임베디드 시스템 설계 및 실험
화요일 분반 3주차 실험 결과 보고서

조 : 6조
조원 : 이주승 김선규 이동현 이지현 최세희

실험목표

1. 임베디드 시스템의 기본 원리 습득
2. 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발 이해

세부 목표

1. Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
2. IAR EW에서 프로젝트 생성 후 관련 설정 변경
3. 조이스틱을 이용한 LED 제어

실험 결과

1. 레지스터 선언 및 설정

```
#define RCC_APB2_ENR *(volatile unsigned int *)0x40021018
#define GPIOB_CRH *(volatile unsigned int *)0x40010C04
#define GPIOC_CRL *(volatile unsigned int *)0x40011000
#define GPIOD_CRL *(volatile unsigned int *)0x40011400
#define GPIOD_BSRR *(volatile unsigned int *)0x40011410
#define GPIOC_IDR *(volatile unsigned int *)0x40011008
#define GPIOB_IDR *(volatile unsigned int *)0x40010C08
#define GPIOD_ODR *(volatile unsigned int *)0x4001140C

int main() {

  RCC_APB2_ENR = 0x00000038; // IO Port B, C, D Enable

  GPIOB_CRH = 0x44444444; // GPIO B Reset
  GPIOC_CRL = 0x44444444; // GPIO C Reset
  GPIOD_CRL = 0x44444444; // GPIO D Reset

  GPIOB_CRH = 0x44444448; // PB8 Set
  GPIOC_CRL = 0x44888844; // PC2, PC3, PC4, PC5 Set
  GPIOD_CRL = 0x34433344; // PD2, PD3, PD4, PD7 Set
  GPIOD_BSRR = 0x00000000; // Port bit Reset
```

① RCC_APB2_ENR의 주소는 RCC의 base 주소 0x40021000에 offset 주소 0x18을 더하여 구한다. 다른 주소들도 같은 방법으로 계산하여 선언한다.

② Schematic에서 사용하려는 Port가 B, C, D임을 확인할 수 있으므로 RCC_APB2_ENR의 값을 변경하여 각 GPIO Port를 Enable한다.

③ Schematic에서 사용하려는 핀의 번호를 확인한다. GPIO B에서 8번, GPIO C에서 2, 3, 4, 5번, GPIO D에서 2, 3, 4, 7번이므로 각각 GPIOB_CRH, GPIOC_CRL, GPIOD_CRL을 통해 값을 설정해 준다.

④ Output으로 사용될 GPIO D는 GPIOD_BSRR을 통해 각 비트에 Set과 Reset을 설정하여야 하므로 선언한 GPIOD_BSRR에 초깃값을 설정해준다.

2. 동작

① UP

```
if (GPIOC_IDR == 0x0000FFDF) { // UP
  GPIOD_BSRR |= 0x00000090; // PD4, PD7 LED ON
  isSelect = 0;
}
```

GPIOC_IDR을 이용하여 GPIO C의 어떤 Input이 들어왔는지 확인할 수 있다.
Pull-up 저항을 사용하고 있으므로 초깃값은 0x0000FFFF이고 Input이 들어온 Bit는 0이 될 것이다. 조이스틱 UP에 해당하는 PC5가 Input으로 들어왔다면 5번 Bit가 0이 될 것이므로 GPIOC_IDR의 값은 0x0000FFDF이다. 이 점을 이용하여 조건문을 만든다.
LED를 켜기 위해서 GPIOD_BSRR을 사용한다. PD4, PD7을 켜려면 4번 Bit와 7번 Bit에 1을 할당해줘야 한다. 계산하여 0x00000090을 할당해준다.
DOWN, LEFT, RIGHT도 위와 같은 방법으로 조건문을 만들어준다.

② DOWN

```
else if (GPIOC_IDR == 0x0000FFFB) { // DOWN
  GPIOD_BSRR |= 0x00900000; // PD4, PD7 LED OFF
  isSelect = 0;
}
```

조이스틱 DOWN에 해당하는 PC2가 Input으로 들어오면 2번 Bit가 0이므로 GPIOC_IDR의 값은 0x0000FFFB이다.
LED를 끄기 위해서는 GPIOD_BSRR의 Reset 영역인 상위 16개 Bit를 Set에서와 같은 방법으로 할당해주면 된다. PD4와 PD7을 끄기 위해 20번 Bit와 23번 Bit에 1을 할당해준다.

