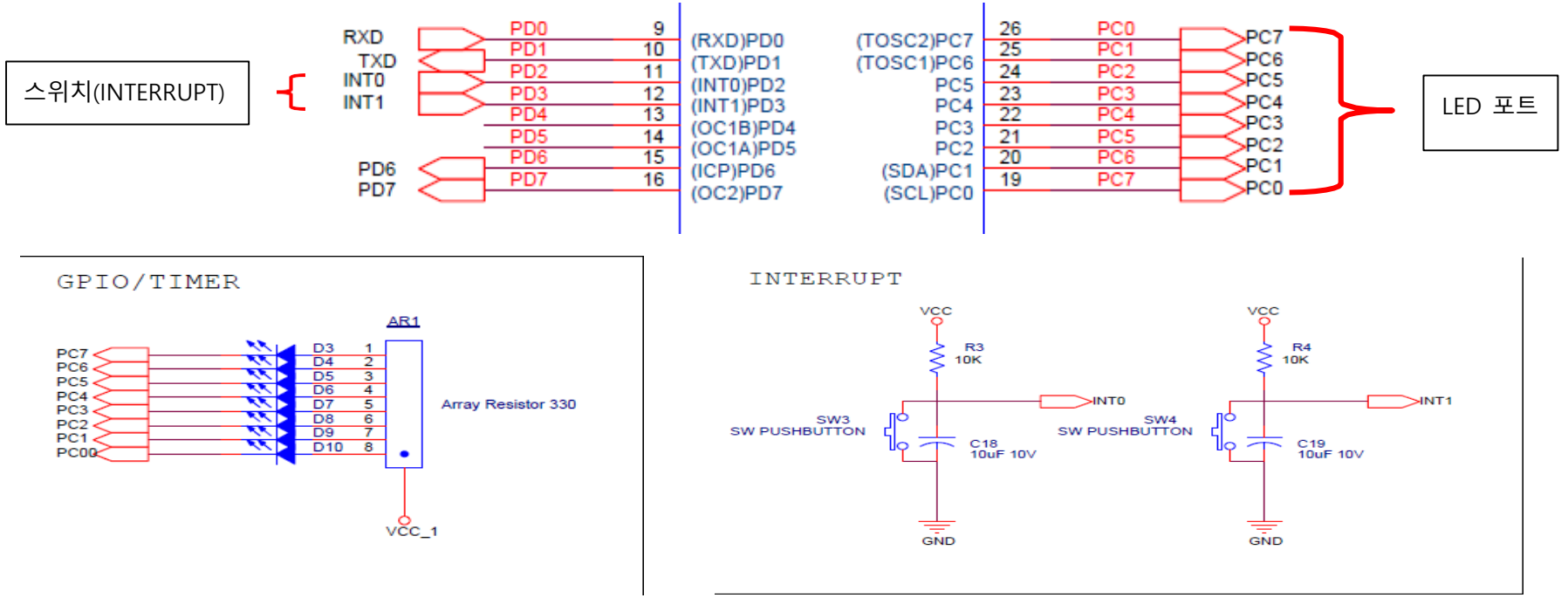
# 로봇학실험3 실험보고서

2014741022 서덕현

동작원리




-회로도만 보면 I/O, INTERRUPT 차이를 모르지만 레지스터에서 차이가 있다.

| DDxn | PORTxn | PUD (in SFIOR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low. |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

**Port C Data Register – PORTC**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTC7 | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 | PORTC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Port C Data Direction Register – DDRC**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDC7 | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 | DDRC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

-LED 설정은 스위치를 I/O 방식이나 INTERRUPT 방식 상관없이 출력이기 때문에 데이터시트와 회로도를 참고하면 DDRC를 1로 설정 해준다. LED를 켤려면 포트 C에 0V가 걸려야 LED에 전류가 흘러서 켜지므로 PORTC를 0 설정 반대로 끌려면 5V가 걸려야 하므로 PORTC를 1로 설정한다.

**Port D Input Pins Address – PIND**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 | PIND |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

- I/O방식

I/O는 포트를 입출력 방식으로 설정한다 라는 것이다. 스위치는 입력 방식이기 때문에 위에 데이터시트와 회로도를 참고하면 포트 D2의 스위치가 사용하고 싶으면 DDRD를 0x00을 설정해줘야 한다. 입력방식 이므로 PIND2를 1로 설정해주면 스위치가 떨어져 있다 는 것이고 PIND2를 스위치가 눌러져 있다는 것이다. 즉 PIND2를 1일 때와 0일 때 조건을 제어하는 방식으로 LED를 토글하는 것이다.

- INTERRUPT 방식

**General Interrupt Control Register – GICR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

1. 먼저 회로도를 보면 스위치 포트D2를 쓰고 싶으면 포트D2가 INT0으로 되어있기 때문에 GICR에서 6번에 있으므로 GICR=0x40 이나 GICR |=(1<<INT0)으로 설정해서 포트D2는 INTERRUPT 방식 중 EXTERNAL INTERRUPT로 쓴다고 설정한다.

