REPORT

임베디드 실습 및 실험(002) 8주차 실험 결과보고서

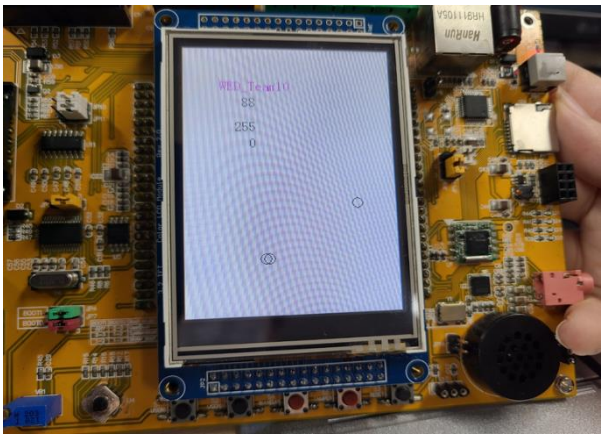
10조

전기컴퓨터공학부	201824446	김윤재
정보컴퓨터공학부	202055558	송세연
정보컴퓨터공학부	2020555889	임연후
바이오소재과학과	201845626	최이한

2022.11.08

◆ 실험 목표

1. TFT-LCD에 Text(팀명) 출력
2. ADC channel과 인터럽트를 사용하여 조도센서 값을 전역 변수에 저장
3. LCD 터치 시 해당 위치에 작은 원을 그리고, 좌표(X,Y) 및 전역 변수에 저장했던 조도센서 값을 출력



ADC란?

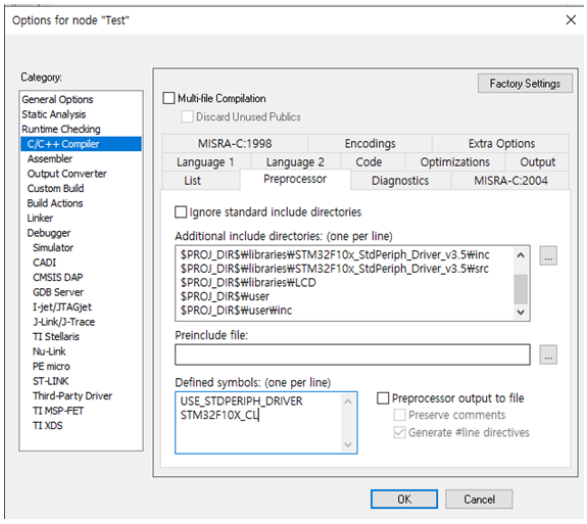
Analog to Digital Converter의 약자로, 아날로그 신호를 디지털 신호로 변환시켜주는 장치이다.

컴퓨터는 아날로그 신호를 감지하지 못하며, 따라서 온도, 습도, 조도와 같은 아날로그 값들을 전달해주기 위해선 ADC를 통해 디지털 신호로 변환후 전달해야한다.

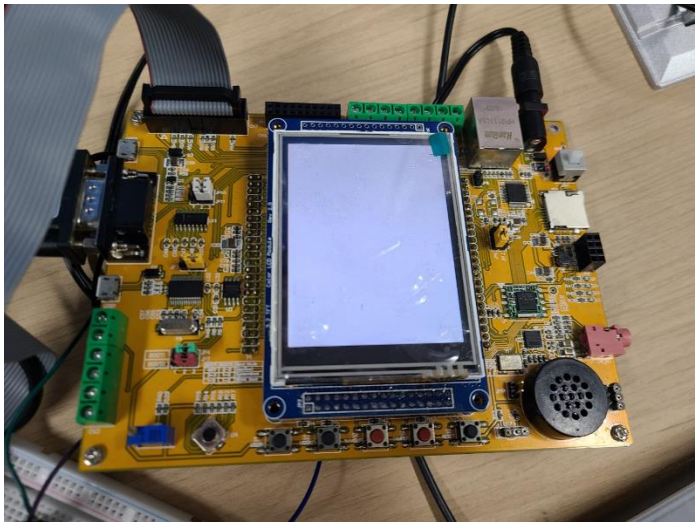
◆ 실험 과정

1. LCD 라이브러리 등록

- Libraries 폴더 밑에 LCD 폴더를 생성하고 제공받은 font.h, lcd.c, lcd.h, touch.c, touch.h 5개의 라이브러리 파일을 추가한다.
- 프로젝트 옵션>C/C++ Compilier>Preprocessor에서 생성한 LCD 라이브러리의 폴더 경로를 추가한다.

