REPORT

12주차 결과 보고서

수강과목: 임베디드 시스템 설계 및 실험

담당교수: 백윤주 교수님

학 과: 전기컴퓨터공학부 정보컴퓨터공학전공

조 원: 201724651 김장환 201724648 김경호

201624481 박윤형 201524588 조창흠

제출일자: 2021. 11. 25

1. 실험 목표

1.1 DMA 기능에 대한 이해 및 사용

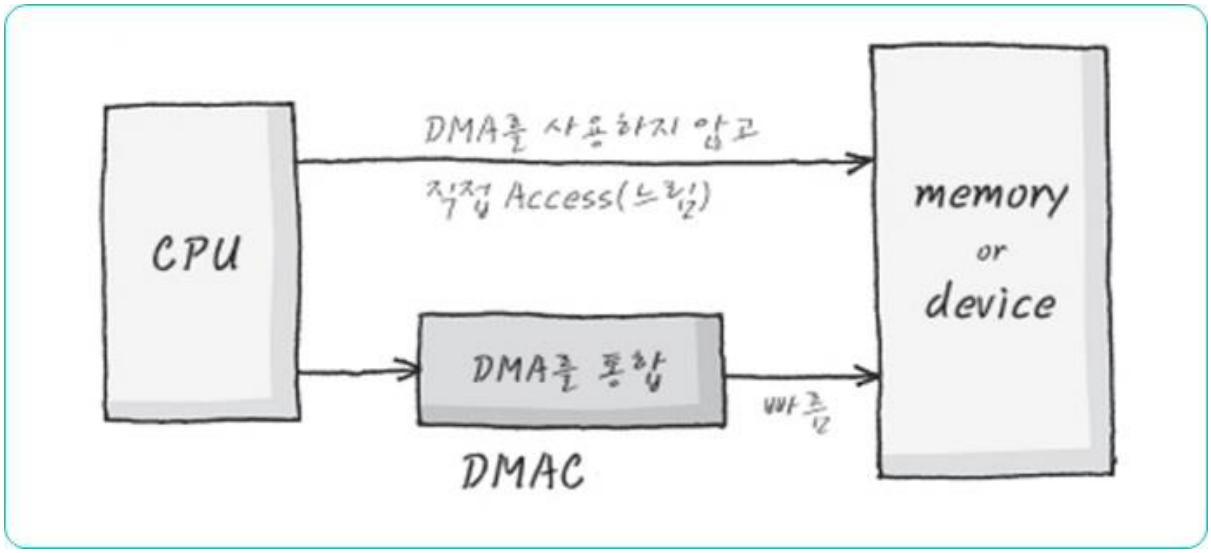
1.2 DMA를 사용하여 DMA Channel 10번, ADC channel 1번을 통해 조도센서 값을 받아오는 코드를 작성한다.

1.3 조도센서 값을 TFT-LCD에 표시한다.

1.4 센서 값에 적당한 기준을 둬서 기준치보다 밝아지면 LED 화면의 배경색에 변화를 준다.

2. 배경 지식

2.1 DMA (Direct Memory Access)



- CPU의 개입 없이 I/O 장치와 기억장치 간 직접적인 데이터 전송방식이다.
- CPU를 통해 I/O장치와 기억장치 간 데이터를 전송할 때는 인터럽트가 발생해서 대기시간이 발생한다.
- CPU를 통하지 않고 DMA Controller를 통해 데이터 교환을 하는 경우 CPU가 별도의 명령을 실행하지 않아서 대기 시간이 발생하지 않게 되어 속도 면에서 효율적이다.
- CPU는 DMA의 상태 정보 및 제어 정보만 주고 받는다.
- 모든 데이터 전송이 끝나면 DMA controller가 CPU에게 인터럽트 신호를 보낸다.

2.2 DMA Channel

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

Table 79. Summary of DMA2 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC3(1)					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO(1)				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1			TIM6_UP/ DAC_Channel 1		
TIM7				TIM7_UP/ DAC_Channel 2	
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

- DMA 레지스터의 종류는 DMA1, DMA2가 있고 각 레지스터의 채널을 통해 메모리 R/W

- STM32 보드 DMA 채널은 12개 -> DMA1 채널 7개, DMA2 채널 5개

- 한 DMA 레지스터의 여러 채널 사이 요청은 priority에 따라 동작(very high, high, medium, low)

- peripheral->memory, memory->peripheral, peripheral-> peripheral 전송

- peripheral에 맞는 channel들이 정해져 있어서 위의 표를 보고 사용할 channel을 정한다.

