

박기홍_07_25_과제보고서

MCU 1일차

작성자 : 박기홍.

업데이트 : 2024.08.01.

과제 1

플로팅(Floating) 상태에 대하여 설명하고, 플로팅 상태가 문제가 되는 원인과 기술을 서술하시오.

설명

전압이 0V와 5V 사이를 왔다갔다 하는 상태를 플로팅 상태(= 떠 있는 상태)라고 합니다.

원인

입력값을 입력하지 않았을 때, 플로팅 상태(0V에서 ~ 5V 사이로 왔다 갔다 하게)로 만드는 것이 플로팅 현상 발생의 원인입니다.

기술

저항을 앞과 뒤에 포함 시켜 전류를 제어한다면, 플로팅 상태를 해결할 수 있습니다.

저항을 포함 시키는 풀업, 풀 다운 상태로 나뉩니다.

1. 풀업(pull-up)

입력 핀과 전원 사이의 저항을 연결합니다.

스위치가 열려진 상태인 경우 입력 핀이 전원과 연결이 되어 전압이 5V가 되고, 플로팅 현상을 해결하게 됩니다.

2. 풀다운(pull-down)

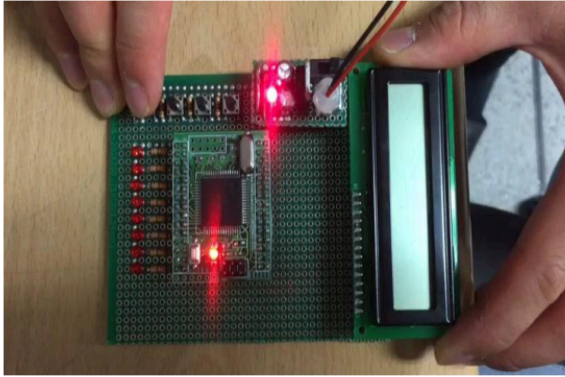
스위치가 눌리면 입력 핀의 전압은 그라운드와 동일하게 0V가 됩니다.

과제 2

HW_002

 과제 설명

과제 2



1. 0.5s 마다 모든 LED 깜빡이기
_delay_ms() 함수를 사용할것

2. SW1이 눌리면 4~7 LED 켜기
3. SW2이 눌리면 0~3 LED 켜기
4. 둘 다 눌리면 모두 켜기

No Ext Interrupt

5. INT3 발생시 LED 좌측 이동
6. INT4 발생시 LED 우측 이동

Ext Interrupt

[과제 풀이]

Github repo : [바로가기](#)

과제 시연 영상 : MCU-1 일차-과제2-시연영상.mp4

기능 모음

<1번 기능>

0.5s 마다 모든 LED 깜빡이기.
_delay_ms() 함수를 사용할 것.

No ext interrupt

<2번 기능>


SW1이 눌리면 4~7 LED 켜기.


<3번 기능>

SW2가 눌리면 0~3 LED 켜기.


<4번 기능>

둘 다 눌리면 모두 키기

 Ext interrupt

 <5번 기능>

INT3 발생시 LED 좌측 이동하기.

 <6번 기능>

INT4 발생시 LED 우측 이동하기.

전체 소스코드

```
/*
 * HW_002.c
 *
 * Created: 2024-07-30 오전 10:39:07
 * Author : lordk
 * SRS:
 * 모든 SRS(명세서)는 과제보고서와 작업일지(개인)에 기록함.
 * 1. 0.5s 마다 모든 LED 깜빡이기. _delay_ms() 함수를 사용할 것.
 * [No Ext Interrupt]
 * 2. SW1이 눌리면 4~7 LED 켜기
 * 3. SW2이 눌리면 0~3 LED 켜기
 * 4. 둘 다 눌리면 모두 켜기
 * [Ext Interrupt]
 * 5. INT3 발생시 LED 좌측 이동
 * 6. INT4 발생시 LED 좌측 이동
 */

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int main(void) {
    DDRA = 0xFF;
    DDRD = 0x00;

    PORTA = 0x00;

    EIMSK = 0b000001100;
```

```

EICRA = 0b10100000;

sei();

while (1) {

    // 스위치 0번 입력 시,
    if(!(PIND & (1 << PIND0))){
        if(!(PIND & (1 << PIND1))) { // <기능4> : 스위치 0번, 1번 동
시에 입력 받을 시,

            /* AVR에서 동시 입력은 없음.
            * 항상 우선으로 입력 받는 키가 있음.
            * 우선키 -> 후발키 순으로 입력 받음(회로 원리)
            * 즉, 0번과 1번을 동시에 입력하는 기능은 Unity Engine
이나 Android Studio 같은 SW 개발 엔진에만 있고,
            * AVR 개발을 위한 Microchip(Atmel) Studio에서는 회로
원리에 의해 첫 번째 입력 받은 값이 흐르고
            * -> 그 뒤에 입력 받은 값이 흐른다는 것을 알게 됨.
            * 이번 케이스에서는 PD0과 PD1의 동시 입력을 원하는데,
PD0이 PD1보다 우선으로 입력 받게 설정되어 있으므로
            * PD0의 세부 조건으로 PD1이 입력 받았을 때를 감지하여
동시 입력 처리를 진행해 주면 됨.

            */
            PORTA = 0x00;
        }else{ // <기능2> : 스위치 0번 단독 입력 시,
            PORTA = 0b00001111;
        }
    }else if(!(PIND & (1 << PIND1))) { // <기능3> : 스위치 1번 단독 입력
시,

        PORTA = 0b11110000;
    }else{ // <기능1> : 깜빡임 반복하기.
        PORTA = 0xFF;
        _delay_ms(500);
        PORTA = 0x00;
        _delay_ms(500);
    }

}

}

// <기능5> : LED 좌측으로 이동하기.
ISR(INT2_vect){
    PORTA = 0b11111110;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11110111;
}

