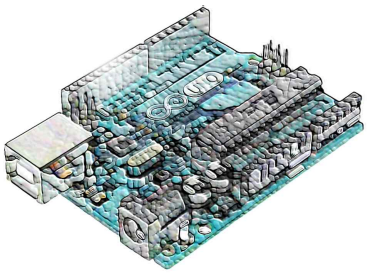


* 본 자료는 서경대학교 아두이노 프로그래밍 수업 수강자를 위해 작성된 강의 교재입니다.

강의교재– 아두이노 프로그래밍

14장 인터럽트

서경대학교 김진헌

14

인터럽트

내용

14.1 인터럽트(Interrupt)의 개념
14.2 아두이노의 인터럽트 벡터와 단자.
14.3 아두이노의 인터럽트 프로그래밍 개요
14.4 인터럽트 관련 아두이노 표준 함수
14.5 외부 인터럽트 프로그래밍 예제
14.6 주기적 인터럽트
14.7 고찰

14.1 인터럽트(Interrupt)의 개념

인터럽트는 프로세서의 동작을 잠시 중단하고 다른 일을 처리하게 만드는 동작이다. 인터럽트 동작은 외부에서 발생한 사건에 대한 처리를 실시간으로 처리할 수 있도록 마련한 프로세서의 주요 사이클[1] 중 하나이다.

인터럽트 처리를 위해 운영체제와 디바이스 드라이버는 사전에 인터럽트 서비스 루틴(**ISR**:Interrupt Service Routine, 혹은 Interrupt Handler)과 인터럽트 벡터 테이블(**IVT**: nterrupt Vector Table)을 사전에 초기화해 둔다. ISR은 인터럽트가 발생하면 처리할 루틴을 말하며, 이는 IVT는 ISR의 위치를 정보를 담고 있다. 인터럽트의 유형에 따라 처리되는 내용이 다른데 이는 인터럽트 타입번호(**type number**)에 의해 관리된다. 프로세서는 인터럽트가 접수되면 어떤 인터럽트 타입번호를 갖고 있는지 단자번호를 통해 알 수 있다.

프로세서는 인터럽트가 들어오면 CPU는 메인 루틴의 수행을 잠시 중단하고 인터럽트 서비스 루틴(ISR)으로 분기한다. 이때 IVT 내에서 참조할 곳의 위치는 인터럽트 type number가 인덱스로 활용된다.

그림 14.1.1에 보인 바와 같이 프로세서가 여러 개의 명령어(instruction 01~n+1 등)로 이루어진 메인 루틴을 수행하는 사례를 예로 들어 인터럽트가 처리되는 과정을 살펴보자. 현재 instruction 04를 수행 중에 event A에 의해 인터럽트가 발생한 것으로 가정하자[2].

현재 수행 중인 명령어(04)의 수행을 마친 후 인터럽트 벡터 테이블의 정보를 참조하여 인터럽트 서비스 루틴(ISR)이 위치한 곳으로 분기한다[3].

---

1) 프로세서의 사이클은 크게 ① 메모리 혹은 I/O 읽기/쓰기 사이클, ② 인터럽트 사이클, 그리고 ③ DMA 동작이 일어나는 idle cycle로 대별하여 분류할 수 있다.

2) 이러한 이벤트의 사례로는 외부에서 키를 입력했다던가, 통신 장치를 통해 처리해야 할 데이터가 유입되었다던가 등의 사례를 들 수 있다. 이러한 이벤트는 CPU의 인터럽트 단자(pin)를 통해 H/W 신호로서 보고된다.

3) 인텔의 x86 프로세서군에서는 IVT에 ISR의 주소가 기록되어 있다. ARM을 비롯한 임베디드 프로세서 계열에서는 IVT의 위치에 ISR로 분기하는 명령어가 들어가 있다. 이 부분에 대한 초기화(주소 혹은 명령어를 넣는 일)는 부팅과정 중에 운영체제 혹은 디바이스 드라이버가 담당한다.