2.3 DMA Mode: DMA의 동작 모드

2.3.1 Normal Mode

- DMA controller는 데이터를 전송할 때마다 NDT(Number of Data to transfer) 값을 감소시킴

- NDT는 DMA를 통해 전송할 데이터의 총 용량을 의미하며 값이 0이 되면 전송 중단

- 데이터 전송을 받고 싶을 때마다 새롭게 요청이 필요

2.3.2 Circular Mode

- 연속적인 값(ex: 온도, 조도)들은 데이터 전송이 연속적으로 일어나므로 데이터 전송을 시도할 때 마다 요청을 해야 하는 Normal Mode는 적합하지 않다.

- NDT 값이 0이 될 경우 설정해준 데이터 최대 크기로 재설정된다.

2.4 DMA Controller

- I/O 장치가 CPU를 거치지 않고 직접 기억 장치로 접근하게 해주는 하드웨어 장치

- I/O 장치가 요청 신호를 보내면 메모리 접근을 허가하는 ACK를 보낸다.

- 여러 개의 I/O 장치에서 요청 신호가 들어올 경우 우선 순위를 정해서 실행

Figure 48. DMA block diagram in connectivity line devices

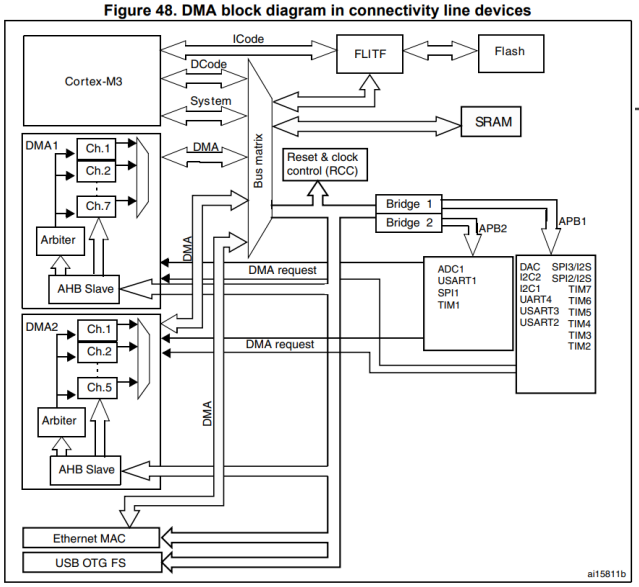
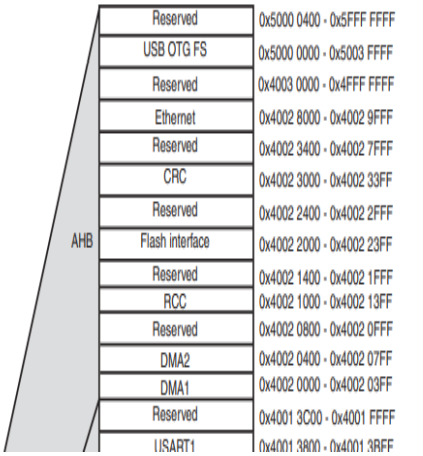
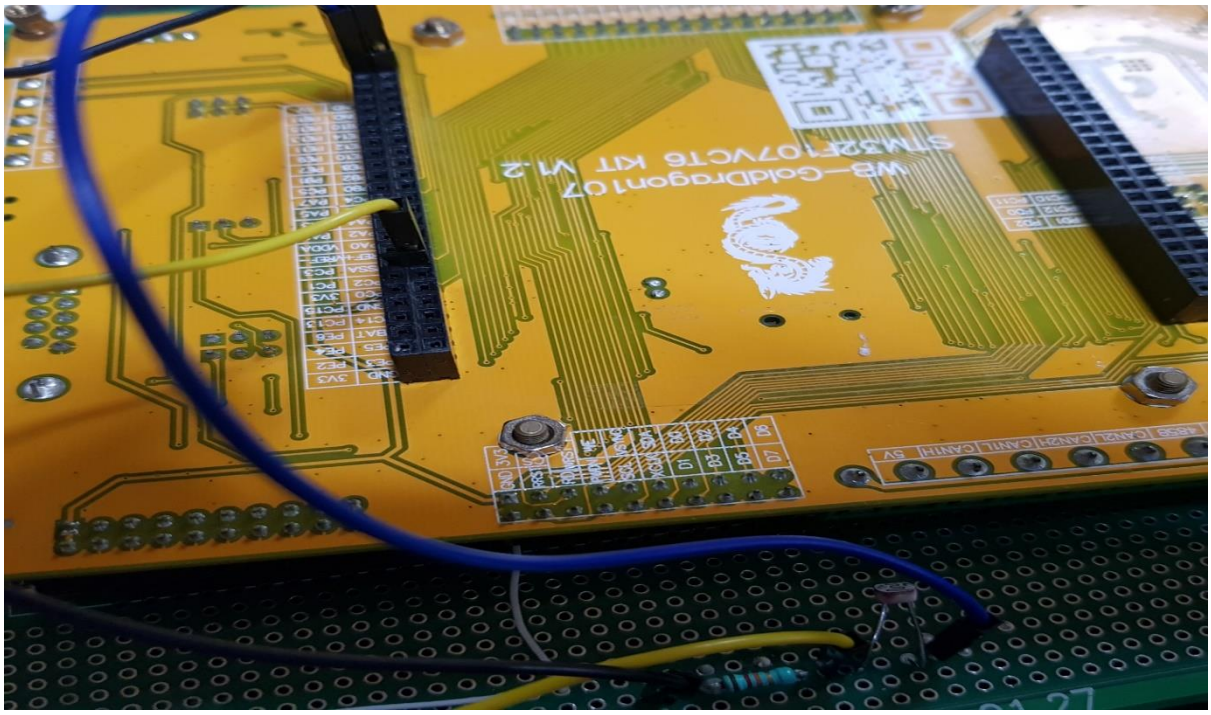


