임베디드 시스템 설계 및 실험

12주차

7조
201424470 서민영
201424421 김시은
201424533 정종진
201424532 정재광

목차

1. 실험목표 ····· 3
2. 배경지식 ····· 3 - 5
3. 실험과정 ····· 5 - 10
4. 소스코드 및 실험결과 ····· 10 - 14
5. 결론 및 느낀점 ····· 14

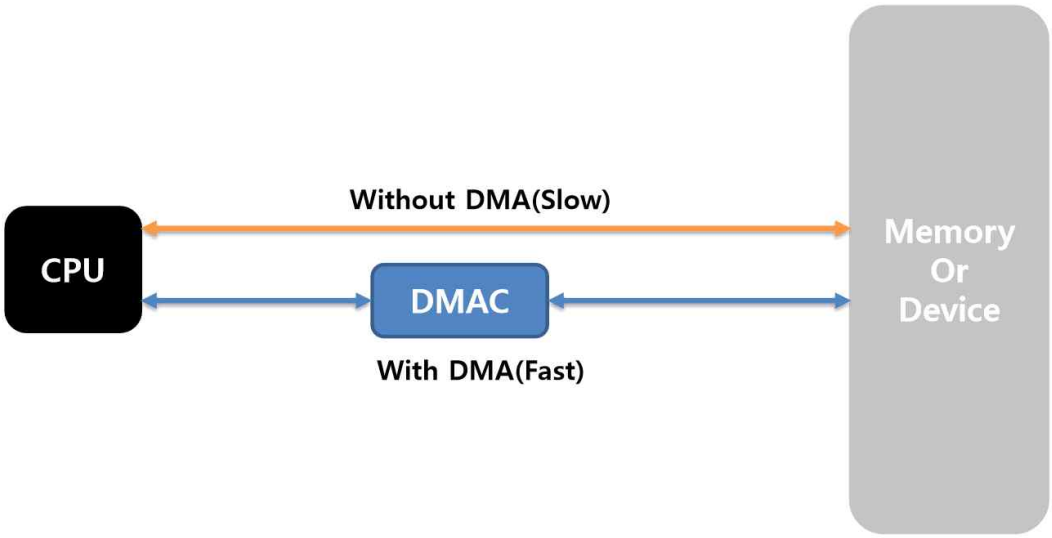
1. .실험목표

1) DMA 구현

2. 배경지식

1) DMA

- CPU 개입 없이, I/O device와 memory device 사이에 data 전송을 가능하게 해주는 것으로 속도가 빠르다.
- Program 수행 I/O를 위한 interrupt를 최소화하여 컴퓨터 효율을 높인다. CPU와 별개로 분리되어 처리하며 disk, printer, tape-drive 등에 이용된다.
- DAM controller는 system bus의 권한을 얻어 원하는 data를 저장하고, bus의 권한을 반환 후, 개별적으로 처리한다. 또, CPU는 DMA의 상태 정보 및 제어 정보만 주고 받는다.
- Interrupt 호출이 많은 기능을 이용할 때, DMA 방식을 사용하는 것이 더 효율적이다.
- DMA와 Interrupt를 혼합해서 사용 가능하다.