③ LEFT

```
else if (GPIOC_IDR == 0x0000FFF7) { // LEFT
  GPIOD_BSRR |= 0x0000000C; // PD2, PD3 LED ON
  isSelect = 0;
}
```

조이스틱 LEFT에 해당하는 PC3가 Input으로 들어오면 3번 Bit가 0이므로 GPIOC_IDR의 값은 0x0000FFF7이다.
PD2와 PD3을 켜기 위해 2번 Bit와 3번 Bit에 1을 할당해준다.

④ RIGHT

```
else if (GPIOC_IDR == 0x0000FFEF) { // RIGHT
  GPIOD_BSRR |= 0x000C0000; // PD2, PD3 LED OFF
  isSelect = 0;
}
```

조이스틱 RIGHT에 해당하는 PC4가 Input으로 들어오면 4번 Bit가 0이므로 GPIOC_IDR의 값은 0x000FFEF이다.
PD2와 PD3을 끄기 위해 18번 Bit와 19번 Bit에 1을 할당해준다.

⑤ SELECT

```
else if ( (((~GPIOB_IDR) << 16) & 0x01000000) && !isSelect ) { // SELECT
  if (GPIOD_ODR & 0x00000004) { // IF PD2 LED ON
    GPIOD_BSRR |= 0x00040000; // PD2 LED OFF
  }
  else { // IF PD2 LED OFF
    GPIOD_BSRR |= 0x00000004; // PD2 LED ON
  }

  if (GPIOD_ODR & 0x00000008) { // IF PD3 LED ON
    GPIOD_BSRR |= 0x00080000; // PD3 LED OFF
  }
  else { // IF PD3 LED OFF
    GPIOD_BSRR |= 0x00000008; // PD3 LED ON
  }

  if (GPIOD_ODR & 0x00000010) { // IF PD4 LED ON
    GPIOD_BSRR |= 0x00100000; // PD4 LED OFF
  }
  else { // IF PD4 LED OFF
    GPIOD_BSRR |= 0x00000010; // PD4 LED ON
  }

  if (GPIOD_ODR & 0x00000080) { // IF PD7 LED ON
    GPIOD_BSRR |= 0x00800000; // PD7 LED OFF
  }
  else { // IF PD7 LED OFF
    GPIOD_BSRR |= 0x00000080; // PD7 LED ON
  }
  isSelect = 1;
}
```

조이스틱의 Select에 해당하는 PB8이 Input으로 들어오면 8번 Bit가 0이므로 해당 Bit를 검사하는 조건문을 만든다.
현재 어떤 LED가 켜져 있고 어떤 LED가 꺼져 있는지 확인하기 위하여 GPIOD_ODR을 이용한다. 각 LED를 위한 조건문 4개를 만든다.
프로그램이 끝나지 않고 계속 실행되게 하고자 위 동작들을 무한루프 안에 정의하였는데, Select를 누르는 아주 짧은 시간 동안 반복문이 무수히 많이 수행되고, 그에 따라 모든 LED가 켜졌다 꺼짐을 반복하여 맨눈으로 보기에는 LED가 계속 켜져 있는 것처럼 보이게 된다. 이러한 현상을 방지하기 위해 isSelect라는 변수를 선언하고, 한번 Select를 누르면 값을 1로 두어 반복문 동안 동작이 여러 번 수행되는 것을 막는다.

한계

```
void delay(int n) {
  time_t cur = clock();
  while (clock() - cur < n) {
    ;
  }
}
```

- 현재 구현한 Select는 연속해서 Select를 두 번 눌렀을 때 두 번째 동작이 수행되지 않는다. 원래는 위 그림과 같이 delay 함수를 사용하려 했으나 테스트해 본 결과 delay 함수가 실행되면 프로그램이 멈춘다. 원인을 밝히지 못하여 불가피하게 현재와 같이 구현하였다.