임베디드 시스템 설계 및 실험

5주차

7조
201424470 서민영
201424421 김시은
201424533 정종진

목차

1. 실험목표

1) 스케터 파일의 이해 및 플래시 프로그래밍
2) 릴레이 모듈의 이해 및 임베디드 펌웨어를 통한 동작

2. 배경지식

1) 보드 연결

① Scatter file이란

링크과정을 통해 만들어진 이미지에는 데이터 요소들이 region과 Output section으로 구성되어 각 region은 서로 다른 실행주소를 가질 수 있다. 이러한 주소설정은 Scatter loading 매커니즘을 통해 구현가능하다. (Simple한 구조를 갖는 경우에도 Scatter file을 이용해 구성가능.)

② Sacatter file이 필요한 이유

- Complex Memory Map: Code, data가 메모리상에 분산되어 위치해야 할 때
- Different Type Memory: Falsh, ROM, DRAM, SRAM등 다양한 메모리를 가진 시스템
- Memory Mapped I/O: 복잡한 I/O map을 다뤄야 하는 경우
- Functions at constant Location: 주변 응용프로그램들에 의해 수정되고 다시 컴파일 될 때 특정 function을 메모리의 동일 장소에 위치시켜야 할 때
- Using symbols to identify the heap and stack: Stack과 heap영역을 위한 심볼을 정의할 수 있고 application이 링크될 때 지정되어질 수 있도록 관련 모듈의 위치 심볼을 정의할 때

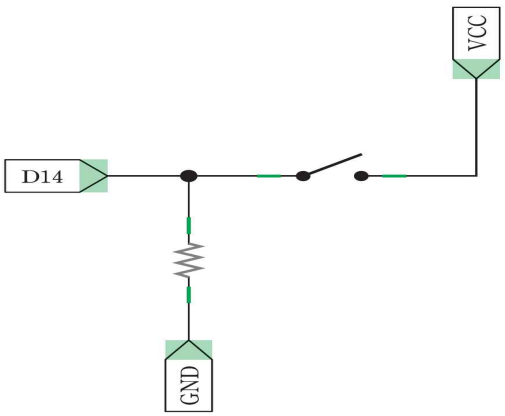
2) 버튼연결

① Floating Input 문제

- Digital 입력 핀에 아무런 회로가 연결되지 않은 상황
- 주변 핀의 상태나 정전기 등에 의해 임의의 값이 입력될 수 있음
- 버튼을 사용할 때는 회로가 오픈되는 경우를 피해야 한다.

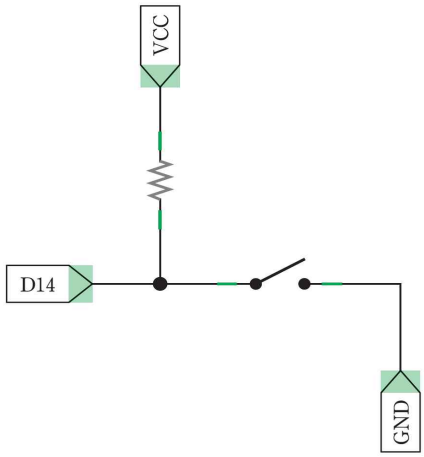
② Pull-Down 저항

- 버튼 on(push): 1(Vcc)
- 버튼 off: 0(Gnd)



<그림 1 : Pull-Down 저항>

③ Pull-Up 저항
- 버튼 on(push): 0(Gnd)
- 버튼 off: 1(Vcc)

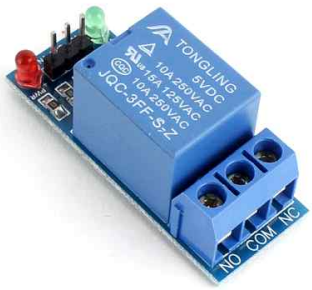


<그림 2 : Pull-Up 저항>

3) CPU 주변장치 요청 처리
① 폴링: CPU가 루프문으로 계속해서 주변장치의 요청여부를 확인하며 작업처리
- 장점: 구현 용이, 단일 이벤트에 유용
- 단점: 다중이벤트의 경우 컨트롤하기 힘듬

② 인터럽트: CPU가 주변장치에 인터럽트 요청이 올때만 해당 요청 실행
- 장점: 다중 이벤트 처리에 효율적
- 단점: 코드 작성에 신경써야 함

4) 릴레이 모듈



<그림 3 : 실험에서 사용한 릴레이 모듈>

- 전기적으로 제어할 수 있는 스위치
- 전통적인 전자기 릴레이는 내부에 전자석이 있어 스위치를 끄고 켜는게 가능
- NO(Normally Opened): 전원 인가시, High -> Low 동작
- NC(Normally Closed): 전원 인가시, Low -> High 동작
- COM(Common): 공통선

3. 실험과정

1) 보드 연결

<그림 1 : 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.