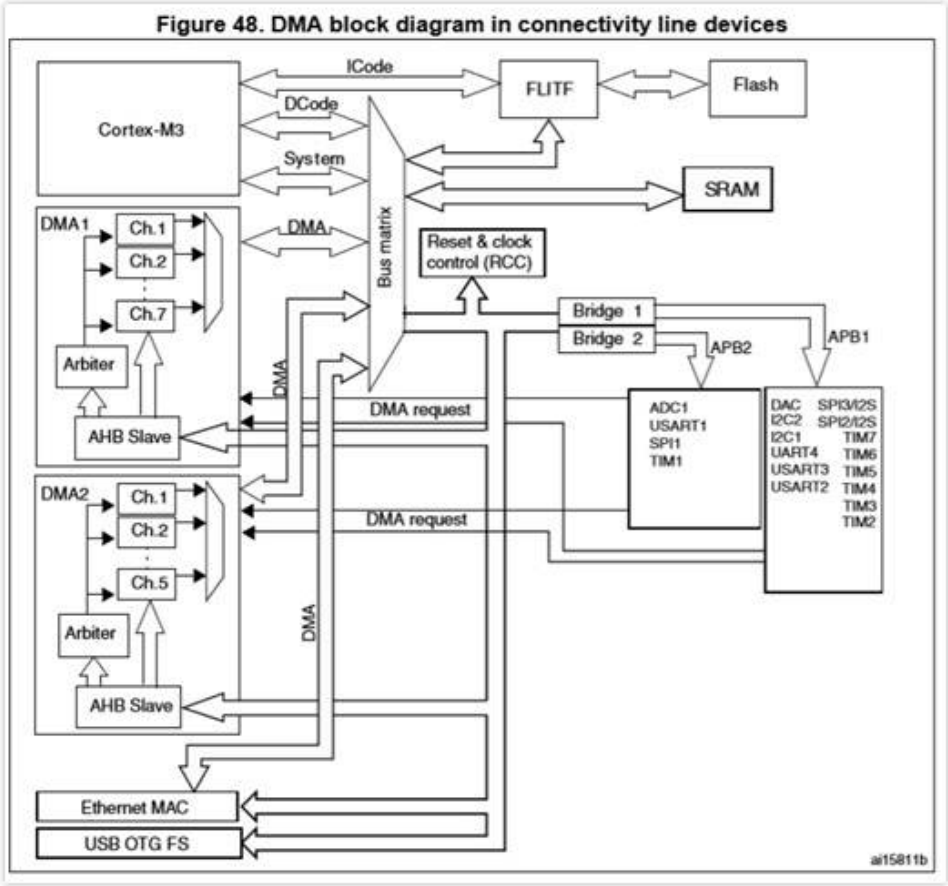


<그림1 : DMA 동작 방식>

2) DMA 동작 방식

- CPU가 DMA controller를 초기화. (memory 시작주소, 크기, I/O device 번호, I/O 선택 등)
- I/O device가 DMA를 요청
- DMA controller가 bus를 CPU에게 요청
- CPU가 bus 승락(Grant)
- DMA controller가 DMA 승락
- I/O device와 memory 사이의 자료전송
- DMA controller가 DMA 완료 interrupt를 CPU에게 보냄

3) DMA Block Diagram



<그림2 : DMA block diagram>

다음은 DMA Block diagram으로 원하는 기능의 bus와 channel을 선택하여 DMA를 구현할 수 있다.

- DMA controller : system bus를 Cortex-M3와 공유하여 직접 memory 전송을 수행한다.
- DMA request : CPU 및 DMA가 동일한 memory나 peripheral를 대상으로 할 때, 일부 bus cycle동안 system bus에 대한 CPU access를 중지할 수 있다.
- Bus Matrix는 Round-robin scheduling을 구현하므로 CPU의 System bus 대역폭의 절반 이상을 보장한다.

4) DMA Channel

- 7개의 DMA1과 5개의 DMA2 channel로 구성되어 있다. 각 channel은 고정 주소에 대한 peripheral register와 memory address 사이의 DMA 전송을 처리한다.
- 동시에 하나의 channel/request만 동작한다. Data 크기는 programming하고 pointer를 증가시킨다. 다음 전송 address는 선택한 data 크기에 따라 이전 address에서 1,2 또는 4씩 증가한다.

5) DMA Mode

- Circular mode : 순환 buffer 및 연속 data의 흐름에 대한 handling이 가능하다.
- Normal mode : 전송할 data의 수가 0이 되면 stream이 disable 된다.

- 추가적으로 data 전송에 필요한 정보들은 뒤에 나오는 register 값으로 set 해주면 된다.

3. 실험과정

1) 보드 연결

<그림3 : Coretex M3/JTAG/DSTREAM을 연결한 모습>

1-1) Coretex M3/JTAG/DSTREAM 연결

다음과 같은 순서로 보드를 연결한다. 이 때 연결 및 분리 순서를 제대로 지키지 않으면 장비가 망가질 수 있으니 주의해야한다.

