5-1강. External Interrupt

박 원 엽
010.5451.0113

목 차

- **인터럽트(Interrupt)란?**

  - 프로그램 수행 시, 작성한 코드에 따라 절차지향적(위에서부터 순서대로 한 문장씩 순차적)으로 처리되는데, 어떤 요인에 의해 인터럽트 요청이 발생되면 현재 수행하는 코드까지 실행하고, 인터럽트 서비스 루틴(ISR: Interrupt Service Routine) 함수를 수행하게 된다.
  인터럽트 서비스 루틴이 종료되면, 기존에 실행 중이었던 프로그램의 흐름으로 복귀한다.

※ 주의사항!

인터럽트 서비스 루틴은 동시에 여러 가지가 실행될 수 없다. 즉, 어떤 ISR이 수행되는 도중에 다른 인터럽트가 요청되면 그 인터럽트는 무시된다! (몇몇 예외도 있음)
따라서 ISR 함수 내부는 간단한 코드만 작성(예를 들면, 상태 변화 플래그 정도만 변화)하는 것이 바람직 하며, 절대 ISR 함수 내부에서는 딜레이 함수나 반복문을 사용하지 않아야 한다!

- **인터럽트(Interrupt)란?**

  - Atmega128에서 인터럽트가 요청되는 요인은 크게 내부적 요인과 외부적 요인으로 나눌수 있다.

  외부적 요인은, 외부적 전기적인 신호에 의해 요청되는 것(External Interrupt 등)이며
  내부적 요인은, 그 외의 소프트웨어적인 상태 변화(UART RX완료, Timer OVF, ADC 변환완료 Interrupt 등)에 의해 요청되는 것이다.

  사용할 수 있는 인터럽트의 종류는 정해져 있으며, 데이터시트 60페이지에 정의 되어있다.

- **인터럽트(Interrupt)란?**

  - 사용할 수 있는 인터럽트의 종류는 다음과 같다. (시트 60페이지)

Table 23. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow

- **인터럽트(Interrupt)란?**
  - 사용할 수 있는 인터럽트의 종류는 다음과 같다.

18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030(3)	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	$0032(3)	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034(3)	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036(3)	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038(3)	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A(3)	TIMER3 OVF	Timer/Counter3 Overflow
31	$003C(3)	USART1, RX	USART1, Rx Complete
32	$003E(3)	USART1, UDRE	USART1 Data Register Empty
33	$0040(3)	USART1, TX	USART1, Tx Complete
34	$0042(3)	TWI	Two-wire Serial Interface
35	$0044(3)	SPM READY	Store Program Memory Ready

- **인터럽트(Interrupt)란?**

  - 위의 표를 인터럽트 벡터 테이블이라고 하며, 인터럽트의 우선순위, 인터럽트 벡터 주소, 인터럽트 이름이 정의되어 있다.

  - 인터럽트의 우선순위의 의미?
  만약 여러 인터럽트가 동시에 요청된다면, 인터럽트 우선순위가 높은 인터럽트가 처리되며, 다른 인터럽트는 무시된다! (몇몇 예외사항 있음)

