REPORT

제목 :4주차 실험보고서

수강과목 :임베디드시스템설계및실험

담당교수 :백윤주 교수님

조 이름 : 2조

이 름 :박건우, 노윤정 ,이동근, 조영진

제출일자 :2022-09-27

1. 실험 목적

1) 스케터 파일의 이해 및 플래시 프로그래밍
2) 릴레이 모듈의 이해 및 임베디드 펌웨어를 통한 동작
3) 센싱에서 폴링 방식의 이해

2. 실험 미션

1) 스케터 파일을 통해 원하는 메모리 위치에 프로그램 다운로드 확인
2) 조이스틱 UP -> 모터 시계 반시계 방향 회전
3) 조이스틱 DOWN -> 모터 시계 방향 회전
4) 스위치 PUSH -> 모터 정지

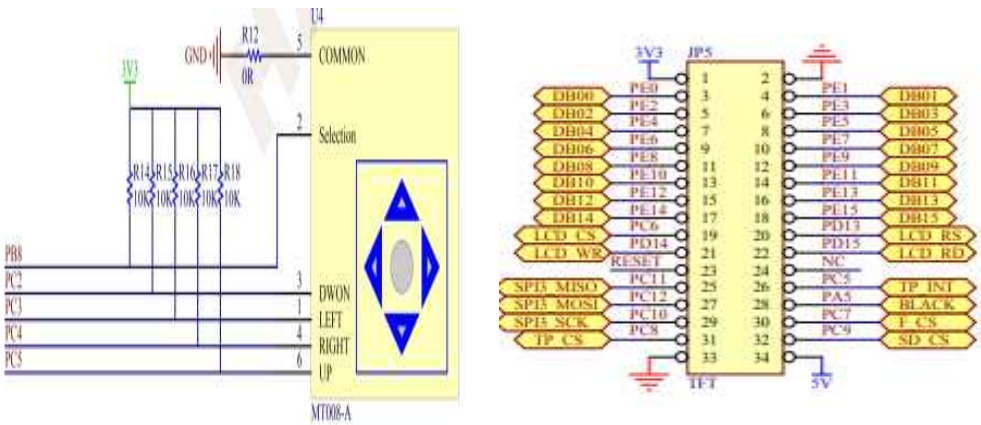
3. 주의 사항

1) 모터 방향 변환 시 과전압 예방을 위한 delay() 함수 삽입
2) 동작 중 케이블 뽑는 행위 금지
3) 실험 장비들의 연결 및 분리 시 모든 전원을 끈 상태로 연결

4. 실험 과정

▶ 사용할 GPIO 목록

① Joystick Down / [Port C, pin 2]
② Joystick Up / [Port C, pin 5]
③ Switch / [Port D, pin 11]
④ Relay 1 / [Port E, pin 0]
⑤ Relay 2 / [Port E, pin 2]



▶ 사용할 GPIO register address 정의

RCC_APB2ENR: peripheral clock enable register

CRL: Port configuration register low

IDR: Input Data register

BRR: Port bit reset register

BSRR: Port bit set/reset register

Register Address: (register address) = (base address) + (register offset)

```
#include "stm32f10x.h"

#define RCC_BASE_ADDR 0x40021000
#define RCC_OFFSET 0x18
#define RCC_ADDR (*(volatile unsigned int*)(RCC_BASE_ADDR + RCC_OFFSET))

#define GPIOC_BASE_ADDR 0x40011000
#define GPIOC_CRL *(volatile unsigned int*) (GPIOC_BASE_ADDR + 0x00)
#define GPIOC_IDR *(volatile unsigned int*) (GPIOC_BASE_ADDR + 0x08)

#define GPIOD_BASE_ADDR 0x40011400
#define GPIOD_CRL *(volatile unsigned int*) (GPIOD_BASE_ADDR + 0x00)
#define GPIOD_CRH *(volatile unsigned int*) (GPIOD_BASE_ADDR + 0x4)
#define GPIOD_IDR *(volatile unsigned int*) (GPIOD_BASE_ADDR + 0x08)
#define GPIOD_BSRR *(volatile unsigned int*) (GPIOD_BASE_ADDR +0x10)
#define GPIOD_BRR *(volatile unsigned int*) (GPIOD_BASE_ADDR + 0x14)

#define GPIOE_BASE_ADDR 0x40011800
#define GPIOE_CRL *(volatile unsigned int*) (GPIOE_BASE_ADDR + 0x00)
#define GPIOE_BSRR *(volatile unsigned int*) (GPIOE_BASE_ADDR +0x10)
#define GPIOE_BRR *(volatile unsigned int*) (GPIOE_BASE_ADDR + 0x14)
```