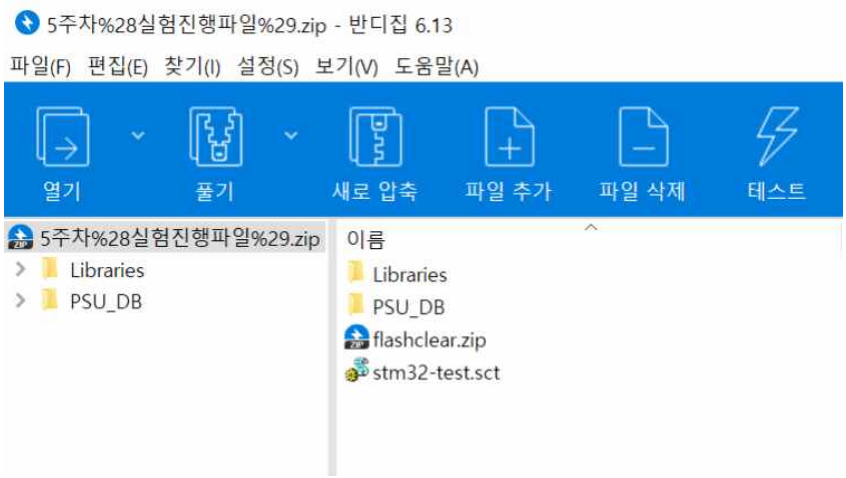
① 보드와 DSTREAM JTAG 연결
② 보드 전원선만 연결
(보드의 전원은 OFF 상태)
③ DSTREAM 전원 연결 및 ON
④ DSTREAM Status LED 점등 확인
⑤ 보드 전원 ON
⑥ DSTREAM Target LED 점등 확인
⑦ DS-5에서 'connect target'

<표 1 : 보드 연결 순서>

2) DS-5 디바이스 데이터베이스 추가

2-1) 수업게시판에서 실습파일로 제공되는 PSU_DB, LIBRARIES, SCATTER FILE을 다운

2-2) 이번 실험에서 include하는 코드의 양이 커서 기존에 scatter file에 설정된 값을 사용하면 load되지 않기 때문에 Data Sheet를 참고하여 scatter file의 메모리 범위를 수정해야 한다.



<그림4 : 실습파일로 제공된 파일들>

```
LR_IROM1 0x08000000 0x00008000 {   : load region size_region
  ER_IROM1 0x08000000 0x00008000 {  : load address = execution address
   *.o(RESET, +First)
   *(InRoot$$Sections)  Data Sheet를 참고하여 0x00080000 으로 수정함
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00008000 {  : RW data
   .ANY (+RW +ZI)
  }
}
```

<그림5 : 스캐터 파일 수정>

2-3) Eclipse에 데이터베이스 추가

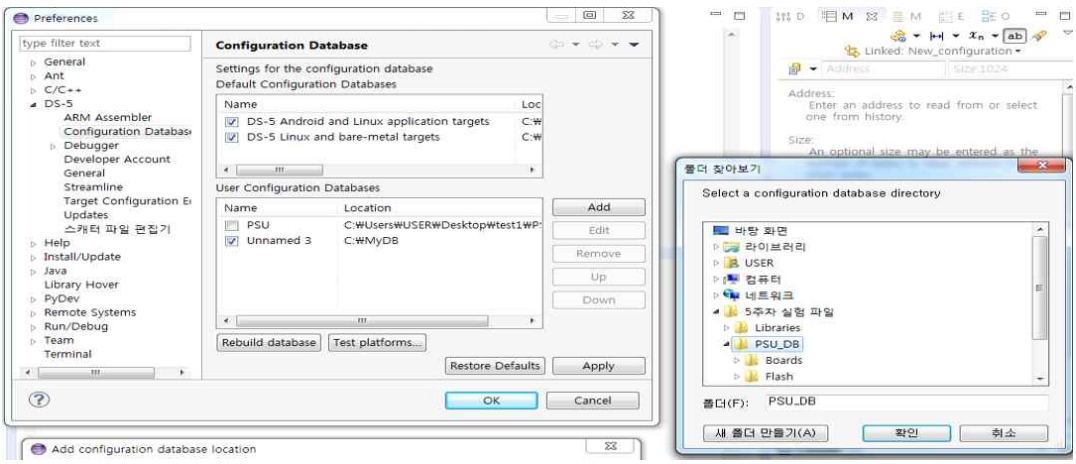
① 시작 → 모든 프로그램 → ARM DS-5 → Eclipse for DS-5 메뉴를 선택

