layout: post author: 한성희, 이호욱, 한경수 date: 2016-09-30 17:48 +0900 tags:

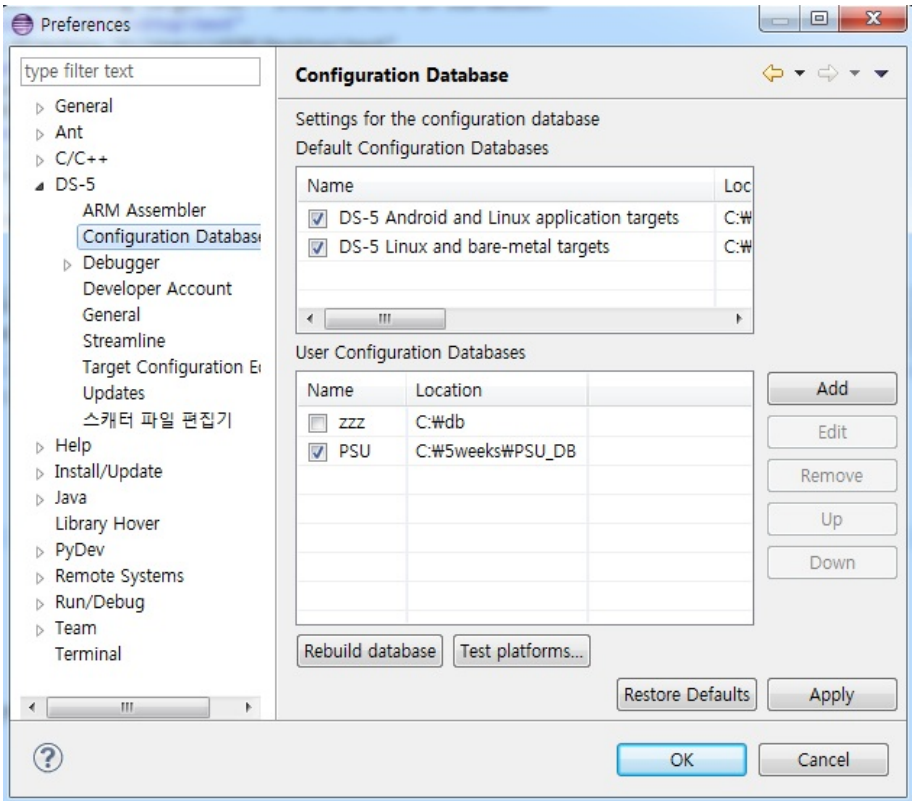
comments: true

개요

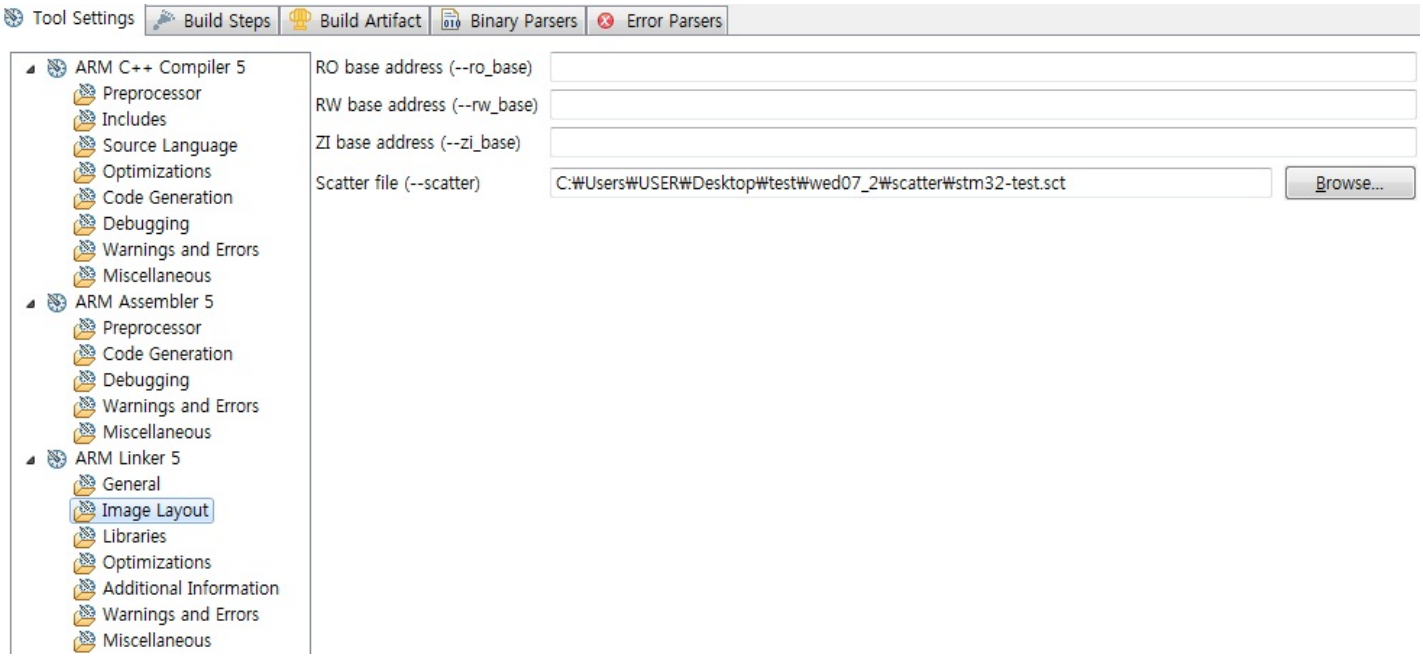
Scatter Loading File 작성 및 Flash Memory로의 Loading 을 통한 Flash Level 의 디버깅.

