임베디드 시스템 설계 및 실험
10주차 실험 결과 보고서

조 : 8조

조원 : 서진욱, 이승민, 하연지, 하태훈

● 실험목표

1. **TFT LCD의 원리와 동작 방법을 이해하고 관련 라이브러리 작성 및 Touch 동작 과정을 이해한다.**
2. **ADC 개념을 이해하고, 이를 활용하여 조도 센서를 사용한다.**

● 개념설명

1. TFT-LCD

- 초 박막 액정 표시장치
- 액체와 고체의 중간 특성을 가진 액정의 상태 변화와 편광판의 편광 성질을 이용하여 통과하는 빛의 양을 조절함으로써 정보를 표시
- RGB 픽셀이 유리판에 코딩 되어 컬러 영상을 구현하는 Color Filter
- 액정을 제어하기 위해 초박형 유리 기판 위에 반도체 막을 형성한 회로인 TFT 기판
- Filter와 기판 사이에 주입된 액정과 광원인 Black light unit으로 구성

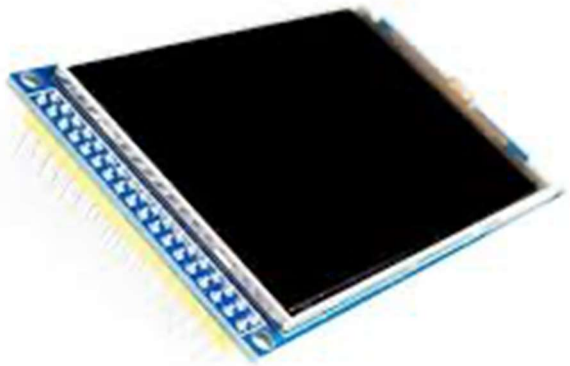

그림 1. TFT-LCD

2. TFT-LCD Timing

- **각 신호들이 시간 별로 처리되는 과정**

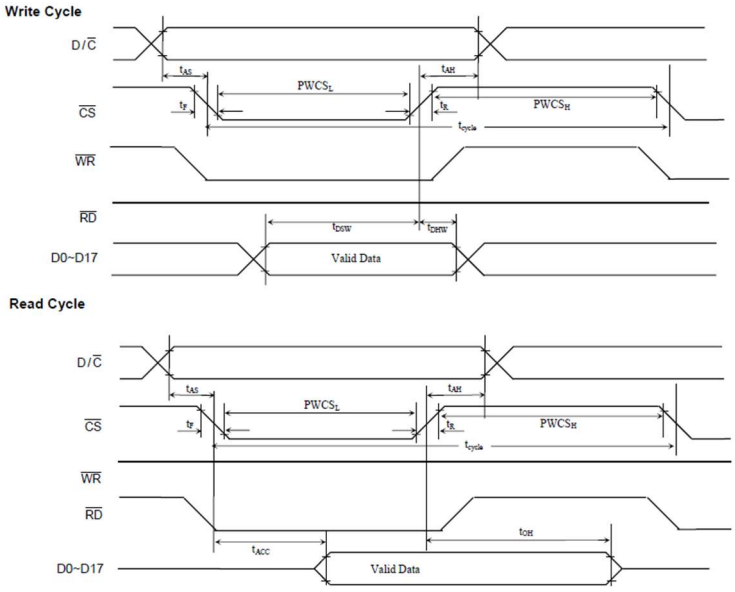

그림 2. TFT-LCD Timing Diagram

- Rising Edge : Low에서 High로 올라가는 구간
- Falling Edge : High에서 Low로 떨어지는 구간

- **Write / Read Cycle**
  - $\overline{CS}$**: Chip Select (Chip Enable)**
    - ▷ High 에서 Low로 Falling Edge 일 때 LCD Chip 을 사용
  - $D/\overline{C}$**: Data / Command (핀맵에서 RS)**
    - ▷ LCD는 Data 와 명령어 레지스터를 함께 사용
    - ▷ High 로 두고 Data를 전송, Low 로 두고 Command를 전송
  - $\overline{WR}$**,** $\overline{RD}$ **: Write / Read**
    - ▷ High 에서 Low로 Falling Edge 일 때, Data를 display에 Write / Read 함
  - 해당 Symbol의 Min Time 내에 Falling / Rising 을 해야 함

