

임베디드 시스템을 위한 SW 구조 설계

(file #4 / 10) ver0.2

Yongseok Chi

1. Develop an understanding of technologies

about the micro controller & processor systems using evaluation kit

2. Skill up a **design ability the micro controller application systems**

(1) technology of **the hardware and software** components

(2) **debugging** technology about the micro controller

(3) understanding of a **circuit design** skill

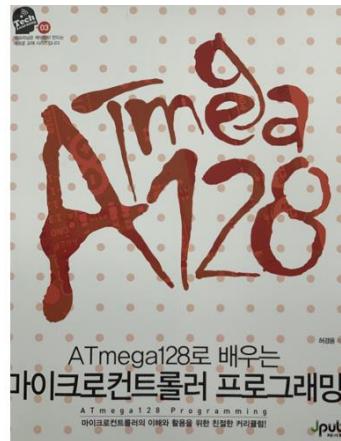
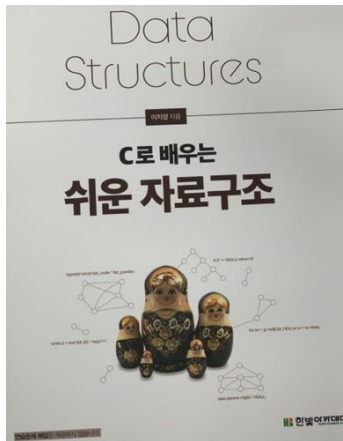
3. Develop an ability of design about the micro controller

1. Reference information

(1) Atmel datasheet, <http://atmel.com> (→ microchip.com)

(2) ARM Architecture Reference Manual, <http://arm.com>

2. Books



3. 실험KIT (Evaluation board)

한백전자 IOT 실험 KIT



KUT-128_Com 실험 키트

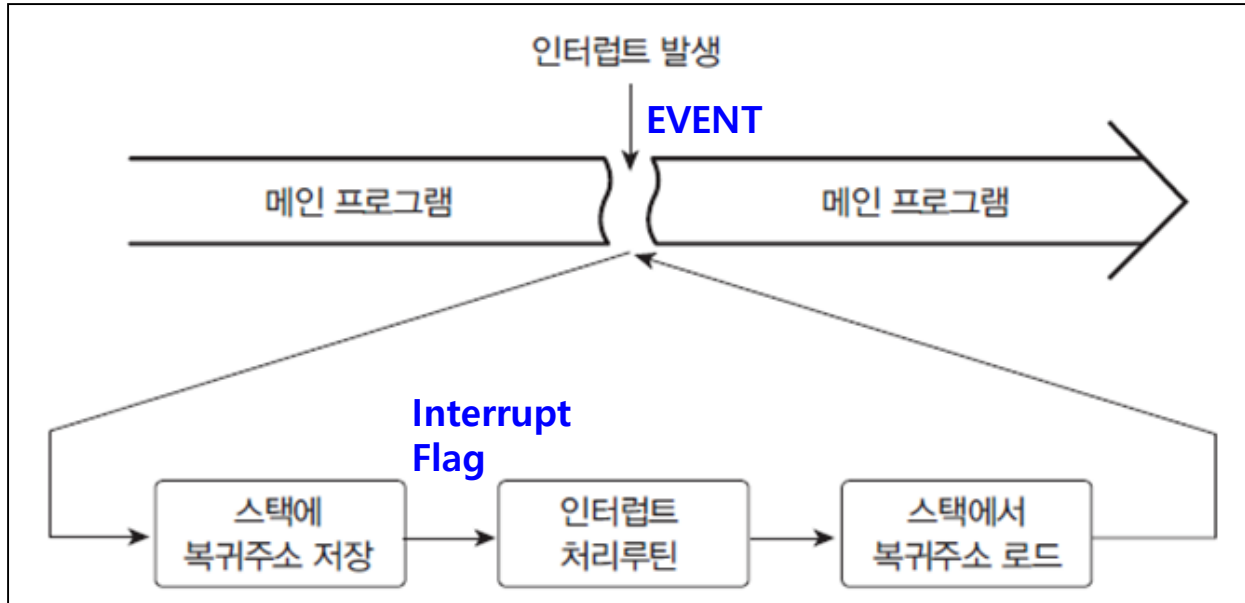
-
1. Micro Processor 원리
 2. Atmel사의 8bit Micro-controller
 3. KUT0128 Evaluation Board 기능과 특징
 4. IO Port 제어
 5. External Interrupt 제어
 6. Timer counter 제어
 7. UART 제어
 8. AD Converter 제어
 9. Comparator 제어
 10. EEPROM 제어 (IIC, Parallel method)
 11. SPI 제어

-
1. Micro Processor 원리
 2. Atmel사의 8bit Micro-controller
 3. KUT0128 Evaluation Board 기능과 특징
 4. IO Port 제어
 5. External Interrupt 제어
 6. Timer counter 제어
 7. UART 제어
 8. AD Converter 제어
 9. Comparator 제어
 10. EEPROM 제어 (IIC, Parallel method)
 11. SPI 제어

Interrupt

- 마이크로프로세서는 **한번에 하나의 프로그램(명령)만 실행 가능**, 이때 긴급히 처리해야 할 프로그램이 있을 경우
- **현재 실행중인 프로그램을 중지하고 긴급한 프로그램의 실행**을 끝낸 후 중지한 프로그램을 계속 실행
- 프로세서의 내/외부 장치가 프로세서에게 **특정 이벤트(event)가 발생함을 알려서 이벤트를 처리하는 과정** (**Event : Interrupt flag, 처리 : Interrupt service routine**)
- **이벤트는 프로세서의 내부 장치나 외부 I/O 장치에서 비정기적으로 발생하기** 때문에 Interrupt 처리를 통해 주변 장치의 서비스(이벤트 처리) 요청을 효율적으로 다룰 수 있음

Interrupt 처리 과정



- 실행 프로그램의 중단점 주소를 Stack에 저장
- 인터럽트 벡터(Interrupt vector)에 위치한 Interrupt 처리 루틴(Interrupt Service Routine)를 실행.
- Stack에서 중단점의 주소를 load하여 원래 프로그램의 중단점으로 복귀

Interrupt vs. Polling

- MCU에서 입력을 받아들이는 방법은 **Polling 방식**과 **인터럽트 방식**이 있음
 - Polling방식
 - : 사용자의 명령어에 의해서 하드웨어의 변경사항을 주기적 으로 읽어 들이는 방식
 - **Polling 방식은 주기적(SW적으로)으로** 하드웨어의 변화를 체크하기 때문에 사용자의 프로그래밍에 따라 다양한 변화에 대응이 가능하지만 CPU의 점유율이 높기 때문에 **반응속도가 상대적으로 느리다.**
 - Interrupt 방식
 - : MCU자체가 하드웨어적으로 변화를 체크하여 변화될 때만 정해진 동작을 수행하는 방식 (**Hardwired, Interrupt vector table**)
 - 하드웨어적으로 정해진 변화에만 대응이 가능하지만, 변화가 있을 때만 MCU에게 신호를 전달하므로 MCU의 점유율이 낮아서 처리속도가 더 빠르다.

5. External Interrupt 제어

Strictly Private and Confidential

Interrupt 구성

- Interrupt 소스

: Interrupt를 발생시킬 수 있도록 만들어진 것
(외부 입력 핀, 타이머, 시리얼 포트 등)

- Interrupt Vector Table (Interrupt 점프 테이블)

: Interrupt 발생시 분기 장소를 기록해 놓은 특정 번지의 내용

- Interrupt Service Routine (Interrupt Handler)

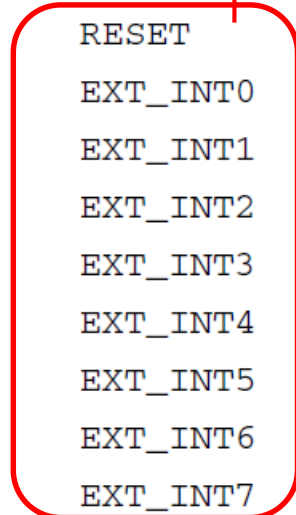
: Interrupt 발생시 처리할 프로그램

예) Interrupt Vector Address

Datasheet pdf file : 61page

Address	Labels	Code	Comments
\$0000	jmp	RESET	; Reset Handler
\$0002	jmp	EXT_INT0	; IRQ0 Handler
\$0004	jmp	EXT_INT1	; IRQ1 Handler
\$0006	jmp	EXT_INT2	; IRQ2 Handler
\$0008	jmp	EXT_INT3	; IRQ3 Handler
\$000A	jmp	EXT_INT4	; IRQ4 Handler
\$000C	jmp	EXT_INT5	; IRQ5 Handler
\$000E	jmp	EXT_INT6	; IRQ6 Handler
\$0010	jmp	EXT_INT7	; IRQ7 Handler

Interrupt Source 명



5. External Interrupt 제어

Strictly Private and Confidential

Interrupt Vector Table

벡터	프로그램 주소	인터럽트 소스	인터럽트 설명
1	0x0000	RESET	외부 리셋, 파워온 리셋/워치독 리셋
2	0x0002	INT0	외부 인터럽트 요구0
3	0x0004	INT1	외부 인터럽트 요구1
4	0x0006	INT2	외부 인터럽트 요구2
5	0x0008	INT3	외부 인터럽트 요구3
6	0x000A	INT4	외부 인터럽트 요구4
7	0x000C	INT5	외부 인터럽트 요구5
8	0x000E	INT6	외부 인터럽트 요구6
9	0x0010	INT7	외부 인터럽트 요구7
10	0x0012	TIMER2 COMP	타이머/카운터2 비교 매치
11	0x0014	TIMER2 OVF	타이머/카운터2 오버플로
12	0x0016	TIMER1 CAPT	타이머/카운터1 캡처 이벤트
13	0x0018	TIMER1 COMP	타이머/카운터1 비교 매치 A
14	0x001A	TIMER1 COMPB	타이머/카운터1 비교 매치 B