Flash Memory로 부터 Source를 가져와 Motor Driver 를 이용한 Motor 제어.

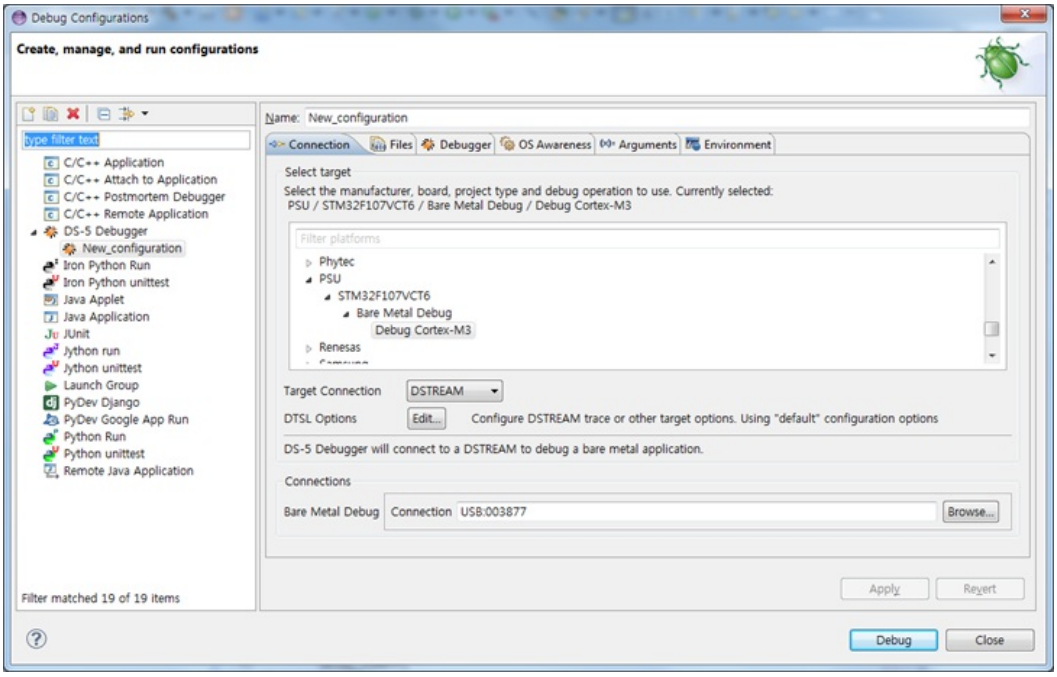
세팅



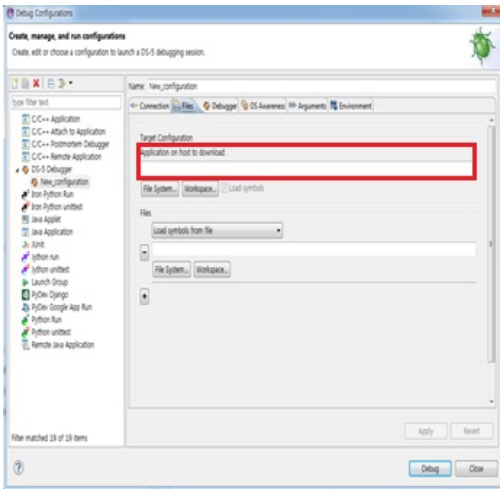
먼저 프로젝트를 생성하고 제공된 PSU로 Database 설정.