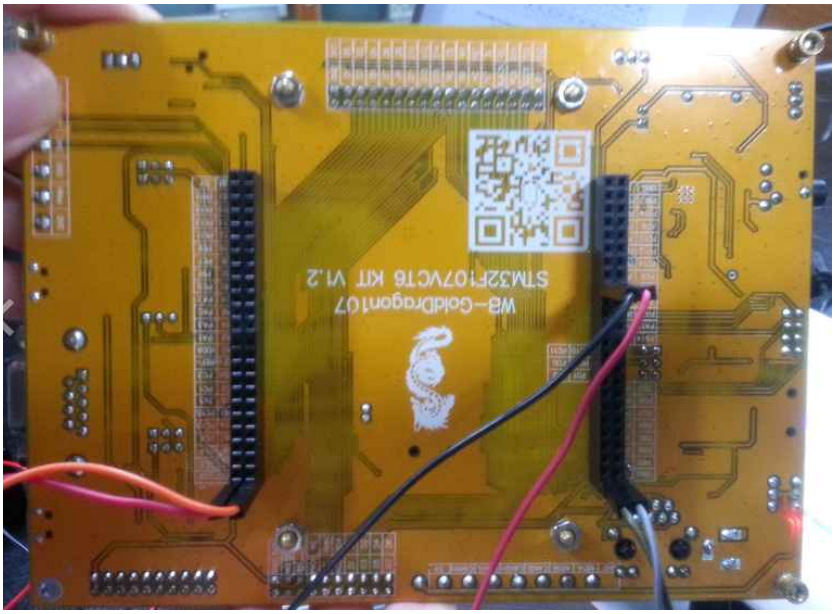
표 1 : 보드 연결 순서

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'

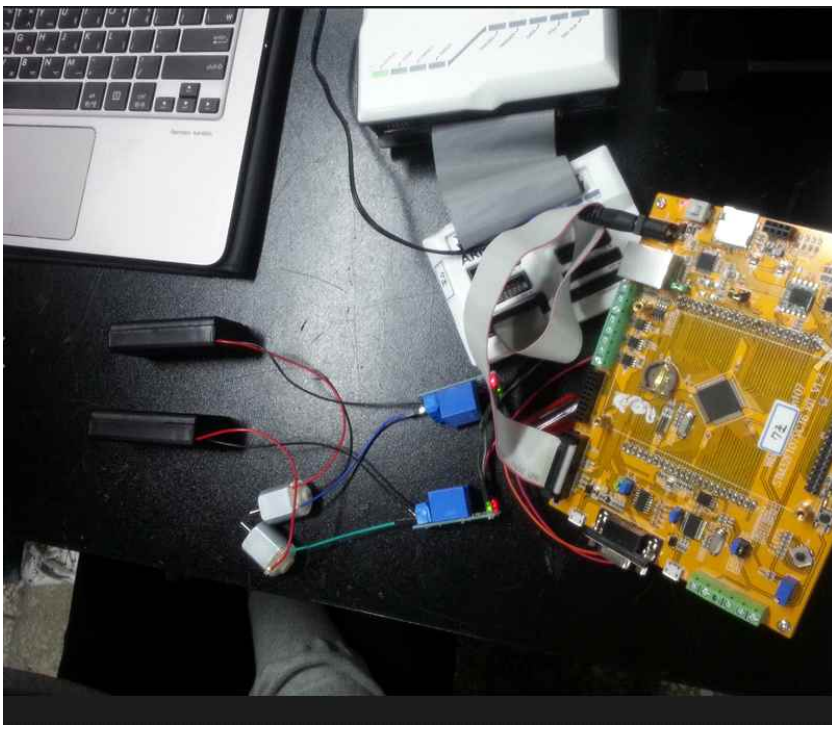
1-2) 모터, 릴레이, 5V 배터리 연결

① 모터, 릴레이, 5V 배터리와 보드를 적절하게 연결

(stm32_ReferenceManual과 stm32_DataSheet를 참고)