② Windows → Preferences

③ DS-5 → Configuration Database 항목을 선택

④ Add 버튼을 클릭하여 사용자 데이터베이스의 디렉터리를 지정

⑤ Rebuild database 버튼을 클릭하여 데이터베이스 추가를 완료



<그림6 : Eclipse에 데이터베이스 추가>

3) C Project 생성 및 환경설정

3-1) C Project 생성

① New Project → C project → Excutable → Empty Project를 선택해주고 Toolchains는 ARM Cmpiler 5(DS-5 built in)로 선택 후 프로젝트 생성

3-2) C Project Properties 설정

① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → ARM Linker 5 → Image Layout → Scatter file 설정

(Scatter file에서 RO/RW base address를 지정해주므로 설정해줄 필요가 없음)

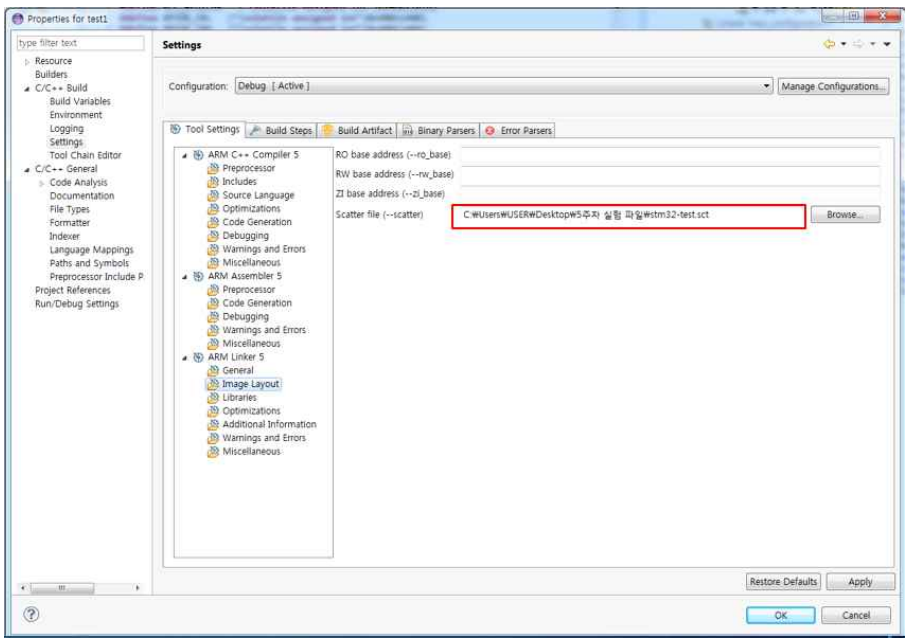
② C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Code Generation과 General의 Target CPU를 Cortex M3로 설정

(entry point를 main으로 지정해주지 않아도 됨)

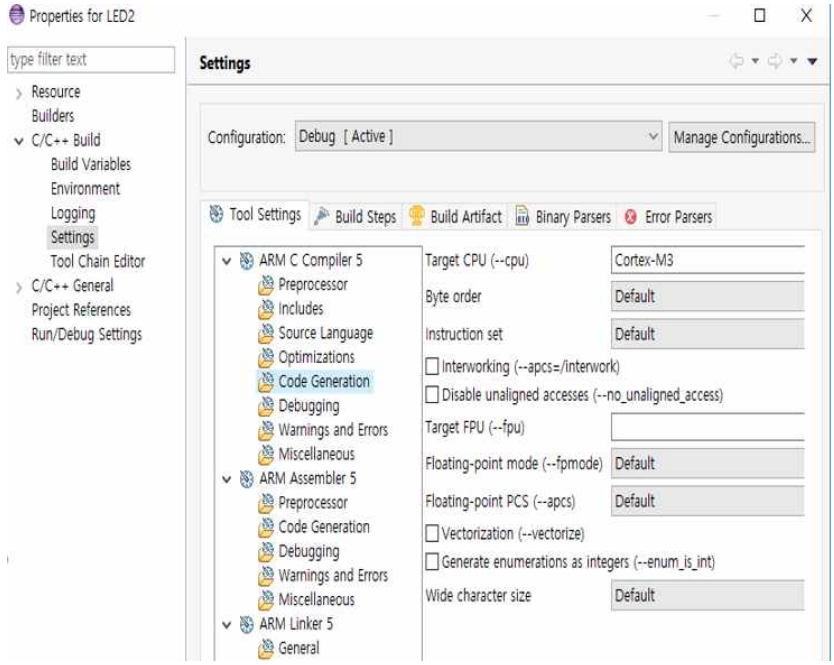
③ C project 우클릭 후 Properties 선택 → C/C++ Build → Settings -> Optimization에서 Optimization level을 High로 설정 (High로 설정 시 컴파일 할 때 코드의 크기가 작아짐)

