REPORT

임베디드 실습 및 실험(002) 4주차 실험 결과보고서

10조

전기컴퓨터공학부	201824446	김윤재
정보컴퓨터공학부	202055558	송세연
정보컴퓨터공학부	2020555889	임연후
바이오소재과학과	201845626	최이한

2022.09.27

◆ 실험 목표

조이스틱과 버튼 입력을 통한 릴레이 모듈 제어와 모터 회전

입력(조이스틱)		출력(모터)
DOWN	→	모터 시계방향 회전
UP	→	모터 반시계방향 회전

입력(버튼)		출력(모터)
S1	→	모터 정지

◆ 실험 과정

1. 제공된 Scatter 파일을 완성한 뒤 프로젝트에 업로드한다.

   Scatter File이란?

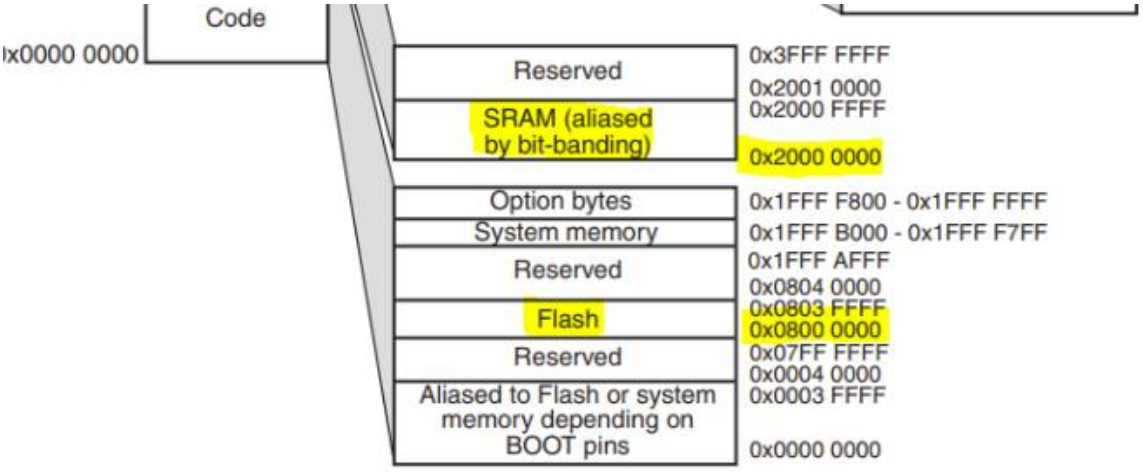
   분산 **적재**, 실행시킬 바이너리 이미지가 메모리에 로드될 때, 바이너리 이미지의 어떤 영역이 **어느 주소에 어느 크기만큼 배치되어야** 할 지 작성한 파일.

   <필요한 이유>

   1. 바이너리의 여러 부분을 각각 별개의 메모리 영역에 로드해야될 때
   2. 자주 사용되거나 빠른 실행을 요구하는 코드 영역을 접근시간이 빠른 메모리에 우선 배치하도록 설정할 수 있음.

```
define symbol __ICFEDIT_region_ROM_start__ = // TODO
define symbol __ICFEDIT_region_ROM_end__   = // TODO
define symbol __ICFEDIT_region_RAM_start__ = // TODO
define symbol __ICFEDIT_region_RAM_end__   = // TODO
```

   ROM 크기는 0x80000, RAM 크기는 0x8000로 주어졌으므로 ROM과 RAM의 시작주소를 Datasheet에서 찾으면 된다.