```

```

        _delay_ms(100);
        PORTA = 0b11101111;
        _delay_ms(100);
        PORTA = 0b11011111;
        _delay_ms(100);
        PORTA = 0b10111111;
        _delay_ms(100);
        PORTA = 0b01111111;
        _delay_ms(100);
    }

// <기능6> : LED 우측으로 이동하기.
ISR(INT3_vect){
    PORTA = 0b01111111;
    _delay_ms(100);
    PORTA = 0b10111111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111110;
    _delay_ms(100);
}

```

코드 리뷰하기

[1. 전처리문]

```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

```

delay() 관련 함수를 사용할 때, F_CPU 지정을 안 해주면, 경고가 발생합니다.
클럭을 지정하지 않았기 때문입니다.
제가 사용하고 있는 MCU는 atmega128이며, 16Mhz의 스펙을 가지고 있습니다.
따라서, 클럭을 MCU에 맞게 변경해 줘야 하므로 F_CPU 관련 정의를 해주었습니다.

참고자료 :

- [F_CPU-Microchip](#),
- [define F_CPU 16000000 왜 써야 하는가? delay](#)
- [ATmega128 Delay함수 사용 방법](#)

avr의 input, output이 가능하게 하기 위해 관련 헤더파일인 <avr/io/h>를 선언하였습니다.
 delay() 함수를 사용하기 위해 관련 헤더파일인 <util/delay.h>를 선언하였습니다.
 <기능5>와 <기능6>을 구현하기 위해 관련 헤더파일인 <avr/interrupt.h>를 선언하였습니다.

[2. main() 함수 내부]

```
int main(void) {
    DDRA = 0xFF;
    DDRD = 0x00;

    PORTA = 0x00;

    EIMSK = 0b00001100;
    EICRA = 0b10100000;

    sei();

    while (1) {

        // 스위치 0번 입력 시,
        if(!(PIND & (1 << PIND0))){
            if(!(PIND & (1 << PIND1))) { // <기능4> : 스위치 0번, 1번 동
시에 입력 받을 시,
                /* AVR에서 동시 입력은 없음.
                * 항상 우선으로 입력 받는 키가 있음.
                * 우선키 -> 후발키 순으로 입력 받음(회로 원리)
                * 즉, 0번과 1번을 동시에 입력하는 기능은 Unity Engine
이나 Android Studio 같은 SW 개발 엔진에만 있고,
                * AVR 개발을 위한 Microchip(Atmel) Studio에서는 회로
원리에 의해 첫 번째 입력 받은 값이 흐르고
                * -> 그 뒤에 입력 받은 값이 흐른다는 것을 알게 됨.
                * 이번 케이스에서는 PD0과 PD1의 동시 입력을 원하는데,
PD0이 PD1보다 우선으로 입력 받게 설정되어 있으므로
                * PD0의 세부 조건으로 PD1이 입력 받았을 때를 감지하여
동시 입력 처리를 진행해 주면 됨.

                */
                PORTA = 0x00;
            }else{ // <기능2> : 스위치 0번 단독 입력 시,
                PORTA = 0b00001111;
            }
        }else if(!(PIND & (1 << PIND1))){ // <기능3> : 스위치 1번 단독 입력
시,
            PORTA = 0b11110000;
```

```

    }else{ // <기능1> : 깜빡임 반복하기.
        PORTA = 0xFF;
        _delay_ms(500);
        PORTA = 0x00;
        _delay_ms(500);
    }
}
}