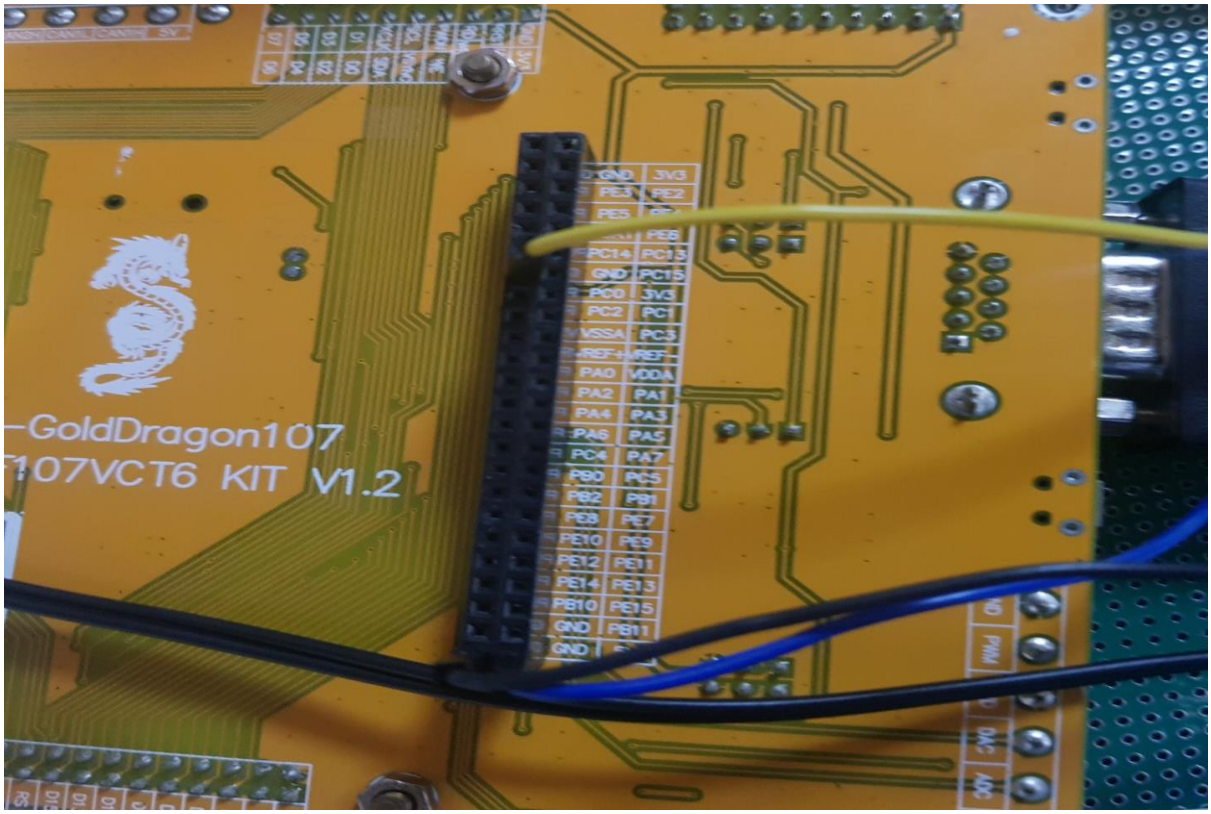
Figure 5. Memory map

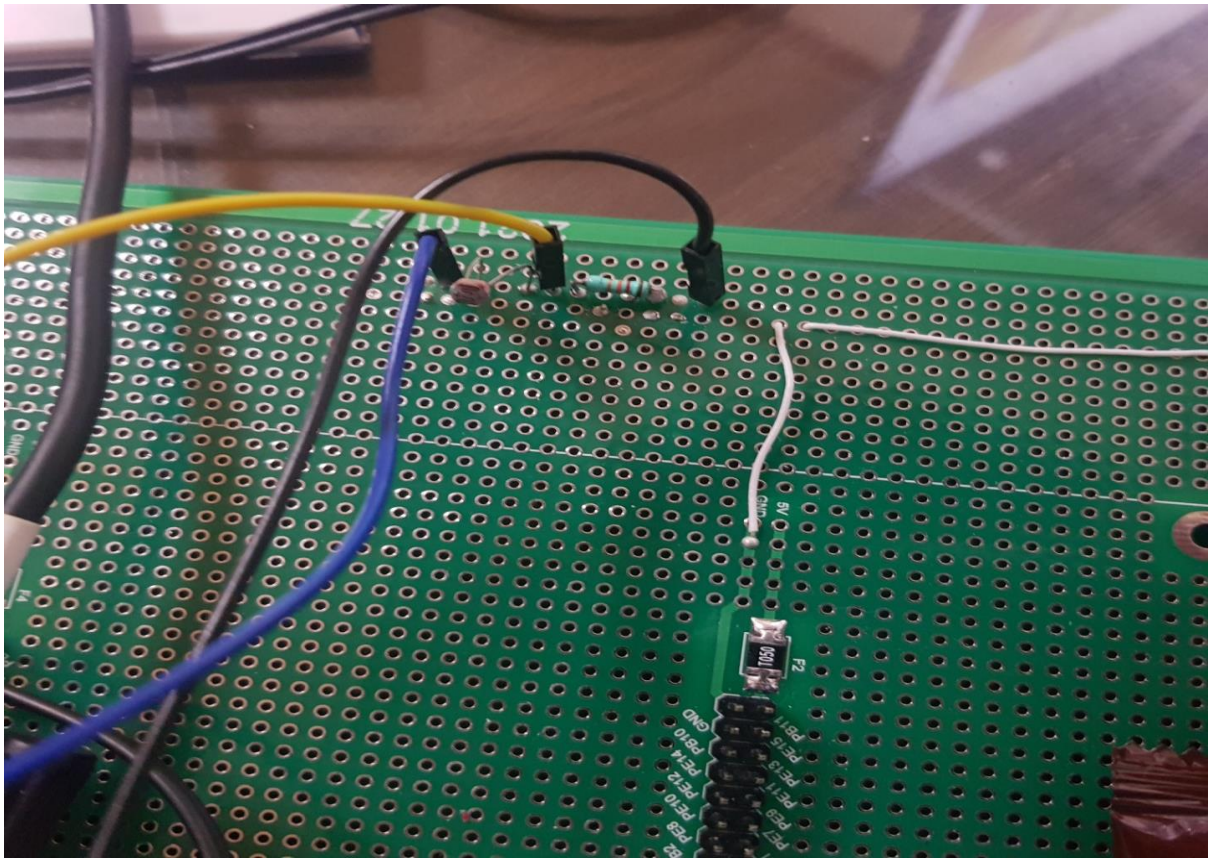