5. External Interrupt 제어

Strictly Private and Confidential

Interrupt Vector Table

벡터	프로그램 주소	인터럽트 소스	인터럽트 설명
15	0X001C	TIMER1 OVF	타이머/카운터1 오버플로
16	0X001E	TIMER0 COMP	타이머/카운터0 비교 매치
17	0X0020	TIMER0 OVF	타이머/카운터0 오버플로
18	0X0022	SPI, STC	SPI 직렬 전송완료
19	0X0024	USART0, RX	USART0 수신완료
20	0X0026	USART0, UDRE	USART0 데이터 레지스터 빔
21	0X0028	USART0, TX	USART0 송신완료
22	0X002A	ADC	ADC 변환 완료
23	0X002C	EE READY	EEPROM READY
24	0X002E	ANALOG COMP	아날로그 비교기
25	0X0030	TIMER1 COMPC	타이머/카운터1 비교 매치 C
26	0X0032	TIMER3 CAPT	타이머/카운터3 캡처 이벤트
27	0X0034	TIMER3 COMPA	타이머/카운터3 비교 매치 A
28	0X0036	TIMER3 COMPB	타이머/카운터3 비교 매치 B

Interrupt Vector Table

벡터	프로그램 주소	인터럽트 소스	인터럽트 설명
29	0X0038	TIMER3 COMPC	타이머/카운터3 비교 매치 C
30	0X003A	TIMER3 OVF	타이머/카운터3 오버플로
31	0X003C	USART1,RX	USART1 수신 완료
32	0X003E	USART1,UDRE	USART1 데이터 레지스터 빔
33	0X0040	USART1,TX	USART1 데이터 전송 완료
34	0X0042	TWI	Two-wire 직렬 인터페이스
35	0X0044	SPM READY	Store Program Memory Ready

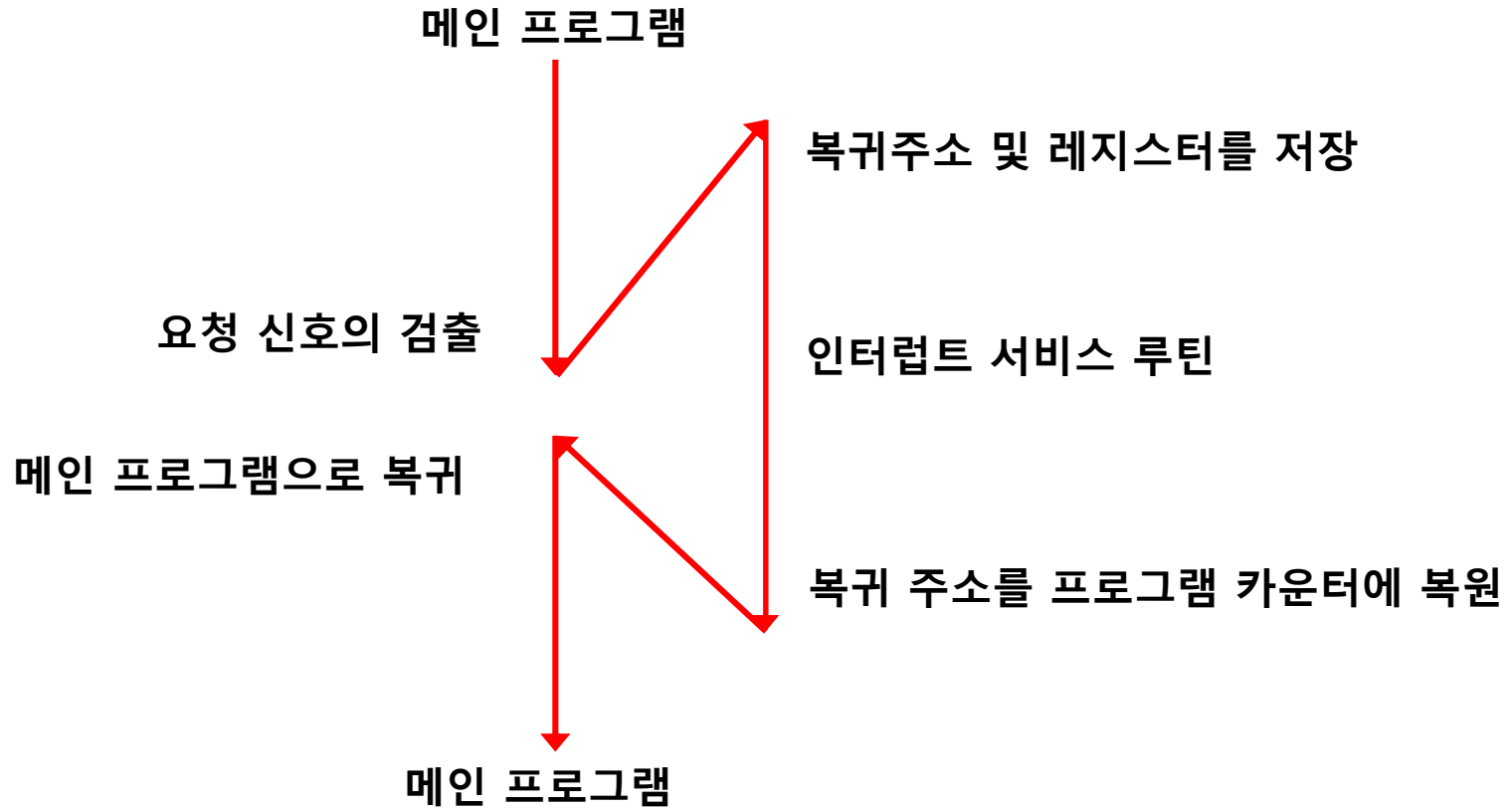
Interrupt 상세 처리 과정

- Interrupt 요청 신호의 검출
: 모든 Interrupt마다 그에 대응되는 **Interrupt flag register**를 가지고 있어서 Interrupt 가 검출되면 Interrupt flag 는 **1(set)**로 되며, **Interrupt Service Routine**이 실행되면 자동으로 **0(clear)**
- Interrupt 우선순위 제어 및 허용 여부 판단
: 각 Interrupt에 해당되는 **Interrupt Mask register**와 SREG (status register)의 **Global Interrupt 허용 bit[7]**를 보고 Interrupt 허용 여부를 판단)
- **Interrupt Service Routine** (인터럽트 처리루틴)의 시작 번지 확인
: 각 Interrupt 에 따라, **Interrupt Vector**의 번지가 **미리 정해져** 있어서 여기에 사용자가 Interrupt Service Routine의 시작번지를 저장해두어야 한다
- 복귀주소 및 register를 저장
: Atmega128은 복귀 주소만을 stack에 저장하며 다른 레지스터는 저장하지 않는다.
- **Interrupt Service Routine**을 실행
: 해당 Interrupt Service Routine으로 점프하여 프로그램을 실행
- Interrupt Service Routine을 종료하고 **원래의 주 프로그램으로 복귀**
: 인터럽트 서비스 루틴을 종료하고 복귀 주소를 되찾아 PC(프로그램 카운터)에 로드 함으로써 원래의 위치로 되돌아온다.

5. External Interrupt 제어

Strictly Private and Confidential

Interrupt 상세 처리 과정



유형별 ATmega128 Interrupt 발생 원인

유형	발생 원인	인터럽트 벡터
리셋	<ul style="list-style-type: none"> • 핀 리셋 • 전원 공급 시작 • 전압 미달 • 와치독 • JTAG 리셋 	0
외부 장치 인터페이스	<ul style="list-style-type: none"> • 전압레벨 Low • 상승에지 • 하강에지 • 외부 인터럽트 0~3 	1
		2
		3
		4
	<ul style="list-style-type: none"> • 전압레벨 Low • 상승에지 • 하강에지 • 상승 또는 하강에지 • 외부 인터럽트 4~7 	5
		6
		7
		8

유형별 ATmega128 Interrupt 발생 원인

유형	발생 원인	인터럽트 벡터
타이머 인터페이스	• 타이머/카운터2 PWM	9
	• 타이머/카운터2 오버플로	10
	• 타이머/카운터1 신호 입력 순간	11
	• 타이머/카운터1 PWM A	12
	• 타이머/카운터1 PWM B	13
	• 타이머/카운터1 오버플로	14
	• 타이머/카운터0 PWM	15
	• 타이머/카운터0 오버플로	16
	• 타이머/카운터1 PWM C	24
	• 타이머/카운터3 신호 입력 순간	25
	• 타이머/카운터3 PWM A	26
	• 타이머/카운터3 PWM B	27
	• 타이머/카운터3 PWM C	28
	• 타이머/카운터3 오버플로	29

유형별 ATmega128 Interrupt 발생 원인

유형	발생 원인	인터럽트 벡터
내부 장치 인터페이스	• SPI 장치 전송 완료	17
	• USART0 수신 완료	18
	• USART0 데이터 쓰기 가능	19
	• USART0 송신 완료	20
	• A/D 변환 완료	21
	• EEPROM 쓰기 가능	22
	• 비교기값 변화 알림	23
	• USART1 수신 완료	30
	• USART1 데이터 쓰기 가능	31
	• USART1 송신 완료	32
	• 2선 직렬 장치 작업 완료	33
메모리 인터페이스	• SPM 완료	34

Interrupt 활성화와 Interrupt Service Routine 연결

[내부 장치 활성화 레지스터와 비트명]

내부 장치	레지스터명	비트명	초깃값(활성 상태)
시리얼 통신*	USARTnB	RXENn TXENn	0(비활성) 0(비활성)
SPI 통신	SPCR	SPE	0(비활성)
2선 직렬통신	TWCR	TWEN	0(비활성)
아날로그 비교기	ACSR	ACD	0(활성)
ADC	ADCSRA	ADEN	0(비활성)

* 시리얼 통신 장치는 2개가 있고, n은 0 또는 1

Interrupt 활성화와 Interrupt Service Routine 연결

장치와 연결핀		인터럽트 벡터	인터럽트 루틴 명칭	레지스터	활성화 비트
장치	연결핀				
핀에 연결된 외부 장치	INT0	1	INT0_vect	EIMSK	INT0
	INT1	2	INT1_vect		INT1
	INT2	3	INT2_vect		INT2
	INT3	4	INT3_vect		INT3
	INT4	5	INT4_vect		INT4
	INT5	6	INT5_vect		INT5
	INT6	7	INT6_vect		INT6
	INT7	8	INT7_vect		INT7

