인터럽트

마이크로프로세서

HRI 연구실

김동한

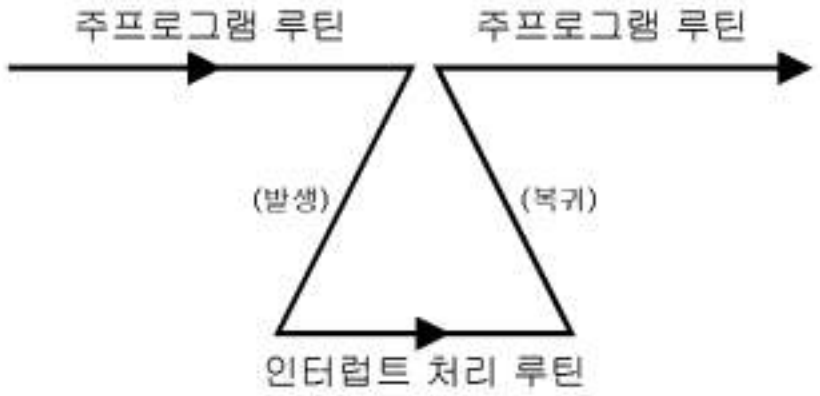
Human-Robot Interaction Laboratory

KYUNG HEE UNIVERSITY

중요. 인터럽트

기본개념

- CPU외부의 하드웨어적인 요구에 의해서 정상적인 프로그램의 실행 순서를 변경하여 보다 시급한 작업을 먼저 수행한 후에 다시 원래의 프로그램으로 복귀하는 것
- 비동기적으로 발생하는 주변장치의 서비스 요청에 CPU가 가장 빠르게 대응할 수 있는 방법
- 비동기적으로 동작하는 CPU(고속)와 주변장치(저속) 사이에서 효율적으로 일을 수행
- 인터럽트가 발생하면 나중에 돌아올 복귀주소(return address)가 자동적으로 스택에 저장되었다가, 인터럽트 서비스루틴의 마지막에서 복귀 명령을 만나면 다시 자동적으로 복귀주소로 돌아온다.



중요. 인터럽트

ATmega128 인터럽트 종류

- 타이머에서 지정된 시간 경과
- 입력장치에서의 서비스 요구
- 출력장치의 작업종료
- A/D 변환의 완료
- DMA 동작의 종료
- 멀티프로세서간의 통신 요구 등 마이크로프로세서와 독립되어 있는 외부장치에 의해 발생되는 순수한 의미에서의 인터럽트

- 차단 가능 인터럽트
  - 프로그래머에 의하여 인터럽트 요청을 받아들이지 않고 무시할 수 있는 것
  - 시간제약이 중요한 프로그램 수행 중에는 인터럽트 요청을 허용하지 않을 수 있다.
- 차단 불가능 인터럽트
  - 프로그래머에 의하여 어떤 방법으로도 인터럽트 요청이 차단될 수 없는 것
  - 전원이상, 비상정지 스위치 등 돌발사태에 대비하기 위한 것

중요. 인터럽트

인터럽트 차단 및 허용

- 인터럽트 마스크 레지스터, 인터럽트 허용 레지스터를 사용하여 개별적으로 차단 및 허용 가능
- EIMSK: 개별적 인터럽트의 차단/허용
- SEI: Set Global Interrupt Flag, Global Interrupt Enable
- CLI: Clear Global Interrupt Flag, Global Interrupt Disable

벡터형 인터럽트

- 인터럽트가 발생할 때마다 인터럽트를 요청한 장치가 인터럽트 서비스 루틴의 시작번지를 CPU에게 전송하거나, 또는 CPU가 각 인터럽트의 종류에 따라 미리 지정된 메모리 번지에서 인터럽트 벡터를 읽어서 이를 인터럽트 서비스 루틴의 시작번지로 사용하는 방식
- 인터럽트 시간이 빠르다
- 주변장치의 많고 적음에 영향이 없다.
- ATmega128의 모든 인터럽트는 이 방식

중요. 인터럽트

외부 인터럽트

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
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	$0039(3)	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A(3)	TIMER3 OVF	Timer/Counter3 Overflow