3. 실험 내용

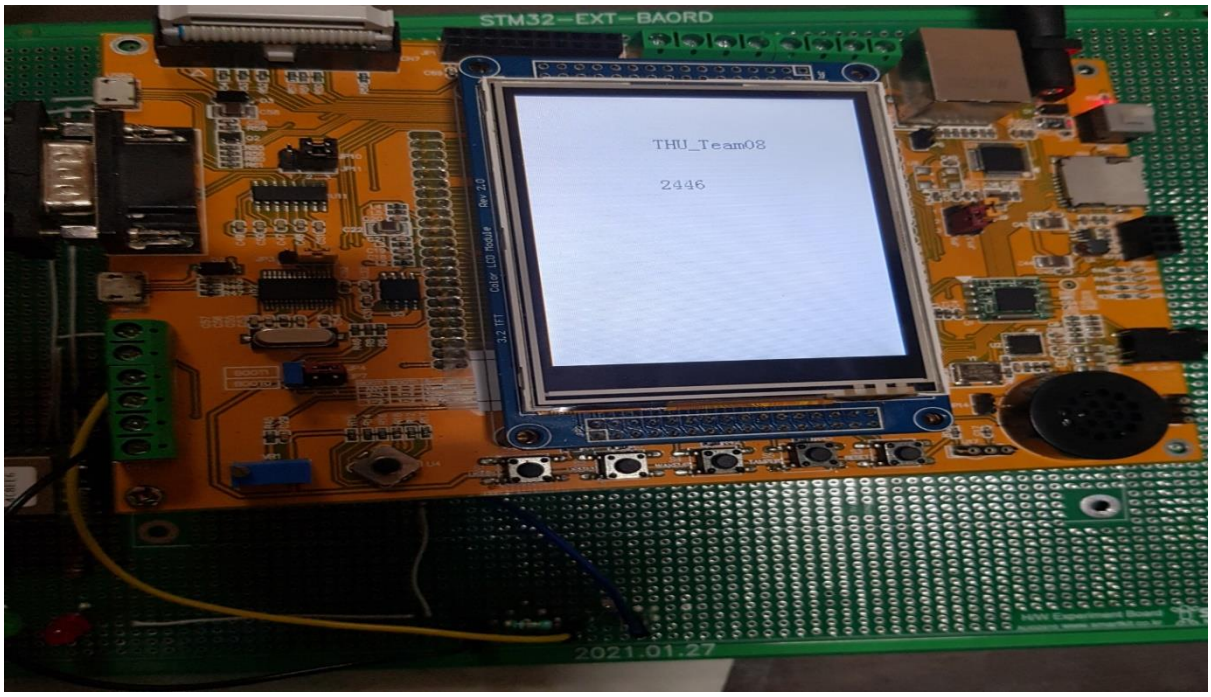
3.1 조도센서와 보드를 연결한다.