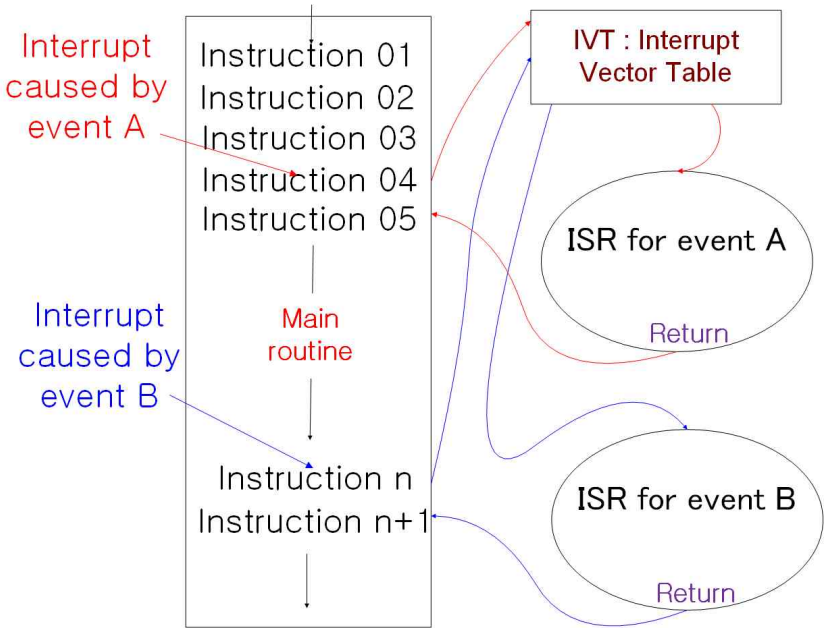


그림 14.1.1 인터럽트 동작의 개념도

그 ISR 루틴은 event A가 발생할 것에 대비한 처리 루틴이 미리 마련되어 있어서 해당되는 요청에 필요한 서비스를 수행한다[4]. ISR의 끝 부분에는 회귀(return) 명령어가 있어서 메인 루틴으로 복귀하도록 준비되어 있다. 본 사례의 경우에는 이벤트 A의 처리가 끝나면 메인 루틴의 05번째 명령어를 계속 수행하게 된다. 같은 방식으로 이벤트 B에 의한 인터럽트가 발생한다면 n반째 instruction의 수행을 마친 후에는 B의 ISR의 처리를 분기하여 해당 서비스를 수행하고 난 후 n+1번째의 instruction을 수행한다.

현재 상황에서 기억해야 할 주요 사항
1. 기술 용어: Interrupt Vector Table, Interrupt Service Routine, Interrupt Type Number
2. 인터럽트 처리 과정(flow)
3. IVT, ISR의 초기화/탑재의 주체

4) 운영체제나 디바이스 드라이버 등은 각 해당 인터럽트의 처리 루틴을 부팅 중에 미리 메모리에 탑재한다.

표 14.1.1 인터럽트 관련 주요 용어

설 명 의 수준	관계된 전문 용어	설명	상세설명
기초	IVT : Interrupt Vector Table	인터럽트가 발생하면 IVT를 참조하여 분기한다.	분기할 곳의 주소가 들어 있다. -> x86 Intel CPU
			분기하라는 명령어가 들어있다. -> ARM, AVR을 비롯한 여타의 임베디드 프로세서, 마이크로콘트롤러가 이에 해당한다.
	ISR : Interrupt Service Routine	인터럽트가 발생하면 처리하는 함수	IVT를 참조하여 ISR로 분기한다. ISR의 끝에는 복귀하는 명령어가 있어야 한다.
중급	PC : Program Counter	다음 수행할 명령어의 위치를 지정하는 레지스터	ISR로 진입하기 전에 PC를 stack에 넣는다. stack은 주기억 장치 안에 존재한다. ->x86 Intel CPU
			ISR로 분기하기 전에 PC를 복귀전용 return address register에 넣는다. -> ARM, AVR을 비롯한 여타의 임베디드 프로세서, 마이크로콘트롤러
	IE : Interrupt Enable flag	IE flag는 외부 터럽트의 허가 여부를 담당한다. IE=0 : 금지 IE=1 : 허가	ISR로 진입하기 전에 flag를 스택에 저장하고 IE=0으로 만든다. 이로서 ISR 내에서는 원천적으로 외부 H/W 인터럽트를 받지 못한다. -> 프로그래머가 ISR 내부 첫 명령어에서 IE=1로 만드는 명령어를 수행시켜 우선순위가 높은 인터럽트가 서비스될 수 있게 한다.
고급	nested interrupt	ISR 수행 중에 외부 인터럽트를 받아 다른 ISR로 진입하는 동작을 말한다.	인터럽트 제어기는 보통 더 높은 순위의 인터럽트만 CPU에 요구한다. 따라서 우선 순위가 낮은 인터럽트에게는 nested interrupt가 허가되지 않는다. -> x86, ARM 등의 일반적인 프로세서 인터럽트 제어기는 가리지 않고 모든 인터럽트를 CPU에게 요구한다. -> AVR 프로세서