5. External Interrupt 제어

Strictly Private and Confidential

Interrupt 활성화와 Interrupt Service Routine 연결

장치와 연결핀		인터럽트 벡터	인터럽트 루틴 명칭	레지스터	활성화 비트
장치	연결핀				
타이머/카운터2	OC2	9	TIMER2_COMP_vect	TIMSK	OCIE2
	—	10	TIMER2_OVF_vect		TOIE2
타이머/카운터1	ICP1	11	TIMER1_CAPT_vect		TICIE1
	OC1A	12	TIMER1_COMPA_vect		OCIE1A
	OC1B	13	TIMER1_COMPB_vect		OCIE1B
	—	14	TIMER1_OVF_vect		TOIE1
타이머/카운터0	OC0	15	TIMER0_COMP_vect		OCIE0
	—	16	TIMER0_OVF_vect		TOIE0
SPI 통신 장치	MISO MOSI SCK /SS	17	SPI_STC_vect	SPCR	SPIE
시리얼 통신 장치(USART0)	RXD0	18	USART0_RX_vect	UCSR0B	RXCIE0
	TXD0	19	USART0_UDRE_vect		UDRIE0
	—	20	USART0_TX_vect		TXCIE0

5. External Interrupt 제어

Strictly Private and Confidential

Interrupt 활성화와 Interrupt Service Routine 연결

장치와 연결핀		인터럽트 벡터	인터럽트 루틴 명칭	레지스터	활성화 비트
장치	연결핀				
ADC 장치	ADC0 ~ ADC7	21	ADC_vect	ADCSRA	ADIE
EEPROM	—	22	EE_READY_vect	EECR	EERIE
아날로그 비교기	AIN0 AIN1	23	ANALOG_COMP_vect	ACSR	ACIE
타이머/카운터1	OC1C	24	TIMER1_COMPC_vect	ETIMSK	OCIE1C
타이머/카운터3	ICP3	25	TIMER3_CAPT_vect		TICIE3
	OC3A	26	TIMER3_COMPA_vect		OCIE3A
	OC3B	27	TIMER3_COMPB_vect		OCIE3B
	OC3C	28	TIMER3_COMPC_vect		OCIE3C
	—	29	TIMER3_OVF_vect		TOIE3
시리얼 통신 장치(USART1)	RXD1	30	USART1_RX_vect	UCSR1B	RXCIE1
	TXD1	31	USART1_UDRE_vect		UDRIE1
		32	USART1_TX_vect		TXCIE1
2선 직렬통신	SDA SCL	33	TWI_vect	TWCR	TWIE
플래시 프로그램 메모리	—	34	SPM_READY_vect	SPMCSR	SPMIE

5. External Interrupt 제어

Strictly Private and Confidential

MCUCR(MCU Control Register)

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	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	

- Bit 1 – IVSEL : **Interrupt Vector Select**

- 비트가 0이면 Interrupt Vector Table는 플래시 메모리의 시작에 위치
- 비트가 1이면 Interrupt Vector Table는 플래시 메모리의 부트 로더 섹션의 선두 위치로 이동
- 부트 로더의 선두 어드레스는 퓨즈 비트 BOOTSZ1..0에 의해 결정
- 이 비트에 새로운 값을 쓰기 위해서는 IVCE 비트를 우선적으로 1로 하고 4cycle 이내에 원하는 값 Write. 이 4cycle 동안 MCU는 인터럽트 금지상태

- Bit 0 – IVCE : Interrupt Vector Change Enable

- IVSEL 비트를 변경할 때 사용
- 1이 되고 4 사이클 후 또는 IVSEL에 값이 set 되었을 때 하드웨어적으로 clear
- IVCE 비트에 1을 쓰고, 4사이클 이내에 IVSEL 비트에 원하는 값 Write

Interrupt 기본 사항

- 인터럽트 벡터가 낮은 주소를 갖는 인터럽트의 우선순위가 높음
 - 0x0000 RESET (외부 Reset)의 우선 순위가 가장 높음
 - 0x0002 INT0 (외부 Interrupt 0)은 두 번째로 높은 우선 순위를 갖는다
- Interrupt 가 발생하여 Interrupt Service Routine 으로 점프하게 되고,
SREG register의 Global Interrupt enable 비트 I는 0으로 clear, 모든 Interrupt 를 무시
→ Nesting 금지
 - Interrupt Service Routine 에서 Interrupt 를 enable 시키기 위해 I 비트를 set 할 수도 있음
- Interrupt Service Routine 을 빠져 나오는 명령 RETI를 사용하면 I비트가 자동적으로 set 된다.

SREG(Status Register) 상태 레지스터

: 가장 최근에 실행된 산술연산의 결과 정보를 저장

비트	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

- I (Global Interrupt Enable)

5. External Interrupt 제어

Strictly Private and Confidential

External Interrupt (외부 인터럽트)

- ATmega128 External Interrupt는 **PD0~PD3(INT0~INT3)**과 **PE4~PE7(INT4~INT7)** 핀에 의해 발생