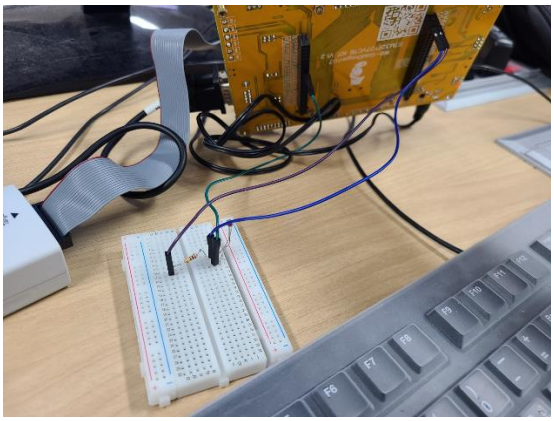


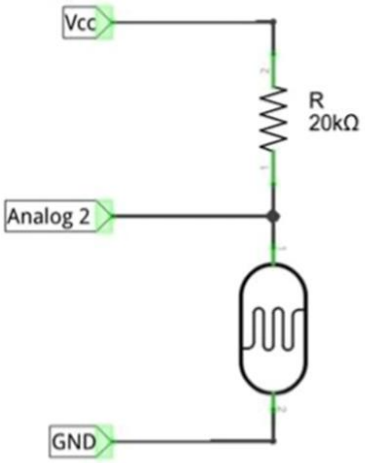
2. TFT-LCD 연결



보드에 TFT-LCD 모듈을 결착시킨다. 이때 오른쪽 한 칸은 비운다.

3. 회로구성





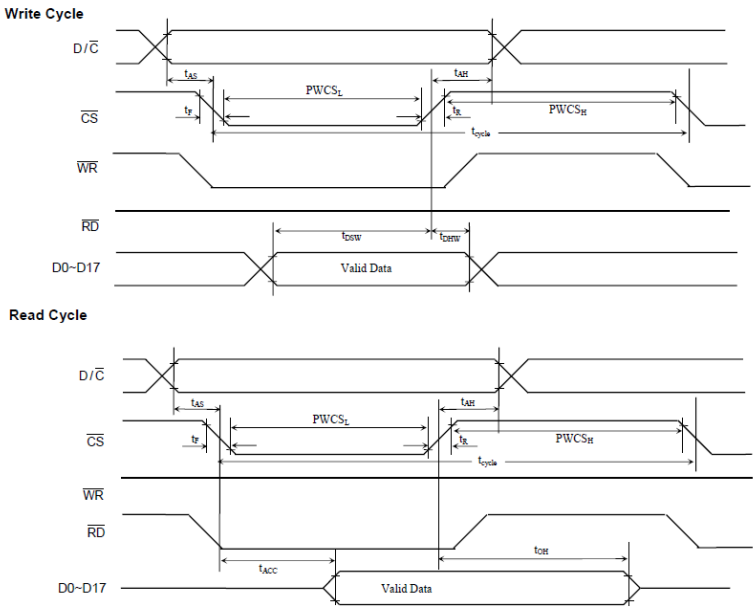
조도센서 회로도를 참고하여 회로를 구성한다. 조도센서가 저항의 역할 또한 해주기에 Vcc와 GND의 위치는 바뀌어도 상관없다.

4. lcd.c 코드 작성

1. 라이브러리 코드 완성

LCD 라이브러리 파일 중 lcd.c의 TODO 부분을 완성한다

LCD에 read,write하기 위해 SetBits와 ResetBits를 사용한다.