지금까지 설명한 내용은 인터럽트의 기본 개념을 설명한 것이라 할 수 있다. 표 14.1.1에는 현재의 상황을 점검하는 용도로 인터럽트의 그 분석 수준

과 그에 따른 주요 기술 용어와 설명을 보였다.

❐ 참고: 인터럽트 적용 사례 - 키보드 동작의 경우

키보드 장치의 입력을 받기 위해 사용자가 키보드를 눌렀는지를 프로세서가 루프를 돌면서 계속 감시한다면 그동안 다른 일을 할 수 없기 때문에 효율적인 방법이 될 수 없다. 키보드 입력 장치에 인터럽트에 활용되는 사례를 예로 하여 ATmega CPU의 인터럽트 동작을 기준으로 설명하여 인터럽트의 개념과 필요성을 생각해 보기로 한다.

1. 키보드 장치는 사용자가 키를 눌렀을 때 이 사실(event A)을 CPU에 연결되어 있는 인터럽트 제어기(혹은 인터럽트 유니트)에게 알린다.
2. 인터럽트 제어기는 CPU의 단자를 통해 인터럽트 사건이 발생하였음을 알린다.
3. CPU는 인터럽트 단자에 신호가 유입되면 현재 수행 중인 명령어의 수행이 완료되는 대로 다음 인터럽트 사이클에 들어간다.
4. 먼저 다음 수행할 명령어의 주소를 복귀주소 전용 레지스터에 대피시킨다[5]. 다음 수행할 명령어의 주소는 보통 (PC: Program Counter)가 담당하므로 PC의 값을 그 레지스터에 저장한고 보면 된다. 이후 IVT에서 event A의 위치에 있는 명령어를 수행한다. AVR CPU의 경우에는 IVT에는 해당 ISR로 분기시키는 명령어들이 나열되어 있다.
5. ISR에 진입하면 임베디드 프로세서는 현재의 레지스터의 일부를 스택에 대비시킨다. 이러한 루틴은 컴파일러에 의해 자동 생성된다.
6. ISR 안에서 키보드 장치의 문자를 읽어 키보드 버퍼에 저장한 후 인터럽트 종료 명령어를 수행한다.
7. 인터럽트 복귀 명령어가 수행되면 복귀 주소 전용 레지스터 혹은 스택에

5) x86 프로세서의 경우에는 복귀주소를 메모리에 위치하는 스택에 대피시킨다. 스택은 용량이 충분하므로 ISR 수행 중에 발생한 다른 ISR의 분기 상황에도 복귀주소를 저장하기에 유리한 점이 있다.

서 복귀 주소를 읽어 복귀한다. 그 주소는 인터럽트가 발생할 당시 수행하던 명령어의 다음 위치이다.

❐ 참고-인터럽트 종류(인터럽트를 발생하게 하는 원인)

인터럽트에 대한 분류체계는 CPU마다 조금씩 다르기 때문에 일괄적으로 정하기는 어렵지만 크게 보면 표 14.1.2와 같이 구분해 볼 수 있다. 이는 인터럽트의 원인이 CPU 외부에서 단자를 통해서 유입되는가 혹은 내부에서 어떤 조건이 되면 스스로 발생하는지에 따라 구분한 것이다[6].