```

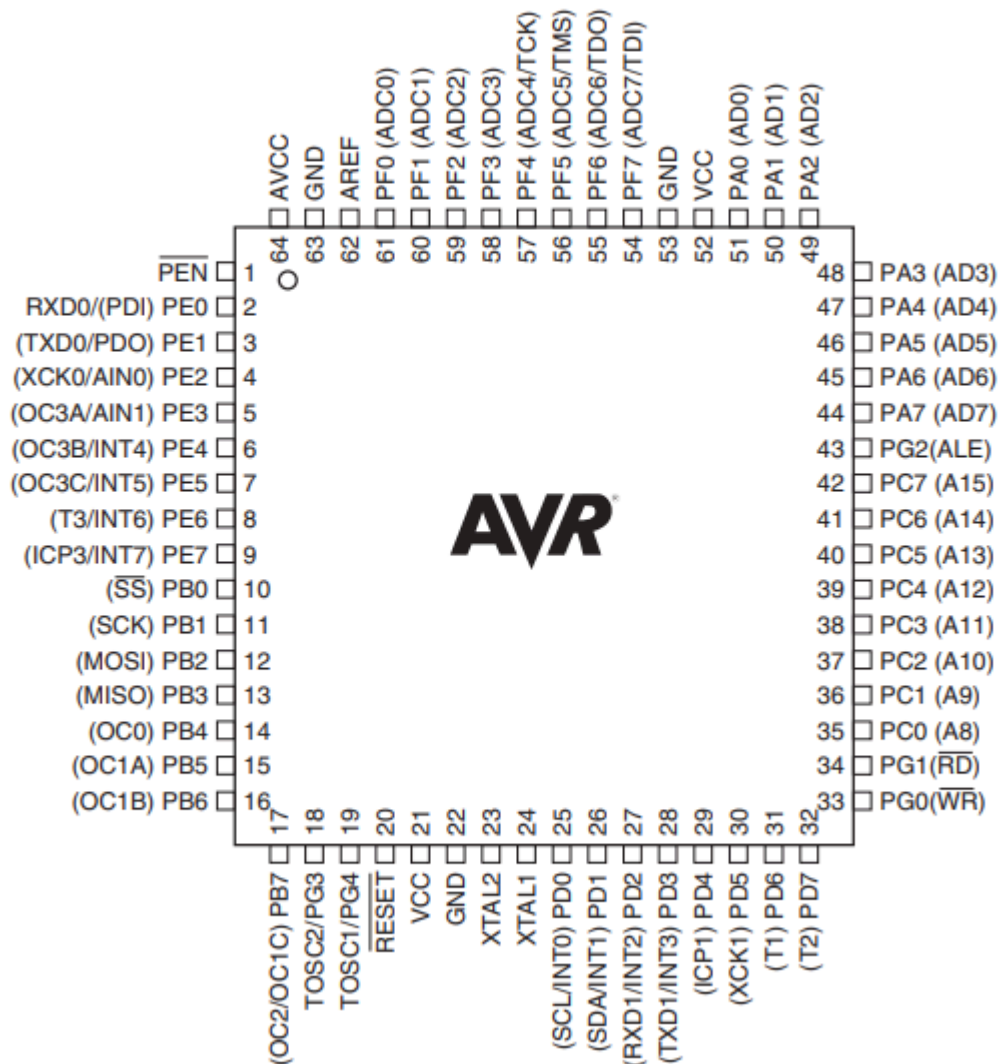
[PORT 관련 리뷰]

ATmega128에서의 포트 입출력 방향은 DDRx 레지스터로 설정이 가능합니다.

DDRA, DDRB DDRC DDRD DDRE DDRF DDRG로 구성되어 있으며 총 일곱 개 입니다.

[Pinout ATmega128]

Figure 1. Pinout ATmega128



ATmega128의 구성도를 보면 GND, VCC, PA0, PA1 ~ PG4 등이 있습니다.

- PA ~ PF : 8비트 양방향 병렬 I/O 포트
- PG : 5비트 양방향 병렬 I/O 포트

ATmega128의 DDRx, PORTx, PINx 레지스터에 대한 내용은 별도의 개념 정리 파일(.md)을 생성하여 private repo 내에서 관리하고 있습니다. 앞으로도 이렇게 개념을 먼저 공부한 후, 알고리즘을 설계하려고 합니다.

다시 코드로 돌아와 이해한 내용을 바탕으로 설명하겠습니다.
DDRA에 16진수 형태의 값을 할당해 주었습니다.

```
DDRA = 0xFF;
DDRD = 0x00;

PORTA = 0x00;

EIMSK = 0b00001100;
EICRA = 0b10100000;

sei();
```

이는 PORTA의 전체 레지스터(0부터 ~ 7까지)에 값 1을 대입해 주는 것을 의미합니다.

= 0번째 PIN부터 ~ 7번째 PIN까지 각 PIN의 값은 1로 대입함.

납땜 처리된 LED PIN의 출력을 위해 DDRA에 값 1을 넣어, PORTA를 출력모드로 지정하였습니다.

마찬가지로 DDRD에 0x00을 할당해 주면, DDRD의 전체 레지스터(0부터 ~7까지)에 값 0을 대입해 주는 것을 의미합니다.

= 0번째 PIN부터 ~ 7번째 PIN까지 각 PIN의 값은 0으로 대입함.

스위치의 입력을 받기 위해 DDRD에 값 0을 넣어, PORTD를 입력모드로 지정하였습니다.

PORTA에 0x00을 대입하여 PORTA에 0V로 지정합니다. 현재 제가 사용하고 있는 회로는 Active-Low 방식으로, 0V가 인가 되면, 연결된 led PIN에 불빛이 들어옵니다.