제공된 Timing diagram png 파일을 참고하여 LCD_CS, LCD_RS, LCD_WR 핀을 제어했다.

- CS: High 에서 Low로 Falling Edge 일 때 LCD Chip 을 사용

- DC(=RS): High 로 두고 Data를 전송, Low 로 두고 Command를 전송
- WR,RD: High 에서 Low로 Falling Edge 일 때, Data를 display에 Write / Read 한다

1. LCD_WR_REG는 Command register에 Command를 작성하는 함수이기에 DC, CS, WR를 low로 두고 command를 전송한 다음 CS,WR을 high로 바꿔주었다.

```
// TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOD, GPIO_Pin_15); //LCD_RD(1)
  GPIO_ResetBits(GPIOD, GPIO_Pin_13); //LCD_RS(0);
  GPIO_ResetBits(GPIOC, GPIO_Pin_6); //LCD_CS(0);
  GPIO_ResetBits(GPIOD, GPIO_Pin_14); //LCD_WR(0);
  GPIO_Write(GPIOE, LCD_Reg);

  // TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOC, GPIO_Pin_6); //LCD_CS(1);
  GPIO_SetBits(GPIOD, GPIO_Pin_14); //LCD_WR(1);
```

2. LCD_WR_DATA는 Data를 Display에 전송하는 함수이기에 DC를 high, CS, WR를 low로 두고 데이터를 전송한 다음 CS, WR을 high로 바꿔주었다.

```
// TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOD, GPIO_Pin_15); //LCD_RD(1)
  GPIO_SetBits(GPIOD, GPIO_Pin_13); //LCD_RS(1);
  GPIO_ResetBits(GPIOC, GPIO_Pin_6); //LCD_CS(0);
  GPIO_ResetBits(GPIOD, GPIO_Pin_14); //LCD_WR(0);

  GPIO_Write(GPIOE, LCD_Data);
  // TODO implement using GPIO_ResetBits/GPIO_SetBits
  GPIO_SetBits(GPIOC, GPIO_Pin_6); //LCD_CS(1);
  GPIO_SetBits(GPIOD, GPIO_Pin_14); //LCD_WR(1);
```

5. Main.c 코드 작성

```
void RCC_Configure(void)
{
   /* Alternate Function IO clock enable */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE);
      // ADC1, port C RCC ENABLE
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

조도센서의 아날로그 값을 디지털로 바꾸기 위한 ADC1과 조도센서를 연결할 portC에 RCC 전압을 인가하기 위해 RCC_APB1ENR 레지스터에 값을 setting한다.

7.3.4 APB2 peripheral reset register (RCC_APB2RSTR)

Address offset: 0x0C

Reset value: 0x00000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 RST	TIM10 RST	TIM9 RST	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 RST	USART1 RST	TIM8 RST	SPI1 RST	TIM1 RST	ADC2 RST	ADC1 RST	IOPG RST	IOPF RST	IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	Res.	AFIO RST
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	rw

```
void GPIO_Configure(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; // 아날로그 입력 설정
   GPIO_Init(GPIOC, &GPIO_InitStructure);
    // C0포트 활성화, 조도센서를 PC0에 연결할 계획
}
```

Input, output을 설정해준다. 이때 아날로그 입력을 사용하기에, GPIO_Mode를 GPIO_Mode_AIN으로 설정해준다.

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

```
typedef enum
{ GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;

#define IS_GPIO_MODE(MODE) (((MODE) == GPIO_Mode_AIN) || ((MODE) == GPIO_Mode_IN_FLOATING) || \
                            ((MODE) == GPIO_Mode_IPD) || ((MODE) == GPIO_Mode_IPU) || \
                            ((MODE) == GPIO_Mode_Out_OD) || ((MODE) == GPIO_Mode_Out_PP) || \
                            ((MODE) == GPIO_Mode_AF_OD) || ((MODE) == GPIO_Mode_AF_PP))
```