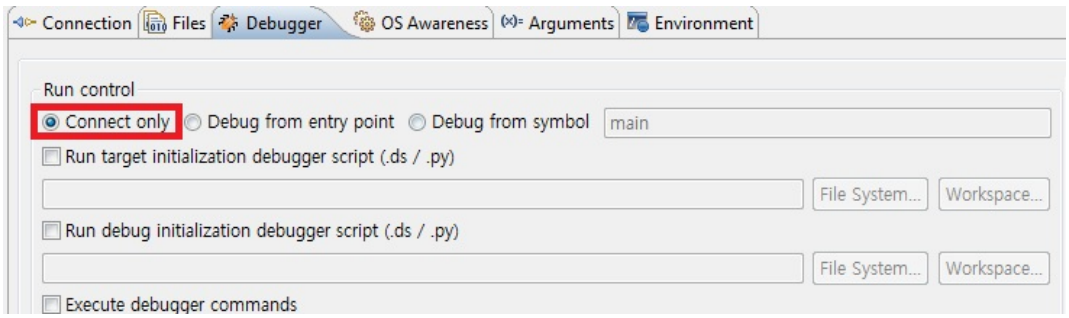
4주차 실험과 동일하게 세팅하되 RO,RW base address 부분의 값을 지우고 새로 Scatter file(--scatter) 부분에 주어진 Scatter 파일 위치 지정.



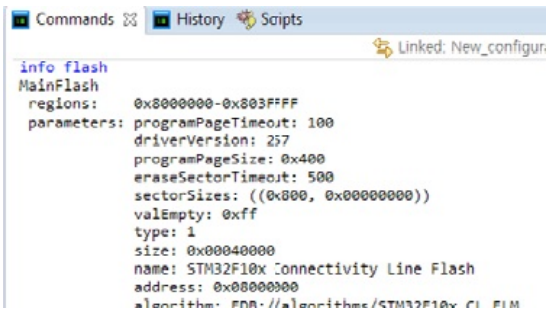
Debug Configuration의 Connection 탭에서 PSU의 Cortex-M3로 연결.



File 탭의 해당 부분의 값을 지운다.



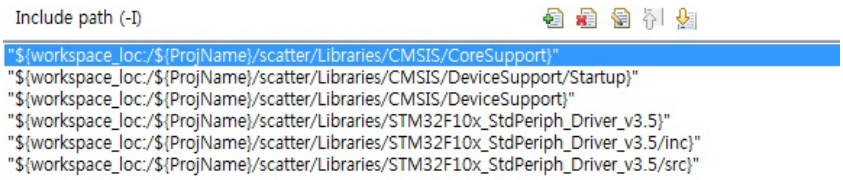
Debugger 탭의 Connect only 를 선택.



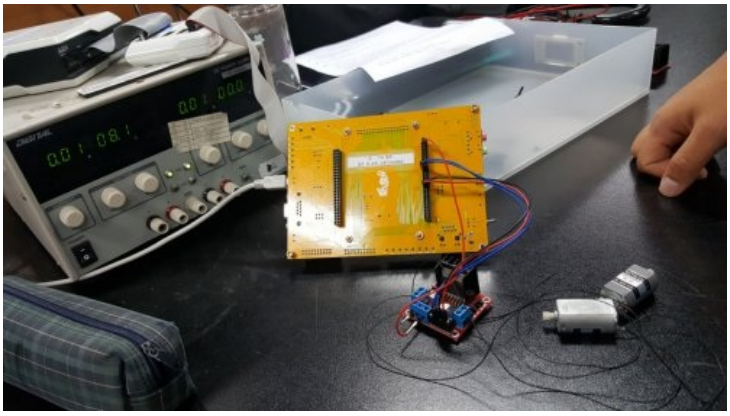
프로젝트 debug 후, Flash Load를 위해 Command 창에 "info flash" 명령어를 쳐서 flash memory의 설정을 확인.

```
WARNING(TAD65): Support & maintenance period for license feature ds_debugger_ice expired on 30-6월-2014
Connected to running target PSU - STM32F107VCT6 on USB:003895
cd "C:\Users\USER\Desktop\test"
Working directory "C:\Users\USER\Desktop\test"
interrupt
Execution stopped at: 0x080005E0
0x080005E0   MOVS     r0,#4
flash load "C:\Users\USER\Desktop\test\wed07_2\Debug\wed07_2.axf"
Writing segment 0x08000000 ~ 0x08000D6C (size 0xD6C)
Flash programming completed OK (target state was not preserved)
quit
Disconnected from stopped target PSU - STM32F107VCT6 on USB:003895
```

확인 후, Flash load ".axf 경로" 를 입력하면, board 로 source가 보내진다.