표 14.1.2 인터럽트의 종류

분류	*명칭 / -발생 원인	
외부 인터럽트 (External Interrupt) : H/W 신호로 CPU에게 접수된다.	NMI(Non Maskable Interrupt)	* 기계 착오 인터럽트(Machine Check Interrupt) - 프로그램 수행 중에 H/W(메모리, 버스) 결함으로 발생 * 전원 이상(Power Fail) 인터럽트 - 정전 혹은 전원공급 이상 시에 발생
	Maskable Interupt	- 외부 I/O 장치로부터 입출력 요청, 종료, 오류 발생 - 타이머에 의한 지정된 시간 경과
내부 인터럽트 (Internal Interrupt): CPU 스스로 발생한다.	* 프로그램 검사 인터럽트(Program Check Interrupt) -MCU에 정의되어 있지 않은 명령 실행 -0(Zero)로 나눗셈 시도 -보호된 메모리 영역 접근 -명령어 인터럽트를 수행	

6) 임베디드 이상의 프로세서를 기준으로 보았다고 할 수 있다. 이를 군에서는 인터럽트라는 표현과 익셉션(exception)을 혼용해서 사용하고 있다.

14.2 아두이노의 인터럽트 벡터와 단자

❐ 아두이노 우노의 인터럽트 단자

그림 14.2.1에 아두이노 우노에 사용되는 프로세서 Atmega328(DIP 28핀)의 단자 배정도를 보였다. 그림에서 1~28번은 프로세서 단자에 할당된 pin number이다[7].

Atmega328에서 제공하는 인터럽트의 개수는 실제로는 총 26개이지만 아두이노 우노에서 프로그래머에게 쓸 수 있도록 지원하는 인터럽트는 외부 인터럽트라 불리는 INT0, INT1 총 2개 뿐이다. 이 인터럽트는 단자 번호 4번(PD2), 5번(PD3) 단자에 외부 장치로부터 디지털 신호로 요구가 접수되면 발생하는 인터럽트이다.

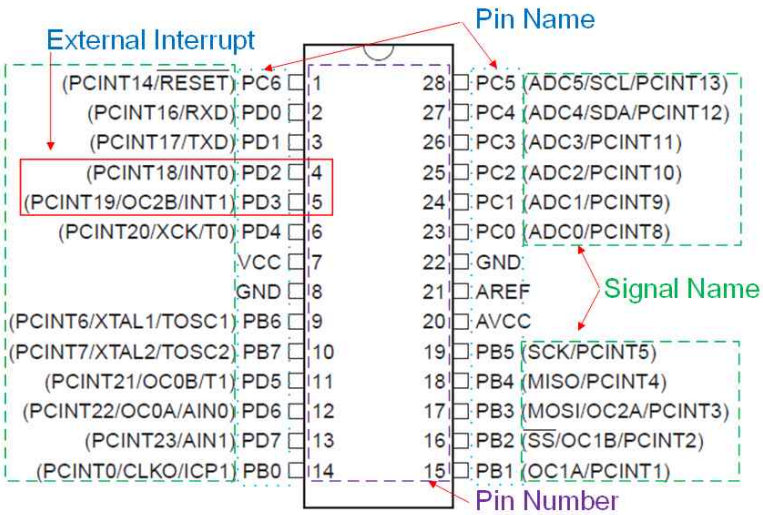


그림 14.2.1 아두이노 우노의 인터럽트 단자

---

7) 이 번호는 아두이노의 단자 번호와는 다르므로 착오 없기를 바란다. 아두이노 프로그램에 쓰이는 단자 번호는 논리적 번호로서 스케치 컴파일러에서 보드를 선택할 때 논리 번호가 실제의 단자 번호인 물리 번호로 매핑되는 변환과정을 거친다고 해석할 수 있다. 우노의 경우 INT0, INT1은 실제로는 단자번호 2번과 3번을 사용한다.

❐ 아두이노 우노의 벡터테이블과 인터럽트 서비스 루틴

표 14.2.1 Atmega328(우노의 프로세서)의 인터럽트

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset a
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

표 14.2.1은 아두이노 우노에 사용되는 프로세서인 Atmega328 프로세서의 인터럽트(vector) 번호에 따른 IVT 기준 번지로부터의 주소, 요구원(source)과 그 활용처를 보인 것이다. 인터럽트 벡터 번호는 x86 CPU에서는 인터럽트 타입이라 불리는 것과 같은 개념으로 IVT의 시작점에서의 인덱스 역할을 담당한다. 만약 벡터 번호가 N이라면 시작 번지로부터의 주소는 (N-1)x2 번지가 된다.