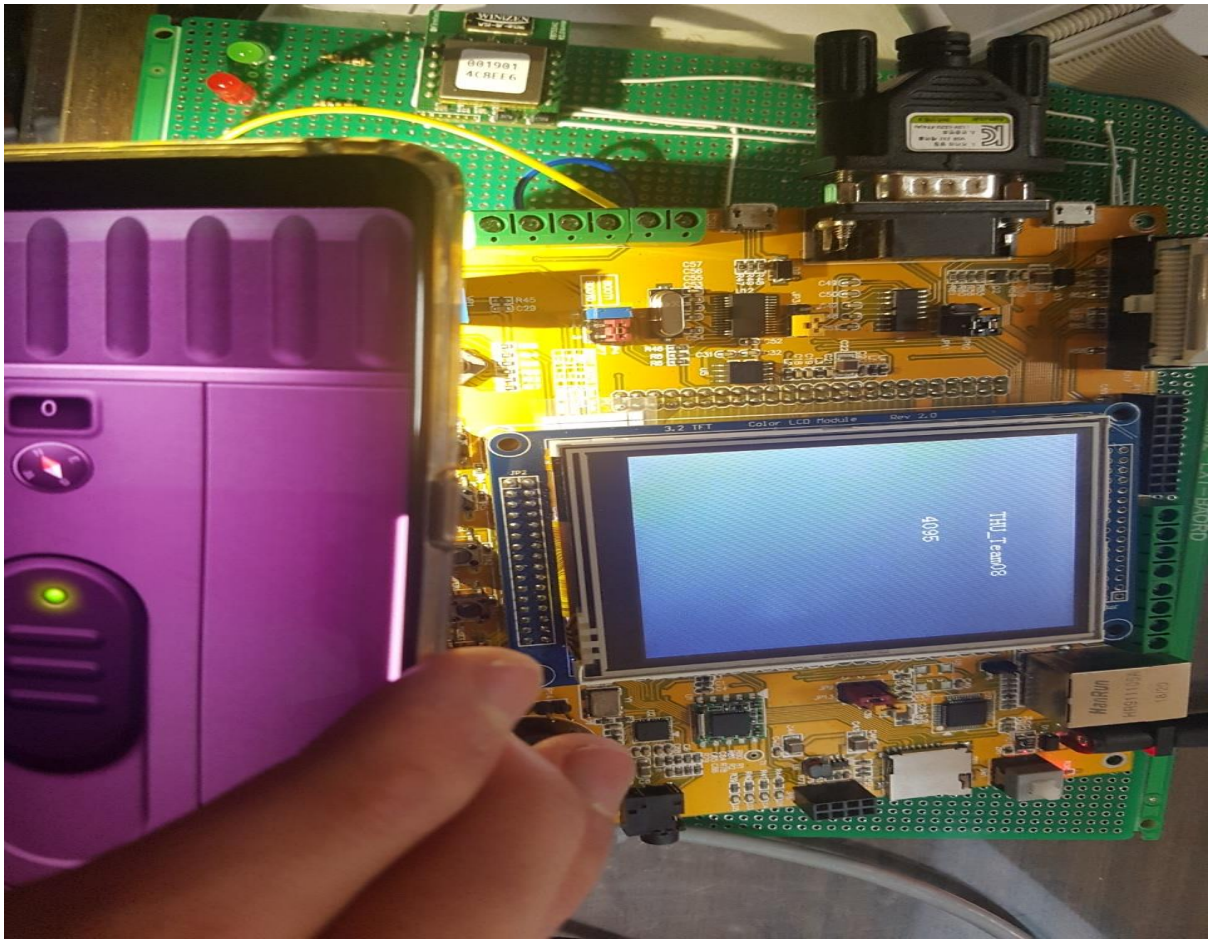




3.2 LCD를 보드에 끼우고 보드를 실행하면 LCD 화면에 팀 이름과 조도 값이 표시된다.



3.3 스마트 폰 불빛을 조도센서에 대주면 LCD 화면의 배경색이 바뀌고 팀 이름과 조도 값이 출력되는 텍스트의 색도 변한다.



4. 코드 설명

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_adc.h"
#include "stm32f10x_dma.h"

//include lcd, touch header files
#include "lcd.h"
#include "touch.h"

#define LCD_TEAM_NAME_X 40
#define LCD_TEAM_NAME_Y 50

void RCC_Configure(void);
void GPIO_Configure(void);
void ADC_Configure(void);
void DMA_Configure(void);
void Delay(void);

// Declare variable to receive ADC value input from illuminance sensor
volatile uint32_t ADC_Value = 0;
//-------------------------------------------------------------------------------------------------
```

조도 센서로부터 입력 받을 ADC 값을 전역 변수로 선언해준다.