```
void ADC_Configure(void)
{
   // Analog to Digital Converter
   ADC_InitTypeDef ADC_InitStruct;

   ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
    // slave-master가 없는 독립 ADC
   ADC_InitStruct.ADC_ScanConvMode = DISABLE;
    // 단일채널이므로 비활성화
   ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
    // 한 번의 트리거로 한 채널의 샘플링 시행
   ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    // 외부 입력핀에 의한 트리거 비활성화
   ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
    // Default
   ADC_InitStruct.ADC_NbrOfChannel = 1;
    // 채널은 하나

   ADC_Init(ADC1, &ADC_InitStruct); // 위 설정을 ADC1에 적용

   ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_28Cycles5);
    // 채널 우선순위 설정, 10채널 단독사용

   ADC_Cmd(ADC1, ENABLE); // ADC1 활성화

   // Calibration reset & start
   ADC_ResetCalibration(ADC1);

   while (ADC_GetResetCalibrationStatus(ADC1))
      ;

   ADC_StartCalibration(ADC1);

   while (ADC_GetCalibrationStatus(ADC1))
      ;

   ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

ADC 설정을 초기화해준다.

```
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_

  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;
```

- ADC_Mode: 조도센서 하나만을 사용하므로 slave master관계가 없는 독립모드를 사용한다.
- ADV_ScanConvMode: 단일 채널을 사용하고 있으므로 비활성한다.
- ADC_continuousConvMode: 한 번의 트리거로 전체 채널을 샘플링 해야하므로 활성화한다.
- ADC_ExternalTrigConv: 외부입력을 사용하지 않으므로 비활성화 한다.
- ADC_DataAlign: 기본설정
- ADC_NbrOfChannel: 하나로 설정

```
void NVIC_Configure(void)
{

   NVIC_InitTypeDef NVIC_InitStructure;

   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

   NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn; // ADC IRQ 인터럽트 활성화
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
}
```

인터럽트를 사용해 ADC값을 읽어온다.

```
void ADC1_2_IRQHandler(void)
{
   // printf("Handle\n");
   if (ADC_GetITStatus(ADC1, ADC_IT_EOC) != RESET)
   {                                  // End Of Conversion, ADC변환이 끝났을때,
      value = ADC_GetConversionValue(ADC1); // value에 조도센서 값 입력
      ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
   }
}
```