▶ GPIO에 인가하기 위한 CLOCK 및 Port C, D, E와 그 Port 들의 필요한 레지스터 활성화 정의

```
void delay(){
  int i;
  for(i = 0; i< 1000000; i++){}
}
```

▶ 주의 사항에 명시된 delay 함수 정의

```
int main(void) {

  RCC_ADDR |= 0x70; // set clock 0111 0000

  GPIOC_CRL &= ~0x00FFFF00; // joy stick
  GPIOC_CRL |= 0x00888800;

  GPIOD_CRH &= ~0x000F0000; // switch
  GPIOD_CRH |= 0x00080000;

  GPIOE_CRL &= ~0x00000F0F; // relay
  GPIOE_CRL |= 0x00000303;
```

▶ RCC_ADDR : Port C, D, E에 Clock 인가

▶ 사용할 GPIO의 Joystick[Port C], Switch[Port D], {Relay 1, Relay 2} [Port E] 활성화

↳ 임의로 선정한 Port E의 Pin 0과 2를 각 Relay의 IN에 연결

```
  while(1) {
    if(!(GPIOC_IDR&0x04)){
      GPIOE_BSRR |= 0x01;
      GPIOE_BRR |= 0x04;
    }
    else if(!(GPIOC_IDR&0x20)) {
      GPIOE_BSRR |= 0x04;
      GPIOE_BRR |= 0x01;
    }
    if(!(GPIOD_IDR&0x0800)) GPIOE_BRR |= 0x05;
    delay();
  }
```

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

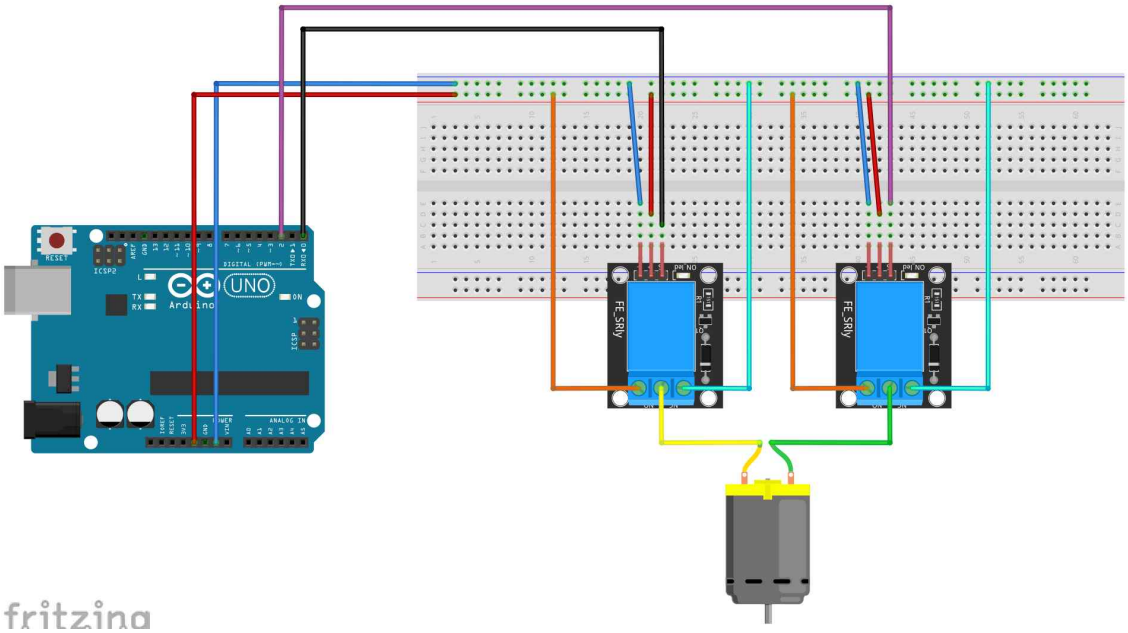
▶ Port C에는 Joystick과 연동돼있고, Port C의 필요한 Down, Up의 Pin Number는 각각 Pin 2와 Pin 5이고, 그 핀의 동작에 해당하는 값을 입력 제어 표에서 살펴보면 각각 0x4와 0x20이다.

▶ 위의 반복문 while 밑의 첫 if문에서 Port C(Joystick)의 입력값이 Down일 때, Port E의 Pin 0(Relay 1)은 BSRR에 의해 활성화되고, Pin 2(Relay 2)는

BRR 에 의해서 비활성화된다.

else if 구문에서 Port C 의 입력값이 Up일 때, Port E의 Pin 0은 BRR에 의해 비활성화 되고, Port E의 Pin 2는 BSRR에 의해 활성화 된다.