- **AVR에서 인터럽트 사용하기**

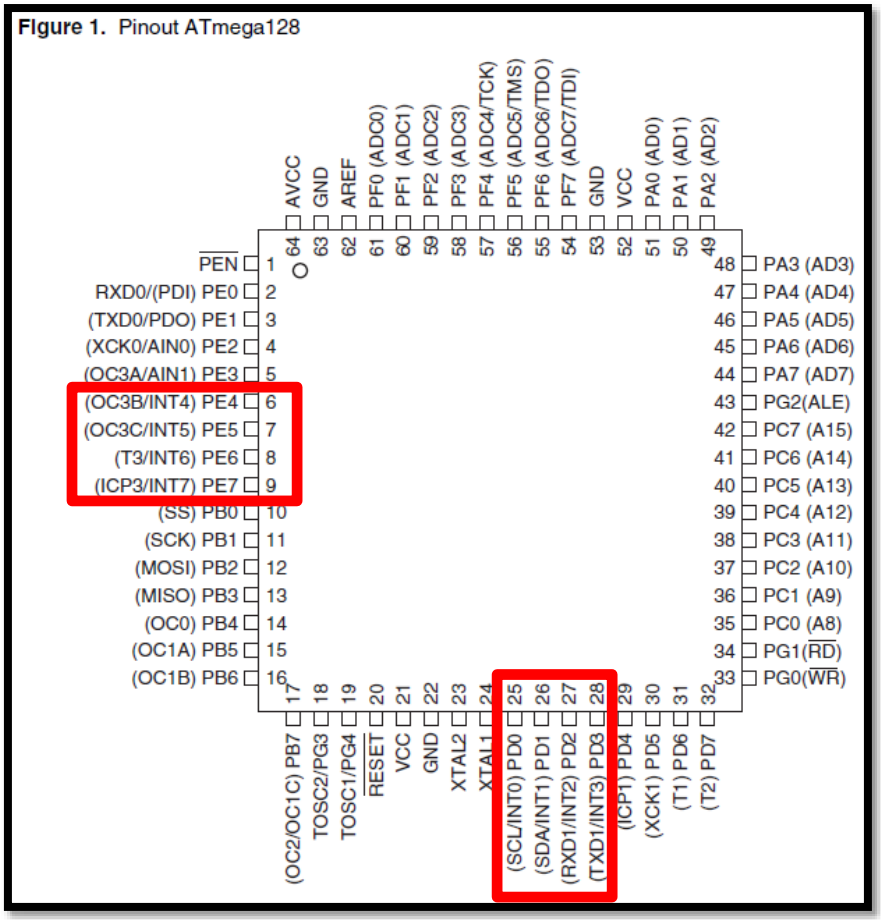
  - Atmega에서 인터럽트를 사용하기 위해서는 다음과 같은 절차가 필요하다.

  1. 지역 인터럽트(UART, External, ADC, Timer등)를 Enable 시킨다.
  2. 전역 인터럽트(SREG 레지스터의 7번 비트)를 Enable 시킨다.
  3. 해당 인터럽트의 ISR 함수를 작성한다.

  이 과정을 코드로 작성하면 인터럽트를 사용할 수 있게 된다.

- **외부 인터럽트(External Interrupt) – 데이터시트 90페이지부터**

  - 외부의 전기적인 신호에 의해 요청되는 인터럽트를 말하며(예를 들면, 하강 엣지 혹은 상승 엣지에서 인터럽트 발생) 외부 인터럽트를 사용할 수 있는 물리적인 핀은 다음과 같다.



- **외부 인터럽트(External Interrupt) – 데이터시트 90페이지**

  - 총 8개의 외부 인터럽트가 존재하며, 4개씩 두 개의 그룹으로 묶여있다.

  첫 번째 그룹은 INT0:3까지(PD.0~PD.3에 각각 매칭되어있음)
  두 번째 그룹은 INT4:7까지(PE.4~PE.7에 각각 매칭되어있음)

  각 그룹은 어느 시점에 인터럽트를 발생 시킬지 설정하는데 차이가 있다.

- **External Interrupt 관련 레지스터들 – 데이터시트 90페이지**

External Interrupt Control Register A – EICRA

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

External Interrupt Control Register B – EICRB

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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	

External Interrupt Mask Register – EIMSK

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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	

External Interrupt Flag Register – EIFR

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
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	

- **EICRA: External Interrupt Control Register A**
  - **Bit7:6 – INT3 Sense Control 비트**
  - **Bit5:4 – INT2 Sense Control 비트**
  - **Bit3:2 – INT1 Sense Control 비트**
  - **Bit1:0 – INT0 Sense Control 비트**

Table 48. Interrupt Sense Control(1)

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Note: 1. n = 3, 2, 1or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Table 49. Asynchronous External Interrupt Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

- **EICRB: External Interrupt Control Register B**
  - **Bit7:6 – INT7 Sense Control 비트**
  - **Bit5:4 – INT6 Sense Control 비트**
  - **Bit3:2 – INT5 Sense Control 비트**
  - **Bit1:0 – INT4 Sense Control 비트**

Table 50. Interrupt Sense Control(1)