이것은 주어진 Library 로 원래 code의 주소와 값을 상수변수로 대체할 때 사용하기 위해 Library include path를 지정한다.



전체적인 기계 연결 모습

기본 개념

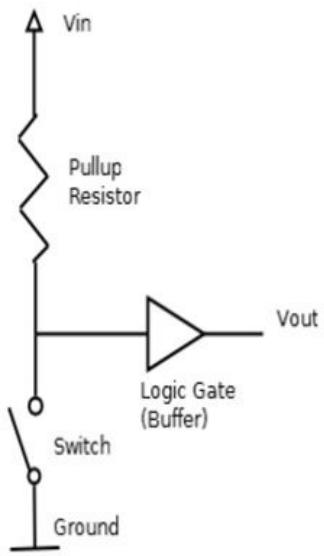
버튼 방식의 종류

1. Floating 방식

플롯 상태란, 디지털 신호에는 1과 0으로 표현되지만 1도 아니고 0도 아닌 정해지지 않은 값이다.

플로팅이란, 마치 0(Low) 과 1(High) 사이를 부유한 상태로 인위적인 전위를 설정하지 않은 것으로 이 경우 일반적인 스위치를 사용하면 값을 알 수 없다. 그래서 Pull-up, Pull-down으로 해결해야 한다.

2. Pull Up 방식



Pull up 방식은 회로에서 논리적으로 High-Level 상태를 유지하기 위해서 신호의 입력 또는 출력 단자와 사이에 접속하는 저항으로

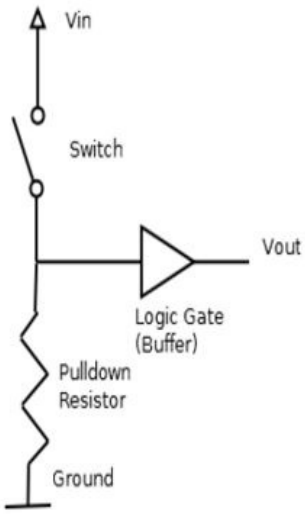
스위치가 닫혔을 때 : 0(Low)

스위치가 열렸을 때 : 1(High)

값을 나타낸다.

Pull Up 방식은 Pull Down 방식보다 noise와 충격에 강해서 더 많이 사용된다.

3. Pull Down 방식



Pull Down 방식은 논리적으로 Low-Level 상태를 유지하기 위해서 신호의 입력 또는 출력 단자와 Ground 단자 사이를 접속하는 저항으로

스위치가 닫혔을 때 : 1(High)

스위치가 열렸을 때 : 0(Low)

값을 나타낸다.

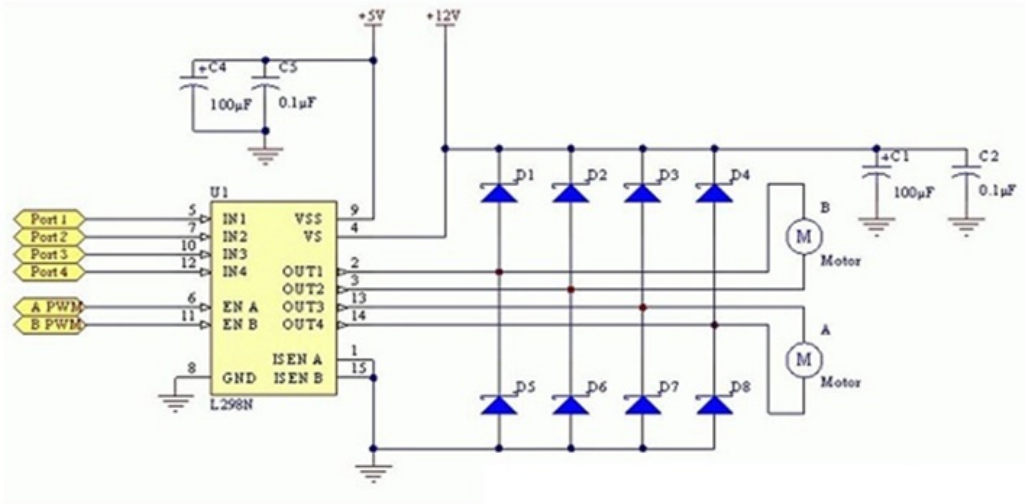
이번 실험에서 우리는 Pull Down 방식을 사용한다.

