임베디드 시스템 설계 및 실험

10주차 LCD 및 ADC

4조

윤소현 김동윤 유주연 팜민두옹

실험 목표

- TFT LCD 제어 및 ADC 사용

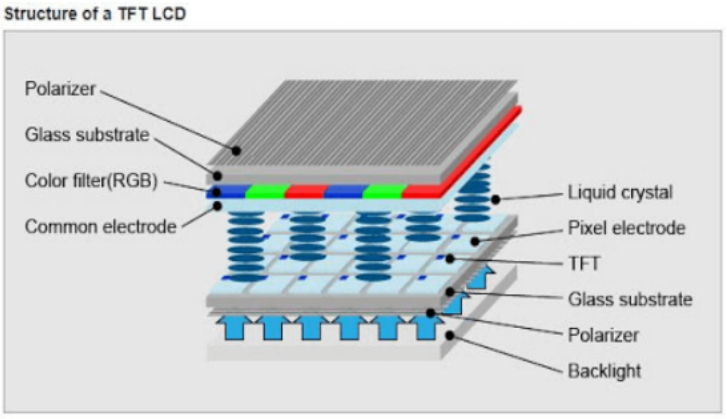
실험 내용

1. TFT-LCD 보드에 올바르게 결착
2. lcd.c에서 write 관련 코드 작성
3. TFT-LCD에 Text(팀명) 출력 ex) THUR_TeamXX
4. **ADC channel과 인터럽트를 사용**하여 조도 센서 값을 전역변수에 저장
5. LCD 터치 시(main에서 폴링 방식) 해당 위치에 작은 **원을 그리고**, **좌표(X, Y)**와 전역 변수에 저장했던 **조도 센서 값 출력** (LCD, Touch 라이브러리 파일 참고, LCD : 240x320)

주요 개념 #1 : TFT-LCD (Thin Film Transistor Liquid Crystal Display)

TFT-LCD란 얇은 필름 형태의 **트랜지스터와 액정을 이용해 이미지를 표시하는 기술**이다. 주로 스마트폰과 모니터 등 화면에 사용된다.

구성 요소 :

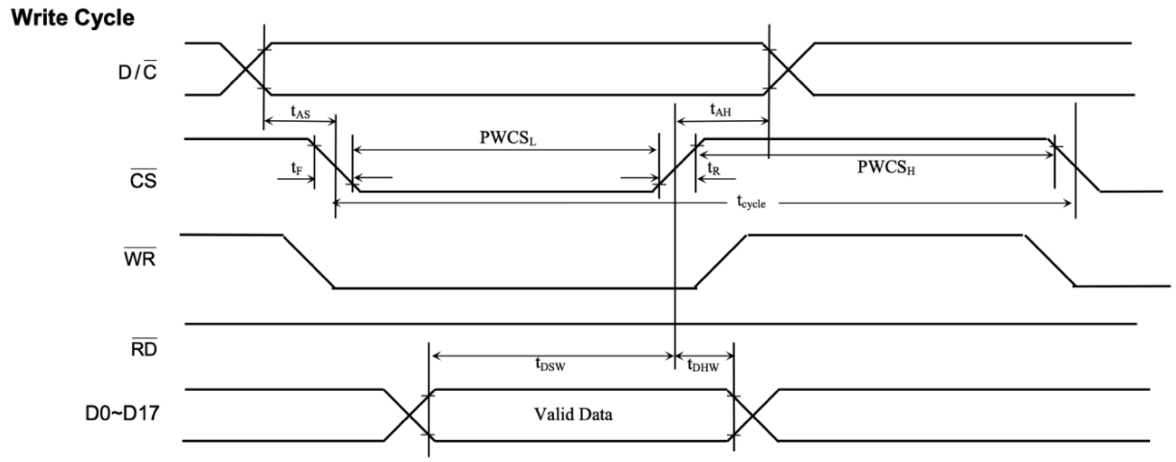


1. **Backlight Unit(백라이트 유닛)** : LCD는 자체적으로 빛을 내지 않으므로, 뒤에서 빛을 비춰주는 광원이 필요하다.
2. **Polarizer(편광판)** : 빛의 특정 방향만 통과시켜, 액정 상태에 따라 빛의 양을 조절해 화면을 표현한다.
3. **TFT Substrate (TFT 기판)** : 각 픽셀을 개별적으로 제어하는 회로이다.
4. **액정 (Liquid Crystal)** : 전기 신호에 따라 상태를 바꿔 빛의 투과량을 조절한다.
5. **RGB Color filter (RGB 컬러 필터)** : RGB 픽셀을 통해 색상을 형성한다.