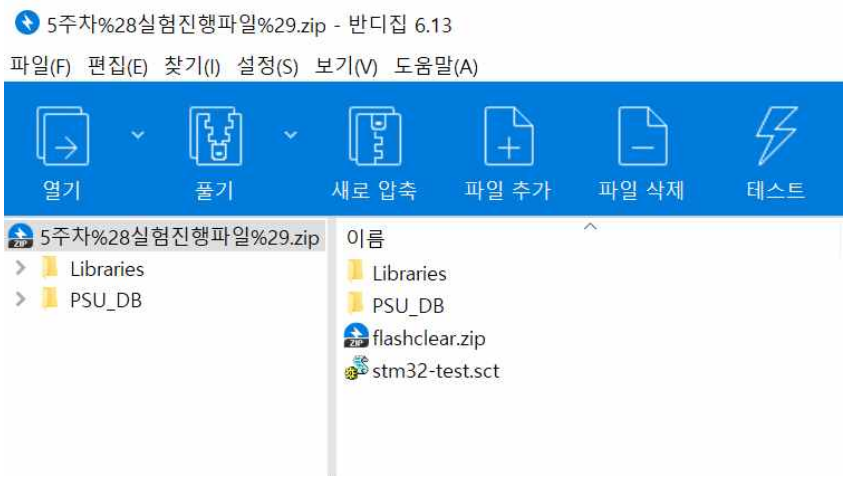
<그림 5 : 보드에 5V 배터리, 릴레이, 모터를 연결한 모습(1)>



<그림 6 : 보드에 5V 배터리, 릴레이, 모터를 연결한 모습(2)>

2) DS-5 디바이스 데이터베이스 추가

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



<그림 7 : 실습파일로 제공된 파일들>

2-3) Eclipse에 데이터베이스 추가

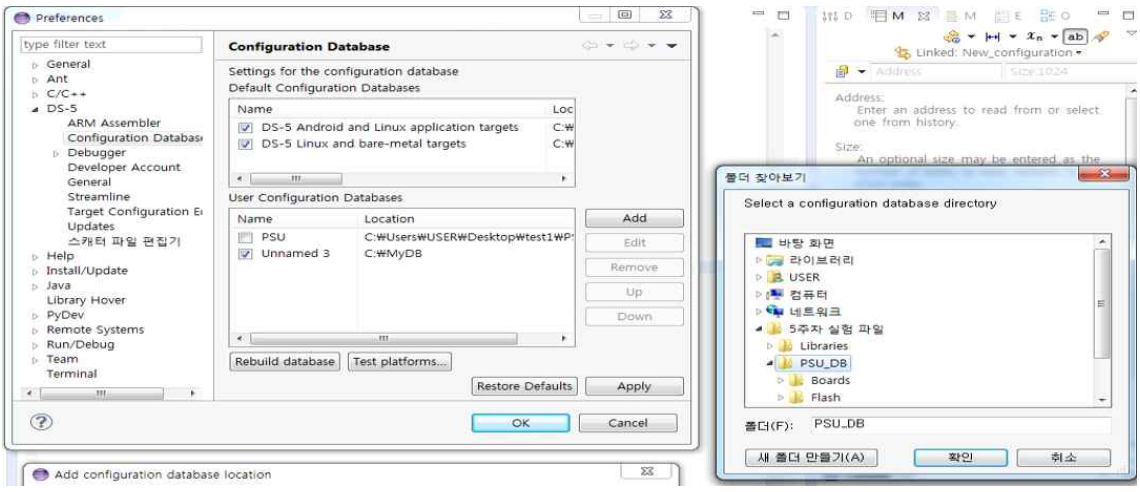
① 시작 → 모든 프로그램 → ARM DS-5 → Eclipse for DS-5 메뉴를 선택