두번째 if 구문에서 Port D의 Pin 11(Switch)가 Pull-down 상태일 경우, Port E의 Pin 0와 Pin 2가 BRR에 의해 비활성화 된다.



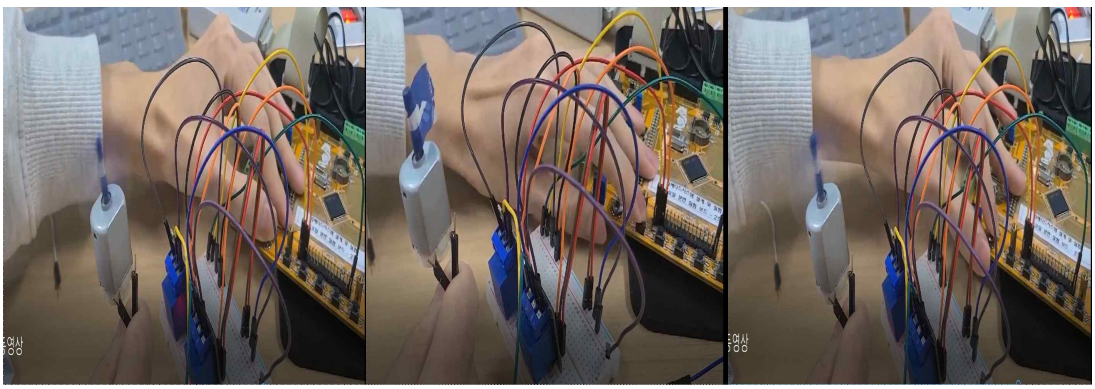
▶ 사용한 회로도의 모습이다. 프로그램이 보드의 모형을 지원하지 않아 부득이하게 Arduino Uno 모형을 사용하였다. 각 Relay의 VCC 부분에 보드의 5V 전원을 인가, GND 부분에 보드의 Ground 포트를 연결해주었고, Relay 1의 IN 에는 Port E Pin 0, Relay 2의 IN에는 Port E Pin 2를 연결해주었다. Motor에도 전원을 따로 인가해주어야 하므로, Relay 각각 NO에 +를, NC에 -를 연결해주었다. 그 후 COM에 Motor를 각각 연결해주었다.

▶ 각각의 Relay 는 IN의 값이 pull-down 일 경우 COM이 NC와 연결되고, IN의 값이 pull-up 일 경우 COM이 NO와 연결되게 된다. 따라서 Relay 1만 pull-up을 주고, Relay 2에는 pull-down을 주게 된다면 모터의 노란색 선에는 NO와 연결된 +가, 초록색 선에는 NC와 연결된 -가 연결되게 된다. 따라서 모터는 시계방향을 회전한다. Relay 1에 pull-down을 주고, Relay 2에

pull-up을 주게된 경우에는 반대로 노란색 선에 NC와 연결된 -가, 초록색 선에는 NO와 연결된 +가 연결되게 되므로 모터는 반시계 방향으로 회전한다. Relay 1과 2 모두 IN에 pull-up을 주게 되면 모터 양쪽에 +가 걸리게되고, 반대로 양쪽 모두 pull-down을 주게 되면 모터 양쪽에 -가 걸리게 된다. 따라서 해당 경우에는 모터가 멈추게 된다.

5. 실험 결과

초기 상태에선 Motor가 멈추어 있다가, Joystick Down 상태가 되면 Motor가 시계방향으로 회전하고, Joystick Up 상태가 되면 Motor가 반시계방향으로 회전한다. 어떠한 상태든, Switch를 누르는 순간 Motor는 정지하게 된다.



6. 애로사항

1) GPIO와 Relay의 직접 연결로 처음에는 시도하였으나, 실패하여 브레드보드와 GPIO를 먼저 연결 후 브레드보드와 Relay를 연결하니 동작하지 않는 문제가 해결되었다.

2) 처음에 모터의 정방향과 역방향의 방법을 GPIO에서 찾았으나, 모터의 동작 원리를 생각해보고, 모터에 걸리는 전극을 서로 바꾸면 동작 방향도 변할 것이라는 생각에 Relay를 2개 사용하여 극을 바꿀 수 있게 설계하니 정상적으로 정방향과 역방향으로 동작했다.

3) 실험 결과를 만족시키는 설계를 실시하였으나, 정상적으로 동작하지 않아 고민하였으나, 단순한 전압 문제로 3.3V 대신 5V에 연결하니 정상적으로 작동하였다.

7. 고찰

이론과 실제 설계 및 실험은 난이도 차가 많이 나는 것을 느꼈고, 사소한 에러 (전압문제, 회로) 등이 실험 시간에 많은 영향을 끼쳤고, 실험마다 주의를 기울이며 임해야 함을 느꼈다.