3-3) LIBRARIES 추가

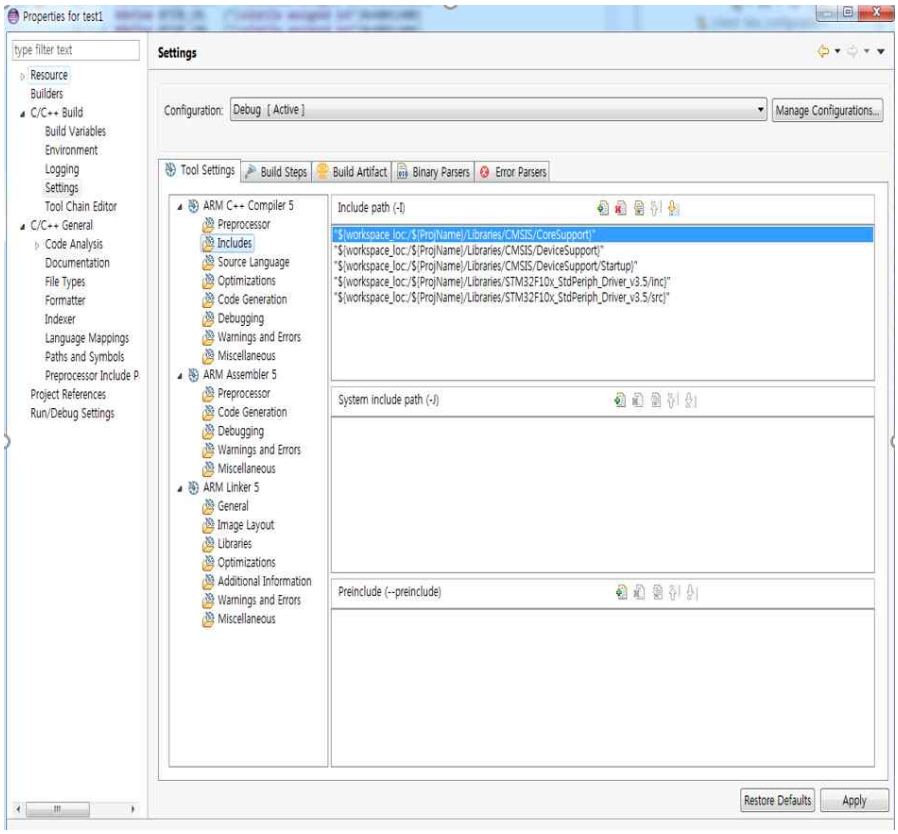
① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Includes에서 제공되는 LIBRARIES 추가



