

# 임베디드시스템 설계 및 실험 보고서(4주차)



8조

작성자: 201824523 안혜준

조원:

201602181 정진성

201824483 박진영

201924660 한병정

202055516 김명서

## 1. 실험 목표


실험 목적은

1. 스캐터 파일의 이해 및 플래시 프로그래밍
2. 릴레이 모듈의 이해 및 사용
3. 폴링 방식의 이해 이다.

스캐터 파일(.icf)을 생성해 원하는 메모리 위치로 프로그램을 다운로드를 하고, 릴레이 모듈을 통하여 모터를 구동시켜 볼 것이다.

## 2. 실험 과정

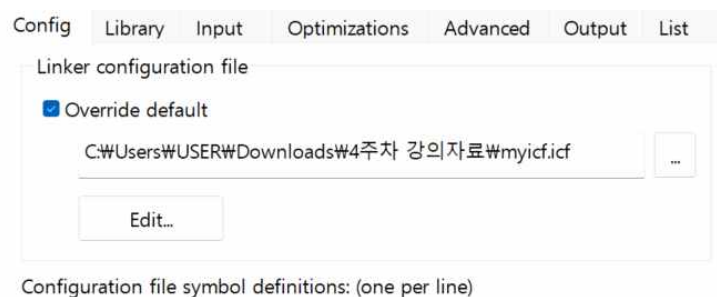
### 2.1. 스캐터 파일을 통한 플래시 프로그래밍



```
stm32f107xC - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$WconfigWideWlcfEditorWcortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__ = 0x0803FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__ = 0x2000FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__ = 0x1000;
/**** End of ICF editor section. ###ICF###*/
```

우리가 사용하는 보드인 stm32f107VCT6은 기본적으로 ROM과 RAM의 범위가 0x08000000~0x0803FFFF / 0x20000000~0x2000FFFF로 설정되어있다.

이를 ROM과 RAM의 범위를 재설정한 myicf.icf를 만들어 적용하였다.



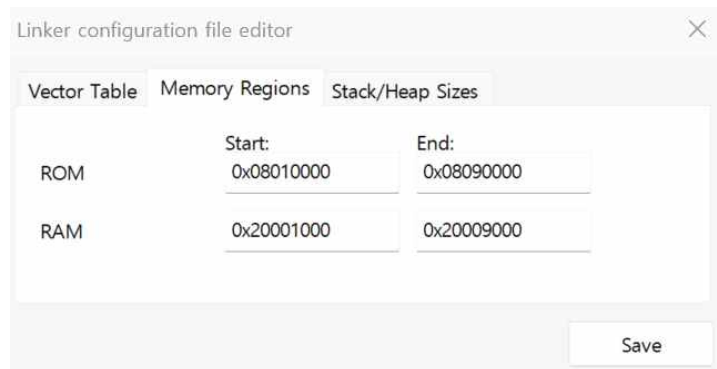
default 대신 myicf.icf를 적용

```

define symbol __ICFEDIT_region_ROM_start__ = 0x08010000;
define symbol __ICFEDIT_region_ROM_end__   = 0x08090000;
define symbol __ICFEDIT_region_RAM_start__  = 0x20001000;
define symbol __ICFEDIT_region_RAM_end__    = 0x20009000;

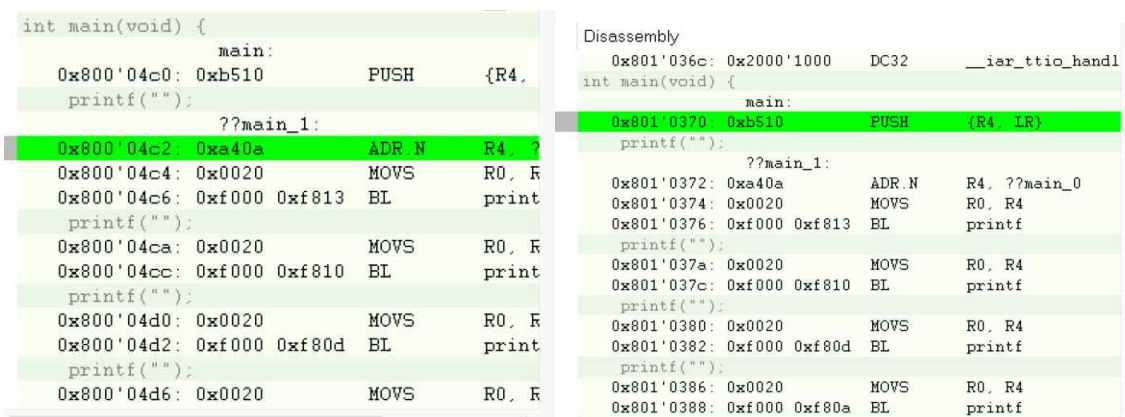
```

myicf.icf의 변경된 부분이다. 이를 통해 ROM을 0x08010000 ~ 0x08090000으로 0x80000의 크기를 할당하였고, RAM에는 0x20001000 ~ 0x20009000으로 0x8000의 크기를 할당하였다.