- **Write Cycle**
  - ***COMMAND***
    - ▷ $D/\overline{C}$를 Low, $\overline{CS}$를 Low, $\overline{WR}$를 Low로 두고 Command를 전송
    - ▷ $\overline{CS}$를 High, $\overline{WR}$를 High로 다시 돌려놓기
  - ***DATA***
    - ▷ $D/\overline{C}$를 High, $\overline{CS}$를 Low, $\overline{WR}$를 Low로 두고 Data를 Display에 전송
    - ▷ $\overline{CS}$를 High, $\overline{WR}$를 High로 다시 돌려놓기

- **Read Cycle**
  - ***$D/\overline{C}$가 high, $\overline{CS}$가 Low, $\overline{RD}$ 신호가 Low 일 때 D0~D17의 Display를 읽음***
    - ▷ Master에게 알림

3. ADC(Analog to Digital Converter)

- 아날로그 신호를 디지털로 변환하는 것
- 아날로그 신호가 들어오면 이를 표본화, 양자화를 거쳐 부호화
- 표본화: 일정한 간격으로 아날로그 신호의 값을 추출
- 양자화: 추출한 표본 샘플 신호의 레벨을 단계를 나누어 나타내는 과정
- 부호화: 양자화로 나눈 레벨에 속한 값을 이진수로 변환

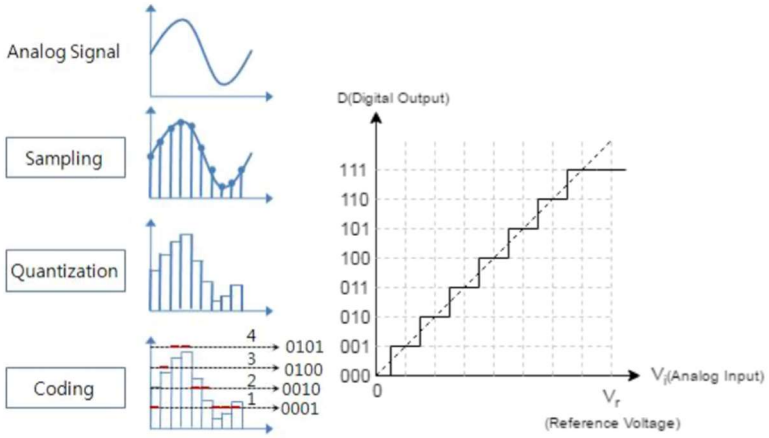

그림 3. ADC(Analog to Digital Converter)

4. 조도센서

- 주변의 밝기를 측정하는 센서
- 빛의 양이 많아질수록 전도율이 높아져 저항이 낮아짐

그림 4. 조도 센서

● 실험 진행

● 코드 작성

1) LCD code

- GPIO_ResetBits 함수를 사용해 GPIO 핀을 설정
  ▷ Chip 선택(LCD_CS), 데이터/명령 선택(LCD_RS), 쓰기 동작(LCD_WR)을 LOW로 설정해 LCD제어
- GPIO_SetBits사용해 LCD_WR 핀과 LCD_CS 핀을 HIGH로 설정해 LCD제어를 완료

```
static void LCD_WR_REG(uint16_t LCD_Reg)
{
        // TODO implement using GPIO_ResetBits/GPIO_SetBits
        GPIO_ResetBits(GPIOC, GPIO_Pin_8); // LCD_CS(0);
        GPIO_ResetBits(GPIOD, GPIO_Pin_13); // LCD_RS(0);
        GPIO_ResetBits(GPIOB, GPIO_Pin_14); // LCD_WR(0);


        GPIO_Write(GPIOE, LCD_Reg);
        // TODO implement using GPIO_ResetBits/GPIO_SetBits
        GPIO_SetBits(GPIOB, GPIO_Pin_14); // LCD_WR(1);
        GPIO_SetBits(GPIOC, GPIO_Pin_8); // LCD_CS(1);

}
```