[Interrupt 관련 리뷰]

EIMSK(External Interrupt Mask Register)를 통해 사용할 Interrupt를 설정할 수 있습니다. 현재 스위치는 네 개이며, 기능을 구현하기 위해서는 index 기준 2번 스위치와 3번 스위치를 사용해야 합니다. 따라서, 2번 스위치와 3번 스위치의 Interrupt를 사용하기 위해 2진수 형태로 지정을 해주었습니다.

```
EIMSK = 0b00001100; // 오른쪽부터 0번째 index(=0번째 스위치) ~ 맨 왼쪽은 7번째 index(=7번째 스위치)
EICRA = 0b10100000; // 오른쪽부터 두 칸씩 0번째 index(=0번째 스위치)에 대한 설정값 ~ 맨 오른쪽 기준 7번째 값(0)부터 8번째 값(1)은 3번째 index(=3번째 스위치)에 대한 설정 값.
```

EICRA(External Interrupt Control Register A)를 통해 각 비트의 선택에 따른 Interrupt 감지 방법을 설정할 수 있습니다. EICRA는 전체 8비트를 차지하며, EICRA의 2비트가 EIMSK의 한 비트에 대한 세부 설정을 참조합니다.

EIMSK의 0번째 비트에 대하여 Interrupt 감지 방법을 설정 하고 싶으면, EICRA의 0번째와 1번째 비트에 값을 수정하여 설정할 수 있습니다. EICRA는 EIMSK의 0번째 index부터 ~ 3번째 index까지만 참조

합니다. EIMSK는 8개의 index로 구성되어 있으며, 4번째 index부터 ~ 7번째 index까지에 대한 세부 설정도 해주어야 합니다. 이를 위해 EICRB(External Interrupt Control Register B)를 통해 설정할 수 있습니다.

```
sei();
```

Interrupt를 얼마나 사용하고, 어떤 방식으로 사용할 것 인지에 대한 세부 설정을 마쳤으니, 이제 Interrupt를 사용하기 위해 활성화를 해주어야 합니다. 위에 코드를 통해 프로그램 내에서 Interrupt를 사용하겠다고 호출을 해줄 수 있습니다.

[2-1. While()문 내부]

```
while (1) {  
    if(!(PIND & (1 << PIND0))){  
        if(!(PIND & (1 << PIND1))) {  
            PORTA = 0x00;  
        }else{  
            PORTA = 0b00001111;  
        }  
    }else if(!(PIND & (1 << PIND1))){  
        PORTA = 0b11110000;  
    }else{  
        PORTA = 0xFF;  
        _delay_ms(500);  
        PORTA = 0x00;  
        _delay_ms(500);  
    }  
}
```

가독성을 높이기 위해 상세 코드 리뷰에서는 주석을 제거하였습니다.

main() 함수에 있는 while문은 무한 반복이 되어야 하므로, while문 value에 '참'을 의미하는 '1'을 입력하였습니다.

<2번 기능>

0번째 스위치가 눌렸을 때는 index 4부터 ~ 7까지의 led 센서가 켜져야합니다. 스위치가 눌린 것을 감지하기 위해서 조건문에 PIN의 감지 여부를 식으로 작성하였습니다. 저희는 Active-Low 방식의 회로를 사용하고 있습니다. 스위치가 눌렸을 때 0V가 되어야 합니다. 즉, 스위치가 눌리지 않은 평소에는 5V가 흐르고 있습니다. 0V일 때를 감지하기 위해서는 스위치가 눌린 상태(5V)에 not 연산자(!)를 사용하여 0V 인지를 감지하게 했습니다.

<4번 기능>

0번째 스위치가 눌렸을 때, 세부조건으로 1번째 스위치가 감지되었는지 판단하는 조건을 추가하였습니다. 이는 <4번 기능>인 0번째 스위치와 1번째 스위치가 동시에 입력 되었을 때를 감지하기 위함입니다.

회로 개념에서 동시 입력은 없다는 것을 알게 되었습니다. 선배님의 가르침 하에 첫 번째 전류가 흐르고, 두 번째 전류를 감지하여 동시에 입력 되었는지를 판단할 수 있다는 것을 알 수 있게 되었습니다. 또한, 입력 감지의 순서도 알 수 있게 되었습니다. ATmega128 Datasheet를 통해 PD0와 PD1이 동시에 눌리더라도, 입력 감지 순서에 따라 PD0에 먼저 전류가 흐른다는 것을 알 수 있었습니다. 따라서, 0번째 스위치(PD0)가 눌리고, 1번째 스위치(PD1)가 눌렸는지 여부를 통해 동시 입력 처리를 진행하였습니다. 만약, 1번째 스위치가 눌리지 않았다면, 0번째 스위치가 단독으로 눌린 것 이므로, <4번 기능>이 활성화 되지 않았을 때 위치에 추가하여 처리했습니다.

<3번 기능>

1번째 스위치가 눌렸을 때, index 0부터 ~ 3까지의 led 센서가 켜지도록 구현하였습니다.