주요 개념 #2 : Timing Diagram

TFT-LCD에 **데이터를 쓸 때** 쓰기 신호, 데이터 신호 등 **신호 간의 순서와 시간 차이**를 맞춰야 작동한다.

Timing Diagram : 이러한 신호들이 시스템에서 시간에 따라 어떻게 변화하는지 시각적으로 나타낸 그래프



- **CS (Chip Select)** :

  High : 비활성화 상태로 LCD는 외부 명령을 무시한다.

  Low : 활성화 상태로 LCD가 MCU의 명령을 받을 준비를 한다.

- **D/C (Data/Command)** :

  High : MCU가 LCD에 데이터를 전송 설정 (Data)

  Low : LCD에 동작 제어 알림 (Command)

- **WR (Write)** :

  High : High 상태로 돌아오면 작업 완료

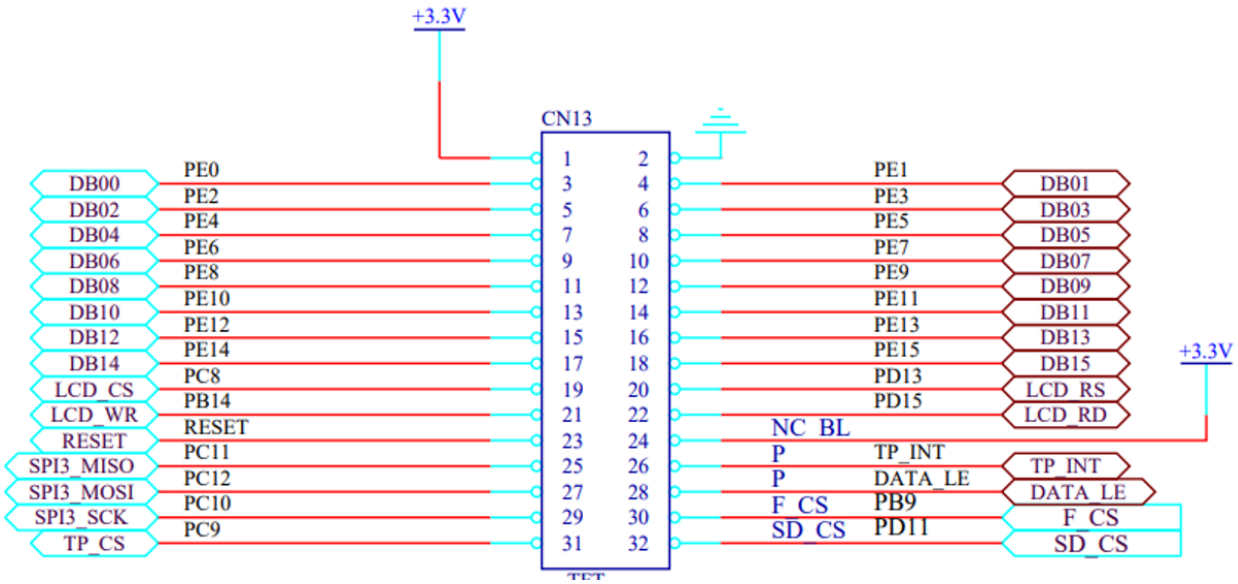
  Low : Wire 작업이 시작된다. 이때 MCU가 LCD에 데이터 전달 가능

- **D0 ~ D17 (데이터라인)** : 실제 LCD에 기록될 데이터가 전송됨

표로 정리하면 다음과 같다.

	CS	D/C	WR
Command 전달 시	Low	Low	Low
DATA 전달 시	High	High	Low

Command, Data 실행 후엔 **CS, WR를 다시 High로 설정**해야 한다.



TFT-LCD Pin 맵

- **D/C 신호는 RS에 해당**한다

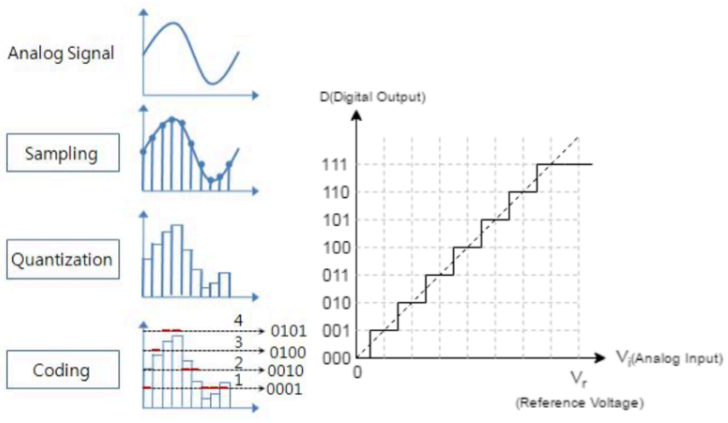
- **Data line**은 DB(**port E**)에 해당한다

주요 개념 #2 : ADC(Analog to Digital Converter)

