
# 임베디드 시스템 설계 및 실험 11주차 보고서

**002분반 3조**

201724601 최성렬

201729163 이희근

202055522 김은지

202055574 이다은

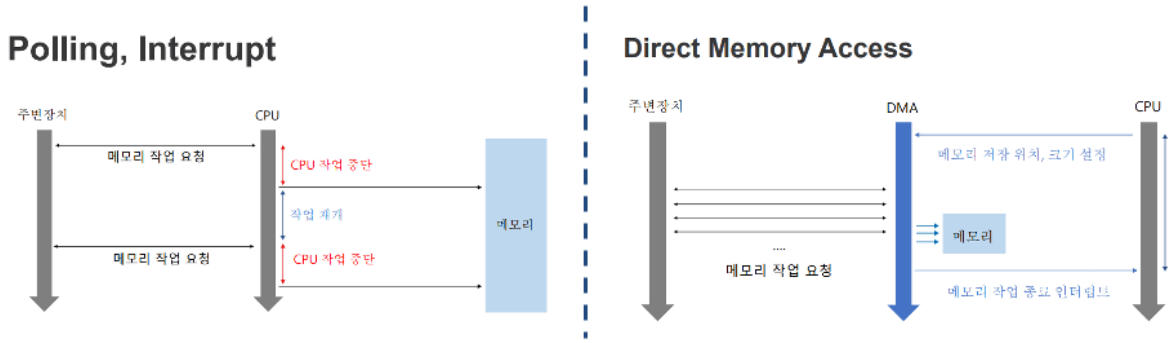
202055590 장서윤

# 실험 목표

1. DMA와 ADC를 사용하여 조도센서의 값을 읽어와 TFT-LCD에 출력한다.
2. 조도센서의 값에 따라 LCD의 배경색을 바꾼다.

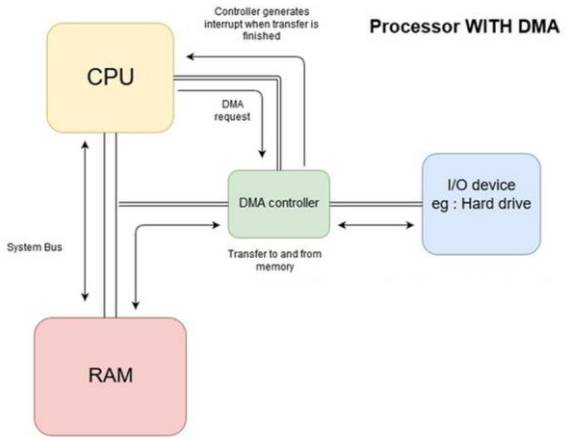
# 이론

### 1) 일반적인 방식과 DMA방식의 차이점




모든 I/O와 Data가 CPU를 통해야 하는 일반적인 메모리 접근 방식과 다르게 DMA방식은 주변 장치들이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능이다.

### 2) DMA방식의 작동 구조




-RAM이 I/O 장치로부터 데이터가 필요해지면 CPU는 DMA 컨트롤러에게 신호를 보냄

-DMA 컨트롤러가 RAM 주소로 데이터를 bus를 통해 주고받음

-모든 데이터 전송이 끝나면 DMA Controller가 CPU에게 Interrupt 신호를 보냄

### 3) DMA Channel

모듈은 DMA Controller의 DMA 채널을 통해 메모리를 읽고 쓴다.

STM32 보드의 DMA 채널은 총 12개로 DMA1 채널 7개, DMA2 채널 5개를 가지고 있다.

하나의 DMA의 여러 채널의 요청들은 Priority에 따라 동작한다.

각 DMA1의 채널과 사용처는 Reference Manual에서 찾아볼 수 있으며 표는 아래 그림과 같다.

**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<br>TIM1_TRIG<br>TIM1_COM | TIM1_UP | TIM1_CH3 | |
| TIM2 | TIM2_CH3 | TIM2_UP | - | - | TIM2_CH1 | - | TIM2_CH2<br>TIM2_CH4 |
| TIM3 | - | TIM3_CH3 | TIM3_CH4<br>TIM3_UP | - | - | TIM3_CH1<br>TIM3_TRIG | - |
| TIM4 | TIM4_CH1 | - | - | TIM4_CH2 | TIM4_CH3 | - | TIM4_UP |

그림을 보면 DMA1의 Channel1이 ADC1의 data를 받아올 수 있음을 알 수 있다. 따라서 이번 실험에서 사용해야할 DMA는 DMA1의 Channel1임을 알 수 있다.

### 4) DMA mode

DMA Mode는 2가지가 있다.

- Normal Mode

  -데이터를 전송할 때마다 NDT값이 감소함

  -NDT는 DMA를 통해 전송할 데이터의 총 용량을 뜻하며 0이되면 데이터 전송을 중단함.