설정에서 Memory Region이 변경되었음을 확인할 수 있다.

디버깅을 통하여 프로그램이 위치가 변경되었는지 확인할 수 있다.



myicf 적용 전

myicf 적용 후

default 설정에서는 0x080004c2에 코드가 들어 있었지만 myicf.icf를 적용하자 0x08010370에 프로그램이 저장되었음을 확인 가능하다.

## 2.2. 릴레이 모듈을 통한 모터 구동

A. 조이스틱 UP:           모터 회전

B. 조이스틱 DOWN:       모터 정지

구현해야 할 기능은 위와 같다.

### 2.2.1. 회로 연결

먼저, 보드와 릴레이 모듈, 모터를 연결해야 한다.

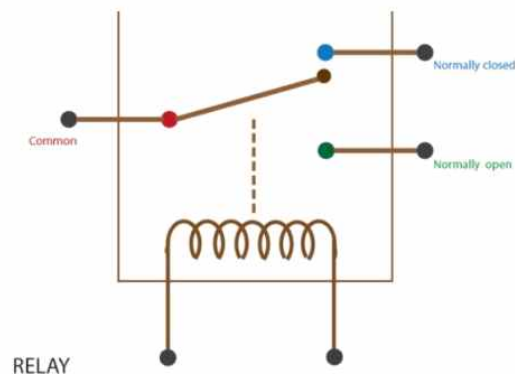


릴레이 모듈

위 그림의 릴레이 모듈의 좌측 부분은 보드와 연결하여 제어 신호를 주는 부분이고, 우측 부분은 모터와 연결되어 모터를 구동시킬 부분이다.

좌측은 VCC, GND, S의 3개의 포트가 존재한다. 각각 보드의 3V3과 GND, 그리고 제어신호를 출력할 GPIO포트와 연결한다.

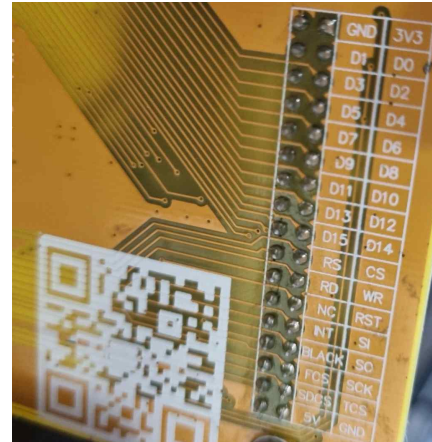
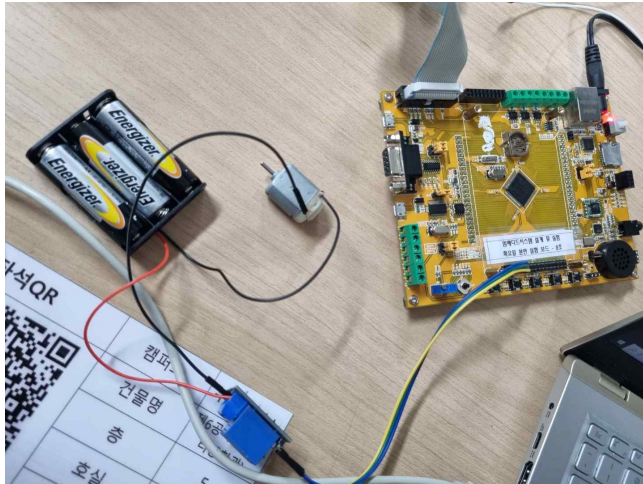
우측은 배터리와 모터를 연결하여 모터를 구동시킨다.



릴레이 모듈의 구조

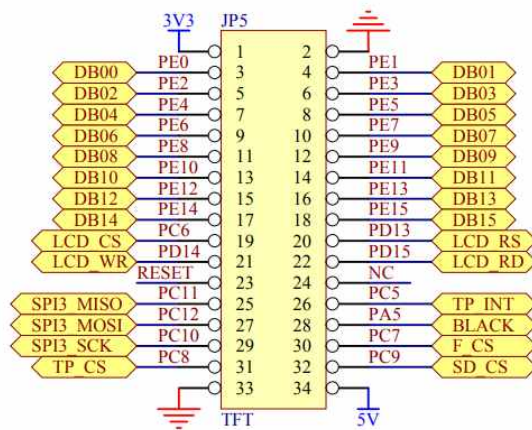
COM은 평소에 NC에 붙어 있다가 제어 신호가 들어오면 NO로 연결된다.