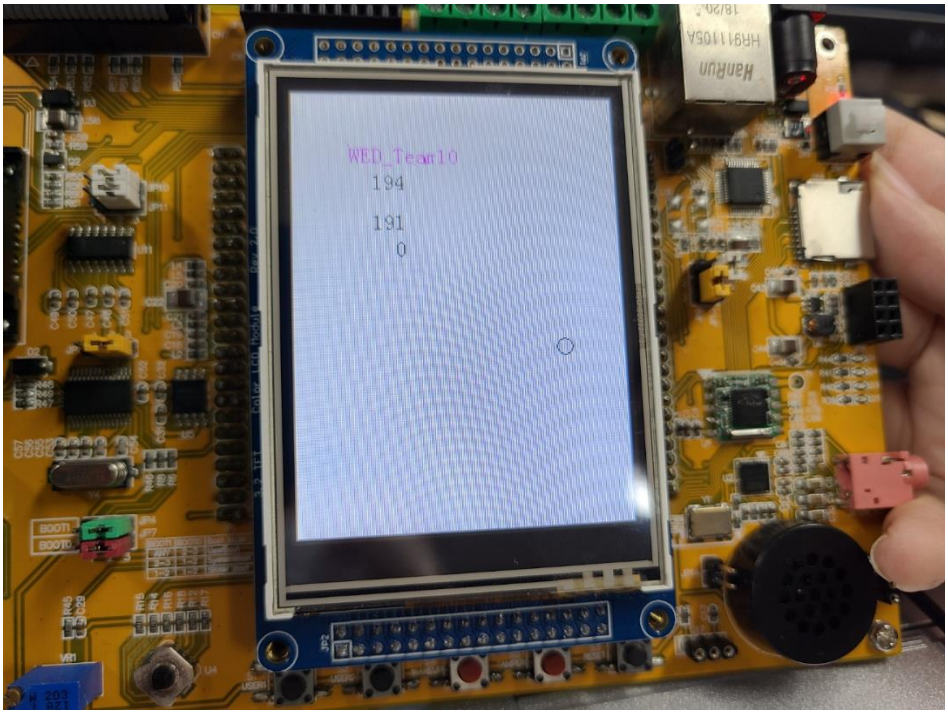
value에 조도센서의 값을 저장하고 ADC_ClearITPendingBit를 사용해 clear해준다.

```
/**
  * @brief  Returns the last ADCx conversion result data for regular channel.
  * @param  ADCx: where x can be 1, 2 or 3 to select the ADC peripheral.
  * @retval The Data conversion value.
  */
uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx)
{
  /* Check the parameters */
  //assert_param(IS_ADC_ALL_PERIPH(ADCx));
  /* Return the selected ADC conversion value */
  return (uint16_t) ADCx->DR;
}
```

```
/**
  * @brief  Clears the ADCx's interrupt pending bits.
  * @param  ADCx: where x can be 1, 2 or 3 to select the ADC peripheral.
  * @param  ADC_IT: specifies the ADC interrupt pending bit to clear.
  *   This parameter can be any combination of the following values:
  *     @arg ADC_IT_EOC: End of conversion interrupt mask
  *     @arg ADC_IT_AWD: Analog watchdog interrupt mask
  *     @arg ADC_IT_JEOC: End of injected conversion interrupt mask
  * @retval None
  */
void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, uint16_t ADC_IT)
{
  uint8_t itmask = 0;
  /* Check the parameters */
  //assert_param(IS_ADC_ALL_PERIPH(ADCx));
  //assert_param(IS_ADC_IT(ADC_IT));
  /* Get the ADC IT index */
  itmask = (uint8_t)(ADC_IT >> 8);
  /* Clear the selected ADC interrupt pending bits */
  ADCx->SR = ~(uint32_t)itmask;
}
```

```c
int main(void)
{

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

   //-----------------

   LCD_Init();            // LCD 초기화
   Touch_Configuration(); // 터치 설정
   Touch_Adjust();        // 화면 터치 초점 맞추기
   LCD_Clear(WHITE);      // LCD 배경 초기화

   //-----------------

   uint16_t pos_temp[2]; // x, y좌표를 담을 배열

   while (1)
   {
      LCD_ShowString(40, 40, "WED_Team10", MAGENTA, WHITE); // 팀명 출력

      Touch_GetXY(&pos_temp[0], &pos_temp[1], 1);
        // 터치 좌표 받아서 배열에 입력
      Convert_Pos(pos_temp[0], pos_temp[1], &pos_temp[0], &pos_temp[1]);
        // 받은 좌표를 LCD 크기에 맞게 변환
      Draw_Big_Point(pos_temp[0], pos_temp[1]);
        // 받은 좌표에 큰 원을 출력한다.
        //이때 touch.c의 Draw_Big_Point함수를 수정하여 이용했다.

      LCD_ShowNum(40, 60, (u32)pos_temp[0], 3, BLUE, WHITE); // x좌표 출력
      LCD_ShowNum(40, 80, (u32)pos_temp[1], 3, BLUE, WHITE); // y좌표 출력

      ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
        // ADC 변환이 끝났을때 인터럽트 발생
      LCD_ShowNum(40, 100, value, 4, BLUE, WHITE);
      ADC_ITConfig(ADC1, ADC_IT_EOC, DISABLE);
        // 무분별한 인터럽트 방지를 위해 비활성화
   }
   return 0;
}
```

◆ 실험 결과



TFT-LCD에 팀명과 터치한 위치의 좌표, 조도값이 출력됨을 확인할 수 있었다.

터치시 원이 그려진다.