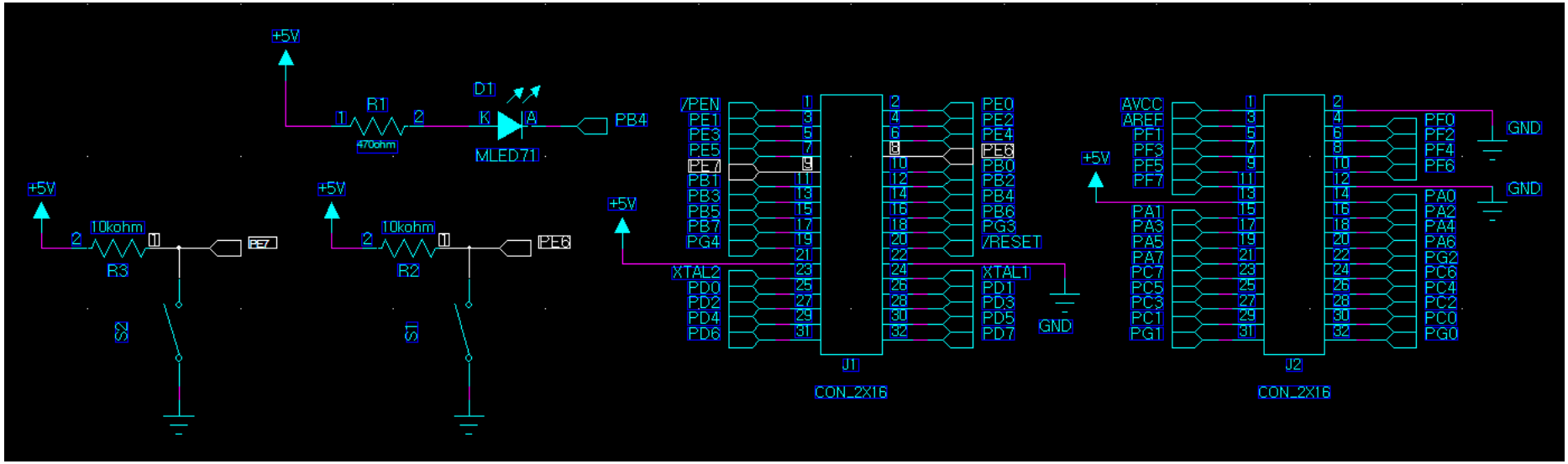
ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Note: 1. n = 7, 6, 5 or 4.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

- **EIMSK: External Interrupt Mask Register**
  - **Bit7 – INT7 외부 인터럽트 요청 허용 비트**
  - **Bit6 – INT6 외부 인터럽트 요청 허용 비트**
  - **Bit5 – INT5 외부 인터럽트 요청 허용 비트**
  - **Bit4 – INT4 외부 인터럽트 요청 허용 비트**
  - **Bit3 – INT3 외부 인터럽트 요청 허용 비트**
  - **Bit2 – INT2 외부 인터럽트 요청 허용 비트**
  - **Bit1 – INT1 외부 인터럽트 요청 허용 비트**
  - **Bit0 – INT0 외부 인터럽트 요청 허용 비트**

- **EIFR: External Interrupt Flag Register**
  - **Bit7 – INT7 외부 인터럽트 플래그 비트**
  - **Bit6 – INT6 외부 인터럽트 플래그 비트**
  - **Bit5 – INT5 외부 인터럽트 플래그 비트**
  - **Bit4 – INT4 외부 인터럽트 플래그 비트**
  - **Bit3 – INT3 외부 인터럽트 플래그 비트**
  - **Bit2 – INT2 외부 인터럽트 플래그 비트**
  - **Bit1 – INT1 외부 인터럽트 플래그 비트**
  - **Bit0 – INT0 외부 인터럽트 플래그 비트**

- **스위치 연결 회로 (Atmega128의 PE6, PE7에 연결)**



위의 회로와 같이 PE6, PE7번에 풀업 방식으로 스위치를 연결함. 풀업 방식으로 회로를 구성하면, 해당 핀에 스위치를 누르지 않으면 H, 누르면 L의 신호가 입력.