<1번 기능>

아무런 스위치가 입력 되지 않았을 때, 0.5초 마다 모든 led 센서가 깜빡이도록 delay() 함수를 사용하여 구현하였습니다.

LED Sensor가 켜졌다가, 꺼졌다가를 0.5초 간격으로 반복되어야 합니다.

따라서, LED Sensor가 납땜 처리된 PA0부터 ~ PA7까지의 PIN을 한꺼번에 관리하기 위해 PORTA로 값을 대입해 주었습니다.

1. PORTA에 2진수 형태로 값을 입력하여 각 PIN에 LED 불빛이 들어오도록 해주었습니다.
2. delay() 함수를 사용하여 0.5초(=500ms) 만큼 지연을 시켜주었습니다.
3. PORTA에 2진수 형태로 값을 입력하여 각 PIN에 LED 불빛이 꺼지도록 해주었습니다.
4. delay() 함수를 사용하여 0.5초(=500ms) 만큼 지연을 시켜주었습니다.

[3. INT2 발생 처리]

```
// <기능 5> : LED 좌측으로 이동하기 .
```

```
ISR(INT2_vect){
    PORTA = 0b11111110;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b10111111;
    _delay_ms(100);
    PORTA = 0b01111111;
    _delay_ms(100);
}
```

과제 설명에는 INT3처리라고 나와 있으나, 2번째 스위치에 할당된 기능은 INT3입니다. 따라서, INT2에 대한 처리를 진행하였습니다. (과제 설명 오타로 생각합니다.)

<5번 기능>

PORTA의 켜짐/꺼짐을 하나하나 작성하여 LED가 좌측으로 이동하도록 구현하였습니다. 시프트 연산자(<<)를 사용하여 기능 구현을 시도하였으나, 예외처리를 구현하는 데 있어 큰 어려움을 느껴 우선은 수동으로 구현하였습니다.

? Question

```
// PORTA의 상태는 현재 1111 1111임.  
PORTA << 1;
```

PORTA를 좌측으로 한 칸 시프트 하면, PORTA의 값은 1111 1110이 됩니다.

또, 다시 좌측으로 한 칸 시프트 하면, PORTA의 값은 1111 1100이 됩니다.

이때 PORTA의 값이 1111 1101이 되게 하고 싶은데, 이 처리를 어떻게 할 수 있는지 잘 모르겠습니다. 여러 레퍼런스를 찾아보기도 하였으나, 관련해서는 더 찾아보고 연구할 필요성을 느꼈습니다. 방법을 찾아내고 이해하게 된다면, 현재의 소스코드를 업데이트 한 후, commit하도록 하겠습니다.

[4. INT3 발생 처리]

```
// <기능6> : LED 우측으로 이동하기 .  
ISR(INT3_vect){  
    PORTA = 0b01111111;  
    _delay_ms(100);  
    PORTA = 0b10111111;  
    _delay_ms(100);  
    PORTA = 0b11011111;  
    _delay_ms(100);  
    PORTA = 0b11101111;  
    _delay_ms(100);  
    PORTA = 0b11110111;  
    _delay_ms(100);  
    PORTA = 0b11111011;  
    _delay_ms(100);  
    PORTA = 0b11111101;  
    _delay_ms(100);  
    PORTA = 0b11111110;  
    _delay_ms(100);  
}
```

<6번 기능>

<6번 기능>은 <5번 기능>의 구현 방식과 동일하게 구현하였으며, LED가 우측으로 이동할 수 있도록

PORTA의 voltage값을 변경해 주었습니다.

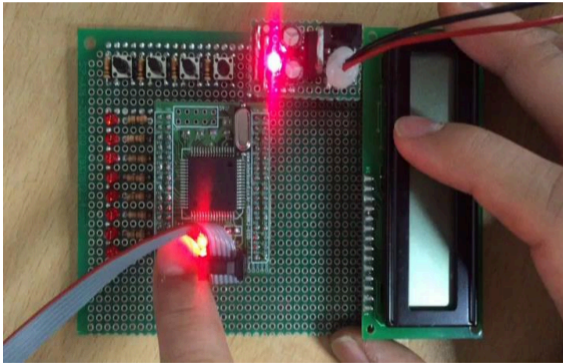
회로 작업(스위치) 작업을 진행하느라 정해진 시간 내에 해당 과제를 수행 및 제출하지 못 했습니다.
현재 진행하고 있는 작업을 완수한 후 과제를 수행할 수 있도록 노력하겠습니다.