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



<그림 8 : 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

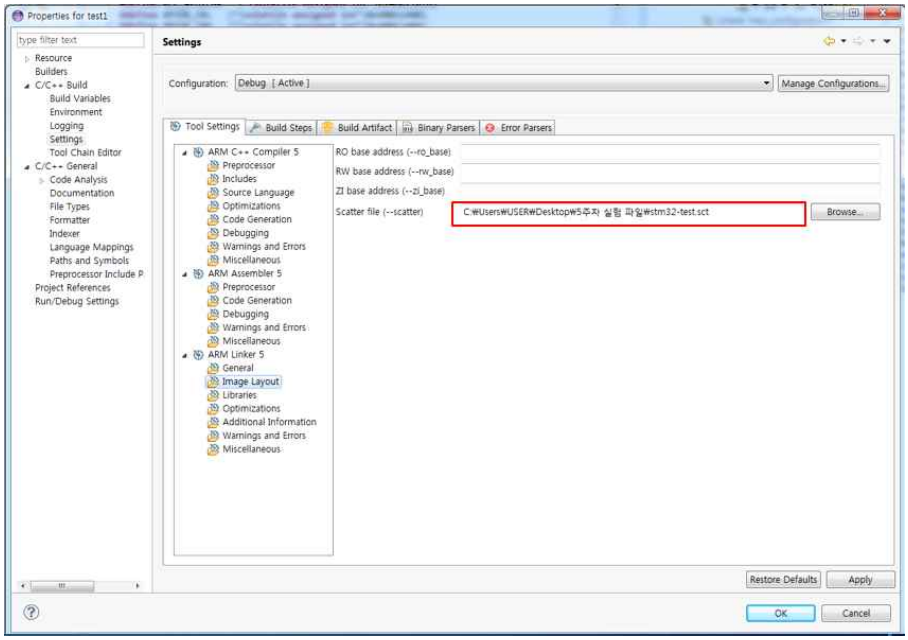
(Scatter file에서 RO/RW base address를 지정해주므로 따로 설정해줄 필요가 없음)

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

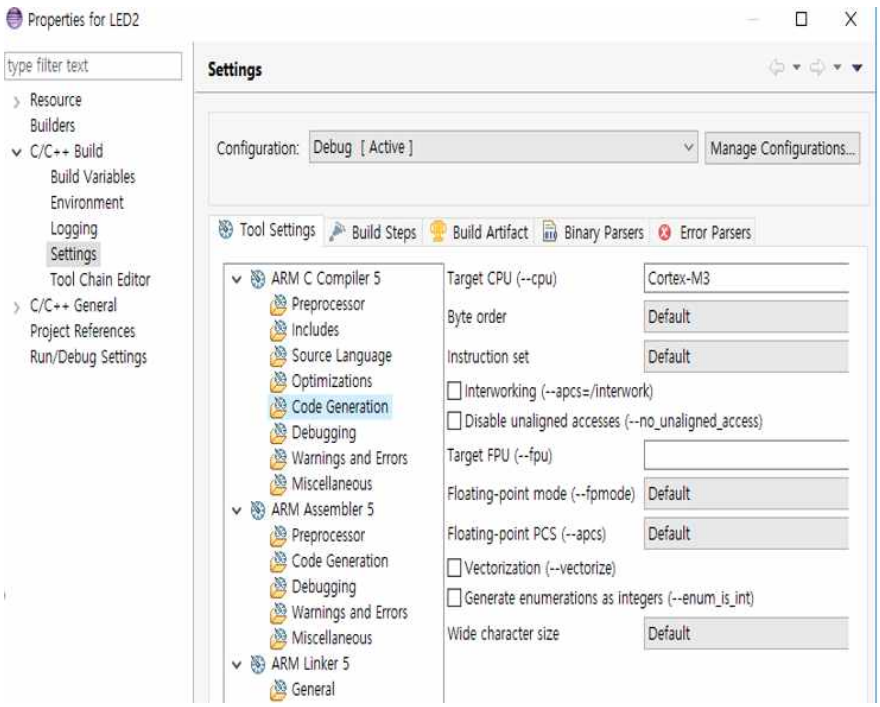
(entry point를 main으로 지정해주지 않아도 됨)