중요. 인터럽트

외부 인터럽트

- 외부 인터럽트가 8개 지원되는데 이는 8개의 외부 핀(INT 7~0)을 통해서 입력되는 신호에 의하여 발생하는 인터럽트를 말한다

- 외부 인터럽트(INT 7~0)의 특징
  - 외부 인터럽트는 INT 7~0 핀의 트리거 동작으로 인터럽트가 발생된다.
  - LOW / 상승엣지 / 하강엣지 의 방식으로 트리거 신호를 선택할 수 있다.
  - 외부 인터럽트는 INT 7~0 핀의 입/출력 방향에 관계없이 인터럽트가 발생된다.
  - (소프트웨어적으로 데이터를 출력하여 인터럽트를 요구 할 수도 있다.)
  - INT 7~4 : I/O클럭이 있어야만 사용가능
  - INT 3~0 : 비동기적 검출이 가능하다.(슬립모드를 깨울 때 이용되기도 한다.)

중요. 인터럽트

외부 인터럽트

- 외부 인터럽트를 제어하기 위해서는 상태레지스터(SREG)와 외부 인터럽트 관련 레지스터(EIMSK, EICRA, EICRB, EIFR)의 사용법

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	

- MCU의 현 상태 및 최근 수치 명령 실행에 대한 결과를 포함한다.
- 상태레지스터는 모든 ALU 연산을 수행 후 갱신된다.
- Bit7. I (Global Interrupt Enable) : 모든 인터럽트 활성화 비트

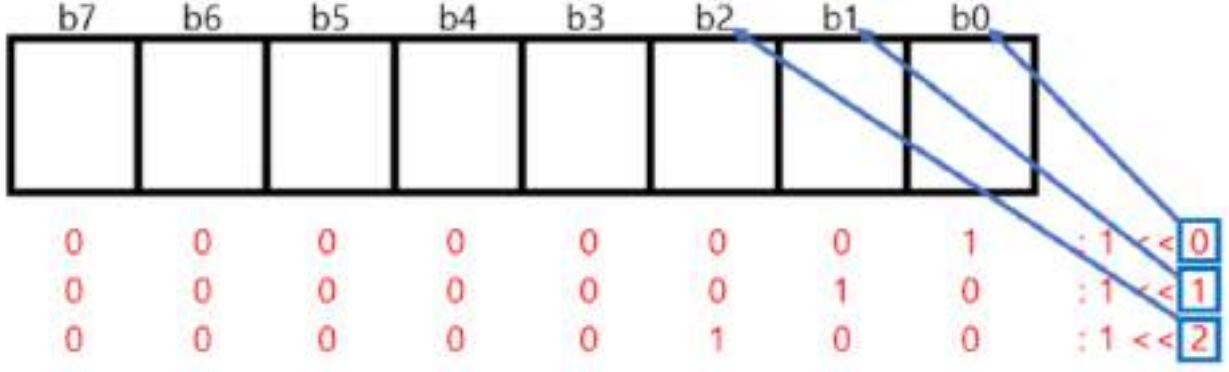
중요. 인터럽트

외부 인터럽트

- **EIMSK(External Interrupt Mask Register)**

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	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	

- **Bit 7..0 – INT 7~0 Q 비트를 셋(1) 시키면 해당 외부 인터럽트 핀이 활성화 된다.**
- **단, SREG의 I비트가 1로 셋된 상태여야 한다.**



쉬프트 연산을 수행할 숫자는 비트의 번호와도 대응이 된다.
즉, 2번비트를 1로 만들고 싶으면 2만큼 쉬프트 연산을 수행해주면 되는것이다.
그러므로 코드를 짤 때, 1<<3 이라는것은 3번비트가 1인 값이라고도 해석 할 수 있어야 한다.

중요. 인터럽트

외부 인터럽트

- **EICRA(External Interrupt Control Register A)**

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	

  - 외부 인터럽트의 트리거 방식을 LOW / 상승엣지 / 하강엣지 중에 선택하는 레지스터

ISCn1	ISCn0	설 명
0	0	INTn의 Low 신호시에 일반적인 인터럽트 요청을 한다.
0	1	(reserved)
1	0	INTn의 하강엣지 신호시에 일반 비동기적인 인터럽트를 요청한다
1	1	INTn의 상승엣지 신호시에 일반 비동기적인 인터럽트를 요청한다

EICRB (External Interrupt Control Register B)

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	

- **EIFR(External Interrupt Flag Register)**
- **Bit 7..0 - 외/내부로부터 INT7:0 핀에 인터럽트가 요청되면 해당 비트가 Set(1) 된다. 그 후에 인터럽트 루틴이 실행될 때 해당 비트가 Clear(0)된다.**