과제 3

HW_003

✎ 과제 설명

과제 3



1. 0.1s 마다 LED 2진 카운터

- 2. INT0 발생시 LED 3개씩 우측 이동 X 2
- 3. INT1 발생시 LED 3개씩 좌측 이동 X 2
- 4. INT2 발생시 LED 1개 좌측 이동 후 우측 이동
- 5. INT3 발생시 2진 카운터 초기화

Ext Interrupt

[과제 풀이]

Github repo : [바로가기](#)

과제 시연 영상 : MCU-1 일차-과제3-시연영상.mp4

기능 모음


✎ <1번 기능>

0.1s 마다 LED 2진 카운터


✎ Ext interrupt

✎ <2번 기능>


INT0 발생시 LED 3개씩 우측 이동 X 2

 <3번 기능>

INT1 발생시 LED 3개씩 좌측 이동 X 2

 <4번 기능>

INT2 발생시 LED1개 좌측 이동 후 우측 이동

 <5번 기능>

INT3 발생시 2진 카운터 초기화

전체 소스코드

```
/*
 * HW_003.c
 *
 * Created: 2024-07-31 오전 10:28:26
 * Author : lordk
 * SRS:
 * 모든 SRS(명세서)는 과제보고서와 작업일지(개인)에 기록함.
 * 1. 0.1s 마다 LED 2진 카운터
 * [Ext Interrupt]
 * 2. INT0 발생시 LED 3개씩 우측 이동 X 2
 * 3. INT1 발생시 LED 3개씩 좌측 이동 X 2
 * 4. INT2 발생시 LED1개 좌측 이동 후 우측 이동
 * 5. INT3 발생시 2진 카운터 초기화
 */

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

// 전역 변수 선언하기 (interrupt와 main에서 공유하기 위함).
unsigned int cnt = 0xFF;

int main(void){
    // 레지스터 선언하기.
    DDRA = 0xFF;
    PORTA = 0xFF;

    EIMSK = 0b00001111;
```

```

    EICRA = 0b10101010;

    sei();

    while (1) {
        PORTA = cnt;
        cnt--;

        if(cnt < 0){ // Active-Low : 0000 0000 (led가 모두 켜졌을 때,)
            cnt = 0xFF; // 1111 1111(led 모두 끄기)
            PORTA = 0xFF;
            _delay_ms(100);
        }

        _delay_ms(100);
    }
}

// <기능2>
ISR(INT0_vect){

    PORTA = 0b11111000;
    _delay_ms(500);
    PORTA = 0b11110001;
    _delay_ms(500);
    PORTA = 0b11100011;
    _delay_ms(500);
    PORTA = 0b11000111;
    _delay_ms(500);
    PORTA = 0b10001111;
    _delay_ms(500);
    PORTA = 0b00011111;
    _delay_ms(500);
    PORTA = 0b00111110;
    _delay_ms(500);
    PORTA = 0b01111100;
    _delay_ms(500);
    PORTA = 0b11111000;
    _delay_ms(500);
    PORTA = 0b11110001;
    _delay_ms(500);
    PORTA = 0b11100011;
    _delay_ms(500);

}

```

// <기능3> : INT1 발생시 LED 3개씩 좌측 이동 X 2

```

ISR(INT1_vect){
    PORTA = 0b00011111;
    _delay_ms(500);
    PORTA = 0b10001111;
    _delay_ms(500);
    PORTA = 0b11000111;
    _delay_ms(500);
    PORTA = 0b11100011;
    _delay_ms(500);
    PORTA = 0b11110001;
    _delay_ms(500);
    PORTA = 0b11111000;
    _delay_ms(500);
    PORTA = 0b01111100;
    _delay_ms(500);
    PORTA = 0b00111110;
    _delay_ms(500);
    PORTA = 0b00011111;
    _delay_ms(500);
    PORTA = 0b10001111;
    _delay_ms(500);
    PORTA = 0b11000111;
    _delay_ms(500);
}

```

// <기능4> : INT2 발생시 LED1개 좌측 이동 후 우측 이동

```

ISR(INT2_vect){

    int tempPORTA = PORTA;

    // 우측 이동하기 .
    PORTA = 0b11111110;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b10111111;
    _delay_ms(100);
    PORTA = 0b01111111;
    _delay_ms(100);

```

```

    // 좌측 이동하기 .

```

```

    PORTA = 0b01111111;
    _delay_ms(100);
    PORTA = 0b10111111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111110;
    _delay_ms(100);

}

// <기능 5> : 2진 카운터 초기화 하기.
ISR(INT3_vect){
    cnt = 0xFF; // 1111 1111(led 모두 끄기)
    unsigned int cnt = 0xFF; // 1111 1111(led 모두 끄기)
    PORTA = 0xFF;
    _delay_ms(100);
}

```

코드 리뷰하기

[1. 전처리문]

```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

```

- <util/delay.h> 헤더파일 내부에 있는 delay() 함수를 사용하기 위해서, 클럭을 지정해 주어야 하므로 F_CPU를 사용하고 있는 MCU인 ATmega128의 클럭인 16Mhz에 알맞는 스펙으로 정의하였습니다.
- avr의 input, output이 가능하게 하기 위해 관련 헤더파일인 <avr/io.h>를 선언하였습니다.
- delay() 함수를 사용하기 위해 관련 헤더파일인 <util/delay.h>를 선언하였습니다.
- Interrupt 관련 처리를 하기 위해 <avr/interrupt.h>를 선언하였습니다.