   참고: GICR |=(1<<INT0)에서 | 는 OR 연산이다. GICR 초기값이 데이터시트 보면 다 0이므로 &(AND) 쓰면 초기값이 0이고 넣 는 값이 1이면 결과는 0이기 때문에 여기서는 | (OR)를 써서 결과가 1이 나오게 해서 INT0를 INTERRUPT 방식으로 쓴다고 할 수 있다.

**MCU Control Register – MCUCR**

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Table 36.** Interrupt 0 Sense Control

| ISC01 | ISC00 | Description |
|---|---|---|
| 0 | 0 | The low level of INT0 generates an interrupt request. |
| 0 | 1 | Any logical change on INT0 generates an interrupt request. |
| 1 | 0 | The falling edge of INT0 generates an interrupt request. |
| 1 | 1 | The rising edge of INT0 generates an interrupt request. |

2. 현재 우리가 INT0를 INTERRUPT 방식으로 쓴다고 설정했으므로 MCUCR 페이지를 보면 INTERRUPT를 제어하는 방식을 알려준 다. 여기서 INT0을 제어할려면 ISC01와 ISC00을 설정해주면 된다. ISC01와 ISC00이 둘 다 0이면 low level 즉 0V에서 INTERRUPT를 건다. 01이면 토글일(LED토글이 아닌 스위치 토글) 때 INTERRUPT를 건다 10이면 falling edge 즉 스위치 누를 때 INTERRUPT를 건다. 11이면 rising edge 즉 스위치를 땠을 때 INTERRUPT를 건다 라고 해석 할 수 있다. 그럼 우리는 LED토글 을 스위치 눌렀을 때 토글한다고 했으므로 falling edge일 때 INTERRUPT를 건다라고 설정해줘야 한다. 즉 MCUCR의 초기값이 0이므로 ISC01와 ISC00를 10으로 설정해줄려면 앞에 GICR와 비슷하게 MCUCR |= (1<<ISC01) | (0<<ISC00)으로 설정한다.

참고 : 1) (1<<ISC01) | (0<<ISC00) 사이에 OR 연산을 한 이유는 (1<<ISC01)는 0b00000010이고 (0<<ISC00)는 0b00000000이다.

이 두 개를 &(AND)로 연산해서 MCUCR에 넣으면 0b00000000이므로 OR연산을 써서 0b00000010로 나오게 해야한다.

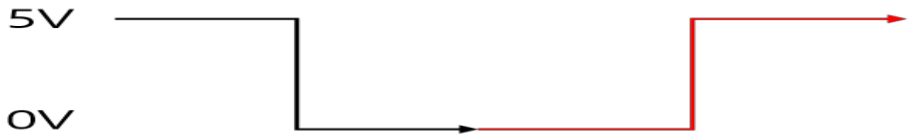
2) ISC01 ISC00 설정을 그림으로 설명하면 빨간 줄일 때 INTERRUPT가 걸린다는 것이다

| | | |
|---|---|---|
| 0 | 0 | The low level of INT0 generates an interrupt request. |


| | | |
|---|---|---|
| 0 | 1 | Any logical change on INT0 generates an interrupt request. |


| | | |
|---|---|---|
| 1 | 0 | The falling edge of INT0 generates an interrupt request. |


| | | |
|---|---|---|
| 1 | 1 | The rising edge of INT0 generates an interrupt request. |