아날로그 신호를 디지털 값으로 변환

변환 과정 :

1. **Sampling (표본화)** : 아날로그 신호의 값을 일정한 간격으로 추출하여 샘플링
2. **Quantization (양자화)** : 샘플링한 값들을 일정한 레벨로 구분하여 단계별로 표현
3. **Coding (부호화)** : 양자화된 각 레벨 값을 이진수로 변환하여 디지털 형태로 나타냄

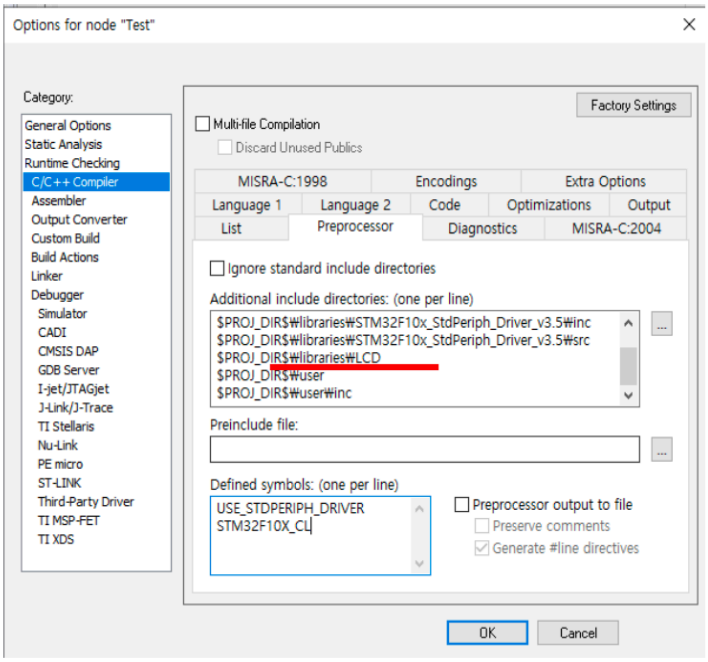


주요 개념 #3 : 조도센서

주변의 **밝기를 측정하는 센서**이다. 빛의 양이 많을 때는 저항이 낮아지고, 적을 때는 저항이 높아진다

1. 저항의 변화는 **아날로그** 신호로 나타난다
2. **ADC**를 통해 **디지털 신호로 변환**된다
3. **MCU**가 디지털 값을 **LCD에 출력**한다

세부 실험 내용



제공받은 font.h, lcd.h, touch.c, touch.h를 사용하기 위해 Preprocessor에 *$PROJ_DIR$₩libraries₩LCD* 를 추가해준다.

Todo #1 : LCD 라이브러리의 LCD_WR_REG, LCD_WR_DATA 함수 구현

- **LCD_WR_REG** 함수는 LCD 레지스터에 **명령어를 쓰는 함수**이다.

- **LCD_WR_DATA** 함수는 LCD에 **데이터를 쓰는 함수**이다.

- 두 함수의 기능을 구현하기 위해 **CS(Chip Select), RS(Data/Command), WR가 사용**된다.
TFT-LCD Pin 맵을 보면 **각각은 PC8, PD13, PD14**임을 알 수 있다.

CS	PC8
RS	PD13
WR	PD14

```
static void LCD_WR_REG(uint16_t LCD_Reg)
{
  // TODO implement using GPIO_ResetBits/GPIO_SetBits
  // PD13은 LCD_RS(LCD 레지스터)
  // D/C(Data/Command)는 데이터와 명령을 구분하는 제어핀. LCD가 받은 값이 명령인지 제어인지 알려주는 역할
  // 0 : 명령모드, 1 : 데이터모드
  GPIO_ResetBits(GPIOD, GPIO_Pin_13); // D/C LOW로 전환 -> 명령모드
  GPIO_ResetBits(GPIOC, GPIO_Pin_8); // CS(Chip select) LOW : LCD와 통신 활성화
  GPIO_ResetBits(GPIOB, GPIO_Pin_14); // WR(Write) LOW : 쓰기 모드로 설정

  GPIO_Write(GPIOE, LCD_Reg);

  // TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOB, GPIO_Pin_14); // WR HIGH -> Setbits (쓰기 작업 완료)
  GPIO_SetBits(GPIOC, GPIO_Pin_8); // CS HIGH -> Setbits (LCD와 통신 비활성화)
}
```