[2. main() 함수 내부]


```

// 전역 변수 선언하기 (interrupt와 main에서 공유하기 위함).
unsigned int cnt = 0xFF;

int main(void){
    // 레지스터 선언하기.
    DDRA = 0xFF;
    PORTA = 0xFF;

    EIMSK = 0b00001111;
    EICRA = 0b10101010;

    sei();

    while (1) {
        PORTA = cnt;
        cnt--;

        if(cnt < 0){ // Active-Low : 0000 0000 (led가 모두 켜졌을 때,)
            cnt = 0xFF; // 1111 1111(led 모두 끄기)
            PORTA = 0xFF;
            _delay_ms(100);
        }

        _delay_ms(100);
    }
}

```

[전역 변수 관련 리뷰]

전역 변수로 unsigned int type의 cnt 변수를 선언하였습니다. 이 변수는 main() 함수 내부와 interrupt 내부에서 공용으로 사용되는 변수입니다.

[레지스터 관련 리뷰]

```

DDRA = 0xFF;
PORTA = 0xFF;

EIMSK = 0b00001111;
EICRA = 0b10101010;

sei();

```

사용할 레지스터와 interrupt 관련 선언을 해주었습니다. PORTA에 연결된 모든 led PIN에 대하여 불빛이 꺼짐(TURN OFF) 상태로 설정하였습니다. 사용할 Interrupt에 대한 설정도 진행하였습니다. 이번에

사용할 Interrupt는 총 네 개로, INT0부터 ~ INT3까지에 대한 Interrupt를 사용합니다. 각 Interrupt에 대한 세부 설정은 **falling edge** 방식으로 지정하였습니다. 마지막으로, Interrupt를 사용하기 위해 sei() 함수를 호출하였습니다.

[2.1 while()문 내부]

```
while (1) {  
    PORTA = cnt;  
    cnt--;  
  
    if(cnt < 0){  
        cnt = 0xFF;  
        PORTA = 0xFF;  
        _delay_ms(100);  
    }  
  
    _delay_ms(100);  
}
```

가독성을 높이기 위해 상세 코드 리뷰에서는 주석을 제거하였습니다.

<1번 기능>

- 0.1초마다 LED 센서가 2진 카운터 하는 기능을 구현하였습니다. PORTA의 켜짐/꺼짐 상태를 cnt로 할당하였습니다. 변수 cnt는 전역으로 선언하였으며, cnt의 값은 0xFF입니다. 이는 PORTA에 연결된 모든 LED 센서의 불빛이 안 들어오는 상태를 의미합니다.
- cnt에 1을 감소시켰습니다. 기존 0xFF는 16진수 형태이며, 이를 2진수로 치환하면 1111 1111가 됩니다. 2진수 1111 1111은 10진수로 255입니다. 255에서 1을 빼주면 10진수로 254가 되며, 2진수로는 1111 1110로 표기할 수 있습니다. 이렇게 되면 0번째 LED 센서에만 불빛이 나타나게 됩니다.
- 다시 cnt에 1을 감소 시키면, 10진수 값은 253이며 2진수에서는 1111 1101로 표기됩니다. 이는 1번째 LED 센서에만 불빛이 나타나게 됩니다. 이런식으로 2진 카운터 기능을 구현하였습니다.
- 조건문을 통해 cnt의 값이 0 미만이라면 모든 불빛이 다 켜진 상태이므로, cnt값과 PORTA의 값을 초기값으로 초기화 시켜주었습니다. 이렇게 구성을 하게 되면, 7번째 LED까지 모두 불빛이 켜지고 다시 LED 센서의 모든 불빛이 꺼진 후 0번째 LED부터 불빛이 들어오게 됩니다.
- 지금 다시 코드를 확인해 보니 if문에서 0.1초를 delay()하고, if문을 탈출한 후에 다시 0.1초를 delay() 한다는 것을 알게 되었습니다. 최종적으로 if문을 충족할 때 delay()가 두 번 충족되므로 if문 내부에 있는 delay() 함수는 제거할 필요가 있습니다.

[3. INT0 처리]

```
// <기능2>  
ISR(INT0_vect){
```

```

PORTA = 0b11111000;
_delay_ms(500);
PORTA = 0b11110001;
_delay_ms(500);
PORTA = 0b11100011;
_delay_ms(500);
PORTA = 0b11000111;
_delay_ms(500);
PORTA = 0b10001111;
_delay_ms(500);
PORTA = 0b00011111;
_delay_ms(500);
PORTA = 0b00111110;
_delay_ms(500);
PORTA = 0b01111100;
_delay_ms(500);
PORTA = 0b11111000;
_delay_ms(500);
PORTA = 0b11110001;
_delay_ms(500);
PORTA = 0b11100011;
_delay_ms(500);
}