코드 1. LCD_WR_REG

- LCD_WR_DATA 함수는 데이터를 LCD에 쓰는 역할
- Chip 선택(LCD_CS), 쓰기 동작(LCD_WR)을 LOW로 데이터/명령 선택(LCD_RS)는 HIGH로 설정
- GPIO_SetBits함수를 사용해 LCD_WR, LCD_CS를 HIGH로 설정해 LCD에 데이터 쓰는 동작 완료

```
static void LCD_WR_DATA(uint16_t LCD_Data)
{
        // TODO implement using GPIO_ResetBits/GPIO_SetBits
        GPIO_ResetBits(GPIOC, GPIO_Pin_8); // LCD_CS(0);
        GPIO_SetBits(GPIOD, GPIO_Pin_13); // LCD_RS(1);
        GPIO_ResetBits(GPIOB, GPIO_Pin_14); // LCD_WR(0);

        GPIO_Write(GPIOE, LCD_Data);
        // TODO implement using GPIO_ResetBits/GPIO_SetBits
        GPIO_SetBits(GPIOB, GPIO_Pin_14); // LCD_WR(1);
        GPIO_SetBits(GPIOC, GPIO_Pin_8); // LCD_CS(1);

}
```

코드 2. LCD_WR_DATA

2) main code

- ADC1가 PB0에 대응되므로 Port B를 활성화하기 위해 RCC_APB2Pheriph_ADC1과 RCC_APB2Periph_GPIOB를 Enable하고 Alternate Function IO clock을 Enable 함

```
void RCC_Configure(void)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOB, ENABLE); // ADC1, port C RCC ENABLE
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

코드 3. RCC_Configure

- ADC1은 PB0을 사용하고 ADC는 Analog 신호를 받아들이므로 Analog In Mode로 설정해주며, 출력 속도는 50MHz로 설정

```
void GPIO_Configure(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  //ADC1
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  //Set Pin Mode Input Pull-up/down UP
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  //Set Pin Mode General Output Push-Pull
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

코드 4. GPIO_Configure

- ADC의 InitStructure를 정의하여 설정 을 할당
  - ▷ ADC1이 독립 모드로 동작하므로 동작 모드는 ADC_Mode_Independent로 설정
  - ▷ 신호의 Conversion 은 Single Mode에서 동작하므로 ADC_ScanConvMode는 Enable로, Conversion은 Continuous하게 이루어지도록 하기 위해 ADC_ContinuousConvMode를 Enable로 설정
  - ▷ External trigger에 의해 Conversion이 일어나도록 하는 것은 아니기 때문에 ADC_ExternalTrigConv_None을 할당
  - ▷ Data 정렬은 오른쪽으로, 채널은 1개만 활용할 것이므로 ADC_NbrOfChannel에는 1을 할당
- ADC1의 Channel 8번을, Rank에는 채널이 1개이므로 1, 가장 긴 주기를 선택하기 위해 ADC_SampleTime은 239Cycle로 설정
- ADC_ITConfig에서는 ADC1의 Conversion이 끝났을 때 Interrupt를 발생시킴
- 설정 완료 후, ADC_Cmd를 통해 ADC1을 Enable하고 Calibration을 Reset 상태로 만든 후 Calibration이 완료되면 Conversion이 실행되도록 구성

```
void ADC_Configure(void) {
  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_8, 1,
                           ADC_SampleTime_239Cycles5);
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE); // interrupt enable
  ADC_Cmd(ADC1, ENABLE); // ADC1 enable
  ADC_ResetCalibration(ADC1);

  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

코드 5. ADC_Configure

- Interrupt 처리를 위해 ADC1_2_IRQn을 Enable하며 NVIC PriorityGroup은 2로 두고, pre-emption priority와 sub priority는 각각 0으로 설정