- 외부 인터럽트 INT0~INT7

: **Edge trigger (rising or falling)** 또는

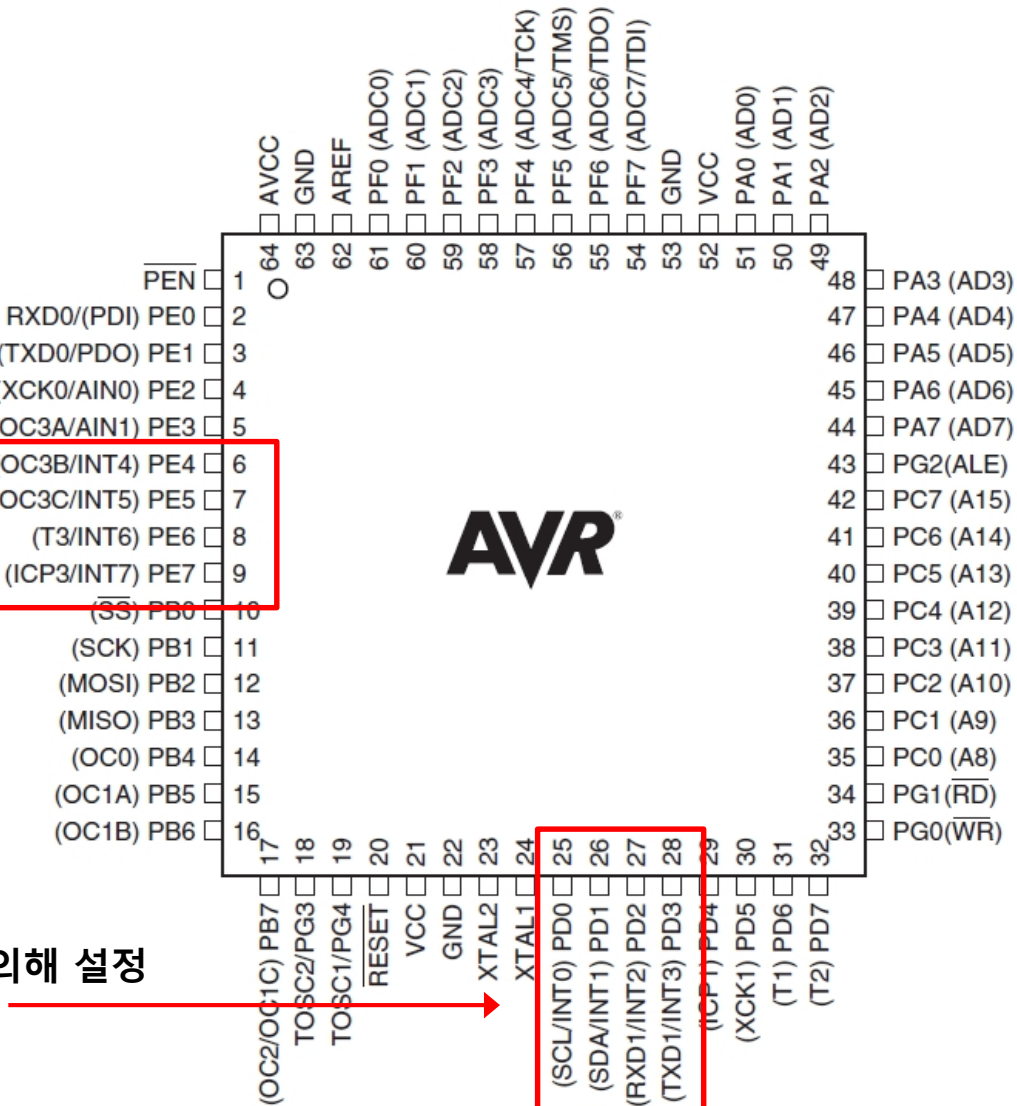
: **Level trigger (Low level)**에 의해

External Interrupt 발생

- **EICRB** register에 의해 설정



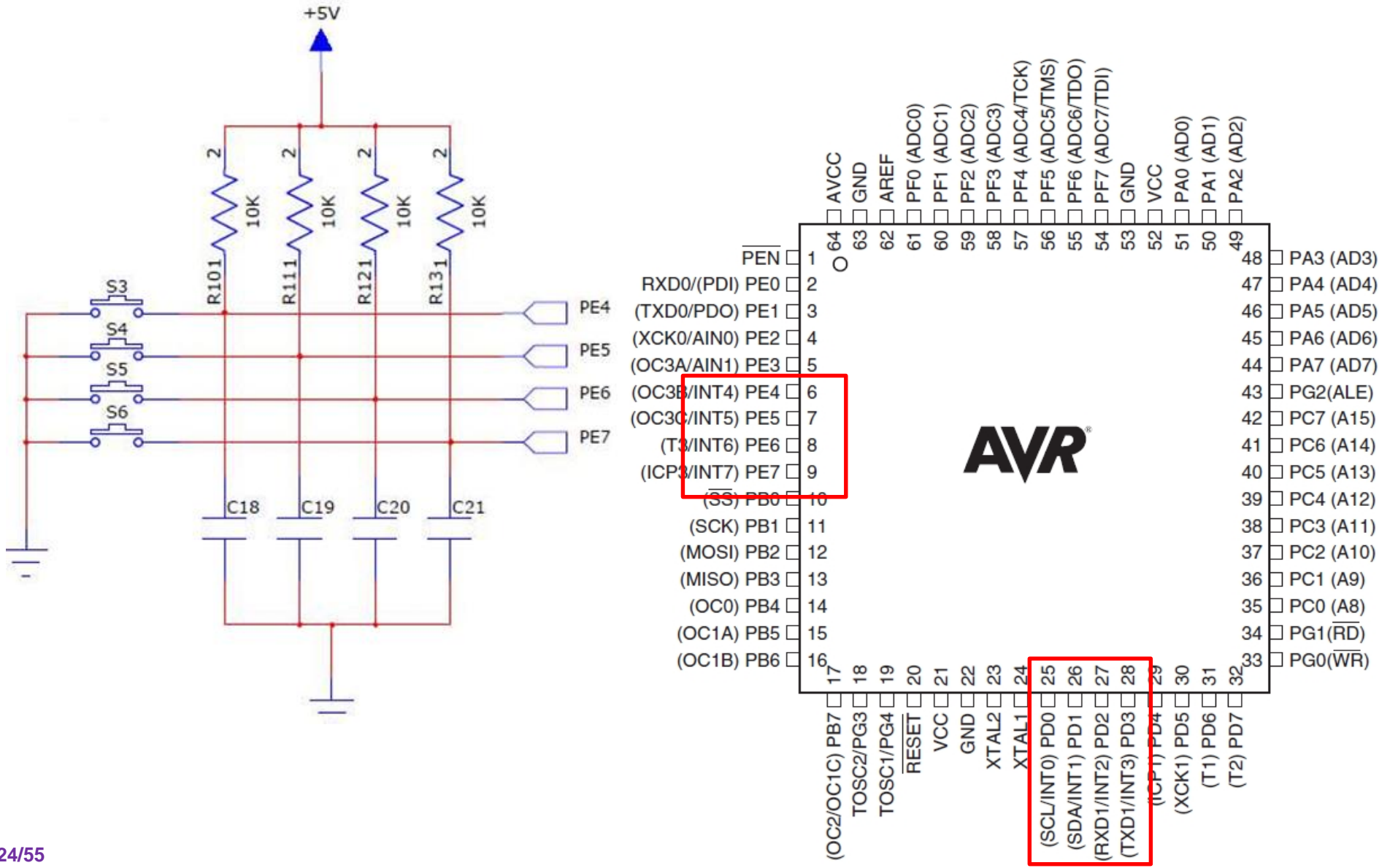
- **EICRA** register에 의해 설정



5. External Interrupt 제어

Strictly Private and Confidential

External Interrupt (외부 인터럽트)



5. External Interrupt 제어

Strictly Private and Confidential

External Interrupt Control register A [EICRA]

Bit	7	6	5	4	3	2	1	0	
	INT3		INT2		INT1		INT0		
	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	

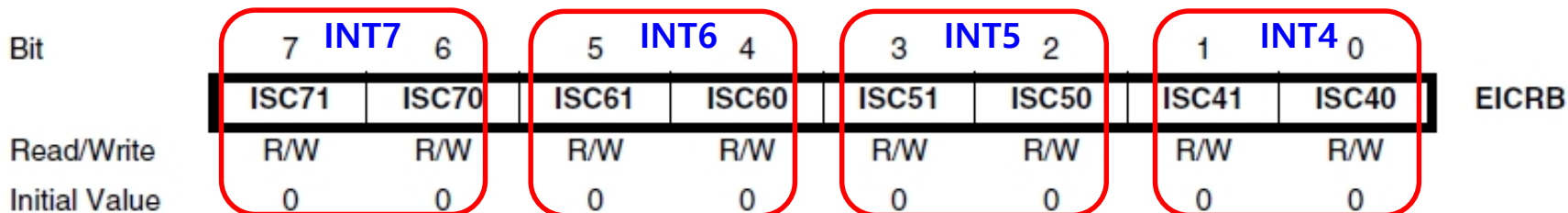
- INT0~INT3에 대한 **Interrupt trigger 방식 설정**
- INT0~INT3의 level trigger 와 edge trigger 방식에 의한 External Interrupt는 **clock에 상관없이 비동기적으로 검출 슬립모드를 해제하는 수단으로 사용이 가능**
- Edge trigger로 사용되는 인터럽트는 최소 **50ns이상의 펄스 폭을 가져야 함**



ISCn1	ISCn0	Trigger 방식
0	0	INTn의 Low level 에서 Interrupt 요구
0	1	-
1	0	INTn의 falling edge 에서 비동기적으로 Interrupt 요구
1	1	INTn의 rising edge 에서 비동기적으로 Interrupt 요구



External Interrupt Control register B [EICRB]



- INT4~INT7에 대한 **Interrupt trigger 방식 설정**
- INT4~INT7의 edge trigger 방식에 의한 External Interrupt는 I/O clock을 필요로 하기 때문에 **Idle mode 이외에 sleeps mode를 해제하는 수단으로 사용불가**
- Level trigger로 사용하는 경우에는 **현재 실행중인 명령어가 끝날 때까지 LOW level 유지 필요**

ISCn1	ISCn0	설 명
0	0	INTn의 Low level 에서 Interrupt 요구
0	1	INTn의 논리적인 변화가 있을 때 Interrupt 요구
1	0	INTn의 falling edge 에서 비동기적으로 Interrupt 요구
1	1	INTn의 rising edge 에서 비동기적으로 Interrupt 요구



5. External Interrupt 제어

Strictly Private and Confidential

External Interrupt Mask register [EIMSK]

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	

- EIMSK는 External Interrupt 를 enable 시키는 비트로 구성
- Bit 7~0-INT7~INT0 : External Interrupt enable bit
 - INT0~INT7 bit가 set 되어 있고 상태 레지스터 SREG의 I 비트가 set 되어 있을 경우에 해당하는 외부 Interrupt enable
 - Interrupt trigger 방식은 EICRA와 EICRB 레지스터에 의해 설정



External Interrupt Flag register [EIFIR]

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	

- EIFR 레지스터는 **External Interrupt**의 발생 여부를 나타냄
- Bit 7~0 – INTF7~INTF0 : **External Interrupt Flag bit**
 - : INT0~INT7 핀에서 Interrupt 요구의 이벤트가 발생했을 때 비트 set
 - : SREG의 I비트와 EIMSK register의 INT0~INT7 비트가 set 되어 있으면, MCU는 해당하는 Interrupt vector로 점프
 - : **Interrupt service routine**이 실행되면 flag는 자동적으로 **clear** 되거나 또는 **flag bit에 1을 쓰면 clear**
 - : **Level trigger**로 설정된 경우에는 항상 **clear** 상태



Code Vision AVR에서 Interrupt 함수 형식

- interrupt [인터럽트 소스명] void 함수명(void)

{

인터럽트 서브루틴

}



```
interrupt [EXT_INT4] void external_int4(void)
{
    led = led << 1;           // 1비트 쉬프트
    led = led | 0b00000001;    // 최하위 비트 셋
    if(led == 0xFF) led = 0xFE; // 모두 off 상태면 초기값 재설정
    PORTC = led;               // 포트 출력
}
```

- Interrupt 소스 명은 CodeVisionAVR에서 제공하는 헤더파일 <mega128.h>에서 정의된 소스 명을 사용
- 함수 명은 사용자가 임의로 지정

5. External Interrupt 제어

Strictly Private and Confidential

Mega128a.h에서 정의된 외부 Interrupt



Interrupt vectors definitions

```
#define EXT_INT0 2
#define EXT_INT1 3
#define EXT_INT2 4
#define EXT_INT3 5
#define EXT_INT4 6
#define EXT_INT5 7
#define EXT_INT6 8
#define EXT_INT7 9
```

```
#define TIM2_COMP 10
#define TIM2_OVF 11
#define TIM1_CAPT 12
#define TIM1_COMPA 13
#define TIM1_COMPB 14
#define TIM1_OVF 15
#define TIM0_COMP 16
#define TIM0_OVF 17
#define SPI_STC 18
#define USART0_RXC 19
#define USART0_DRE 20
#define USART0_TXC 21
#define ADC_INT 22
#define EE_RDY 23
#define ANA_COMP 24
#define TIM1_COMPC 25
#define TIM3_CAPT 26
```

```
#define TIM3_COMPA 27
#define TIM3_COMPB 28
#define TIM3_COMPC 29
#define TIM3_OVF 30
#define USART1_RXC 31
#define USART1_DRE 32
#define USART1_TXC 33
#define TWI 34
#define SPM_RDY 35
```

External Interrupt 사용법 요약

- 사용하고자 하는 External Interrupt enable bit의 set 사용 레지스터 : **EMISK**
- **Trigger** 방식 설정(**rising or falling or level trigger** 중 선택) 사용 레지스터 : **EICRA(INT0~INT3, EICRB(INT4~INT7))**
- Global(전역) Interrupt enable I bit의 set 사용 레지스터 : **SREG (최상위 비트)**
 - ➔ 가능하면 **모든 초기 설정이 끝난 후 제일 마지막에 I bit의 set**
- 인터럽트 함수 작성

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-1) External Interrupt 4에 의한 LED 점멸 실험 (falling edge)

- KUT-128_COM 보드의 SW/INT4이 눌러질 때마다 External Interrupt 4가 발생하여, LED가 순차적으로 점멸하도록 프로그램을 작성하라.
- 외부 인터럽트4의 입력 Trigger 는 falling edge 에서 발생하도록 한다.