  -데이터 전송을 받고 싶을 때 마다 새롭게 요청이 필요

- Circular Mode

  -주기적인 값의 전송(업데이트)이 필요할 때 사용하는 모드

  -NDT값이 0이될 경우 설정한 데이터 최대 크기로 재설정 됨

이번 실험에서는 조도센서의 값을 주기적으로 받아와야 하므로 Circular Mode를 사용하였다.

# 실험 과정

## Main.c 코드

1. RCC_Configure

ADC, DMA 사용을 위해 각각의 clock을 활성화시킨다.

```
void RCC_Configure(void)
{
    /* Alternate Function IO clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

    /* ADC */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

    /* DMA */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}
```

2. ADC_Configure

Interrupt가 아닌 DMA를 이용하여 작성해야 하므로 이전에 사용했던 ADC_ITConfig 함수 대신 ADC_DMACmd 함수를 사용하였다.

```
void ADC_Configure(void) {
    ADC_InitTypeDef ADC_InitStructure;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_NbrOfChannel = 1;

    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
    // ADC_ITConfig -> ADC_DMACmd
    ADC_DMACmd(ADC1, ENABLE);
    ADC_Cmd(ADC1, ENABLE);
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

void DMA_Configure(void) {
```

3. DMA_Configure

- DMA_InitStructure.DMA_PeripheralBaseAddr에서는 우리가 값을 받아야할 주소를 정해준다. ADC1의 값을 받아와야 하므로 ADC1->DR로 설정해주었다.
- DMA_InitStructure.DMA_MemoryBaseAddr 에서는 DMA가 가져온 값을 저장할 메모리의 주소를 정한다. ADC값을 전역변수로 사용하여 항상 참조할 수 있도록 미리 선언하였고 이 주소를 그대로 사용하였다.
- DMA_InitStructure.DMA_BufferSize에서는 조도센서 값 1개만 들어가기 때문에 ADC채널을 1개 사용한다. 따라서 1로 설정하였다.
- DMA_InitStructure.DMA_PeripheralDataSize, DMA_InitStructure.DMA_MemoryDataSize는 Peripheral data 와 Memory data의 크기를 설정한다. 여기서는 word단위 즉 32bit로 설정하였다.
- DMA_InitStructure.DMA_Mode는 조도센서 값의 주기적인 업데이트를 위해 Circular로 설

정하였다.

- DMA_InitStructure.DMA_Priority는 채널사이의 우선순위를 정해준다. 이번 실험에서는 채널을 1개만 사용하기 때문에 큰 상관은 없다.

```
// ADC값 저장
volatile uint32_t ADC_Value[1];
```

```
void DMA_Configure(void) {

    DMA_InitTypeDef DMA_InitStructure;
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) &ADC1->DR;  //ADC 어디서 불러올지
    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) &ADC_Value; // 저장할 주소(버퍼)
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC; // Read from peripheral
    DMA_InitStructure.DMA_BufferSize = 1; // 조도센서 값 한개
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; // 주변 레지스터의 추가 안 됨

    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;//여기를 Enable해야 메모리 주소르 증가시키면서 다음메모리에 정보 씀
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word; // 32bit
    DMA_InitStructure.DMA_MemoryDataSize =  DMA_MemoryDataSize_Word; // 32bit
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;//정해진 크기의 메모리에 데이터를 쓰면 처음으로 돌아와서 씀
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;//전송할 data의 수가 0이 되면 stream이 disable

    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
    DMA_Cmd(DMA1_Channel1, ENABLE);

}
```

4. GPIO_Configure

조도센서를 C0핀에 연결하였기 때문에 C0의 input mode를 설정하였다.