<그림7 : Scatter file 설정>



<그림8 : Target CPU 설정>



<그림9 : 제공되는 LIBRARIES 추가>

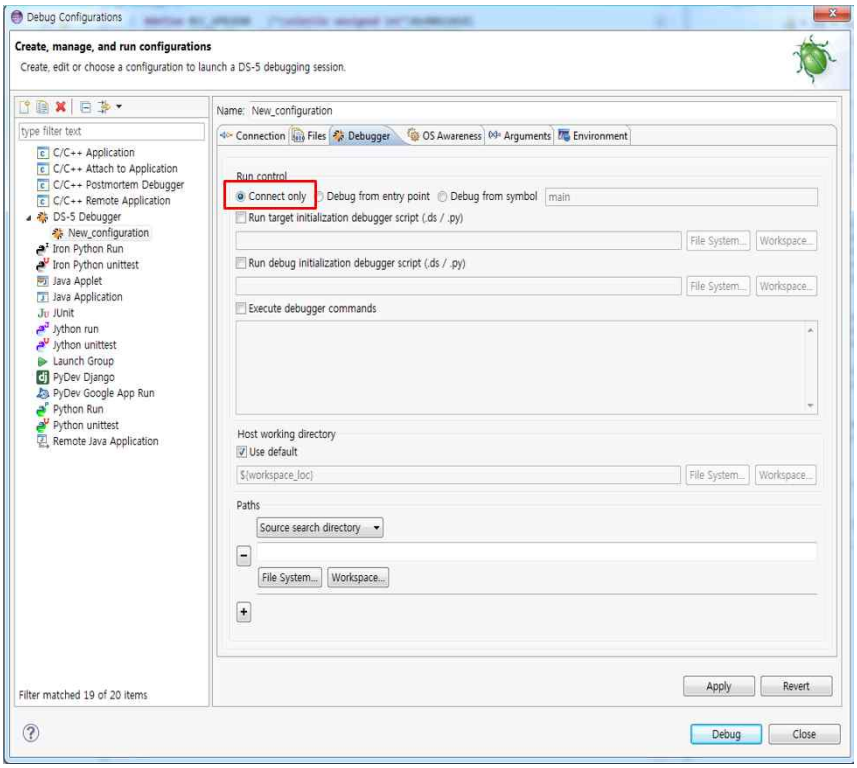
4) DS-5 Debugger 연결

4-1) Debug Configuration 설정

① Run → Debug Configuration 메뉴를 선택

② DS-5 Debugger 더블 클릭하여 새로운 하위 오브젝트 생성
③ Name, Platform 등 Debug 환경설정을 변경
④ Browse 버튼을 클릭하여 DSTREAM 장비를 detection

4-2) Debug
① Debugger 탭에서 Connect only 체크
② Apply 클릭



<그림10 : Debug Configurations에서 Debugger탭의 설정화면>

5) 세부 실험 내용
5-1) 실험 요구 조건 및 구현 내용
① 2개의 조도센서 값이 LCD에 표시되도록 코드 작성
② 인터럽트가 아닌 DMA로 동작(인터럽트 헨들러 사용 x)
③ 조도센서의 수치에 따라 LED점등
5-2) C source code 작성
① ADC설정
② DMA설정

7) C project 빌드 및 .axf 업로드
7-1) C project 빌드 및 Debug As
① C Project 빌드
(.axf 파일이 생성된 것을 확인할 수 있음)
② C Project 우클릭 → Debug → Debug As
7-2) flashclear.axf 및 생성된 .axf 업로드

① command 라인에 다음과 같은 명령어를 입력해서 flashclear.axf를 업로드
flash load “flashclear.axf 파일경로”
② disconnect한 다음 보드를 껐다가 켬
③ command 라인에 다음과 같은 명령어를 입력해서 생성된 .axf를 업로드
flash load “생성된 .axf 파일경로”
④ disconnect한 다음 보드를 껐다가 켬
(flash load 후에는 반드시 diconnect를 하고 보드를 껐다가 켜야 함)

7-3) LCD에 나타나는 2개의 조도센서 값 확인

7-4) 조도센서 값에 따른 LED1, 2점등 확인
① 조도센서에 빛을 비추면서 LED1, 2의 변화를 관찰.
② 제대로 동작하지 않으면 5) C source code 작성으로 돌아감

8) 보드 연결 해체

앞서 보드 연결과 마찬가지로, 보드 연결 해체 시에도 순서를 제대로 지키지 않으면 보드가 망가질 수 있으므로 유의해야한다. 보드 연결 해체 순서는 다음과 같다.

① DS-5에서 ‘disconnect target’ ② 보드 전원 OFF ③ DSTREAM 전원 해제 및 OFF ④ 보드 전원선 분리 ⑤ DSTREAM과 보드 JTAG 분리