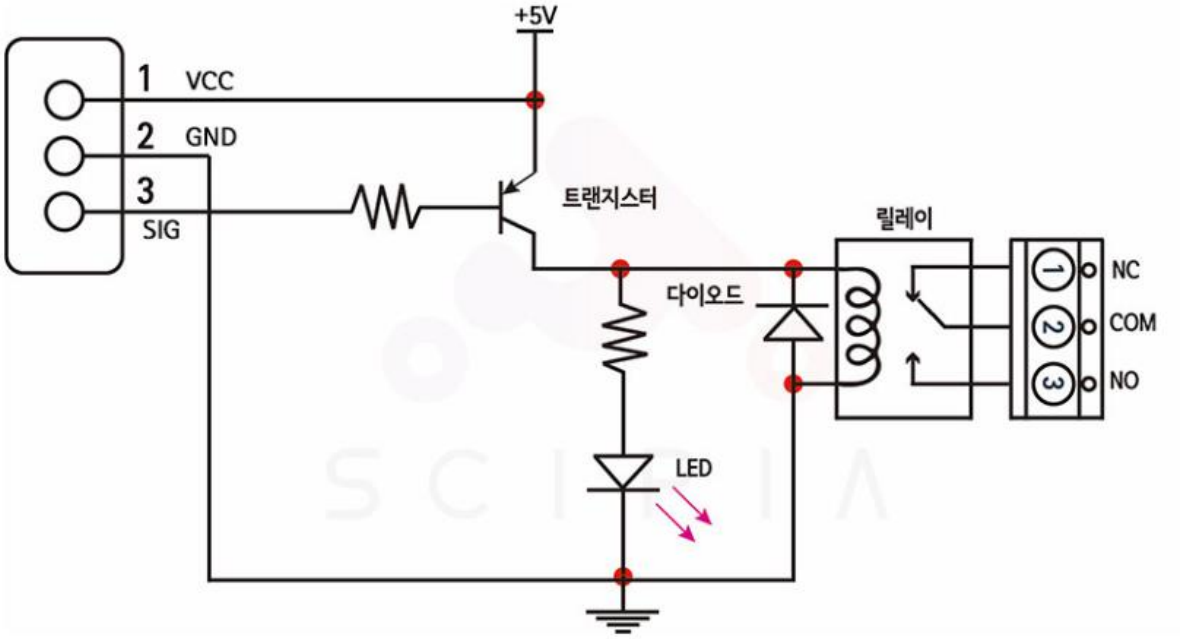


결과적으로

define symbol __ICFEDIT_region_ROM_start__ = 0x08000000
define symbol __ICFEDIT_region_ROM_end__ = 0x08080000
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000
define symbol __ICFEDIT_region_RAM_end__ = 0x20008000

2. 릴레이모듈 & DC모터 회로 구성

릴레이 모듈: 전자기 유도 원리를 이용해 스위치를 제어하는 모듈.

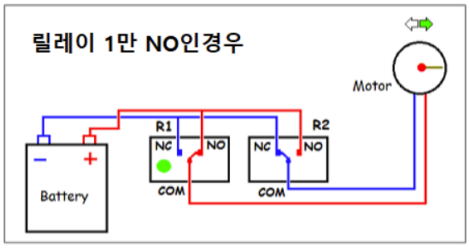


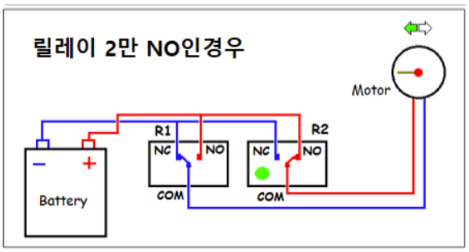
- NC**(Normally Close)** : 평상시에 닫혀있다(on). 릴레이에 전류가 흐르면(=데이터 핀의 value가 1이면) on→off
- **NO(Normally Open)** : 평상시에 열려있다(off). 릴레이에 전류가 흐르면 off→on

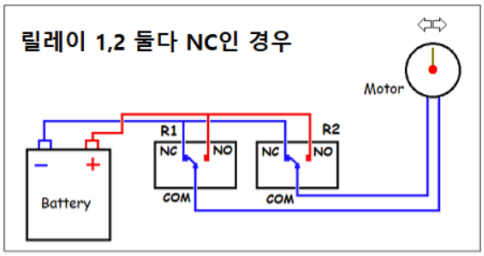
- **COM(Common port) :** 공통단자, 항상 전력 또는 외부 기기의 한쪽 선을 연결해야 함.

