD 라이브러리 사용을 위해 lcd.h와 touch.h를 include 하였다.

```
void ADC_Configure(void){
    ADC_InitTypeDef  ADC_InitStructure;

    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    // 변환을 여러 번 하므로 ENABLE로 설정한다.
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    // 데이터 정렬을 오른쪽으로 하도록 설정한다.
    ADC_InitStructure.ADC_Mode =  ADC_Mode_Independent;
    // ADC1과 다른 ADC 간의 sync가 필요하지 않으므로 independent로 설정한다.
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    // 변환이 단일채널 상태에서 수행되므로 DISABLE로 설정한다.
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    // 몇 개의 채널을 이용해 변환을 수행할지 선택한다. 1개를 선택하였다.
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    // 외부 trigger edge는 따로 선택하지 않는다.
    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_239Cycles5);
    // 채널, 우선순위, 샘플링 주기 채널에 따라 독립적으로 설정이 가능하다.
    ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);      // ADC1 interrupts
    ADC_Cmd(ADC1, ENABLE);                       // ADC1 사용 허가
    ADC_ResetCalibration(ADC1);                  // ADC1 reset
    while(ADC_GetResetCalibrationStatus(ADC1)); // reset 대기
    ADC_StartCalibration(ADC1);                  // ADC1 start
    while(ADC_GetCalibrationStatus(ADC1));       // start 대기
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);      // ADC1 변환 허가
}
```

그림 12 ADC_Configure_comment

**ADC_Configure :**
변환을 여러 번, 데이터 정렬은 오른쪽, ADC1과 다른 ADC 간의 sync 필요 없음, 단일채널 상태에서 변환, 1개 채널, 외부 트리거 엣지는 없음으로 ADC_InitStructure를 설정하여 init 한다.

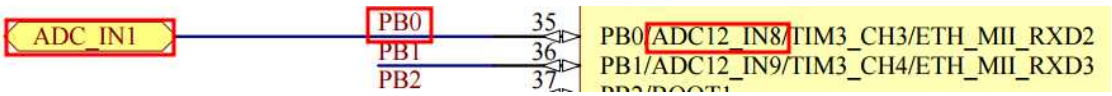


그림 13 STM32_schemetic

channel_8 : 사용하는 핀이 PB0이기에 channel_8을 선택했다.
EOC : 조도 센서 값 Conversion 후에 진행하므로 이 flag를 사용한다.

```
void ADC1_2_IRQHandler(){
  if(ADC_GetITStatus(ADC1, ADC_IT_EOC)){
        value = ADC_GetConversionValue(ADC1);
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
  }
}

int main() {
  uint16_t x, y,nx,ny;

  SystemInit();
  RCC_Configure();
  GPIO_Configure();
  ADC_Configure();
  NVIC_Configure();

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

  char str[11] = "THU_Team02";
  while(1){
  // TODO
      LCD_ShowString(0, 0, str, BLACK, WHITE);
      Touch_GetXY(&x,&y,1);
      Convert_Pos(x,y,&nx,&ny);
      LCD_DrawCircle(nx, ny, 5);

      LCD_ShowNum(50, 20, nx, 4, BLACK, WHITE);
      LCD_ShowNum(50, 40, ny, 4, BLACK, WHITE);
      LCD_ShowNum(0, 60, value, 4, BLACK, WHITE);
  }
}
```

그림 14 Source code 3

**ADC_1_2_IRQHandler :**
ADC1의 interrupt를 check 해서 값을 받아와 value에 저장하고, interrupt pending bit를 초기화한다.

**main :**
주어진 순서대로 함수를 호출한 후 LCD를 Clear 한다.
이후 주어진 미션지에 맞게 팀명을 출력하고, 터치하면 작은 원을 그리고 좌표와 저장된 조도 센서값을 출력하도록 작성하였다. 아래와 같은 LCD 라이브러리 함수들을 활용하였다.

**LCD_ShowString**(0, 0, str, BLACK, WHITE)
: (0, 0) 위치부터 가로 방향으로 배경은 WHITE, 글자 색은 BLACK으로 str을 작성한다.
**Touch_GetXY**(&x,&y,1)
: x와 y를 업데이트한다. 함수 내부의 T_INT는 $\overline{CS}$ 체크용이다.
**Convert_Pos**(x,y,&nx,&ny)
: 주어진 x, y를 변환하여 nx, ny를 업데이트하는 함수이다. Touch_GetXY의 결과를 올바르게 변경하는 데 사용된다.
**LCD_DrawCircle**(nx, ny, 5)
: (nx, ny)를 원점으로 직경이 5인 원을 그린다.
**LCD_ShowNum**(50, 20, nx, 4, BLACK, WHITE)
: (50, 20) 위치부터 가로 방향으로 배경은 WHITE, 글자 색은 BLACK으로 전체 길이 4의 공간에 숫자 nx를 작성한다.