3-3) LIBRARIES 추가

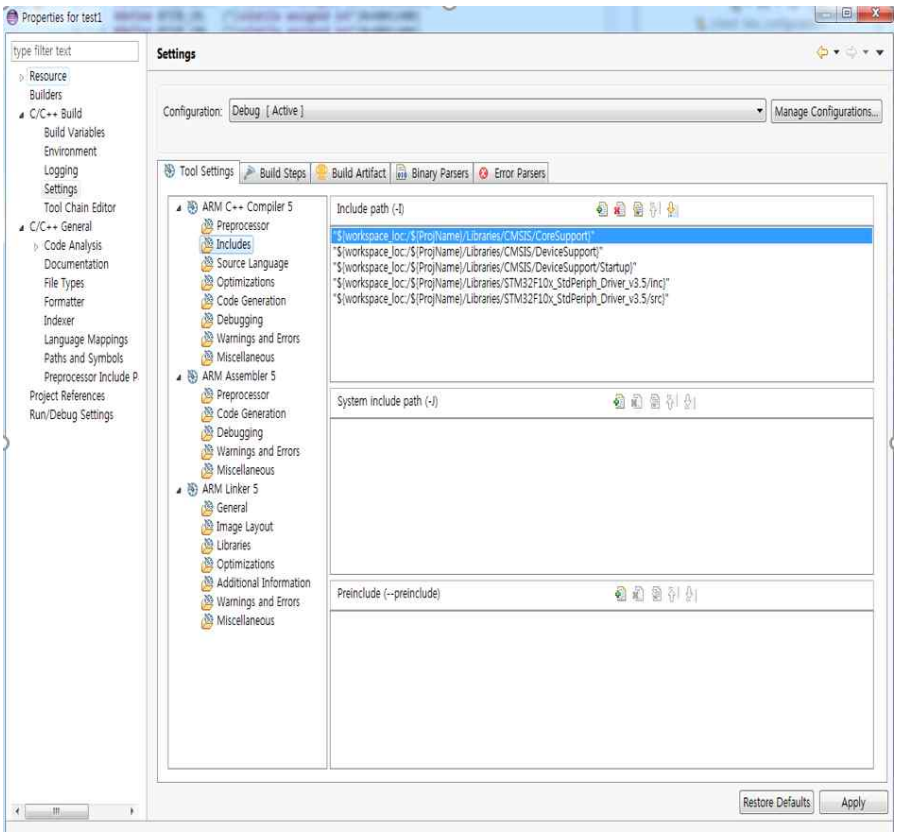
① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Includes에서 제공되는 LIBRARIES 추가



<그림 9 : Scatter file 설정>



<그림 10 : Target CPU 설정>



<그림 11 : 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

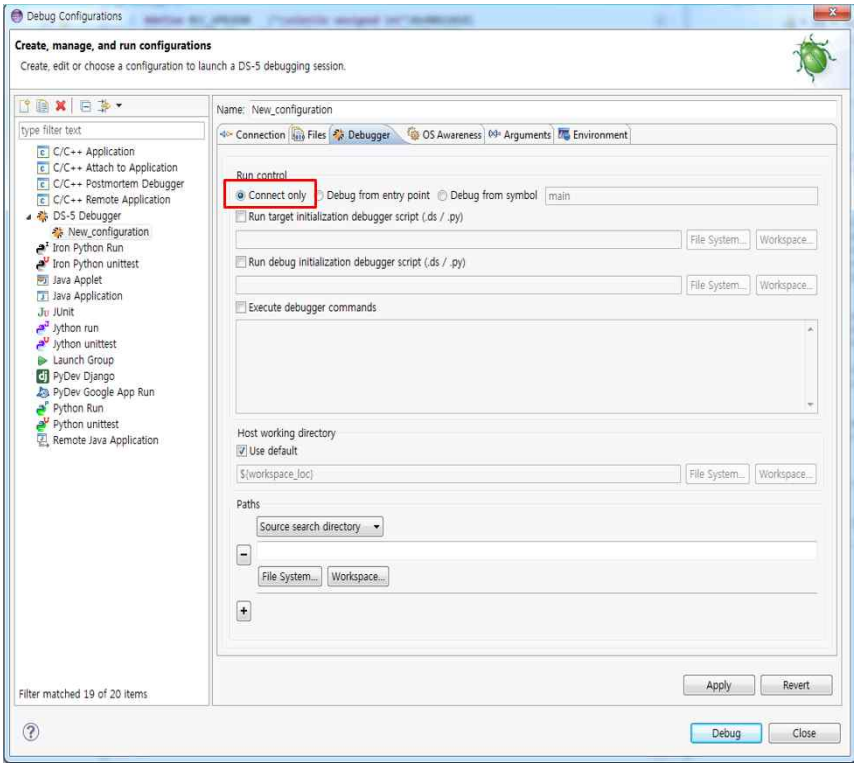
③ Name, Platform 등 Debug 환경설정을 변경

④ Browse 버튼을 클릭하여 DSTREAM 장비를 detection

4-2) Debug

① Debugger 탭에서 Connect only 체크

② Apply 클릭



<그림 12 : Debug Configurations에서 Debugger탭의 설정화면>

5) C source code 작성

5-1) 해당 Port를 Enable 및 초기화

① LED를 켜기 위한 Port D, 조이스틱, 모터를 동작시키기 위한 Port B, C를 enable 시키고 적절한 값으로 초기화를 해줌

(stm32_ReferenceManual과 stm32_DataSheet를 참고)