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	

Interrupt enable [4] = 1

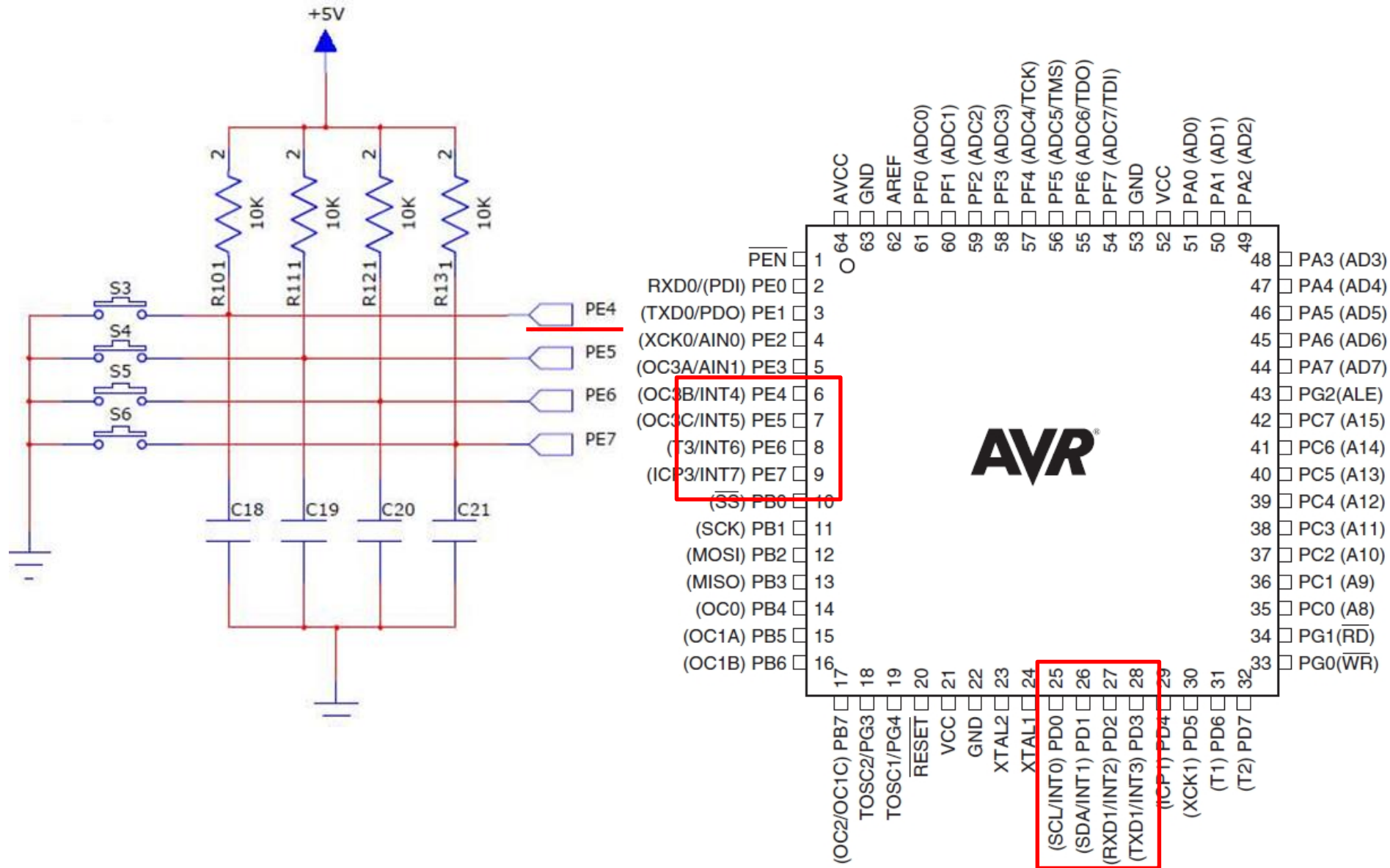
Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Falling edge [1][0] = 10

- SREG : MSB=1 → SREG = 0x80

5. External Interrupt 제어

Strictly Private and Confidential



5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-1) External Interrupt 4에 의한 LED 점멸 실험 (falling edge)

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

```
unsigned char led = 0xFE;
```

```
void main(void)
```

```
{
```

```
    // 포트 초기화
```

```
    DDRC = 0xFF;
```

```
    DDRE = 0b00000010;
```

```
    PORTC = led;
```

```
    EIMSK = 0b00010000;
```

```
    EICRB = 0b00000010;
```

```
    SREG = 0x80;
```

```
    while(1);
```

```
}
```

UART Tx (출력)



```
    // 포트 C 출력 설정
```

```
    // 포트 E 입력(PE1 출력)
```

```
    // 포트 C에 초기값 출력
```

```
    // 외부 인터럽트4 enable
```

```
    // 외부 인터럽트4 falling edge
```

```
    // 전역 인터럽트 enable set
```

```
// 외부 인터럽트4 서비스 루틴
```

```
interrupt [EXT_INT4] void external_int4(void)
```

```
{
```

```
    SREG &= 0x7F;
```

```
    // All Interrupt disable
```

```
    led = led << 1;
```

```
    // 1비트 쉬프트
```

```
    led = led | 0b00000001; // LSB bit set
```

```
    if(led == 0xFF) led = 0xFE;
```

```
    PORTC = led;
```

```
    // 포트 출력
```

```
    SREG |= 0x80;
```

```
    // All Interrupt enable
```

```
}
```

- SW1/INT4이 눌러질 때마다 점멸

5. External Interrupt 제어

Strictly Private and Confidential

The screenshot displays the CodeVisionAVR IDE interface. The main window shows the source code for `ex4-1.c`, which is configured for an external interrupt 4. The code includes the `mega128.h` header and defines a global variable `led` of type `unsigned char`. The `main` function initializes the DDRC register to `0xFF` and the DDRE register to `0b00000010`, then enters an infinite loop.

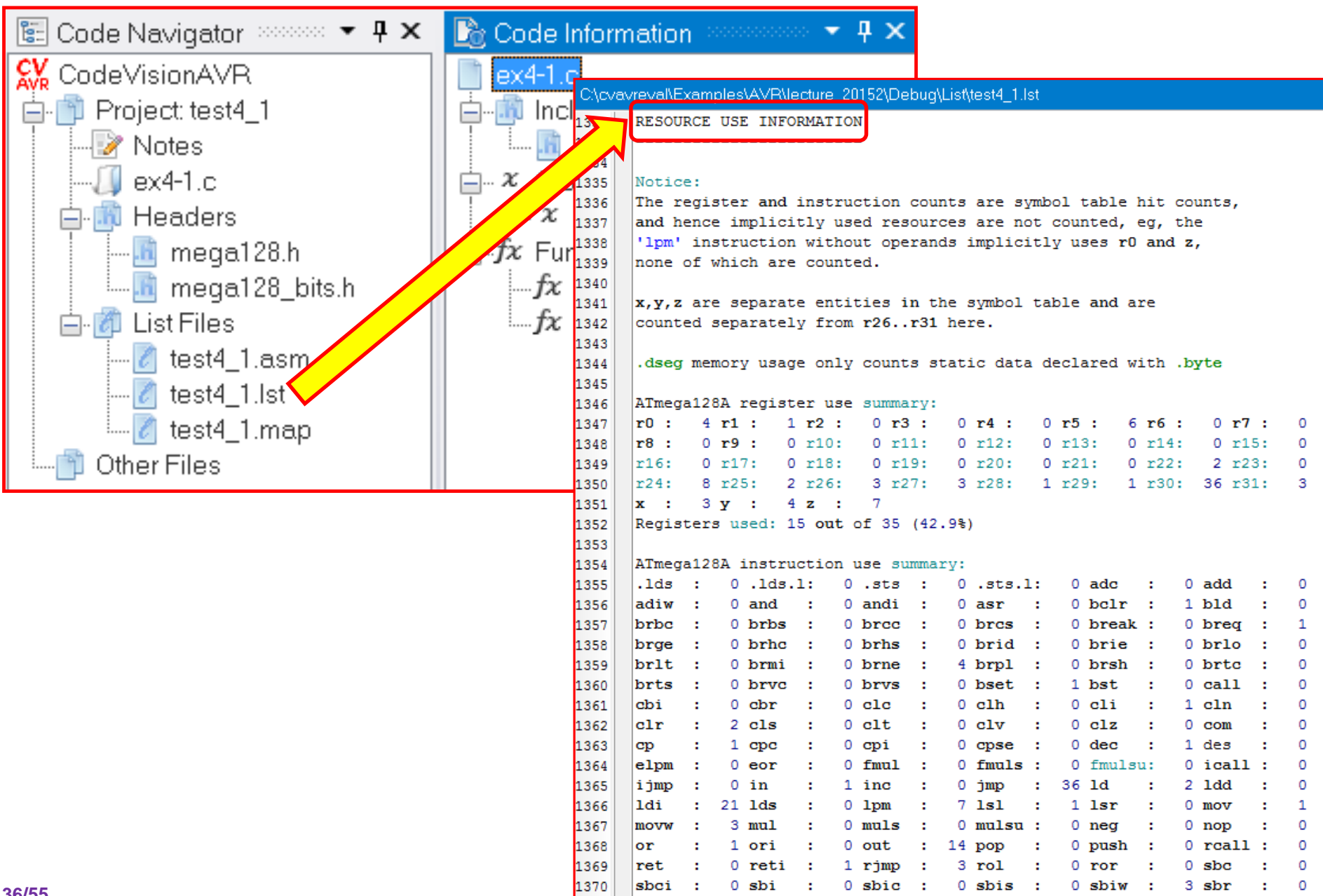
Two inset windows provide a detailed view of the project structure:

- Code Navigator:** Shows the project hierarchy for `Project: test4_1`. It includes `Notes`, `ex4-1.c`, `Headers` (`mega128.h`, `mega128_bits.h`), `List Files` (`test4_1.asm`, `test4_1.lst`, `test4_1.map`), and `Other Files`. A red box highlights the `List Files` section.
- Code Information:** Shows the file structure for `ex4-1.c`. It includes `Includes` (`mega128.h`), `Global/Static Variables` (`led -> R5`), and `Functions` (`external_int4(void)`, `main(void)`). A red box highlights the `Functions` section.

A large yellow arrow points from the `Code Navigator` inset to the `Code Information` inset, indicating the relationship between the project files and the code structure.