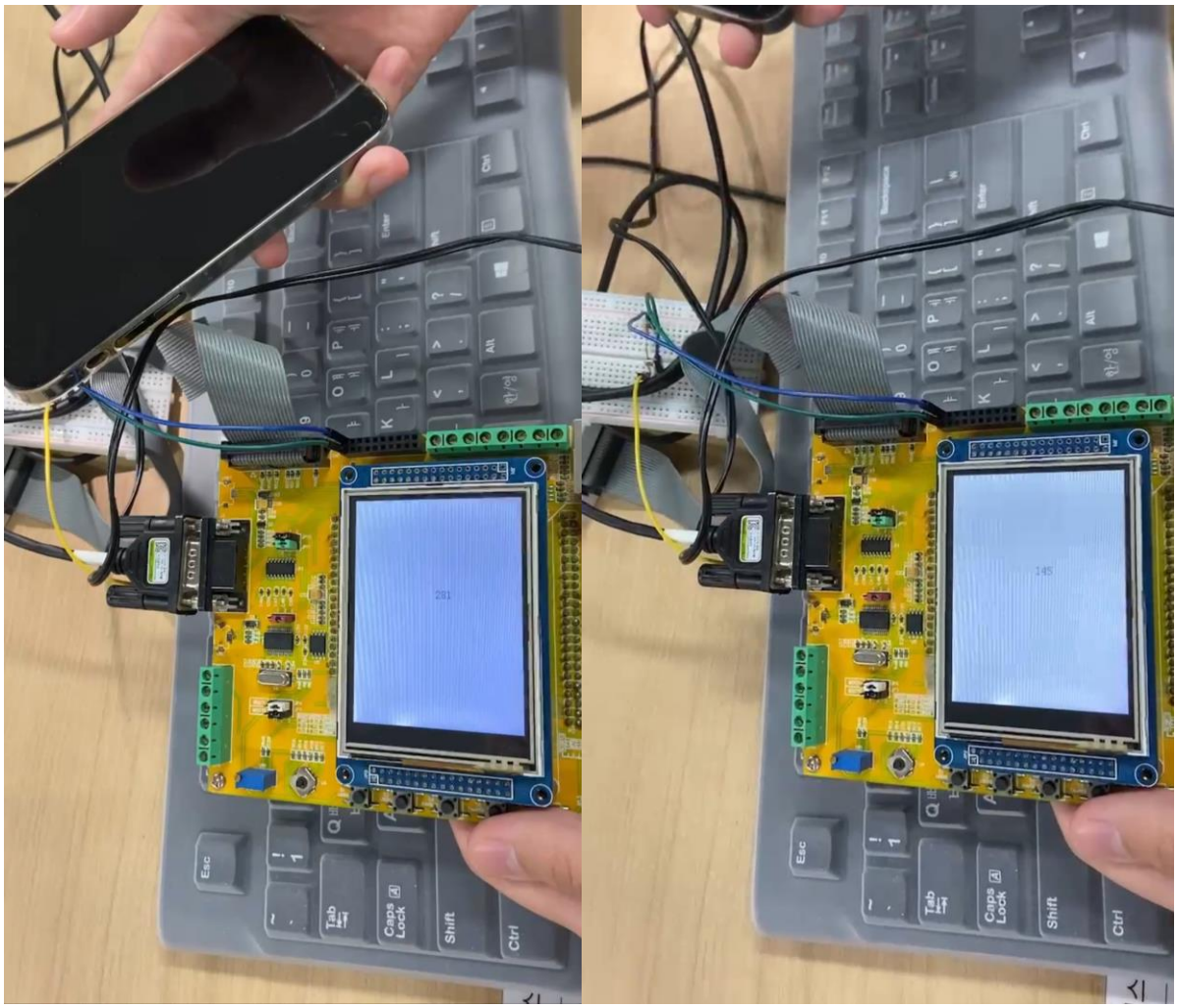
```
void GPIO_Configure(void) // 조도센서 C0 input mode 설정
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

5. main

조도센서의 값에 따라 LCD 화면의 색을 흰색 또는 회색으로 바꿔야한다. 따라서 if문을 사용해 ADC_Value의 값에 따라 LCD 화면의 색과 글자색을 바꿔주었다.

```
int main(void)
{

    SystemInit();
    RCC_Configure();
    GPIO_Configure();

    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();
    LCD_Clear(WHITE);
    ADC_Configure();
    DMA_Configure(); // DMA


    while(1) {
        if(  ADC_Value[0] < 200 ){ // 플래시 비출때
            // 화면 회색
            LCD_Clear(GRAY); //배경 바꾸면 글자색도 사라짐 -> 업데이트
            LCD_ShowNum(0x10, 0x80, ADC_Value[0], 16, BLACK, GRAY);
        }
        else { //플래시 안 비출때
            // 화면 흰색
            LCD_Clear(WHITE);
            LCD_ShowNum(0x10, 0x80, ADC_Value[0], 16, BLACK, WHITE);
        }


    }
    return 0;
}
```

## 실험 결과



플래시를 비출 때 LCD의 배경이 회색, 비추지 않을 때 흰색으로 바뀌는 것을 확인 할 수 있었다.

## 고찰

처음에는 LCD배경의 색만 바꿔 글자 배경색과 이질감이 드는 현상이 있었다. 따라서 직접 LCD_ShowNum의 배경색을 변경해주었다. 글자 배경색을 LCD배경과 자동으로 같게 만들어 주는 변수가 있는지 찾아보았지만 다른 정보는 찾을 수 없었다.

LCD의 화면이 계속해서 깜박이는 현상이 있다. 아마도 main의 while문 안에서 LCD_Clear 함수가 계속해서 실행되기 때문이라고 추측된다. 따라서 flag와 같은 변수를 이용하여 LCD_Clear 함수가 한 번만 실행되도록 바꾼다면 가시성을 개선가능 할 수 있을 것이라 예상된다.