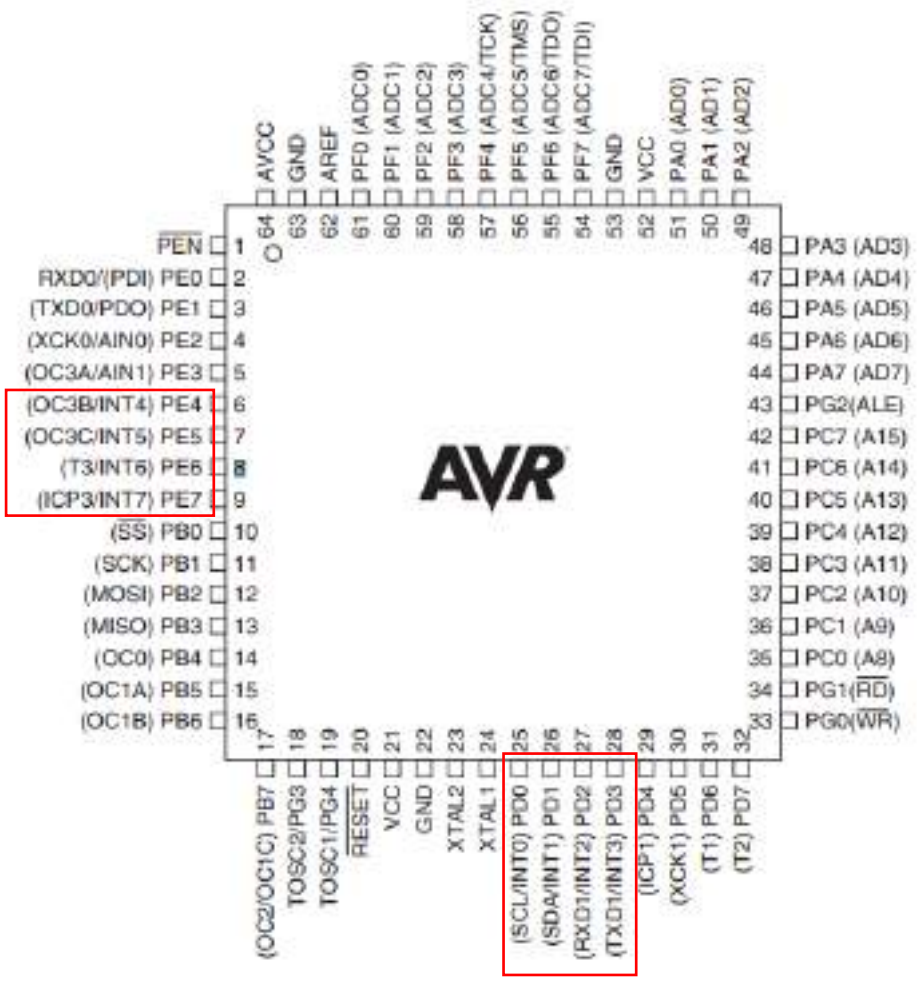
중요. 인터럽트

외부 인터럽트

- **외부 인터럽트 동작**
  - 인터럽트가 활성화(SREG.7 / EIMSK 해당 비트 활성화) 되어 있는 상태에서
  - 외부INT의 동작 엣지나 논리신호에 의해 인터럽트가 요청되면
  - 제일 우선 INTFn=1 상태로 플래그가 Set되고
  - 실행 중이던 메인 프로그램의 프로그램카운터 값을 스택에 저장하게 된다.
  - 그 후 해당 인터럽트 벡터로 점프하게 되며
  - 지정된 인터럽트 루틴을 실행하는 동시에 INTFn=0 상태로 플래그가 Clear 된다.
  - 해당 인터럽트의 루틴이 처리된다.
  - 해당 인터럽트의 루틴이 끝나게 되면 RETI 명령을 받는다.
  - 스택으로부터 저장된 프로그램카운터 값을 로드 한다.
  - 동작 중이던 프로그램 위치로 복귀하여 실행 중이던 부분부터 메인 프로그램이 재작동된다.

중요. 인터럽트

외부 인터럽트 핀



(SCL/INT0) PD0 25
(SDA/INT1) PD1 26
(RXD1/INT2) PD2 27
(TXD1/INT3) PD3 28

(OC3B/INT4) PE4 6
(OC3C/INT5) PE5 7
(T3/INT6) PE6 8
(ICP3/INT7) PE7 9

ATmega128에서 외부인터럽트를 사용하기 위한 입력 핀은 미리 할당 되어있다.