5. External Interrupt 제어

Strictly Private and Confidential



The screenshot displays the CodeVisionAVR IDE interface. On the left, the 'Code Navigator' window shows a project tree for 'test4_1'. The 'List Files' folder is expanded, showing 'test4_1.asm', 'test4_1.lst', and 'test4_1.map'. A yellow arrow points from 'test4_1.lst' to the 'Code Information' window on the right. The 'Code Information' window shows the assembly listing for 'test4_1.lst'. A red box highlights the 'RESOURCE USE INFORMATION' section, which contains the following text:

Notice:
The register and instruction counts are symbol table hit counts, and hence implicitly used resources are not counted, eg, the 'lpm' instruction without operands implicitly uses r0 and z, none of which are counted.

x,y,z are separate entities in the symbol table and are counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

ATmega128A register use summary:

Register	Count
r0	4
r1	1
r2	0
r3	0
r4	0
r5	6
r6	0
r7	0
r8	0
r9	0
r10	0
r11	0
r12	0
r13	0
r14	0
r15	0
r16	0
r17	0
r18	0
r19	0
r20	0
r21	0
r22	2
r23	0
r24	8
r25	2
r26	3
r27	3
r28	1
r29	1
r30	36
r31	3

x : 3 y : 4 z : 7

Registers used: 15 out of 35 (42.9%)

ATmega128A instruction use summary:

Instruction	Count
.lds	0
.lds.l	0
.sts	0
.sts.l	0
adc	0
add	0
adiw	0
and	0
andi	0
asr	0
bclr	1
bld	0
brbc	0
brbs	0
brcc	0
brcs	0
break	0
breq	1
brge	0
brhc	0
brhs	0
brid	0
brie	0
brlo	0
brlt	0
brmi	0
brne	4
brpl	0
brsh	0
brtc	0
brts	0
brvc	0
brvs	0
bset	1
bst	0
call	0
cbi	0
cbr	0
clc	0
clh	0
cli	1
cln	0
clr	2
cls	0
clt	0
clv	0
clz	0
com	0
cp	1
cpc	0
cpi	0
cpse	0
dec	1
des	0
elpm	0
eor	0
fmul	0
fmuls	0
fmulsu	0
icall	0
ijmp	0
in	1
inc	0
jmp	36
ld	2
ldd	0
ldi	21
lds	0
lpm	7
lsl	1
lsr	0
mov	1
movw	3
mul	0
muls	0
mulsu	0
neg	0
nop	0
or	1
ori	0
out	14
pop	0
push	0
rcall	0
ret	0
reti	1
rjmp	3
rol	0
ror	0
sbc	0
sbcui	0
sbi	0
sbic	0
sbis	0
sbiw	3
sbr	0

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-2) External Interrupt 4에 의한 LED 점멸 실험 (rising edge)

- KUT-128_COM 보드의 SW/INT4이 눌렀다 떴어지는 순간(rising edge) External Interrupt 4가 발생하여, LED가 순차적으로 점멸하도록 프로그램을 작성하라.

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	

Interrupt enable [4] = 1

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	

rising edge [1][0] = 11

- SREG : MSB=1 → SREG = 0x80

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-2) External Interrupt 4에 의한 LED 점멸 실험 (rising edge)

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

```
unsigned char led = 0xFE;
```

```
void main(void)
```

```
{
```

```
    // 포트 초기화
```

```
    DDRC = 0xFF;
```

```
    DDRE = 0b00000010;
```

```
    PORTC = led;
```

```
    EIMSK = 0b00010000;
```

```
    EICRB = 0b00000011;
```

```
    SREG = 0x80;
```

```
    while(1);
```

```
}
```

반드시

Global variable로 선언



```
// 외부 인터럽트4 서비스 루틴
```

```
interrupt [EXT_INT4] void external_int4(void)
```

```
{
```

```
    SREG &= 0x7F;
```

```
// All Interrupt disable
```

```
    led = led << 1;
```

```
// 1비트 쉬프트
```

```
    led = led | 0b00000001; // LSB bit set
```

```
    if(led == 0xFF) led = 0xFE;
```

```
    PORTC = led;
```

```
// 포트 출력
```

```
    SREG |= 0x80;
```

```
// All Interrupt enable
```

```
}
```

- SW1/INT4이 눌렀다 떼어지는 순간 점멸

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-3) 외부 Interrupt 4에 의한 LED 전자 롤렛 실험[Level trigger : Low level]

- External Interrupt 4에 대한 실험으로 입력 trigger 방식을 Level trigger 방식으로 하여 SW이 눌러지고 있는 동안 LED가 순차 점멸하도록 하고,
- SW에서 손을 떼면 순차 점멸하던 LED가 정지하도록 프로그램을 작성하라.

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	

Interrupt enable [4] = 1

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	

Level trigger[1][0] = 00

- SREG : MSB=1 → SREG = 0x80

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-3) 외부 Interrupt 4에 의한 LED 전자 롤렛 실험[Level trigger : Low level]

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

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

```
unsigned char led = 0xFE;
```

반드시

Global variable로 선언

```
void main(void)
```

```
{
```

```
    // 포트 초기화
```

```
    DDRC = 0xFF;
```

```
    // 포트 C 출력 설정
```

```
    DDRE = 0b00000010;
```

```
    // 포트 E 입력(PE1 출력)
```

```
    PORTC = led;
```

```
    // 포트 C에 초기값 출력
```

```
    EIMSK = 0b00010000;
```

```
    // 외부 인터럽트4 enable
```

```
    EICRB = 0b00000000;
```

```
    // 외부 인터럽트4 level trigger
```

```
    SREG = 0x80;
```

```
    // 전역 인터럽트 enable-bit set
```

```
    while(1);
```

```
}
```

```
// 외부 인터럽트4 서비스 루틴
```

```
interrupt [EXT_INT4] void external_int4(void)
```

```
{
```

```
    SREG &= 0x7F;
```

```
    // All Interrupt disable
```

```
    led = led << 1;
```

```
    // 1비트 쉬프트
```

```
    led = led | 0b00000001;
```

```
    // LSB bit set
```

```
    if(led == 0xFF) led = 0xFE;
```

```
    PORTC = led;
```

```
    // 포트 출력
```

```
    delay_ms(500);
```

```
    SREG |= 0x80;
```

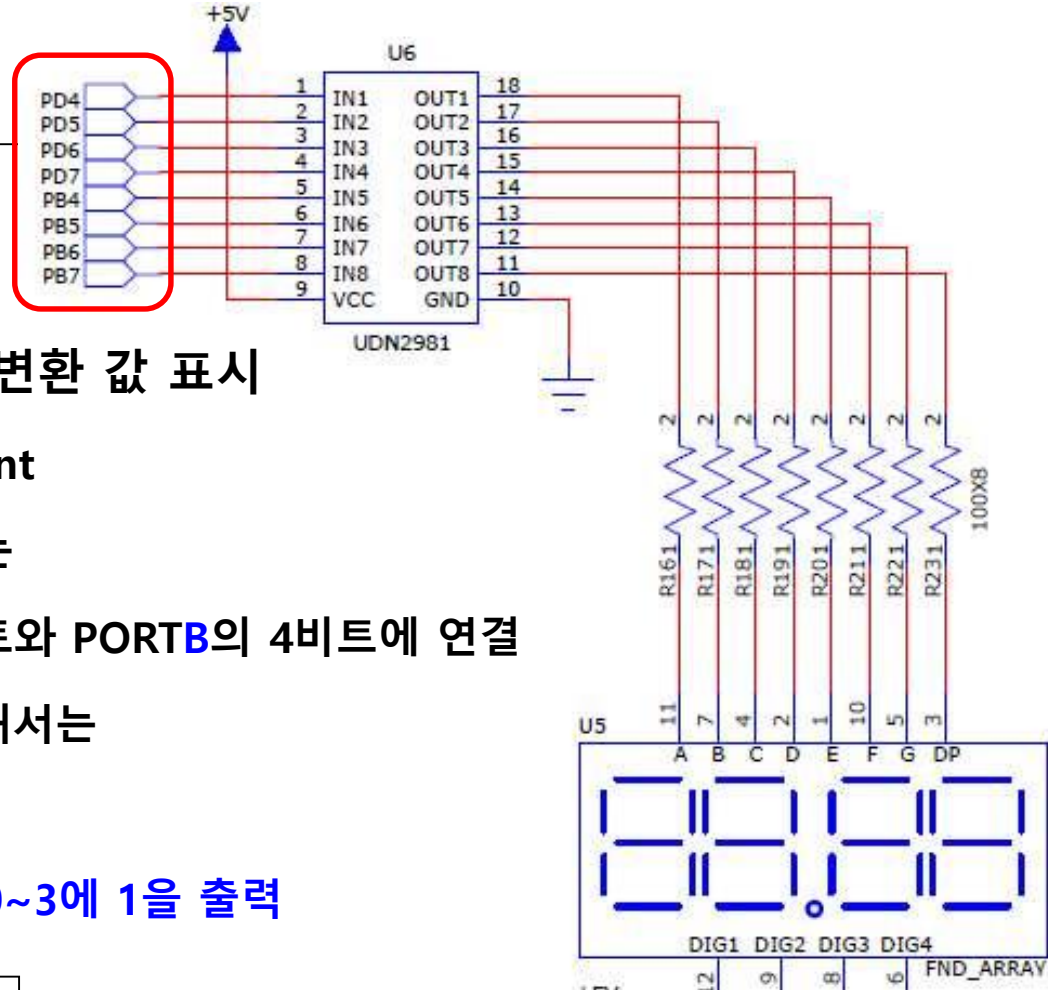
```
    // All Interrupt enable
```

```
}
```

- SW/INT4이 눌러지고 있는 동안 점멸

(예제 4-4) 외부 Interrupt 4에 의한 스위치 입력 실험 (응용)