DC 모터와의 연결: DC 모터는 한쪽 단자가 VCC, 다른 쪽 단자가 GND로 설정되면 동작한다. 두 핀의 방향을 반대로 설정하면 역방향으로 돈다

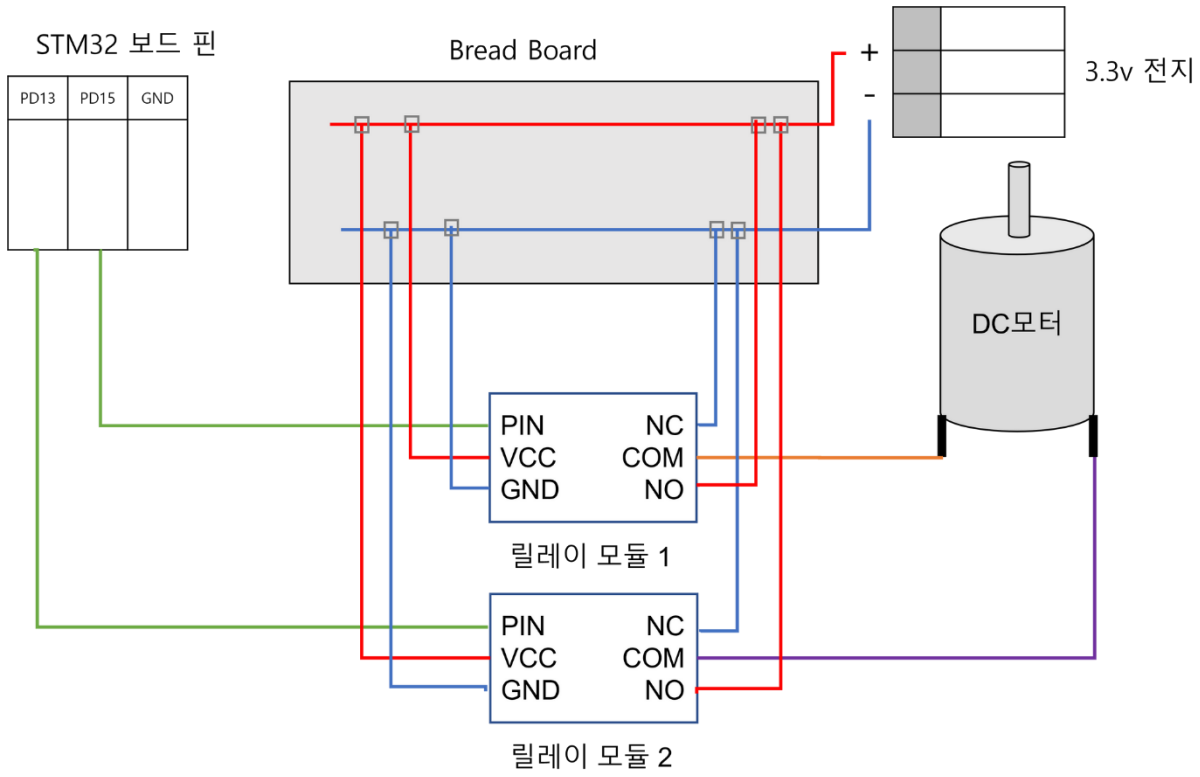
➔ 릴레이 모듈의 상태(high/low)에 따라 모터에 공급되는 전원의 극성을 반대로 바꿔주는 것으로 방향 전환 구현







입력	포트&핀 번호	출력(보드핀)	결과
조이스틱 DOWN	PC5	PD13 리셋, PD15 셋	모터 반시계 회전
조이스틱 UP	PC2	PD13 셋, PD15 리셋	모터 시계 회전
버튼 S1	PC11	PD13 리셋, PD15 리셋	모터 정지



3. stm32_Datasheet.pdf 파일을 참고하여 RCC 및 IO 포트로 사용될 포트 C, 포트 D의 base address 값을 구한다.