구현

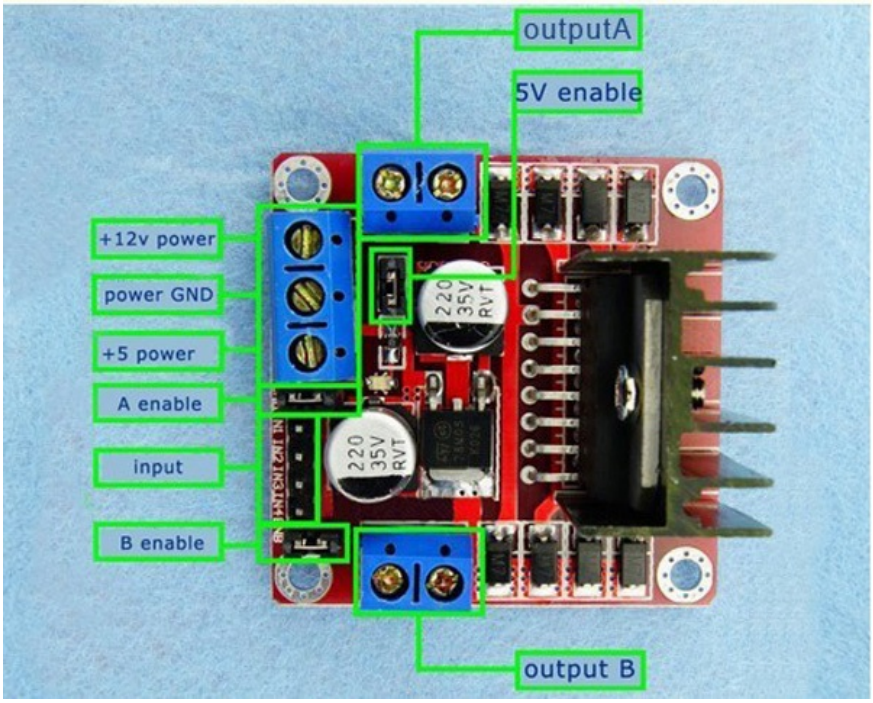
LED 불키기와 조이스틱 탐지에 대해서는 4주차 실험을 참조한다.