5-2) 주어진 조건에 맞는 적절한 함수를 작성

① 조이스틱 조작에 맞게 LED와 모터가 주어진 동작을 하도록 적절한 함수를 작성

6) C project 빌드 및 .axf 업로드

6-1) C project 빌드 및 Debug As

① C Project 빌드

(.axf 파일이 생성된 것을 확인할 수 있음)

② C Project 우클릭 → Debug → Debug As

6-2) 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)

6-3) 동작 유무 확인

① 조이스틱을 조작하여 LED와 모터가 적절하게 동작하는 지 확인
② 제대로 동작하지 않으면 5) C source code 작성으로 돌아감

7) 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.

표 2 : 보드 연결 해체 순서

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

4. 작성한 소스코드 및 실험결과

1) 작성한 소스코드

전체 소스코드

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

//#include <stdio.h>
const int ALE1= 0x04;
const int ALE2= 0x08;
const int ALE3= 0x10;
const int ALE4= 0x80;

const int UP = 0x20;
const int DOWN = 0x04;
const int LEFT = 0x08;
const int RIGHT = 0x10;

const int MOTOR1 = 0x100;
const int MOTOR2 = 0x200;

void delay(num) {
    int i;
    for(i=0;i<num;++i)
    ;
}
void ledOn(int led) { // 특정 led를 켜는 동작을 수행
    (*(volatile unsigned *)0x40011410) = led;
}
void ledOff(void) { // 특정 led를 끄는 동작을 수행
    (*(volatile unsigned *)0x40011414) = ALE1 | ALE2 | ALE3 | ALE4;
}
void forward() { // 왼쪽, 오른쪽 모터가 동시에 돌면서 앞으로 전진
    (*(volatile unsigned *)0x40011010) = MOTOR1 | MOTOR2; // BSRR에 해당 값을 넣어줌
}
void  rightRotate() { // 오른쪽 모터만 돌면서 오른쪽 회전
    (*(volatile unsigned *)0x40011010) = MOTOR1; // BSRR에 해당 값을 넣어줌
    delay(100000);
}
```

```
void  leftRotate() { // 왼쪽 모터만 돌면서 왼쪽 회전
    (*(volatile unsigned *)0x40011010) = MOTOR2; // BSRR에 해당 값을 넣어줌
    delay(100000);
}
void rotate() { // 오른쪽 모터가 길게 돌면서 오른쪽으로 180도 회전
    (*(volatile unsigned *)0x40011010) = MOTOR1; // BSRR에 해당 값을 넣어줌
    delay(18000000);
}
void stop() { // 왼쪽, 오른쪽 모터 모두 멈춤
    (*(volatile unsigned *)0x40011014) = MOTOR1|MOTOR2; // BRR에 해당 값을 넣어줌
}
int main(void) {
    (*(volatile unsigned *)0x40021018) |= 0x38; // PORT B, C, D enable
    (*(volatile unsigned *)0x40011400) = 0x44444444;
    (*(volatile unsigned *)0x40011400) = 0x10011100; // PORT D의 CRL set
    (*(volatile unsigned *)0x40011000) = 0x00888800; // PORT C의 CRL set
    (*(volatile unsigned *)0x40011004) = 0x00000033; // PORT C의 CRH set
    (*(volatile unsigned *)0x40010C04) = 0x8; // PORT B의 CRH set
    (*(volatile unsigned *)0x40011008) = 0x00000000; // PORT C의 IDR set