RCC의 base address : 0x4002 1000

Port C의 base address : 0x4001 1000

Port D의 base address : 0x4001 1400

AHB	Ethernet	0x4002 8000 - 0x4002 9FFF
	Reserved	0x4002 3400 - 0x4002 7FFF
	CRC	0x4002 3000 - 0x4002 33FF
	Reserved	0x4002 2400 - 0x4002 2FFF
	Flash interface	0x4002 2000 - 0x4002 23FF
	Reserved	0x4002 1400 - 0x4002 1FFF
	RCC	0x4002 1000 - 0x4002 13FF
	Reserved	0x4002 0800 - 0x4002 0FFF
	DMA2	0x4002 0400 - 0x4002 07FF
	DMA1	0x4002 0000 - 0x4002 03FF
APB2	Reserved	0x4001 3C00 - 0x4001 FFFF
	USART1	0x4001 3800 - 0x4001 3BFF
	Reserved	0x4001 3400 - 0x4001 37FF
	SPI1	0x4001 3000 - 0x4001 33FF
	TIM1	0x4001 2C00 - 0x4001 2FFF
	ADC2	0x4001 2800 - 0x4001 2BFF
	ADC1	0x4001 2400 - 0x4001 27FF
	Reserved	0x4001 1C00 - 0x4001 23FF
	Port E	0x4001 1800 - 0x4001 1BFF
	Port D	0x4001 1400 - 0x4001 17FF
	Port C	0x4001 1000 - 0x4001 13FF
	Port B	0x4001 0C00 - 0x4001 0FFF
	Port A	0x4001 0800 - 0x4001 0BFF
	EXTI	0x4001 0400 - 0x4001 07FF
	AFIO	0x4001 0000 - 0x4001 3FFF
	Reserved	0x4000 7800 - 0x4000 FFFF

RCC, Port C, Port D의 base address

4. RCC를 사용하여 사용하고자 하는 GPIO(포트&핀)에 clock을 인가한다.

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

RCC 레지스터의 사용방법

C, D 포트이므로 4, 5번째 비트를 1로 설정해줌으로써 clock을 인가해준다.

0000 0000 0000 0000 0000 0000 0001 0000⇒C포트
0000 0000 0000 0000 0000 0000 0010 0000⇒D포트
*(RCC의 base address+offset)=0x40021000+0x18=0x30

5. 사용하려는 GPIO Port, Pin input/output을 설정한다.(=Port Configuration)

- C포트의 2, 5번 핀을 CRL 레지스터를 사용해 input configuration한다
- D포트의 11번 핀을 CRH레지스터를 사용해 input configuration한다.
- D포트의 13, 15번 핀을 CRH레지스터를 사용해 output configuration한다.

레퍼런스에 나와있는 대로, 입력포트인 C포트에 대해, 사용할 핀 번호 위치에서 Mode는 00, CNF는 10(조이스틱 입력은 pull-up 방식을 쓰므로)을 넣어준다.

출력포트인 D포트에 대해, 마찬가지로 사용할 핀 번호 위치에서 Mode는 가장 빠른 주파수를 가지는 11, CNF는 표준 출력인 00을 넣어준다.

0000 0000 **10**00 0000 0000 **10**00 0000 0000 ⇒ **0x00800800**
00**11** 0000 00**11** 0000 **10**00 0000 0000 0000 ⇒ **0x30308000**
*(C포트의 base address+offset)=0x4001100+0x00=0x00800800
*(D포트의 base address+offset)=0x4001100+0x00=0x30308000

6. GPIO의 Input(IDR), output(BSRR&BRR)을 통해 입출력 장치를 제어한다.

- Input

9.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy:** Port input data (y= 0 .. 15)
These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

IDR 레지스터의 사용 방법

조이스틱 입력은 Pull-up 방식이기 때문에, 입력이 들어오지 않을 때엔 비트 값이 1이었다가 입력이 들어올 때 해당 핀 번호에 해당하는 비트 값이 0이 된다.

이 점에 유의하여 if문 코드를 짜면 특정 핀번호에 입력이 들어올 때를 다음과 같이 처리할 수 있다.

```
If ( !(*GPIOC_IDR & BIT_5) )      //5번 핀, 즉 조이스틱 up 입력이 들어왔을 때
```

GPIOC_IDR은 C포트의 IDR 레지스터 주소, BIT_5는 0x20(=0b100000)

특정 핀의 value를 set/reset하고자 할 때 해당 핀 번호 위치의 bit 값을 1로 설정해주면 된다.