```
24  void RCC_Configure(void)
25  {
26      /* DMA clock enable */
27      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
28
29      /* ADC clock enable */
30      RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
31      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
32      RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
33
34  }
```

사용할 DMA와 ADC에 clock을 인가해준다. 사용할 ADC는 ADC1이고 PC0을 사용하기 위해 GPIOC에 clock을 인가해주고 ADC1가 GPIOC는 APB2로 연결된다.

```
36  void GPIO_Configure(void)
37  {
38      GPIO_InitTypeDef GPIO_Struct;
39
40      /* ADC pin setting */
41      GPIO_Struct.GPIO_Pin = GPIO_Pin_0;
42      GPIO_Struct.GPIO_Mode = GPIO_Mode_AIN;                  // Input Analog value
43      GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
44      GPIO_Init(GPIOC, &GPIO_Struct);
45  }
```

ADC가 받는 값은 아날로그 값이기 때문에 PC0을 입력으로 사용하기 위해 GPIOC의 0번 핀을 analog input mode로 설정해준다.

```
47  void ADC_Configure(void) {
48      /* ADC1 channel 10 config */
49      ADC_InitTypeDef ADC;
50
51      ADC.ADC_Mode = ADC_Mode_Independent;
52      ADC.ADC_ContinuousConvMode = ENABLE;                    // continuity of data
53      ADC.ADC_DataAlign = ADC_DataAlign_Right;
54      ADC.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;   // ExternalTriger
55      ADC.ADC_NbrOfChannel = 1;                               // number of ADC channel
56      ADC.ADC_ScanConvMode = DISABLE;                         // Whether multichannel
57
58      ADC_Cmd(ADC1, ENABLE);
59      ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
60      ADC_Init(ADC1, &ADC);
61      ADC_DMACmd(ADC1, ENABLE);                               // Don't use ADC_ITConfig
62
63      ADC_ResetCalibration(ADC1);
64      while(ADC_GetResetCalibrationStatus(ADC1)) ;
65
66      ADC_StartCalibration(ADC1);
67      while(ADC_GetCalibrationStatus(ADC1)) ;
68
69      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
70  }
```

ADC 설정 값을 준다.

조도는 연속적인 값이기 때문에 ContinousConvMode를 enable 해주고 외부 트리거를 사용하지 않기 때문에 ExternalTrigConv는 None으로 해주고 ADC가 받는 값의 종류는 조도 하나 이기 때문에 채널의 수는 1개이다. ADC 스캔은 다중채널에서 설정하는 것이니 disable 해준다.

인터럽트를 쓰지 말고 DMA를 이용해야 하므로 ADC_ITConfig 함수 대신 ADC_DMACmd 함수를 써야 한다. 이후 Calibration을 Reset하고 Calibration을 시작하는 동작을 수행하도록 하였다.

```
72  void DMA_Configure(void) {
73      /* DMA1 channel 1 config */
74      DMA_InitTypeDef DMA_InitType;
75
76      DMA_DeInit(DMA1_Channel1);                                          // DMA1 DeInit
77      DMA_StructInit(&DMA_InitType);
78      DMA_InitType.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
79      DMA_InitType.DMA_MemoryBaseAddr = (uint32_t)&ADC_Value;
80      DMA_InitType.DMA_BufferSize = 1;                                    // Initialize memory buffer size
81      DMA_InitType.DMA_MemoryInc = DMA_MemoryInc_Disable;                 // Whether memory to memory
82      DMA_InitType.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;  // Initialize peripheral memory size
83      DMA_InitType.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;          // Initialize memory data size
84      DMA_InitType.DMA_Mode = DMA_Mode_Circular;                          // Initialize DMA Mode
85      DMA_InitType.DMA_Priority = DMA_Priority_High;                      // Initialize priority
86      DMA_Init(DMA1_Channel1, &DMA_InitType);
87      DMA_Cmd(DMA1_Channel1, ENABLE);
88  }
```

DMA 설정을 해준다.

DMA의 사용할 channel 값을 DeInit해준다.

PeripheralBaseAddr은 DMA를 사용할 peripheral과 memory간의 변수 address이고 어떤 I/O 장치의 정보를 받을지 메모리 주소를 설정해주는 것으로 ADC1을 통해 조도를 받으므로 ADC1의 adress값을 DR에 더해준다.

MemoryBaseAddr은 변수를 통해 실제로 저장될 memory adress이고 ADC_Value를 값으로 주면 조도센서의 data가 update 된다.

BufferSize는 변수에 저장할 데이터의 개수이고 변수가 조도만 있으니까 1로 설정해준다.

메모리 간의 전송은 일어나지 않으므로 Memoryinc는 활성화하지 않는다.

조도는 연속적인 값이므로 DMA mode를 normal이 아닌 circular로 설정해준다.

DMA이 우선순위를 High로 설정해주어 즉각적으로 ADC의 값을 받아올 수 있도록 한다.