    while(1) {
        if((~(*(volatile unsigned *)0x40011008) & UP)) {
            ledOn(ALE4);
            forward();
        }
        else if((~(*(volatile unsigned *)0x40011008) & DOWN)) {
            ledOn(ALE1);
            rotate();
        }
        else if((~(*(volatile unsigned *)0x40011008) & LEFT)) {
            ledOn(ALE3);
            leftRotate();
        }
        else if((~(*(volatile unsigned *)0x40011008) & RIGHT)) {
            ledOn(ALE2);
            rightRotate();
        }
        else {
            ledOff();
            stop();
        }
    }
    return 0;
}
```

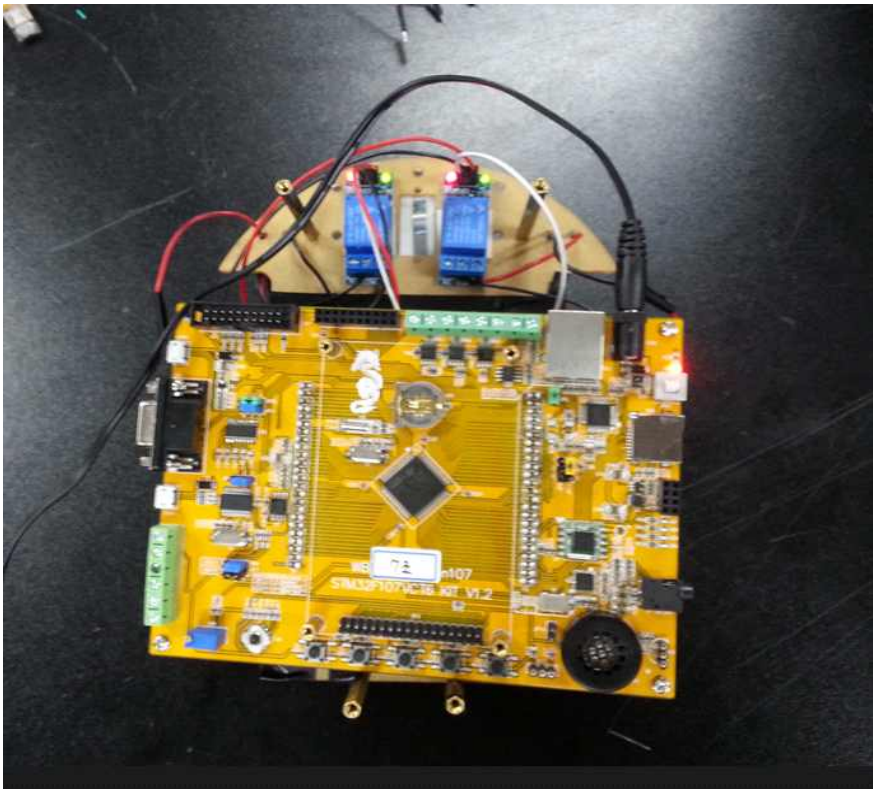
2) 실험결과

2-1) Up : LED 4 점등하며 전진

2-2) Down : LED 1 점등하며 180도 회전

2-3) Left : LED 3 점등하며 왼쪽으로 회전

2-4) Right : LED 2 점등하며 오른쪽으로 회전



<그림 13 : 조이스틱을 조작에 따라 모터와 LED가 특정 동작을 수행>

5. 결론 및 느낀점

이번 실험을 통해 Scatter File이 여러 개의 입력 Section을 출력 Section으로 그룹화하고, Memory Map에 효율적으로 배치하기 위해서 사용한다는 것을 알았고, Flash Memory로의 Loading을 통한 Flash Level의 디버깅을 할 수 있었다.

3주차에 LED를 다루는 실험을 하여, 대체로 무난하게 진행할 수 있었지만, Motor, Battery, Relay Module에 대해 이해하고 연결하는데 시간이 걸렸다. 이것들은 연결한 후에는 Motor를 사용할 Port를 열어주고, 저번 실험과 마찬가지로 GPIOx_CRH, GPIOx_BRR 과 GPIOx_BSRR을 이용하여 쉽게 소스 코드를 쉽게 작성할 수 있었다. Cortex-M3 보드에 있는 입/출력 센서뿐만 아니라 외부 입/출력 센서를 연결하여 사용하는 방법을 알게 되었고, 인터럽트, 폴링 방식의 차이를 정확하게 이해할 수 있었고, 다양한 버튼 방식(Floating, Pull-up, Pull-down)의 종류를 알게 되었다.

6. 참고 문헌 및 사이트

[1] 부산대학교 ENS 연구실 (https://enslab.pusan.ac.kr/)

[2] 2018_5주차_임베디드_시스템_설계_및_실험 예비조(9조, 10조) 발표자료