Motor

Motor Driver 구조

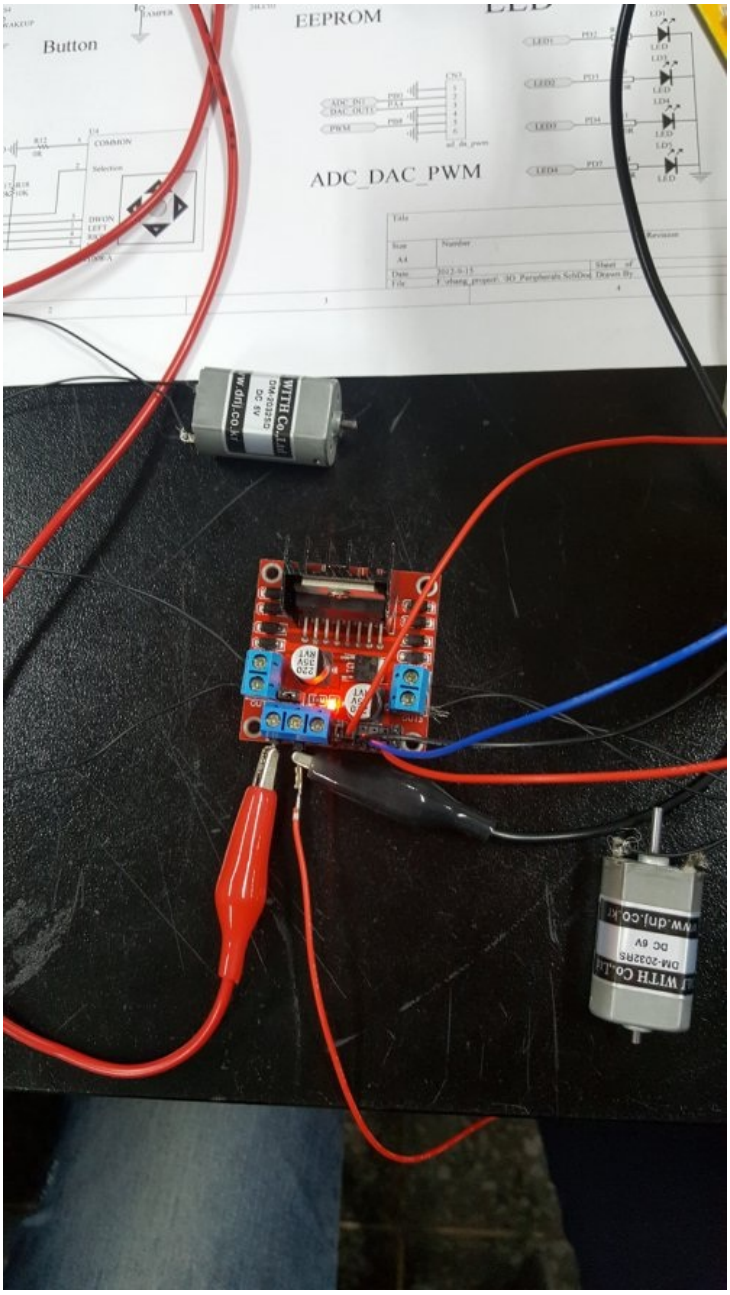


다음은 Motor Driver 회로도이다.



+12v power 에 (+) 전압을 인가하고, power GND 에 (-) 전압을 인가한다. motor 를 output A 와 input1,2 가 연결되어있고 output B 와 input3,4 가 연결되어있다.

Motor Driver 의 IN1, IN2, IN3, IN4 를 Cortex-M3 board 의 PC8, PC9, PC10, PC11에 순서대로 연결.



Motor Driver와 Cortex 연결 모습.

IN1(IN3)	IN2(IN4)	출력A(출력B)
High	Low	정방향
Low	High	역방향
High	High	정지
Low	Low	정지

입력 IN 에 인가된 전압에 따른 Motor 회전 방향을 나타내는 표이다. 이를 참조하여 Motor 회전 방향을 제어한다.

Motor 작동시키기

4주차 실험에서 사용한 조이스틱 제어에 따른 LED 불을 키는 code 에 motor 를 작동 시키는 조건을 추가한다.

motor는 PC8, PC9, PC10, PC11을 사용한다.

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

위의 CRH 레지스터에서 PC8, PC9, PC10, PC11은 MODE8, CNF8부터 MODE11, CNF11까지 순서대로이다.

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

각 비트에 0x3의 값을 주어 Output mode, General purpose output push-pull로 설정 하였다.

```
GPIOC->CRH = (GPIO_CRH_MODE8 | GPIO_CRH_MODE9 | GPIO_CRH_MODE10 | GPIO_CRH_MODE11);
```

GPIOx*BSRR*과 *GPIOx* BRR은 다음과 같다.

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

9.2.6 Port bit reset register (GPIOx_BRR) (x=A..G)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

Bits 15:0 **BRy:** Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

BS8, BS9, BS10, BS11 비트를 set, unset하여 motor를 회전을 조절하고 BR8, BR9, BR10, BR11 비트를 set하여 motor를 정지 시킨다.

motor 회전

```
GPIOC->BSRR = GPIO_BSRR_BS8 | GPIO_BSRR_BS9 | GPIO_BSRR_BS10 | GPIO_BSRR_BS11;
```

motor 정지

```
GPIOC->BSRR = 0X00000000;

GPIOC->BRR = GPIO_BRR_BR8;
GPIOC->BRR = GPIO_BRR_BR10;
GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR10;
GPIOC->BRR = GPIO_BRR_BR9 | GPIO_BRR_BR11;
GPIOD->BRR = GPIO_BRR_BR2;
GPIOD->BRR = GPIO_BRR_BR3;
GPIOD->BRR = GPIO_BRR_BR4;
GPIOD->BRR = GPIO_BRR_BR7;
```