→PD0~PD3, PE4~PE7

중요. 인터럽트

[실습 15]

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

interrupt [EXT_INT0] void ext_int0_isr (void)          //핵심 인터럽트 걸릴 시 이곳으로 점프
{
    PORTA = 0x0F;                                      //하위 4비트만 LED를 켬
    delay_ms(1000);
}

void main(void)
{
    PORTA = 0x00;
    DDRA = 0xFF;

    EIMSK = 0x01;                                      //INT0 의 인터럽트를 허용
    EICRA = 0x03;                                      //인터럽트0 의 하강에지 트리거

    #asm("sei");                                       //모든 인터럽트 허용 (어셈블리어 명령어)
    //SREG |= 0x80;                                    //모든 인터럽트 허용 (C 명령어)

    while(1)
    {
        PORTA = 0xFF;                                  // PORTA의 LED를 1초에 한번씩 켰다가 껐다가 반복
        delay_ms(1000);
        PORTA = 0x00;
        delay_ms(1000);
    }
}
```

중요. 인터럽트

[실습 16]

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

interrupt [EXT_INT0] void ext_int0_isr (void)  //외부 인터럽트 0의 서비스 루틴
{                                              //LED 1개가 오른쪽으로 이동하는 카운트 동작
    unsigned char i;

    for(i=0x80; i; i >>= 1)
    {
        PORTA = i;
        delay_ms(1000);
    }
    PORTA = 0xFF;                              //전체 LED를 켜겠습니다.
}

interrupt [EXT_INT1] void ext_int1_isr (void)  //외부 인터럽트 1의 서비스 루틴
{                                              //전체 점등 상태에서 LED가 1개씩 오른쪽으로 이동하면서 소등
    unsigned char i;

    for(i=0xFF; i; i >>= 1)
    {
        PORTA = i;
        delay_ms(1000);
    }
    PORTA = 0xFF;                              //전체 LED를 켜겠습니다.
}
```

중요. 인터럽트

[실습 16]

```
interrupt [EXT_INT2] void ext_int2_isr (void)  //외부 인터럽트 2의 서비스 루틴
{                                              //전체 점등 상태에서 LED가 1개씩 왼쪽으로 이동하면서 소등
    unsigned char i;

    for(i=0xFF; i; i <<= 1)
    {
        PORTA = i;
        delay_ms(1000);
    }
    PORTA = 0xFF;                              //전체 LED를 켜겠습니다.
}

void main(void)
{
    PORTA = 0xFF;
    DDRA = 0xFF;
    EIMSK = 0x07; //INT0, INT1, INT2 의 인터럽트를 허용!
    EICRA = 0x03; //인터럽트0 의 하강 에지 트리거
    EICRA |= 0x08; //인터럽트1 의 하강 에지 트리거
    EICRA |= 0x20; //인터럽트2 의 하강 에지 트리거
    #asm("sei"); //모든 인터럽트 허용
    while(1)
    {
        PORTA = 0xFF;
        delay_ms(1000);
        PORTA = 0x00;
        delay_ms(1000);
    }
}
```

중요. 인터럽트

[실습 16]

- 코드 설명을 덧붙이자면 외부 인터럽트 2, 1, 0 세 개를 이용하여서 각각의 인터럽트를 걸어주었습니다. 외부 인터럽트의 우선 순위는 항상 벡터 번호가 작을 수록 더 높은 우선 순위를 가집니다.

- 또한 인터럽트 서브 루틴이 실행 되고 있는 도중에 그보다 더 높은 우선순위의 인터럽트가 걸린다 하더라도, 현재 진행하고 있는 서비스 루틴을 마저 끝내고 그 다음 인터럽트 서브 루틴을 실행합니다.

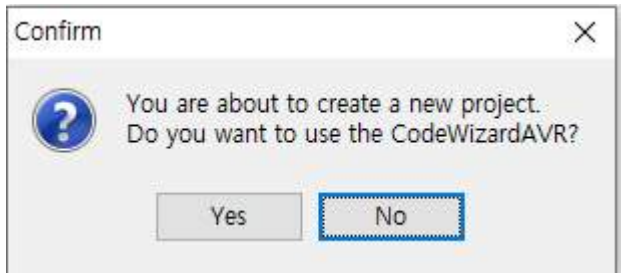
- 이 때, 새롭게 요구된 인터럽트가 2개 이상일 경우에는 요청된 순서와 관계없이 가장 우선순위가 높은 인터럽트가 먼저 걸립니다.