# 4. 실험 결과

10주차 미션
- 팀명 출력(THU_Team02)
- LCD 터치 시 해당 위치에 작은 원 그리기
- 좌표와 조도 센서값 출력

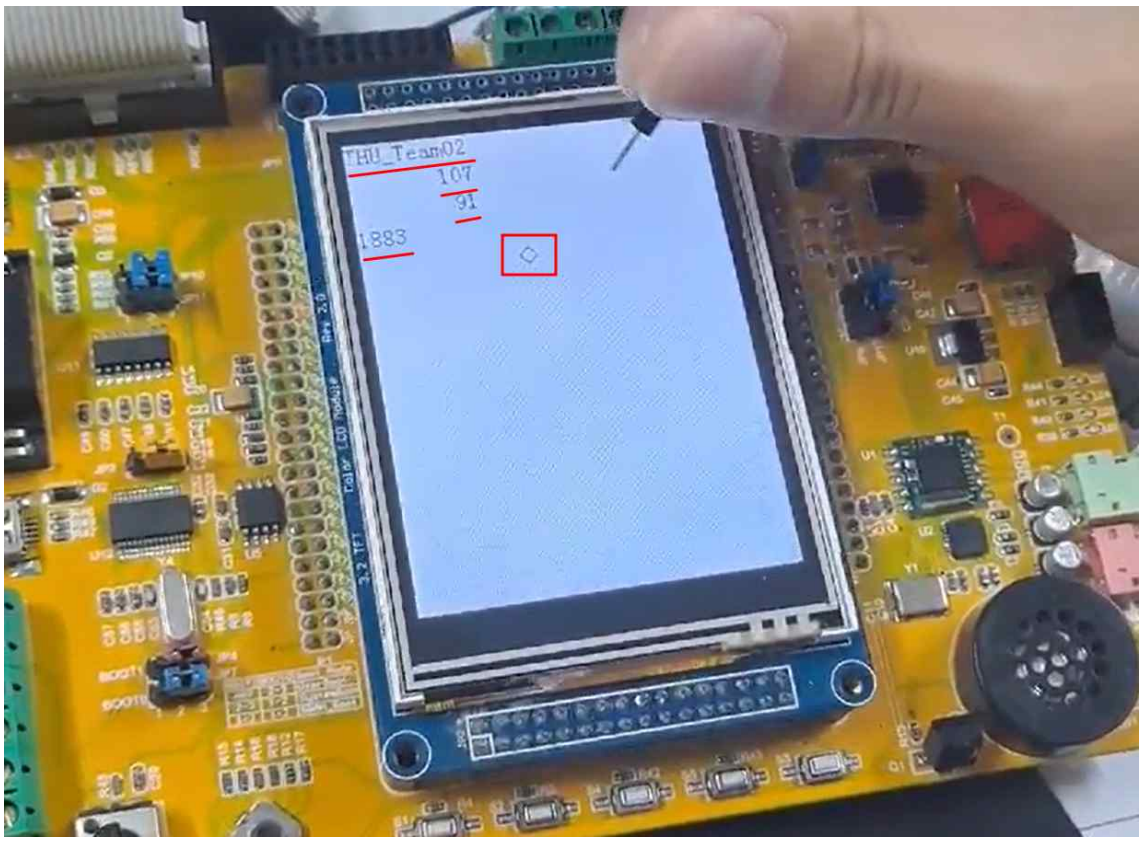


그림 15 올바르게 동작하는 TFT-LCD

# 5. 결론

이번 실험에서는 ADC가 무엇인지 알고, 직접 값을 받아 LCD에 출력해보는 시간을 가졌다. lcd.c를 제외한 라이브러리가 제공되어 있어 이전 실험에서 사용되었던 코드를 응용하면 어렵지 않게 완성할 수 있을 것이라 기대했으나, 조도 센서값이 변하지 않아 그 문제를 해결하는데 시간이 꽤 소요되었다. 또한, 조교님이 질문하셨던 independent의 의미를 자세히 몰랐는데, 찾아보니 master인 ADC1만 사용하므로 다른 ADC Block과 sync를 맞출 필요가 없어 independent로 설정하는 것이라는 것을 알게 되었다. 앞으로 있을 텀 프로젝트에서도 외부의 값을 읽어야 하는 상황이 있기에 이 경험이 유용하게 쓰일 것으로 본다.