1강 General Purpose Input/Output (GPIO)

다음과 같은 코드를 작성하여 스위치를 눌렀을 때, LED가 켜지는 프로그램을 작성(Polling 방식)

```
#include <mega128.h>
#include <delay.h>
main()
{
    DDRB = 0x10;      //B포트 4번 핀을 출력으로 설정, 나머지는 don't care
    PORTB = 0x00;     //모든 핀 0으로 초기화.
    DDRE = 0x00;      //E포트 6, 7번 핀에 스위치 연결. 풀업 되어있음.

    while(1)
    {
        if(PINE & 0x80) //E포트 7번 핀이 H일때. 즉, 스위치가 눌리지 않았을 때.
        {
            PORTB = PORTB & ~(0x10);    //B포트 4번 핀 L신호 출력. 즉, LED 켜짐
        }
        else //E포트 7번 핀이 L일때. 즉, 스위치를 눌렀을 때.
        {
            PORTB = PORTB | 0x10;       //B포트 4번 핀 H신호 출력. 즉, LED 꺼짐
        }
    }
}
```

다음과 같은 코드를 작성(Interrupt 방식)

```
#include <mega128.h>
#include <delay.h>

main()
{
      /**   GPIO 입출력 설정   **/
      DDRB = 0x10; //PB4에 LED 연결
      PORTB = 0x00; //B포트 출력 초기화
      DDRE = 0x00; //PE6, 7에 스위치 연결
      DDRC = 0xff; //PC0~7에 7segment 연결. 따라서 전부 출력으로 설정
      PORTC = 0xff; //C포트 출력 초기화
      /************************/

      // External Interrupt(s) initialization
      // INT6: On
      // INT6 Mode: Falling Edge
      // INT7: On
      // INT7 Mode: Falling Edge
      EICRA=0x00;
      EICRB=0xA0;
      EIMSK=0xC0;
      EIFR=0xC0;

      // Global enable interrupts
      //#asm("sei")
      SREG = 0x80;
```

위의 레지스터 설정의 의미를 한번 생각해보자.
위와 같이 설정하면 EXT6, EXT7을 사용할 수 있게 된다!
또한 둘 다 하강 엣지에서 인터럽트가 발생하게 된다!
왜 하강 엣지에서 발생하도록 설정하였는가에 대해서도 생각해보자.

다음과 같은 코드를 작성(Interrupt 방식)

```
33  interrupt [EXT_INT6] void ext_int6_isr(void)
34  {
35
36  }
37
38  interrupt [EXT_INT7] void ext_int7_isr(void)
39  {
40
41  }
```

외부 인터럽트의 모든 설정이 완료되었으면, 그에 해당하는 Interrupt Service Routine 함수를 작성한다.

만약 다음과 같은 ISR 함수가 작성되었다면..

```
33  int cnt = 0;
34  interrupt [EXT_INT6] void ext_int6_isr(void)
35  {
36        cnt++;
37  }
```

외부 인터럽트6이 걸릴때 마다 cnt변수를 1씩 증가하게 하였다.
즉, 스위치가 한번 눌릴때 마다 cnt가 1씩 증가하는 것이다.

실제 동작도 원하는대로 cnt가 1씩 증가하는지 확인해보자.

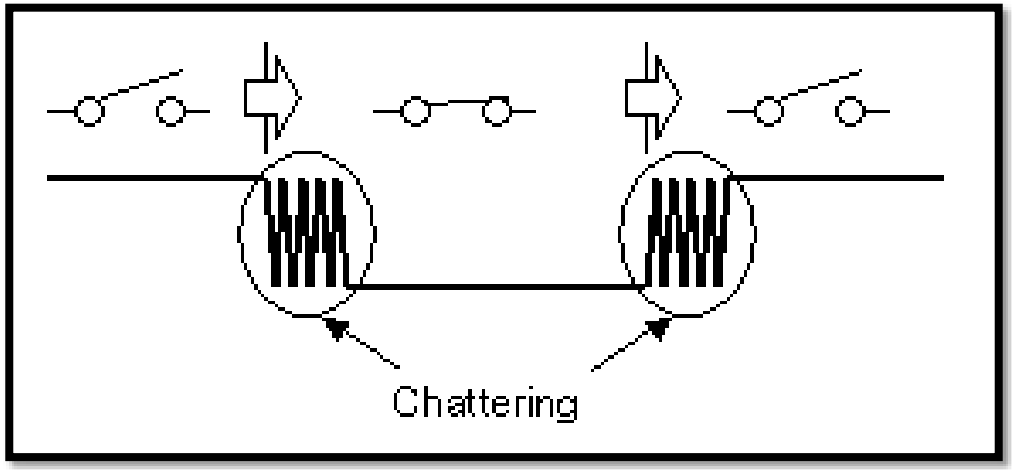
확인은 4강에서 진행한 character LCD나 2강에서 진행한 USART를 통해서 할 수 있다.