모든 인터럽트는 IVT내에 독립적인 인터럽트 벡터를 보유하고 있으며 인터럽트의 벡터번호가 낮을수록 더 높은 우선순위의 인터럽트를 보유한다. 만약 두 종류가 인터럽트가 동시에 발생하게 되면 더 우선순위가 높은 인터럽트의 ISR이 먼저 수행되고 난 후 다음 우선순위의 인터럽트에 해당하는 ISR이 수행된다.

표에서 인터럽트 벡터를 담고 있는 IVT의 시작 주소(혹은 base address)는 설정에 따라 표 14.2.2와 같이 선택하여 바꿀 수도 있다.

표 14.2.2 Reset과 IVT의 base Address(Start Address)

BOOTRST	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x000	0x002
1	1	0x000	Boot Reset Address + 0x0002
0	0	Boot Reset Address	0x002
0	1	Boot Reset Address	Boot Reset Address + 0x0002

만약 BOOTRST=1, IVSEL=0의 설정이라면 그림 14.2.2와 같이 메모리 $0000번지부터는 인터럽트 벡터 테이블이 위치하게 될 것이고 이후에 ISR과 응용 프로그램 등이 설치될 것이다.

```
Address  Labels  Code                Comments
0x0000           jmp    RESET        ; Reset Handler
0x0002           jmp    EXT_INT0     ; IRQ0 Handler
0x0004           jmp    EXT_INT1     ; IRQ1 Handler
0x0006           jmp    PCINT0       ; PCINT0 Handler
0x0008           jmp    PCINT1       ; PCINT1 Handler
0x000A           jmp    PCINT2       ; PCINT2 Handler
0x000C           jmp    WDT          ; Watchdog Timer Handler
0x000E           jmp    TIM2_COMPA   ; Timer2 Compare A Handler
0x0010           jmp    TIM2_COMPB   ; Timer2 Compare B Handler
0x0012           jmp    TIM2_OVF     ; Timer2 Overflow Handler
0x0014           jmp    TIM1_CAPT    ; Timer1 Capture Handler
0x0016           jmp    TIM1_COMPA   ; Timer1 Compare A Handler
0x0018           jmp    TIM1_COMPB   ; Timer1 Compare B Handler
0x001A           jmp    TIM1_OVF     ; Timer1 Overflow Handler
0x001C           jmp    TIM0_COMPA   ; Timer0 Compare A Handler
0x001E           jmp    TIM0_COMPB   ; Timer0 Compare B Handler
0x0020           jmp    TIM0_OVF     ; Timer0 Overflow Handler
0x0022           jmp    SPI_STC      ; SPI Transfer Complete Handler
0x0024           jmp    USART_RXC    ; USART, RX Complete Handler
0x0026           jmp    USART_UDRE   ; USART, UDR Empty Handler
0x0028           jmp    USART_TXC    ; USART, TX Complete Handler
0x002A           jmp    ADC          ; ADC Conversion Complete Handler
0x002C           jmp    EE_RDY       ; EEPROM Ready Handler
0x002E           jmp    ANA_COMP     ; Analog Comparator Handler
0x0030           jmp    TWI          ; 2-wire Serial Interface Handler
0x0032           jmp    SPM_RDY      ; Store Program Memory Ready Handler
;
0x0034   RESET:  ldi    r16, high(RAMEND); Main program start
0x0035           out    SPH,r16      ; Set Stack Pointer to top of RAM
0x0036           ldi    r16, low(RAMEND)
0x0037           out    SPL,r16
0x0038           sei                 ; Enable interrupts
0x0039           <instr>  xxx
  ...     ...    ...     ...
```

그림 14.2.2 IVT와 RESET의 ISR이 설치된 메모리 내용

사례 1) 그림에서 RESET 인터럽트가 들어오면 0x0000 번지의 IVT의 명령어를 수행하여 0x0034 번지로 분기한다. 이 번지는 RESET의 ISR 프로그램의 시작 번지이다.