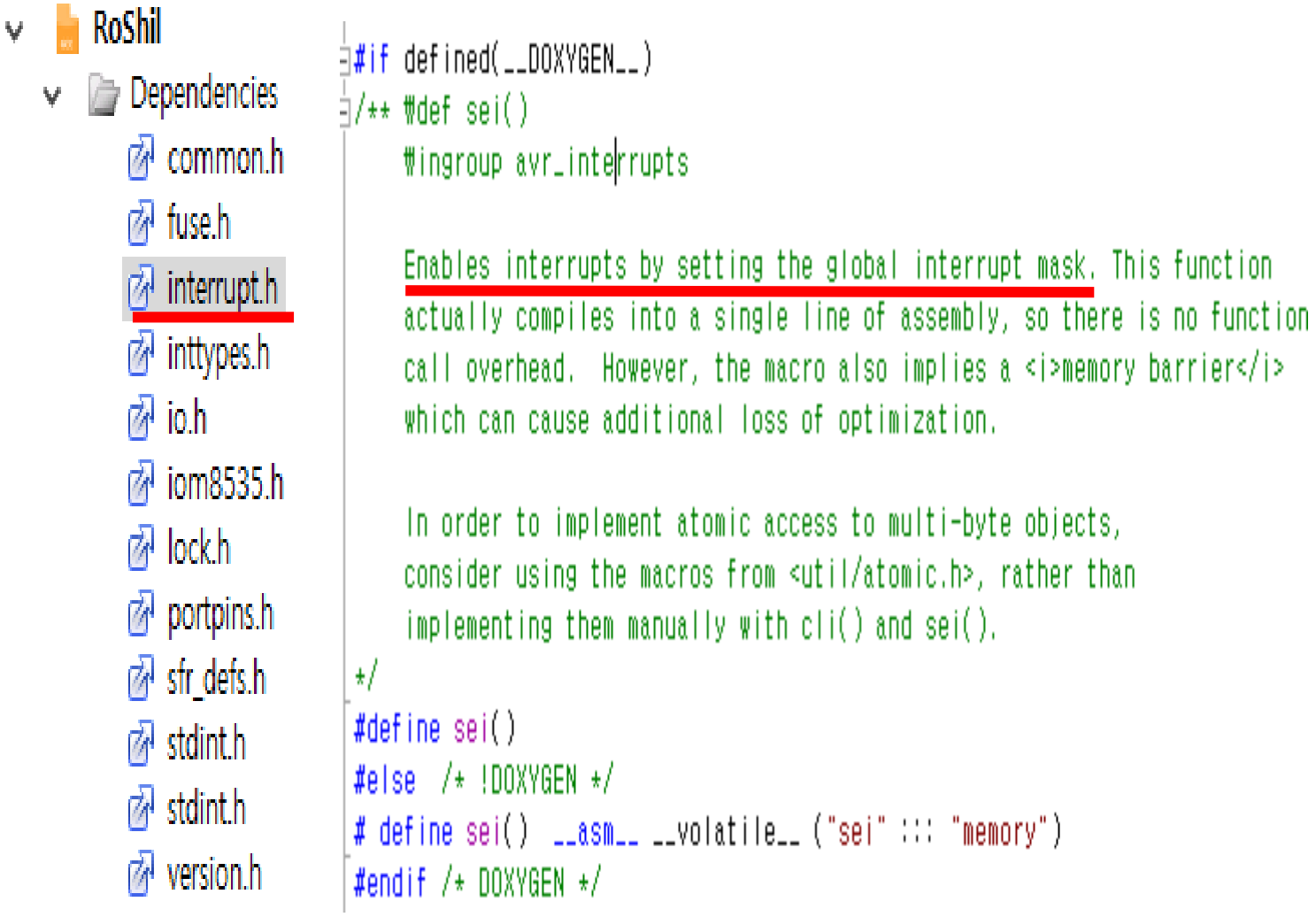
The AVR Status Register – SREG – is defined as:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | I | T | H | S | V | N | Z | C | SREG |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

3. 마지막으로 INTERRUPT를 Enable 할려면 7비트에 있는 Global Interrupt를 활성화 해줘야한다. SREG |= 0x80으로 설정하면 된다.




참고 : SREG |= 0x80 말고 sei()로 써도 된다. sei()로 쓰고 빌드하면 왼쪽 위 그림 같이 interrupt.h 파일이 생긴다. 클릭해서 sei()를 찾으면 오른쪽 위 그림 같이 Global Interrupt를 마스킹으로 활성화준다고 설명하고 있다.

-소스 비교

I/O 방식 소스

```
#include<avr/io.h>

 int main(void)
 {
     int count=0;//변수 초기화
     DDRC = 0xff; //포트 C를 출력으로 쓴다(LED)
     DDRD = 0x00; //포트 D를 입력으로 쓴다(SWITCH)

     PORTC = 0xff;//LED OFF

     |

     while(1)
     {
         if((PIND & (1<<PIND2))) //스위치 안 눌림
         {
             if( count==0)//count라는 변수가 0이라 일치할 시
             PORTC = 0xff;//LED OFF

             if( count==1)//count라는 변수가 1이라 일치할 시
             PORTC = 0x00;//LED ON


         }
         else//스위치 눌림
         {
             if(PORTC == 0x00)//LED ON
               count=0;//count라는 변수에 0을 대입
             if(PORTC == 0xff)//LED OFF
               count=1;//count라는 변수에 1을 대입
         }
     }
 }
```

INTERRUPT 방식 소스

```
#include <avr/io.h>
#include <avr/interrupt.h>

SIGNAL(INT0_vect)//INT0 Interrupt 발생 /
{
    PORTC = ~PORTC;//LED 토글


}

int main(void)
{
    DDRC = 0xff;//포트 C를 출력으로 쓴다.
    PORTC = 0xff;//LED OFF
    GICR |=(1<<INT0);//INT0 Interrupt Enable
    MCUCR |= (1<<ISC01)|(0<<ISC00);//falling edge 일 때 Interrupt Enable
    sei();//SREG = 0x80;// Global Interrupt Enable
    while(1)
    {
    }
}
```

비교 : I/O 방식은 반복문인 while에서 계속 LED를 토글 해주는 방식이고 INTERRUPT는 while에서 있다가 인터럽트가 발생하면 SIGNAL(INT0_vect)로 가서 그 함수 안에 있는 명령을 하고 인터럽트가 끝나면 다시 main으로 돌아온다.

소스 구현 영상

LED토글 구현 영상에는 I/O와 INTERRUPT는 똑같이  구현된다.