확인해 보면 1보다 더 큰 수만큼 증가하는 것을 확인할 수 있다!!
이러한 현상을 채터링(Chattering) 혹은 바운싱(Bouncing)현상이라고 한다.

- **채터링(Chattering) 현상 이란?**

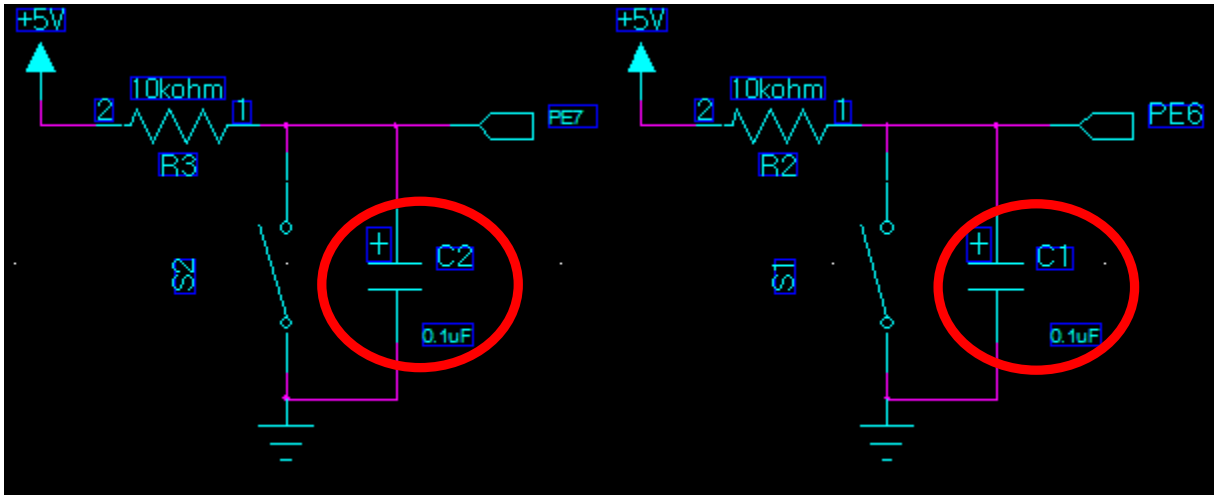
  - 스위치나 릴레이 등의 접점이 개폐될 때 기계에서 발생하는 진동이다. 스위치를 ON 또는 OFF로 했을 때, 접점 부분의 진동으로 말미암아 단속(斷續) 상태가 반복되는 일. 키보드에서 이 상태가 발생하면 같은 문자가 여러 개 찍힌다. 이를 방지하기 위해 하드웨어 내부나 소프트웨어에 특수 장치가 첨가되는데, RS 플립 회로가 대표적인 것이다.

  출처: [네이버 지식백과] 채터링 [chattering] (용어해설)



- 채터링(Chattering)현상을 제거하기 위한 방법 (Debounce)

  - 하드웨어적인 방법: 가장 간단한 방법으로는 스위치에 커패시터를 그라운드와 병렬로 연결. 그 외에 슈미트트리거, 버퍼회로를 구성하는 방법등도 있음.
  - 소프트웨어적인 방법: 스위치가 눌렸을 때, 또는 떨어졌을 때 약간의 딜레이를 주는 방법.



- 위의 그림과 같이 C1, C2 (0.1uF)를 연결. (하드웨어적 방법)

5-1강 External Interrupt

만약 다음과 같은 ISR 함수가 작성되었다면..

```
33  int cnt = 0;
34  interrupt [EXT_INT6] void ext_int6_isr(void)
35  {
36        cnt++;
37  }
```

하드웨어적인 방법으로 입력 핀과 그라운드 사이에 0.1uF정도의 커패시터를 달아준다.

그 후 동작 변화를 비교!

소프트웨어적인 방법으로는 ISR 함수 안에 100us정도의 딜레이를 준다.
(좋은 방법은 아니다.)

그 후 동작 변화를 비교!