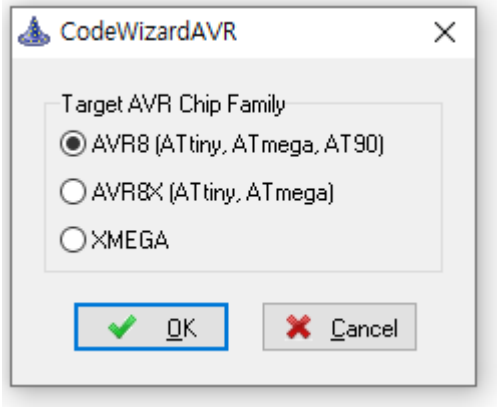
- 또한 동일한 인터럽트가 2번 이상 발생하면 두 번째 인터럽트 요청부터는 무시되게 됩니다.이와 같은 이유는 I 비트의 인터럽트 플래그에 의해서 인터럽트 요청을 기억하는 거라서 다중으로 요청된 인터럽트는 기억할 수 없기 때문입니다.

중요. 인터럽트

외부인터럽트 코드 작성법 – CodeWizard 이용

1. CodeWizard를 실행하지 않은 상태로 프로젝트를 만들고, CodeWizard 실행(Shift+F2)

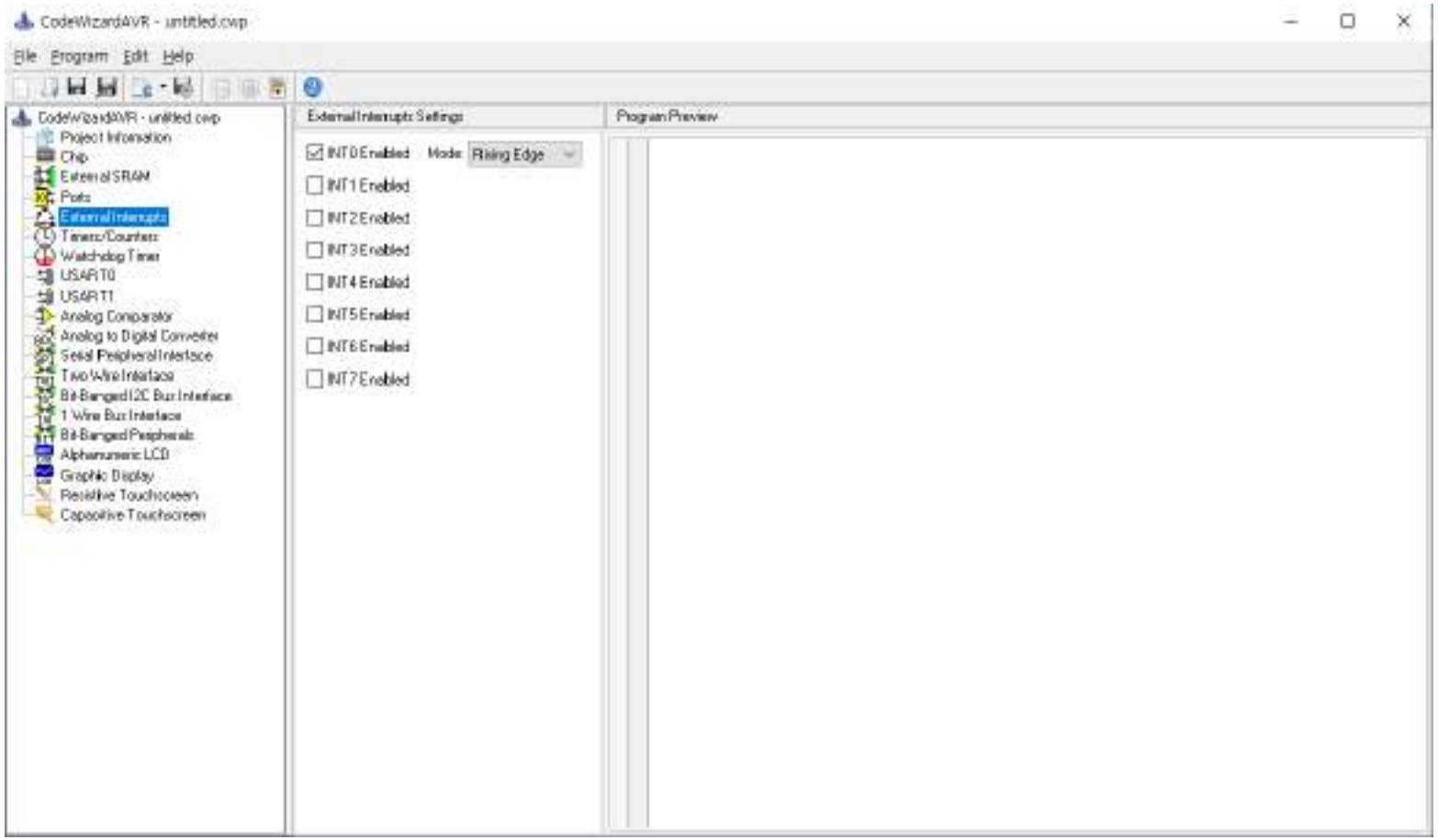




중요. 인터럽트

외부인터럽트 코드 작성법 – CodeWizard 이용

2. External Interrupts를 클릭, INT0을 체크하고, Generate code 아이콘을 클릭. 



중요. 인터럽트

외부인터럽트 코드 작성법 – CodeWizard 이용

3. 오른 쪽 창에 긴 코드가 생성되는데, 우리가 생성한 External Interrupts에 필요한 부분만 복사하여 프로그램 하는데 사용하면 된다.

```
#include <mega128.h>

// Declare your global variables here

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here

}
```

```
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=(0<<ISC31) | (0<<ISC30) | (0<<ISC21) | (0<<ISC20) | (0<<ISC11) | (0<<ISC10) | (1<<ISC01) | (1<<ISC00);
EICRB=(0<<ISC71) | (0<<ISC70) | (0<<ISC61) | (0<<ISC60) | (0<<ISC51) | (0<<ISC50) | (0<<ISC41) | (0<<ISC40);
EIMSK=(0<<INT7) | (0<<INT6) | (0<<INT5) | (0<<INT4) | (0<<INT3) | (0<<INT2) | (0<<INT1) | (1<<INT0);
EIFR=(0<<INTF7) | (0<<INTF6) | (0<<INTF5) | (0<<INTF4) | (0<<INTF3) | (0<<INTF2) | (0<<INTF1) | (1<<INTF0);