```

<2번 기능>

- <2번 기능>은 INT0 발생 시, LED 센서의 불빛이 3개씩 우측 이동하는 것 입니다. 선배님께서 제공해 주신 시연 영상을 보니, 우측으로 끝까지 이동한 후, 다시 왼쪽에서 불빛이 들어와 가운데 까지 이동하는 것을 확인하였습니다. 시연 영상에 알맞게 코드를 구성하였습니다.
- 2024년 8월 1일 20시 30분 쯤, MCU 세미나를 교육해 주신 로빛 16기 고준호 선배님께 리팩토링 관련해서 여쭙보았습니다. 현재 저의 코드는 PORTA 레지스터를 사용하여 수동으로 LED 센서의 불빛을 설정하고 있습니다. 이는 직관적이지 않으며 효율적이지 못하므로 상당히 안 좋은 코드라고 말할 수 있습니다. 선배님께서도 시프트 연산 관련해서 더 찾아보라고 조언해 주셨습니다. 제가 원하는 기능을 구현하기 위한 제가 알지 못한 시프트 연산이 더 있는 것 같습니다. 과제를 제출한 후, 그동안 배정 받은 모든 과제를 완수한 후에 다시 시프트 연산 관련 찾아보고 리팩토링을 진행하도록 하겠습니다.

[4. INT1 처리]

```

// <기능3> : INT1 발생시 LED 3개씩 좌측 이동 X 2
ISR(INT1_vect){
    PORTA = 0b00011111;
    _delay_ms(500);
    PORTA = 0b10001111;
    _delay_ms(500);
    PORTA = 0b11000111;
}

```

```

    _delay_ms(500);
    PORTA = 0b11100011;
    _delay_ms(500);
    PORTA = 0b11110001;
    _delay_ms(500);
    PORTA = 0b11111000;
    _delay_ms(500);
    PORTA = 0b01111100;
    _delay_ms(500);
    PORTA = 0b00111110;
    _delay_ms(500);
    PORTA = 0b00011111;
    _delay_ms(500);
    PORTA = 0b10001111;
    _delay_ms(500);
    PORTA = 0b11000111;
    _delay_ms(500);
}

```

<3번 기능>

- <3번 기능>은 INT0 발생 시, LED 센서의 불빛이 3개씩 좌측 이동하는 것 입니다. 선배님께서 제공 해 주신 시연 영상을 보니, 좌측으로 끝까지 이동한 후, 다시 오른쪽에서 불빛이 들어와 가운데 까지 이동하는 것을 확인하였습니다. 시연 영상에 알맞게 코드를 구성하였습니다.
- <3번 기능>도 시프트 연산 관련해서 더 찾아본 후, 리팩토링을 진행하도록 노력하겠습니다.

[5. INT2 처리]

```

// <기능4> : INT2 발생시 LED1개 좌측 이동 후 우측 이동
ISR(INT2_vect){

```

```

    int tempPORTA = PORTA;

```

```

    // 우측 이동하기.

```

```

    PORTA = 0b11111110;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b10111111;

```

```

    _delay_ms(100);
    PORTA = 0b01111111;
    _delay_ms(100);

    // 좌측 이동하기.

    PORTA = 0b01111111;
    _delay_ms(100);
    PORTA = 0b10111111;
    _delay_ms(100);
    PORTA = 0b11011111;
    _delay_ms(100);
    PORTA = 0b11101111;
    _delay_ms(100);
    PORTA = 0b11110111;
    _delay_ms(100);
    PORTA = 0b11111011;
    _delay_ms(100);
    PORTA = 0b11111101;
    _delay_ms(100);
    PORTA = 0b11111110;
    _delay_ms(100);

}

```

<4번 기능>

- <4번 기능>은 LED 센서 한 개씩 좌측 이동 후, 우측으로 이동하는 기능입니다. 본 기능 역시 시연 영상에 알맞게 구성하였습니다.
- 마찬가지로 본 기능 역시 시프트 연산 관련해서 더 찾아본 후, 리팩토링을 진행하도록 노력하겠습니다.

[6. INT3 처리]

```

// <기능 5> : 2진 카운터 초기화 하기.
ISR(INT3_vect){
    cnt = 0xFF; // 1111 1111(led 모두 끄기)
    unsigned int cnt = 0xFF; // 1111 1111(led 모두 끄기)
    PORTA = 0xFF;
    _delay_ms(100);
}

```

<5번 기능>

- <5번 기능>은 2진 카운터를 초기화 하는 기능입니다. 2진 카운터를 초기화 하기 위해서는 PORTA 를 관리하는 하나의 변수가 필요합니다. 그래서 전역 변수로 PORTA의 불빛의 꺼짐/켜짐을 관리 하는 변수를 선언하였습니다.

- 3번째 스위치(=INT3)가 눌리면, 2진 카운터가 초기화 되도록 구현하였습니다.

전체 과제 리뷰

회로 작업이 늦게 끝나 과제를 늦게 완수하여 제출드립니다. 과제를 포기하지 않고, LED 센서에 불빛을 하나씩 켜보면서 모든 LED 센서를 다룰 수 있게 되어 그 성취감이 높습니다. 시프트 연산 종류를 세부적으로 공부하여 이해한 내용을 기록하면서 작성한 코드의 효율을 극대화 할 수 있도록 계속 도전하겠습니다. 앞으로 남은 과제를 진행하여 갈망하는 로봇을 개발하기 위해 동기들과 함께 전진해 나아가겠습니다.