```
void NVIC_Configure(void) {
  NVIC_InitTypeDef NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
  //ADC1
  //NVIC Line ADC1
  NVIC_EnableIRQ(ADC1_2_IRQn);
  NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

코드 6. NVIC_Configure

- ADC1에서 interrupt가 발생하면 조도센서에서 값을 읽어와 Conversion하여 이를 Global Variable로 선언한 value에 할당하고 PendingBit를 초기화

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

코드 7. ADC1_2_IRQHandler

- main의 while문에서는 "THU_Team08"을 출력하도록 LCD_ShowString함수를 이용
  - ▷ LCD에 터치 동작이 들어왔을 때 해당 위치값을 읽고 변환하여 LCD에 x1, y1을 각각 (50, 50), (50, 70) 위치에 LCD_ShowNum 함수를 이용해 출력
  - ▷ 조도센서의 값 또한 읽어 (60, 100) 위치에 출력하고, 터치 동작이 들어온 위치에 LCD_DrawCircle 함수를 이용해 반지름 4인 원을 그림

```
while (1) {
  LCD_ShowString(10, 10, "THU_Team08", BLACK, WHITE);
  Touch_GetXY(&x1, &y1, 1);
  Convert_Pos(x1, y1, &x1, &y1);

  LCD_ShowNum(50,50, x1, 4, BLACK, WHITE);
  LCD_ShowNum(50, 70, y1, 4, BLACK, WHITE);

  ADC_ITConfig(ADC1,ADC_IT_EOC,ENABLE);
  LCD_ShowNum(60,100, value, 4, BLACK, WHITE);
  ADC_ITConfig(ADC1,ADC_IT_EOC,DISABLE);
  LCD_DrawCircle(x1, y1, 4);
}
```