```

```
// Globally enable interrupts
#asm("sei");
```

→모든 인터럽트를 허용하는 명령, 인터럽트를 사용할 때는 꼭 입력해 주도록 한다.

중요. 인터럽트

외부인터럽트 예제

#. 500ms 간격으로 LED 8개를 모두 깜빡이게 하고, 도중에 외부인터럽트를 걸어
하던 동작을 멈추고 LED 상위 4비트 만을 500ms 동안 켜는 동작을 하는 프로그램

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

interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here
  PORTA = 0xF0;
  delay_ms(500);

}

void main(void)
{


PORTA = 0x00;
DDRA = 0xFF;
DDRD = 0x00;


EICRA=(0<<ISC31) | (0<<ISC30) | (0<<ISC21) | (0<<ISC20) | (0<<ISC11) | (0<<ISC10) | (1<<ISC01) | (1<<ISC00);       //INT0~3에 대한 설정
EICRB=(0<<ISC71) | (0<<ISC70) | (0<<ISC61) | (0<<ISC60) | (0<<ISC51) | (0<<ISC50) | (0<<ISC41) | (0<<ISC40);       //INT4~7에 대한 설정
EIMSK=(0<<INT7) | (0<<INT6) | (0<<INT5) | (0<<INT4) | (0<<INT3) | (0<<INT2) | (0<<INT1) | (1<<INT0);  //INT7-INT0 중 어느 핀을 외부인터럽트 핀으로 사용할지
EIFR=(0<<INTF7) | (0<<INTF6) | (0<<INTF5) | (0<<INTF4) | (0<<INTF3) | (0<<INTF2) | (0<<INTF1) | (1<<INTF0);



// Globally enable interrupts
#asm("sei");

while (1)
      {
      // Place your code here
         PORTA = 0xFF;
         delay_ms(500);
         PORTA = 0x00;
         delay_ms(500);
      }

}
```