<표 2 : 보드 연결 해체 순서>

4. 작성한 소스코드 및 실험결과

1) 작성한 소스코드

team07.c

```
// flash load "C:\Users\Team07\week12\team07\flashclear\flashclear.axf"
// flash load "C:\Users\Team07\week12\team07\Debug\team07.axf"

#include "stm32f10x.h"
#include "stm32f10x_dma.h"
#include "core_cm3.h"
#include "misc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_adc.h"
#include "lcd.h"
#include "touch.h"

vu32 ADC_VAL[2];
int color[12] =
{WHITE,CYAN,BLUE,RED,MAGENTA,LGRAY,GREEN,YELLOW,BROWN,BRRED,GRAY};

void RCC_Configure(void) {
```

```
            RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
            RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
            RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
            RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}

void GPIO_Configure(void) {
        GPIO_InitTypeDef GPIO_InitStructure;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
}

void GPIO_In() {
        GPIO_InitTypeDef GPIOD_LED;
        GPIOD_LED.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
        GPIOD_LED.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIOD_LED.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOD, &GPIOD_LED);
}

void ADC_Configure(void) {
        ADC_InitTypeDef ADC_InitStructure;
        ADC_DeInit(ADC1);
        ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        ADC_InitStructure.ADC_NbrOfChannel = 2;
        ADC_InitStructure.ADC_ScanConvMode = ENABLE;

        ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);

        ADC_Init(ADC1,&ADC_InitStructure);
        ADC_RegularChannelConfig(ADC1,          ADC_Channel_11,          1,
ADC_SampleTime_239Cycles5);
        ADC_RegularChannelConfig(ADC1,          ADC_Channel_12,          2,
ADC_SampleTime_239Cycles5); // ?
        ADC_Cmd(ADC1, ENABLE);
        ADC_DMACmd(ADC1,ENABLE);
        ADC_ResetCalibration(ADC1);
        while(ADC_GetResetCalibrationStatus(ADC1)!=RESET);
        ADC_StartCalibration(ADC1);
        while(ADC_GetCalibrationStatus(ADC1)!=RESET);

}

void DMA_Cofigure(void) {
```

```
	DMA_InitTypeDef DMA_InitStructure;
	DMA_DeInit(DMA1_Channel1);
	DMA_InitStructure.DMA_BufferSize = 2;
	DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
	DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
	DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)ADC_VAL;
	DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
	DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
	DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
	DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR; // ?
	DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_Word;
	DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
	DMA_InitStructure.DMA_Priority = DMA_Priority_High;
	DMA_Init(DMA1_Channel1, &DMA_InitStructure);
	DMA_Cmd(DMA1_Channel1,ENABLE);

}

void delay(int i){
	int j;
	for(j=0; j<=i * 100000; j++);
}


int main() {

	char adc1[10];
	char adc2[10];

	unsigned int ledOn[] = { GPIO_BSRR_BS2, GPIO_BSRR_BS3,
GPIO_BSRR_BS4,
				GPIO_BSRR_BS7 };
	unsigned int ledOff[] = { GPIO_BRR_BR2, GPIO_BRR_BR3, GPIO_BRR_BR4,
				GPIO_BRR_BR7 };

	LCD_Init();
	Touch_Configuration();
	Touch_Adjust();
	LCD_Clear(WHITE);

	RCC_Configure();
	GPIO_Configure();
	ADC_Configure();
	DMA_Cofigure();
	GPIO_In();

	while(1){
		ADC_SoftwareStartConvCmd(ADC1, ENABLE);
		while(ADC_GetFlagStatus(ADC1, SET)!=RESET);
		//temp = ADC_GetConversionValue(ADC1);
		sprintf(adc1,"%d",ADC_VAL[0]);
		sprintf(adc2,"%d",ADC_VAL[1]);
		//LCD_ShowNum(100,100,8,ADC_VAL[0],BLACK, WHITE);
```

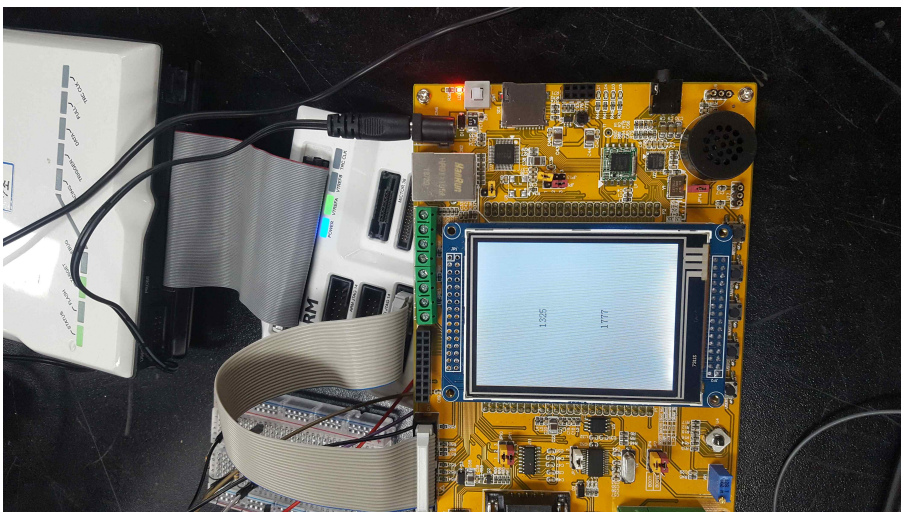
```
                    //LCD_ShowNum(100,200,8,ADC_VAL[1],BLACK, WHITE);
                    LCD_ShowString(100, 100, adc1, BLACK, WHITE);
                    LCD_ShowString(100, 200, adc2, BLACK, WHITE);

                    if((int)ADC_VAL[0] > 2300)
                            GPIO_SetBits(GPIOD, GPIO_Pin_2);
                    else
                            GPIO_ResetBits(GPIOD, GPIO_Pin_2);

                    if((int)ADC_VAL[1] > 2300)
                            GPIO_SetBits(GPIOD, GPIO_Pin_3);
                    else
                            GPIO_ResetBits(GPIOD, GPIO_Pin_3);


                    delay(10);
            }

}
```