즉, NO와 모터를 연결함으로써 제어신호를 입력 받았을 때 모터를 구동시킨다.



회로를 연결한 모습이다.

릴레이 모듈과 연결된 보드의 부분은 보드의 뒷부분을 보면 알 수 있다. 3V3과 GND, D0이다.



STM32F107 보드 회로도

보드의 회로도를 보면 D0부분이 PE0이라는 것을 알 수 있다.

이로써 회로 연결이 끝났다.

### 2.2.2. 레지스터 주소 설정

우리는 조이스틱의 UP, DOWN(PC5, PC2), 그리고 릴레이 모듈에 신호를 출력할 PE0를 사용할 것이다.

포트 인가를 위한 RCC\_APB2ENR

GPIO포트의 모드를 선택해줄 GPIOE\_CRL, GPIOC\_CRL

조이스틱의 입력을 읽을 GPIOC\_IDR

릴레이 모듈에 신호를 출력할 GPIOE\_BSRR

이 필요하다.

Reference 파일을 보고 다음과 같이 주소를 찾았다.

```
#define RCC_APB2ENR (*(volatile unsigned int *)0x40021018)
#define GPIOE_CRL (*(volatile unsigned *)0x40011800)
#define GPIOE_BSRR (*(volatile unsigned *)0x40011810)
#define GPIOC_CRL (*(volatile unsigned *)0x40011000)
#define GPIOC_IDR (*(volatile unsigned *)0x40011008)
```

### 2.2.3. GPIO 포트 설정

PC와 PE에 Clock을 넣고, PC2와 PC5에 입력모드, PE0에 출력모드를 설정한다.

### 2.2.4. 신호제어

PC2(Down)에 입력이 들어오면 GPIOE\_BSRR |= 0x00010000;으로 rest 신호를 보내고  
PC5(Up)에 입력이 들어오면 GPIOE\_BSRR |= 0x00000001;으로 set 신호를 보낸다.

이렇게 하여, 조이스틱이 UP 되었을 때 릴레이 모듈로 set 신호가 입력되고 COM이 NO에 붙어 회로가 연결되어 모터가 동작하게 된다. 조이스틱이 DOWN이 되었을 때는 릴레이 모듈에 reset 신호가 들어가 COM이 NC로 붙어 회로 연결이 끊어져 모터가 동작 하지 않게 된다.

## 2.3. 폴링 방식의 이해 및 문제점

현재 우리의 프로그램은 폴링 방식으로 만들어졌다.

```
while(1) {

    if (check_bit(GPIOC_IDR, 2) == 0) { // DOWN
        GPIOE_BSRR |= 0x00010000;
    }
    if (check_bit(GPIOC_IDR, 5) == 0) { // UP
        GPIOE_BSRR |= 0x00000001;
    }
}
```



폴링 방식은 Hardware의 변화를 지속적으로 읽어들이어 변화를 감지하는 방식으로, 신호를 판단하기 위해서는 지속적인 확인이 필요하다. 이는 다른 일을 수행할 때 문제가 생기는데, CPU가 신호를 감지하여 어떠한 일을 수행한다면 그 일이 끝날 때까지 변화를 감지할 수 없어 전체 시스템이 멈춰버린다.

극단적인 예시로 신호가 감지 되었을 때 delay(10000)을 수행한다고 가정하자. 그렇다면 폴링 방식에서는 delay(10000)이 끝날 때까지 다른 어떠한 신호도 감지 할 수 없고 정지한채로 다른 일을 수행할 수 없을 것이다.

이를 방지하기 위해선 폴링 방식이 아닌 인터럽트 방식을 사용할 수 있다. 인터럽트 방식이란, Hardware의 변화를 감지하여 외부로부터의 전기신호 입력을 CPU가 알아차리는 방법이다. CPU가 진행 중인 일을 처리하다가 외부로부터 신호 입력이 감지되면 interrupt가 들어오고 interrupt 처리 루틴을 수행하여 신호를 처리하게 된다.

폴링 방식이 간단하지만 프로그램의 규모가 커지면 인터럽트 방식을 사용해야 자원의 낭비가 줄어들 것이다.

### 3. 실험 결과

처음에는 보드의 전원을 사용하여 모터를 구동하려 했지만, 전압이 낮아 외부 전원을 사용하였다. 릴레이 모듈에 불이 들어왔을 때가 UP을 하여 신호를 준 상황으로 모터가 돌아간다(동영상 참고). DOWN을 하자 릴레이 모듈이 꺼지고 모터가 멈춘다. 코드와 동영상은 별도 첨부하였다.