코드 8. main()의 LCD 구현 code

● 실습 결과

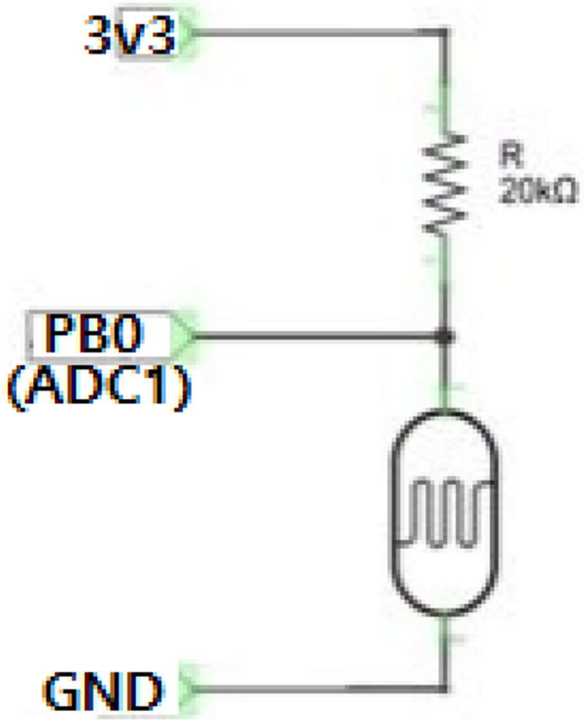

그림 5. 실험에서 구성한 조도센서 회로도 및 핀 구성

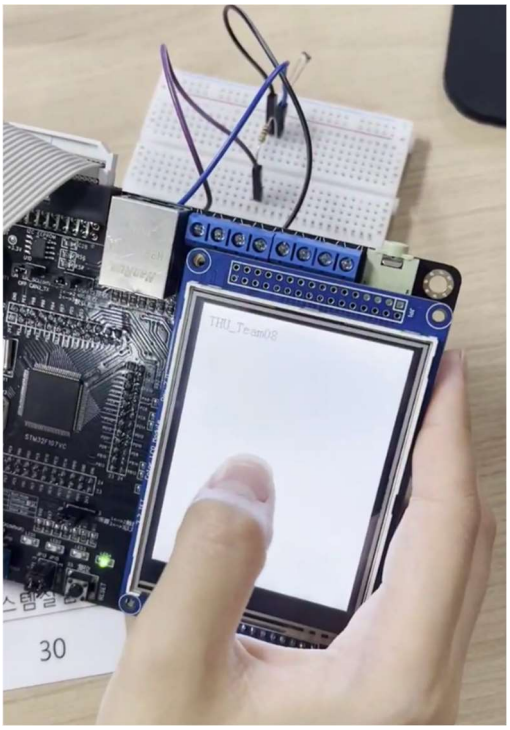

그림 6. 초기 보정 과정 후 첫 화면(조 이름 출력)

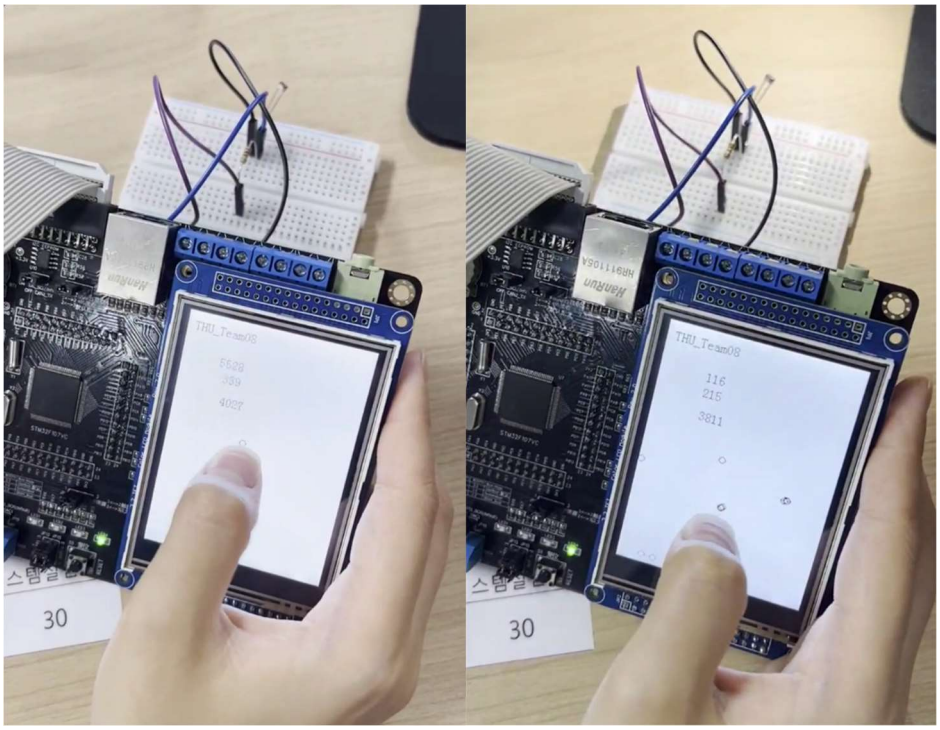

그림 7. 터치 동작 화면

- 터치를 할 때 마다 화면이 갱신되고, 터치를 한 위치에 도형(원)이 출력됨과 동시에 ADC 를 통한 조도센서 값과 터치한 곳의 좌표가 출력됨
- 스마트폰의 플래시를 활용하여 밝기 변화를 주자 값이 변화함

● 결론

- ADC의 역할과 사용 방법에 대해 알게 되었고, DAC의 역할 또한 함께 알게 되었음
- LCD 터치 패널은 초기에 터치 보정이 필요하다는 것을 알게 되었고, LCD패널에 텍스트 및 도형을 출력 하는 방법을 익히게 되었음
- 조도센서의 값이 가끔 부정확한 경우가 있었는데 해당 경우에 대한 재확인이 필요하다는 것을 확인 함
- 터치 인식이 너무 빨리 되어 한 자리에 도형이 여러 개 그려지는 경우가 있었는데 이 역시 interrupt를 통한 입력이니 delay를 통해 해결 할 수 있는 문제일지 파악할 필요가 있음을 확인 함