Library 이용

주어진 Library 이용하기

주어진 Library 를 이용하여 위의 code 속 주소와 값을 상수변수로 대체할 수 있다.

```
#define GPIO_BSRR_BS2                    ((uint32_t)0x00000004)
```
예를 들어,
include 부분을 이용하여 다음과 같이 code를 수정할 수 있다.

```
//원래 code
(*(volatile unsigned *)0x40011410) = 0x04;

//Library를 이용하여 수정한 code
GPIOD->BSRR = GPIO_BSRR_BS2;
```

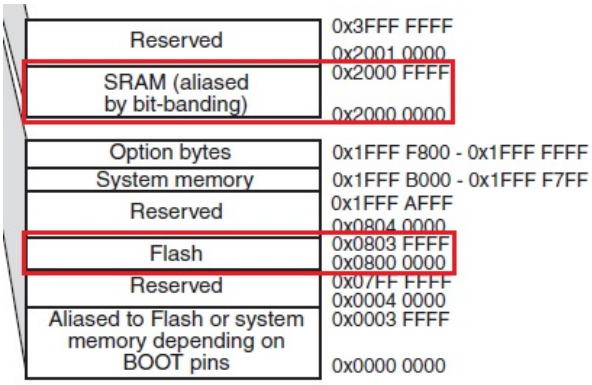
Library를 이용한 전체 Code

```cpp
//주어진 Library 를 include 시킨다
#include "stm32f10x_gpio.h"
#include "stm32f10x.h"

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

void on1()
{
  GPIOD->CRL &= ~(GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 | GPI
O_CRL_MODE7_0);

  GPIOD->CRL |= GPIO_CRL_MODE2_0; // 1
  GPIOD->BSRR = GPIO_BSRR_BS2;
  GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR9 | GPIO_BRR_BR10 | GPIO_BRR_BR11;

  GPIOC->BSRR = GPIO_BSRR_BS8 | GPIO_BSRR_BS10;
  delay();
  GPIOC->BRR = 0x0;
}

void on2()
{
  //  GPIOD->CRL &= ~0x10011100;
  GPIOD->CRL &= ~(GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 | GPI
O_CRL_MODE7_0);
  GPIOD->CRL |= GPIO_CRL_MODE3_0; // 1
  GPIOD->BSRR = GPIO_BSRR_BS3;

  GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR9 | GPIO_BRR_BR10 | GPIO_BRR_BR11;

  GPIOC->BSRR = (GPIO_BSRR_BS8 | GPIO_BSRR_BS11); // 0x900;
  delay();
  GPIOC->BRR = 0x0;
}

void on3()
{
  GPIOD->CRL &= ~(GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 | GPI
O_CRL_MODE7_0);
  GPIOD->CRL |= GPIO_CRL_MODE4_0; // 1
  GPIOD->BSRR = GPIO_BSRR_BS4;

  GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR9 | GPIO_BRR_BR10 | GPIO_BRR_BR11;

  GPIOC->BSRR = (GPIO_BSRR_BS9| GPIO_BSRR_BS10); //0x600;
  delay();
  GPIOC->BRR = 0x0;
}
void on4()
{
  GPIOD->CRL &= ~(GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 | GPI
O_CRL_MODE7_0);
  GPIOD->CRL |= GPIO_CRL_MODE7_0; // 1
  GPIOD->BSRR = GPIO_BSRR_BS7;
  GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR9 | GPIO_BRR_BR10 | GPIO_BRR_BR11;


  GPIOC->BSRR = (GPIO_BSRR_BS9 | GPIO_BSRR_BS11); // 0xa00;
  delay();
  GPIOC->BRR = 0x0;
}


int main() {

  RCC->APB2ENR |= (RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCE
N | RCC_APB2ENR_IOPDEN); //0x3c; // LED portD enable
  GPIOD->CRH = 0x44444444; //
  //        GPIOD->CRL = 0x10011100; // 1

  GPIOD->CRL = (GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 | GPIO_
CRL_MODE7_0);
  GPIOC->CRH = (GPIO_CRH_MODE8 | GPIO_CRH_MODE9 | GPIO_CRH_MODE10 | GPIO_CRH_M
ODE11);

  GPIOC->IDR = 0x00000000;

  while(1) {