- **Command 전달 시** CS, RS(D/C), WR **모두 Low**여야 한다. **Command 실행 후** CS와 WR를 **High로** 바꿔준다.

```
static void LCD_WR_DATA(uint16_t LCD_Data)
{
  // TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOD, GPIO_Pin_13); // D/C HIGH -> 데이터모드
  GPIO_ResetBits(GPIOC, GPIO_Pin_8); // CS LOW
  GPIO_ResetBits(GPIOB, GPIO_Pin_14); // WR LOW

  GPIO_Write(GPIOE, LCD_Data); // 데이터 버스에 전송할 데이터를 쓴다

  // + implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOB, GPIO_Pin_14); // WR HIGH -> Setbits
  GPIO_SetBits(GPIOC, GPIO_Pin_8); // CS HIGH -> Setbits
}
```

- **Data 전달 시 CS, RS(D/C)는 High, WR는 Low**여야 한다. **Data 전달 후 CS를 Low**로 하여 통신을 종료하고 WR을 High로 설정하여 쓰기 작업이 완료되었음을 알린다.

Todo #2 : RCC_Configure 구현

```
void RCC_Configure(void)
{
   /* Port A, C, D, E clock enable */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE, ENABLE);

   /* ADC1 enable */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

}
```

- LCD를 사용하기 위해 A, B, C, D, E 포트를 활성화 시켜준다. 조도 센서도 사용할 것으므로 ADC도 활성화 시켜준다.

Todo #3 : GPIO_Configure 작성

```
void GPIO_Configure(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   /* ADC1 pin setting */
   //Input
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

- 조도센서를 사용하기 위해 ADC12(PC2)를 활성화 시킨다.

- **AIN** 모드는 **아날로그 입력 모드**다. 조도센서는 아날로그 값을 신호로 전송하기 때문에 AIN으로 설정

Todo #4 : ADC_Configure 작성

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

   // ADC의 동작 모드를 독립 모드(Independent)로 설정
   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

   // ADC 스캔 변환 모드를 비활성화 (단일 채널 변환)
   ADC_InitStructure.ADC_ScanConvMode = DISABLE;

   // ADC의 연속 변환 모드를 활성화하여 지속적으로 데이터를 읽음
   ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;

   // 외부 트리거를 사용하지 않고 소프트웨어로 변환 시작 (자동으로 변환 시작)
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

   // 변환할 채널 수를 1로 설정 (단일 채널 변환)
   ADC_InitStructure.ADC_NbrOfChannel = 1;
```

```
    // 설정한 내용을 ADC1에 적용하여 초기화
    ADC_Init(ADC1, &ADC_InitStructure);

    // ADC1의 Regular Channel 12(채널 12)로 설정, 우선 순위 1, 샘플링 시간 239.5 사이클
    ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 1, ADC_SampleTime_239Cycles5);

    // 변환 완료 인터럽트(End of Conversion) 기능을 활성화
    ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);

    // ADC1을 활성화
    ADC_Cmd(ADC1, ENABLE);

    // ADC 캘리브레이션을 리셋 (보정 초기화)
    ADC_ResetCalibration(ADC1);

    // 캘리브레이션 리셋이 완료될 때까지 대기
    while (ADC_GetResetCalibrationStatus(ADC1));

    // ADC1의 캘리브레이션 시작
    ADC_StartCalibration(ADC1);

    // 캘리브레이션이 완료될 때까지 대기
    while (ADC_GetCalibrationStatus(ADC1));

    // ADC1의 소프트웨어 시작 명령을 활성화하여 변환 시작
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

- ADC_InitTypeDef 구조체를 통해 설정을 한 뒤 ADC_Init으로 초기화한다
- 보정 (ADC_ResetCalibration, ADC_StartCalibration)을 통해 정확성을 높여준다

Todo #5 : NVIC_Configure 작성

- **NVIC**는 STM32 같은 마이크로컨트롤러에서 **인터럽트를 관리**하는 하드웨어 모듈이다.