- 외부 Interrupt 4 를 이용하여 SW/INT4를 누를 때마다 (falling edge), 맨 우측 7-Segment의 표시 값이 0 → 1 → 2 → 3... → 9 → 0 → 1.. 이 표시되도록 프로그램을 작성하라.
- `EIMSK = 0b00010000;` `// Interrupt 4 enable`
- `EICRB = 0b00000010;` `// Interrupt 4 falling edge`
- `SREG = 0x80;` `// Global Interrupt bit set`
: `MSB=1` → `SREG = 0x80`



- 디지털시계의 제작, A/D변환기의 변환 값 표시

: **Common Cathode** 형 4채널 Segment

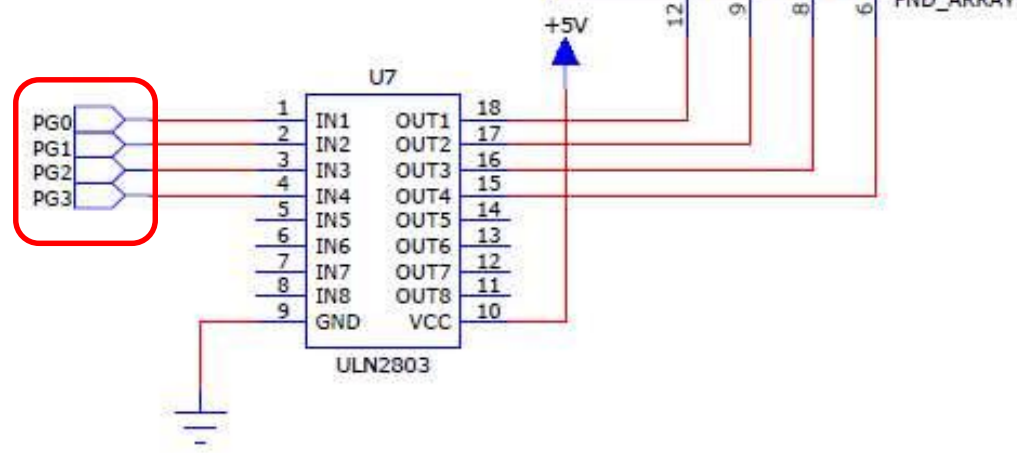
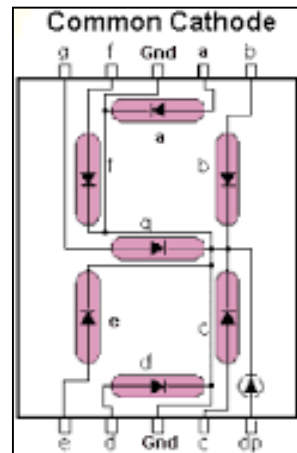
: 각 Segment LED(A,B,C,D,E,F,G,DP)는

UDN2981을 거쳐 포트 PORTD 4비트와 PORTB의 4비트에 연결

: 각 Segment의 LED를 ON시키기 위해서는

PORTB, PORTD의 해당비트에 1출력

: Segment를 ON시키기 위해서는 PG0~3에 1을 출력



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

```
const char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
unsigned char N1 = 0;
```

```
void main(void)
{
    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
    DDRC = 0x0F;           // 포트 C 하위 4비트 출력 설정
    DDRE = 0x0F;           // 포트 E 하위 4비트 출력 설정
    DDRC = 0x0F;           // 포트 G 하위 4비트 출력 설정
```

// 외부 인터럽트4 서비스 루틴

interrupt [EXT_INT4] void external_int4(void)

{

N1 = (N1 + 1) % 10; // 값 +1

}

← Interrupt service routine 간단하게

```
// 인터럽트 초기화
EIMSK = 0b00010000;      // 외부 인터럽트4 enable
EICRB = 0b00000010;      // 외부 인터럽트4 : falling edge
SREG = 0x80;              // 전역 인터럽트 enable-bit set

PORTG = 0b00001000;      // 맨 우측 7-Segment DIG4 ON(PG3=1)
```

PORTD 하위 4bits
PORTB 하위 4bits
변경되지 않게

```
while(1) {
    PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시
    PORTB = (seg_pat[N1] & 0x70) | (PORTB & 0x0F);        // E, F, G 표시
}
```

(예제 4-5) 2개의 외부 Interrupt 를 이용한 10진 네 자리 입력 실험

- 두 개의 스위치 **SW/INT4**와 **SW/INT5**만을 이용하여 4자리를 입력하는 프로그램을 작성하라.
- 여기서, **SW/INT5**는 입력될 자리(위치)를 나타내고,
- **SW/INT4** 눌러지면 현재 선택된 자리의 값을 +1하도록 한다.
- **SW/INT5**가 눌러지면 입력될 위치를
1자리 ➔ 10자리 ➔ 100자리 ➔ 1000자리 ➔ 1자리 ➔ 10자리..로
이동하도록 한다

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

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

```
typedef unsigned char U8;
```

```
const U8 seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,0x7f, 0x6f};
```

```
U8 N1 = 0, N10 = 0, N100 = 0, N1000 = 0; // 0 : 1자리, 1 : 10자리 , 2 : 100자리 , 3 : 1000자리
```

```
U8 pos = 0;
```

```
void Seg4_out(void);           // 네 자리수 7-Segment 출력
```

```
void main(void)
```

```
{
```

```
    DDRB = 0xF0;           // 포트 B 상위 4비트 출력 설정
```

```
    DDRD = 0xF0;           // 포트 D 상위 4비트 출력 설정
```

```
    DDRG = 0x0F;           // 포트 G 하위 4비트 출력 설정
```

```
    EIMSK = 0b00110000;    // 외부 인터럽트 4,5 enable
```

```
    EICRB = 0b00001010;    // 외부 인터럽트 4,5 : falling edge
```

```
    SREG = 0x80;           // 전역 인터럽트 enable-bit set
```

```
    while(1)
```

```
    {
```

```
        Seg4_out();
```

```
    }
```

```
}
```

```

void Seg4_out(void)
{
    PORTG = 0b00001000; // 7-Seg DIG4 ON(PG3=1), 1자리 표시
    PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시
    PORTB = (seg_pat[N1] & 0x70) | (PORTB & 0x0F); // E, F, G 표시
    delay_ms(5);

    PORTG = 0b00000100; // 7-Seg DIG3 ON(PG2=1), 10자리 표시
    PORTD = ((seg_pat[N10] & 0x0F) << 4) | (PORTD & 0x0F); // A, B, C, D 표시
    PORTB = (seg_pat[N10] & 0x70) | (PORTB & 0x0F); // E, F, G 표시
    delay_ms(5);

    PORTG = 0b00000010; // 7-Seg DIG2 ON(PG1=1), 100자리 표시
    PORTD = ((seg_pat[N100] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N100] & 0x70) | (PORTB & 0x0F);
    delay_ms(5);

    PORTG = 0b00000001; // 7-Seg DIG1 ON(PG0=1), 1000자리 표시
    PORTD = ((seg_pat[N1000] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N1000] & 0x70) | (PORTB & 0x0F);
    delay_ms(5);
}

```

PORTD 하위 4bits
 PORTB 하위 4bits
 변경되지 않게

// 외부 인터럽트4 서비스 루틴

interrupt [EXT_INT4] void external_int4(void)

```
{  
    if(pos == 0) N1 = (N1 + 1) % 10;           // 1자리 +1  
    else if(pos == 1) N10 = (N10 + 1) % 10;    // 10자리 +1  
    else if(pos == 2) N100 = (N100 + 1) % 10;  // 100자리 +1  
    else N1000 = (N1000 + 1) % 10;             // 1000자리 +1  
}
```

// 외부 인터럽트5 서비스 루틴

interrupt [EXT_INT5] void external_int5(void)

```
{  
    pos = (pos + 1) % 4;                       // 입력 자리 이동  
}
```


5. External Interrupt 제어

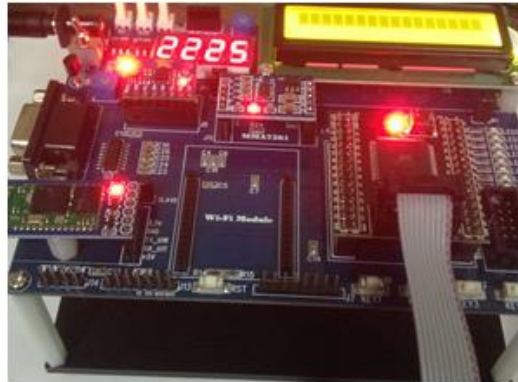
Strictly Private and Confidential

(예제 4-6) 외부 Interrupt 를 이용한 간이 시계 실험

- delay_ms() 함수를 이용하여

시간과 분을 7-Segment에 출력하는 프로그램을 작성하라.

시간과 분의 조정은 [예제 4-5]와 같이 두 개의 스위치를 이용하여 입력하도록 한다.



5. External Interrupt 제어

Strictly Private and Confidential

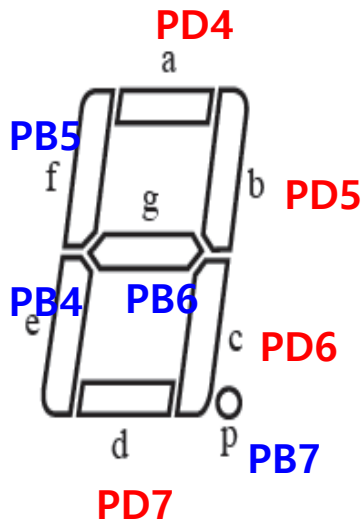
(예제 4-6) 외부 Interrupt 를 이용한 간이 시계 실험

- delay_ms() 함수를 이용하여

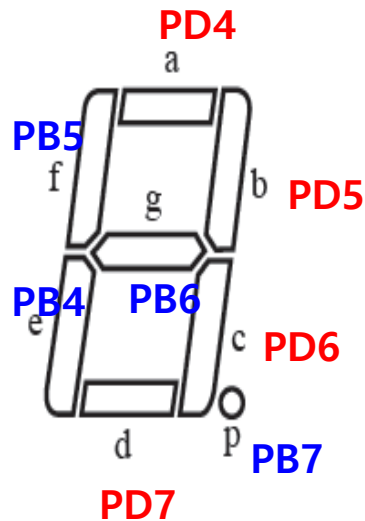
시간과 분을 7-Segment에 출력하는 프로그램을 작성하라.