```c
int main(void) {

    SystemInit();
    RCC_Configure();
    GPIO_Configure();
    ADC_Configure();
    DMA_Configure();


    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();
    LCD_Clear(GRAY);

    int flag = 0;
    LCD_ShowString(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y, "THU_Team08", WHITE, GRAY);
    LCD_ShowNum(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y+50, ADC_Value, 4, WHITE, GRAY);

    while (1) {
       if(ADC_Value >= 4050){
         if(flag == 1){
           LCD_Clear(GRAY);
           flag = 0;
         }
             LCD_ShowString(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y, "THU_Team08", WHITE, GRAY);
             LCD_ShowNum(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y+50, ADC_Value, 4, WHITE, GRAY);
       }
       else{
         if(flag == 0){
           LCD_Clear(WHITE);
           flag = 1;
         }
         LCD_ShowString(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y, "THU_Team08", GRAY, WHITE);
         LCD_ShowNum(LCD_TEAM_NAME_X+50, LCD_TEAM_NAME_Y+50, ADC_Value, 4, GRAY, WHITE);
       }

       // TODO: implement
       // Delay
       Delay();
    }
    return 0;
}
```

main문

인터럽트를 쓰지 않기 때문에 Handler 호출 함수를 쓰지 않는다.

TFT LCD의 배경을 변경하기 위해 flag를 선언한다.

LCD_ShowString() 함수로 TFT LCD화면의 (90,50) 좌표에 "Tue_team 08"을 회색 배경에 흰색 텍스트로 출력한다.

LCD_ShowNum() 함수로 TFT LCD화면의 (90,100) 좌표에 조도 값을 회색 배경에 흰색 텍스트로 출력한다.

일반적으로 대략 3600 정도의 조도 값이 측정되는데 플래시가 켜지면 4095로 증가하고 최대 값이 4096임을 알 수 있다.

4096 = 4M이기 때문에 최대 범위를 초과하지 않는 것으로 간주할 수 있다.

따라서 기준치를 4050으로 설정하고 조도센서에 측정되는 조도 값이 기준치에 도달하면 LCD 화면의 배경이 회색으로 변경되고 flag값이 0으로 변경되고 이 경우 팀 이름과 조도 값은 흰색 텍스트로 출력된다.

반대로 조도센서에 측정되는 조도 값이 기준보다 낮아질 경우 LCD 배경이 흰색으로 바뀌고 flag 값이 1로 변경된다. 이 경우 팀 이름과 조도 값은 지정된 위치에서 회색 텍스트로 출력된다.

5. 결론

이번 실험은 DMA를 사용하여 DMA Channel 1번, ADC1 Channel 10번을 통해 조도센서 값을 받아오는 방법에 대해 알아보았다. CPU를 이용하지 않고 메모리에 direct로 접근해 성능 감소를 줄여 효율적을 사용할 수 있게 되었다.

DMA에서 필요하거나 불필요한 channel이나 기능을 enable, disable하고 우선순위를 직접 설정할 수 있도록 하였으며 이를 이용해 조도센서 값을 받아와 센서 값을 TFT-LCD 화면에 표시 하였다.

또한 받아오는 조도 값에 기준치를 둬서 밝다고 생각하면 TFT-LCD 화면의 배경색에 변화를 주고 조도 값이 출력되는 텍스트 색에도 변화를 주는 실험을 통해 다른 센서를 사용하게 될 때도 DMA를 이용해 효율적으로 data 전송을 처리할 수 있을 것이다.