```
void NVIC_Configure(void) {
   NVIC_InitTypeDef NVIC_InitStructure;
   NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;// ADC1_2
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
}
```

- NVIC를 설정하여 ADC1 인터럽트를 활성화하고 우선순위를 설정한다.

- ADC1 인터럽트가 NVIC에서 활성화되어, **ADC 변환이 완료되면 인터럽트가 발생**한다.

- 설정한 NVIC 초기화 구조체를 NVIC_Init 함수에 넣어 ADC1 인터럽트가 설정한 우선순위와 함께 활성화되도록 한다.

Todo #6 : ADC1_2_IRQHandler 구현

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

- ADC1_2_IRQHandler 함수는 **인터럽트 핸들러**로, **ADC 변환 완료 인터럽트가 발생했을 때 실행**된다.

- ADC_GetITStatus 함수는 지정한 인터럽트의 상태를 확인하여 인터럽트가 발생한 경우 RESET이 아닌 값을 반환한다.

- if문에서 ADC1의 **변환 완료 인터럽트(End Of Conversion)**의 상태를 확인한다.

- 인터럽트가 발생한 경우 ADC1에서 변환된 값을 value에 저장한다.

- ADC_ClearITPendingBit 함수를 통해 ADC 변환 완료 인터럽트를 초기화한다. 이 작업을 통해 다음 변환 완료 시에 새로운 인터럽트를 발생시킬 수 있다.

Todo #7 : main 작성 (LCD 값 출력 및 터치 좌표 읽기)

```
uint16_t value = 0;
uint16_t x, y;
uint16_t l_x, l_y;
```

- 전역변수 설정

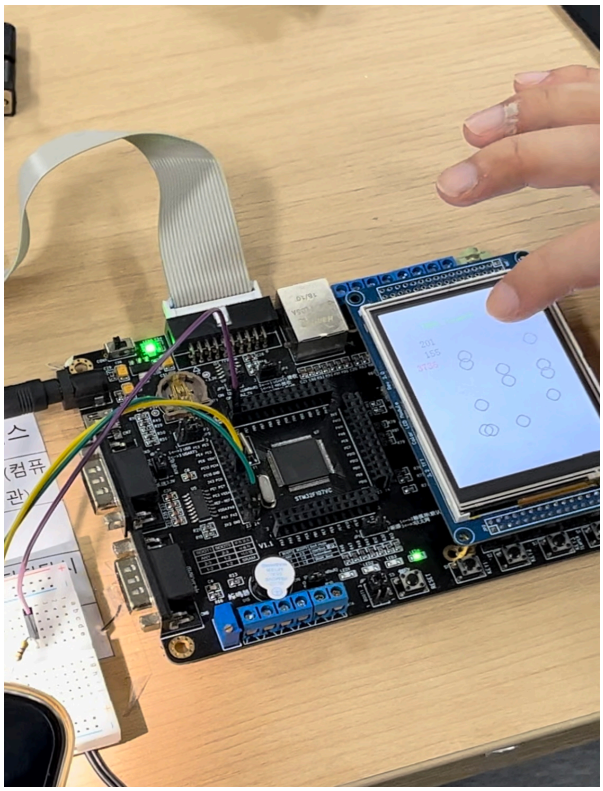
```
while (1) {
	// x=50, y=30 위치에 THUR_Team04 출력
	LCD_ShowString(50, 30, "THUR_Team04", GREEN, WHITE);

	// 터치 패널에서 터치된 좌표를 읽어와서 x, y에 저장
	Touch_GetXY(&x, &y, 1);

	// x, y에 저장된 값을 각각 l_x, l_y에 저장
	Convert_Pos(x, y, &l_x, &l_y);

	// x=40, y=60 위치에 x좌표 값을 출력
	LCD_ShowNum(40, 60, l_x, 3, BLACK, WHITE);
	// x=40, y=80 위치에 y좌표 값을 출력
	LCD_ShowNum(40, 80, l_y, 3, BLACK, WHITE);

	// x=20, y=100 위치에 ADC로 변환한 조도센서 값 출력
	LCD_ShowNum(20, 100, value, 4, RED, WHITE);

	// 눌렀던 좌표 위치에 10 크기의 원을 그린다
	LCD_DrawCircle(l_x, l_y, 10);
  }
```

- THUR_Team04를 출력한다

- 함수를 이용하여 터치한 위치에 원을 출력한다

- 터치한 위치 x, y와 조도센서의 값도 출력한다

실험 결과

	
THUR_Team04 출력 확인	휴대폰 손전등을 비추면 조도 값이 변화한다