시간과 분의 조정은 [예제 4-5]와 같이 두 개의 스위치를 이용하여 입력하도록 한다.

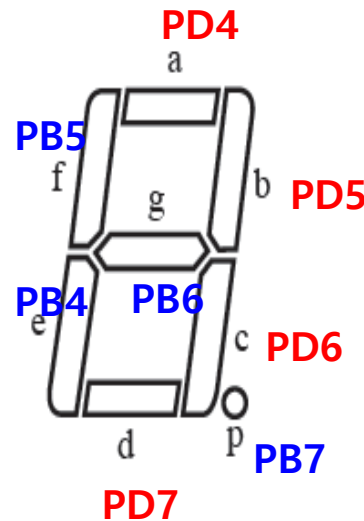
- 시계 값 조정은 어떻게 해야 될까요? (5분 토론)



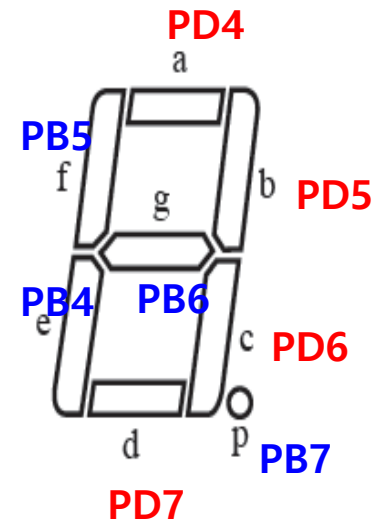
시간



시간



분



분

5. External Interrupt 제어

Strictly Private and Confidential

(예제 4-6) 외부 Interrupt 를 이용한 간이 시계 실험

- 시계 조정

시계 자리	SW 누르기	SW 누름 위치	의미	표현 가능 숫자
	안 누름	POS = 0	1단위 분	0 - 9
	1번 누름	POS = 1	10단위 분	0 - 5
	2번 누름	POS = 2	1단위 시간	만일 POS=3의 값이 : 0,1경우, 0 - 9 가능 : 2경우, 0 - 3 가능
	3번 누름	POS = 3	10단위 시간	만일 POS=2의 값이 : 0,1,2,3 경우, 0, 1, 2 가능 : 4, - ,9 경우, 0, 1 가능

5. External Interrupt 제어

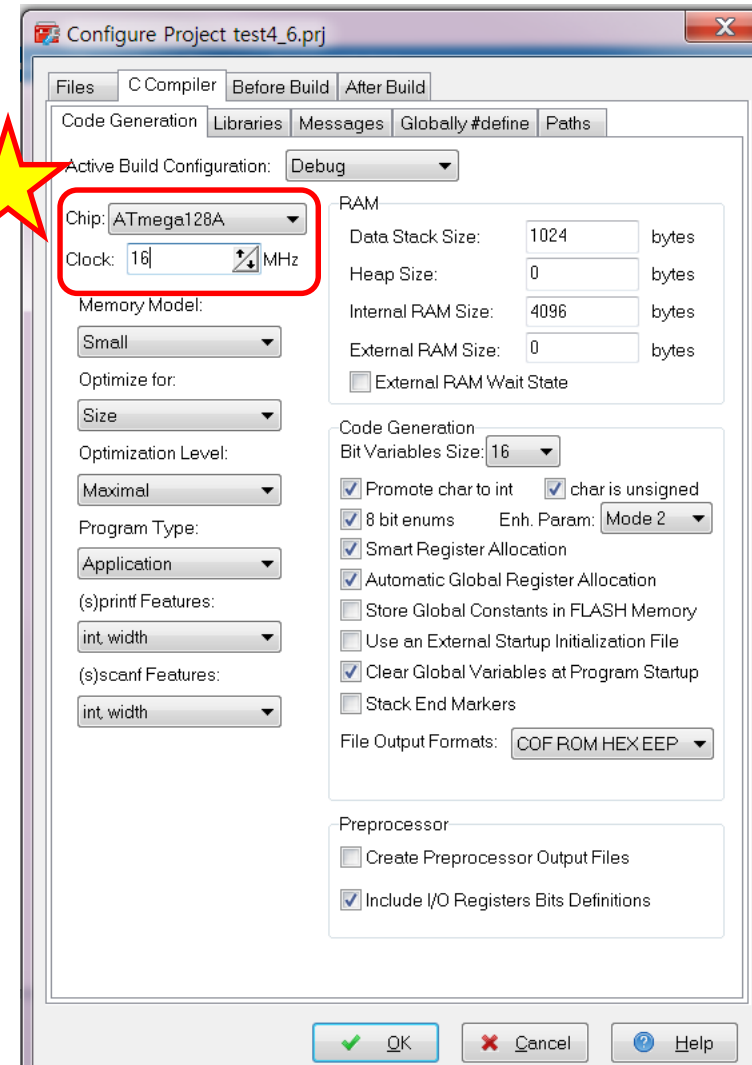
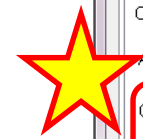
Strictly Private and Confidential

(예제 4-6) 외부 Interrupt 를 이용한 간이 시계 실험

- **delay_ms()** 함수를 이용하여

시간과 분을 7-Segment에 출력하는 프로그램을 작성하라.

- Project 생성시,
C-Compiler에서 **clock 8MHz를 16MHz로**
- **delay_ms()**를 사용하여 **시계**를 만들기 때문



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

```
typedef unsigned char U8;
```

```
typedef unsigned short U16;
typedef unsigned int U32;
typedef signed char S8;
typedef signed short S16;
typedef signed int S32
```

```
const U8 seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
U8 N1, N10, N100, N1000;
U8 pos = 0;
U8 hour = 12, min = 0, sec = 0;
```

```
void Time_out(void); // 시간 표시 함수
```

```
void main(void) {
    unsigned char i;

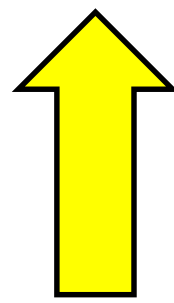
    DDRB = 0xF0;
    DDRD = 0xF0;
    DDRC = 0x0F;
    EIMSK = 0b00110000; // 외부 인터럽트 4,5 enable
    EICRB = 0b00001010; // 외부 인터럽트 4,5 falling edge
```

```
SREG = 0x80; // 전역 인터럽트 enable-bit set
```

```
while(1) {
    for(i = 0; i < 49; i++) {
        Time_out(); // 약 1초간 반복 표시
    } // end of for
```

```
sec = sec + 1; // 초값 +1
if(sec == 60) {
    sec = 0;
    min = min + 1; // 분값 +1
    if(min == 60) {
        min = 0;
        hour = (hour + 1) % 24; // 시간 +1
    } // end of min
} // end of sec

} //end of while
} //end of main
```



```
void Time_out(void) // display
```

```
{  
    PORTG = 0b00001000; // 7-Seg DIG4 ON(PG3=1), 분 1자리 표시  
    PORTD = ((seg_pat[min % 10] & 0x0F) << 4) | (PORTD & 0x0F);  
    PORTB = (seg_pat[min % 10] & 0x70) | (PORTB & 0x0F);  
    delay_ms(5);  
  
    PORTG = 0b00000100; // 7-Seg DIG3 ON(PG2=1), 분 10자리 표시  
    PORTD = ((seg_pat[min / 10] & 0x0F) << 4) | (PORTD & 0x0F);  
    PORTB = (seg_pat[min / 10] & 0x70) | (PORTB & 0x0F);  
    delay_ms(5);  
  
    PORTG = 0b00000010; // 7-Seg DIG2 ON(PG1=1), 시간 1자리 표시  
    PORTD = ((seg_pat[hour % 10] & 0x0F) << 4) | (PORTD & 0x0F);  
    PORTB = (seg_pat[hour % 10] & 0x70) | (PORTB & 0x0F);  
    delay_ms(5);  
  
    PORTG = 0b00000001; // 7-Seg DIG1 ON(PG0=1), 시간 10자리 표시  
    PORTD = ((seg_pat[hour / 10] & 0x0F) << 4) | (PORTD & 0x0F);  
    PORTB = (seg_pat[hour / 10] & 0x70) | (PORTB & 0x0F);  
    delay_ms(5);  
}
```

PORTD 하위 4bits

PORTB 하위 4bits

변경되지 않게

interrupt [EXT_INT4] void external_int4(void)

```
{  
    N1 = min % 10;  
    N10 = min / 10;  
    N100 = hour % 10;  
    N1000 = hour / 10;  
  
    if(pos == 0) N1 = (N1 + 1) % 10;  
    else if(pos == 1) N10 = (N10 + 1) % 6;  
    else if(pos == 2) {  
        if(N1000 == 2) N100 = (N100 + 1) % 4;  
        else N100 = (N100 + 1) % 10;  
    }  
    else {  
        if(N100 < 4) N1000 = (N1000 + 1) % 3;  
        // else if(N1000 != 1) N1000 = (N1000 + 1) % 3;  
        else N1000 = (N1000 + 1) % 2;  
    }  
}
```

```
// 현재 분 1자리 추출  
// 현재 분 10자리 추출  
// 현재 시간 1자리 추출  
// 현재 시간 10자리 추출  
  
// 현재 분 1단위 표시 +1 값  
// 현재 분 10단위 표시 +1 값 (60분)  
// 시간 1단위 +1  
// 24시간의 앞자리 2이므로 20,21,22,23,24  
// 0,1,2,3,-9,10,11,12,-,18,19  
  
// pos=3 (세 번 눌러짐)  
// N100<4은 04 (00,01,02,03), (10,11,12,13),  
// (20,21,22,23) =>10의 시간 자리는 0,1,2
```

```
hour = N1000 * 10 + N100;    // 시간 계산 (시간 세팅 값을 void Time_out(void) 로,  
min = N10 * 10 + N1;        // 분 계산
```

interrupt [EXT_INT5] void external_int5(void)

```
{  
    pos = (pos + 1) % 4;    // 입력 자리 이동
```

감사합니다