```

```
    if(~GPIOC->IDR & (GPIO_IDR_IDR2)) // 0x4)
      on1();
    else if(~GPIOC->IDR & GPIO_IDR_IDR5) // 0x20)
      on4();
    else if(~GPIOC->IDR & GPIO_IDR_IDR3) // 0x8)
      on2();
    else if(~GPIOC->IDR & GPIO_IDR_IDR4) // 0x10)
      on3();
    else {//if ( (GPIOC->IDR & (0xFFFE) != (0xFFFE))){

      GPIOC->BSRR = 0X00000000;

      GPIOC->BRR = GPIO_BRR_BR8;
      GPIOC->BRR = GPIO_BRR_BR10;
      GPIOC->BRR = GPIO_BRR_BR8 | GPIO_BRR_BR10;
      GPIOC->BRR = GPIO_BRR_BR9 | GPIO_BRR_BR11;
      GPIOD->BRR = GPIO_BRR_BR2;
      GPIOD->BRR = GPIO_BRR_BR3;
      GPIOD->BRR = GPIO_BRR_BR4;
      GPIOD->BRR = GPIO_BRR_BR7;
    }


    //    if(~GPIOB_IDR & 0x100)

    }
  }
```

Scatter file

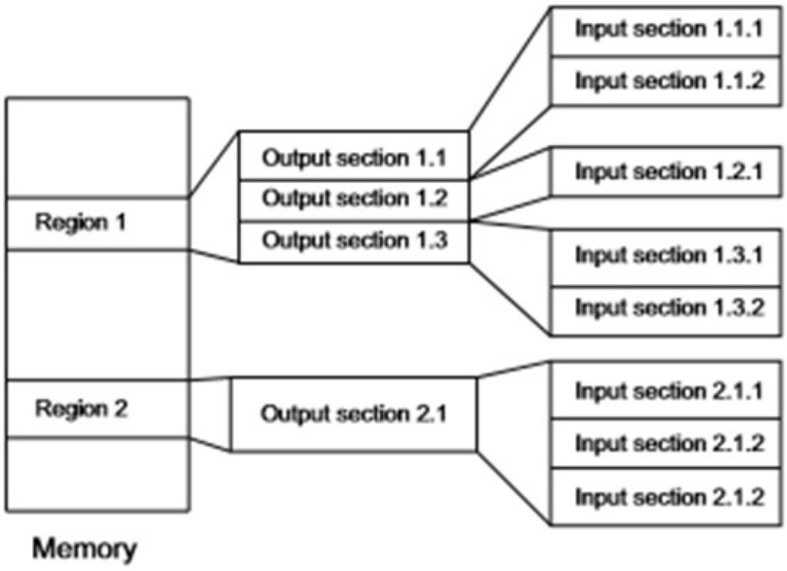
Scatter file 분석



SRAM 과 Flash Memory(ROM)의 Memory address 값

```
//Datasheet를 참고하면, LR_IROM1 0x08000000 0x00008000 에서 뒤에 16 진수 중 첫 번
째는 load section 의 시작주소, 두 번째는 section 의 크기를 말한다
LR_IROM1 0x08000000 0x00008000
{
  ER_IROM1 0x08000000 0x00008000
  {
    *.o(RESET, +First)        //*.o(RESET, +First)은 모든 .o 코드를 RESET Sectio
n 의 처음(First)에 위치시킨다
     *(InRoot$$Sections)        //ARM Library section 을 root section 에 자동
배치시킨다
      .ANY (+RO)        //나머지 code 와 RO data 를 ROM에 위치시킨다
  }
  RW_IRAM1 0x20000000 0x00008000  //이건 RAM Section 이므로 위의 두개와 달리 SRAM
의 시작주소 0x20000000 를 사용
  {
    .ANY (+RW +ZI)        //RW 와 ZI data section 을 RAM 에 위치시킨다
  }
}
```

Scatter file 이용 목적



여러 개의 입력 section을 출력 section으로 "그룹화" 하고 memory map에 효율적으로 "배치"하기 위해서 사용한다.

결론

저번 실험에서 LED와 나머지 설정을 다 한 상태여서 대체로 무난하게 진행할 수 있었다. 하지만 Motor Driver 에 연결할 선이 많아 관리하기 힘들어 그부분에서 시간을 많이 끌었다. 그래도 Motor Driver 가 있어서 추가 작업 없이 선 연결과 각 해당 Port 설정만으로 motor를 동작시킬 수 있었다. Scatter 파일 또한 처음 접한 것이지만 알아서 입력 섹션의 값을 그룹화 시켜줘서 도움이 되었다. 그리고 인터럽트와 폴링 방식의 차이를 정확히 이해할 수 있었고 다양한 버튼 방식(Floating, Pull-up, Pull-down)의 종류를 알게 되었다.