※touch.c는 주어진 파일을 수정하지 않고 그대로 활용하였으므로, 따로 소스코드를 첨부안함

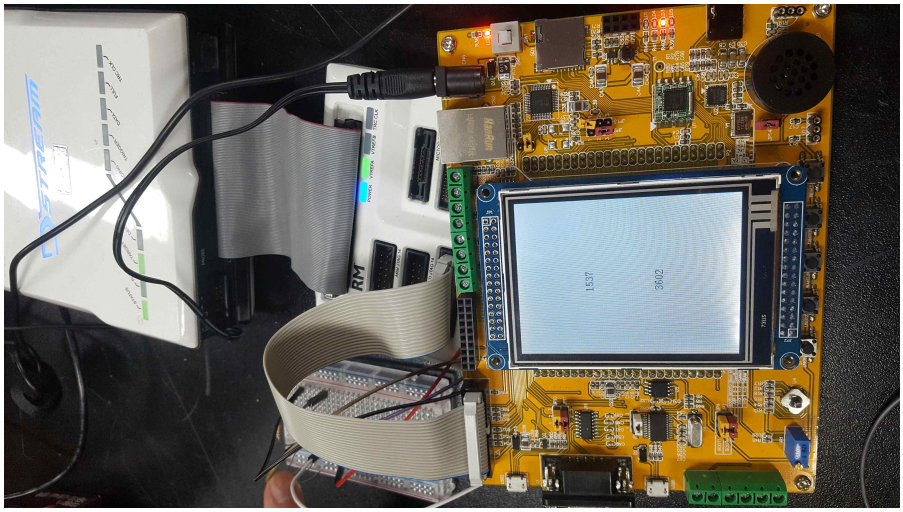
2) 실험결과

2-1) LCD 화면에 2개의 조도센서 값 출력 확인

2-2) 조도센서의 값에 따른 LED1, 2 점등 확인



<그림11 : LCD화면에 조도센서 값 출력>



<그림 12 : 조도센서의 값에 따라 LED 점등>

5. 결론 및 느낀점

이번 실험에서는 DMA에 관해 배웠다. 조도 센서를 이용하여 값을 읽어 ADC로 값을 변환하는데 이 과정은 원래 CPU가 제어하지만 이번에는 DMA를 통해 빠른 제어로 구현하고 그 값을 LCD에 출력하였다. 또한 읽은 조도 센서 값으로 밝기에 따라 LED를 제어하는 동작도 구현하였다. 이전에 했던 실험과 중복 되는 부분이 많아서 구현 난이도가 높지 않아 빨리 구현할 수 있었다.

이번 주차 실험을 끝으로 총 8번의 실험을 하였다. 이론으로 배웠던 인터럽트, 클럭 제어, 모듈 간 통신 방법 등 시스템 아키텍처를 본 실험 강의를 통해 구현하면서 많은 시행착오도 겪었지만 직접 구현을 통해 그 원리들을 체득할 수 있었다. 남은 기간동안 배운것을 바탕으로 텀 프로젝트를 수행하게 된다. 시스템을 구현하는 과정 중에 예상치 못한 결과가 발생하거나 오류를 겪을 수도 있을것이다. 팀원들과 자주 소통하여 발생되는 오류를 처리하고 또한 적절하게 분업하여 조화롭게 텀프로젝트를 수행해야 될 것이다. 의미있는 텀프로젝트가 되었으면 좋겠다.