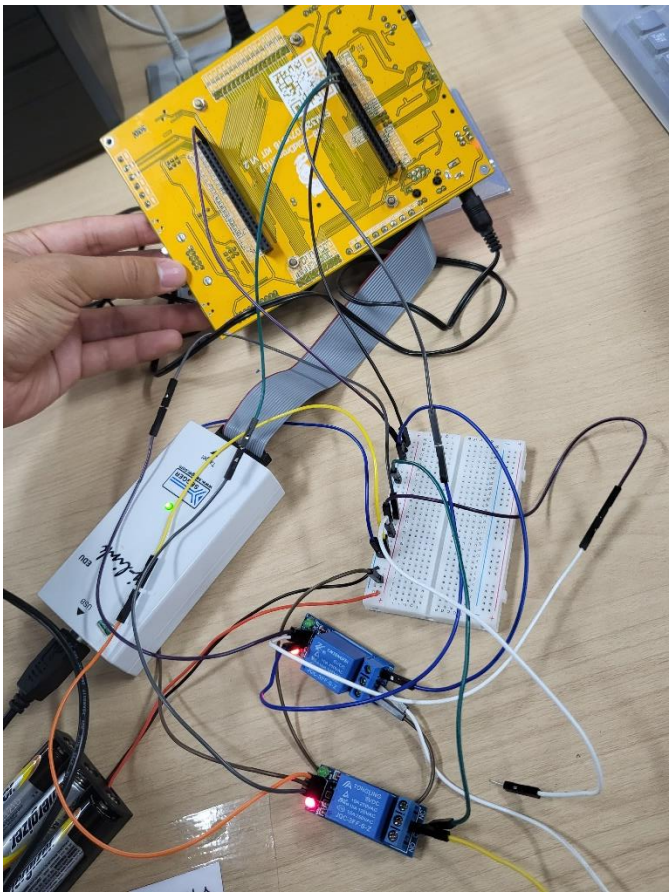
```
*GPIOD_BSRR |= 0x2000;     //13번 핀 set
*GPIOD_BRR |= 0x2000;     //13번 핀 reset
*GPIOD_BSRR |= 0x8000;      //15번 핀 set
*GPIOD_BRR |= 0x8000;      //15번 핀 reset
```

GPIOD_BSRR, GPIOD_BRR은 D포트의 BSRR, BRR 레지스터의 주소

✧ 혹시 모를 오작동을 피하기 위해 코드가 실행되기 전 모든 output 핀의 value를 reset해준다.

```
*GPIOD_BRR |= (BIT_15|BIT_13);
```

◆ 실험 결과



실제 구성한 회로도

◆ 추가) 릴레이 모듈 하나만을 사용한 방식

각각 NC와 NO는 각각 DC모터의 양 극 단자에 연결해준다. 임의의 output pin PD13를 COM에 연결해주고, 임의의 output pin PD15를 릴레이 모듈의 외부입력 단자(pin)에 연결해준다.(나머지 회로는 동일)

PD13를 reset하면 NC와 NO 둘 다 low로, 모터가 정지한 상태이고, PD13를 set하면 가장 처음에 NC가 high, NO가 low인 상태이므로 특정 방향으로 돌게 된다. 이 상태에서 PD15의 값을 set해주게 되면 접점이 바뀌어 NC가 low, NO가 high가 되므로 기존 방향에서 역방향으로 회전하게 된다.

이를 정리해서 case문만 수정하면 다음과 같이 구현할 수 있다

```
...
int main()
{
  *RCC_APB2 |= RCC_APB2_IOPCEN | RCC_APB2_IOPDEN;
  *GPIOC_CRL &= ~0x00F00F00;
  *GPIOC_CRL |= 0x00800800;
  *GPIOD_CRH &= ~0xF0F0F000;
  *GPIOD_CRH |= 0x30308000;
  *GPIOD_BRR |= (BIT_15|BIT_13);

  int i = 0;
  while(1) {
        //8번 셋
    if(!(*GPIOC_IDR & BIT_5))
          i=1;
    if(!(*GPIOC_IDR & BIT_2))
          i=2;
    if(!(*GPIOD_IDR & BIT_11))
          i=3;
    switch(i)
    {
    case 1:    //반시계
     *GPIOD_BSRR |= 0x2000;     //13번 셋
      *GPIOD_BSRR |= 0x8000;     //15번 셋
      break;
    case 2:     //시계
      *GPIOD_BSRR |= 0x2000;     //13번 셋
       *GPIOD_BRR |= 0x8000;     //15번 리셋
      break;
    case 3:     //정지
     *GPIOD_BRR |= 0x2000;     //13 리셋
      break;
    }
  }
 }
```