사례 2) 외부 인터럽트 EXT_ INT0가 발생하면 이는 인터럽트 벡터가 2이므로 (2-1)x2 번지의 명령어(jmp EXT_INT0)를 수행한다. 그리하여 해당 서비스 루틴으로 분기한다.

시작 번지를 $0000을 사용할 경우의 IVT와 ISR의 가상 배치 상황을 그림 14.2.3에 보였다.

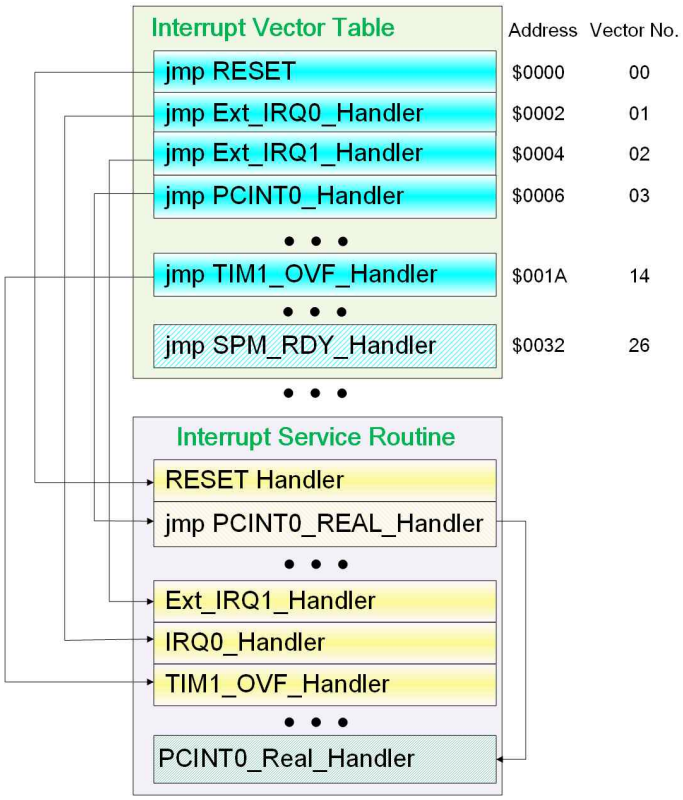


그림 14.2.3 Atmega328의 IVR와 ISR

사례 1) PCINT0 인터럽트는 벡터 3으로 서비스 된다. 해당 지점에는 ICINT0의 핸들러로 분기하는 명령어가 들어있어야 한다. 분기를 하면 그곳에 PCIINT0_REAL_Handler로 분기하는 명령어가 들어 있는데 이는 해당 ISR이 더 먼 곳에 있거나 프로그램의 길이가 길어 다 담지 못할 때 여유있는 메모리 공간으로 분기시키는 사례를 가상하여 개념적으로 기술한 것이다.

사례 2) 벡터 26 지점에는 원래 SPD_RDY의 ISR로 분기하는 명령어가 들어 있어야 한다. 그러나 이를 사용지 않을 때는 그 메모리 장치가 의미없는 쓰레기 데이터로 메워져 있거나, dummy ISR로 분기시키는 명령어가 들어 있을 수도 있다.

❐ 메가 2560의 인터럽트

표 14.2.3 ATmega2560의 인터럽트 벡터 테이블의 일부(총 1~57까지)

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	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt

.....(생략)...

47	$005C(3)	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066(3)	USART2 RX	USART2 Rx Complete
53	$0068(3)	USART2 UDRE	USART2 Data Register Empty
54	$006A(3)	USART2 TX	USART2 Tx Complete
55	$006C(3)	USART3 RX	USART3 Rx Complete
56	$006E(3)	USART3 UDRE	USART3 Data Register Empty
57	$0070(3)	USART3 TX	USART3 Tx Complete

아두이노 메가 2560보드는 Atmega2560 프로세서를 내장하고 있다 이 프로세서의 인터럽트는 표 14.2.3에 그 일부를 보인 바와 같이 총 57개가 지원된다. 그림 14.2.4은 인터럽트 벡터와 IVT 및 ISR의 관계를 사례를 가정하여 보인 것이다.

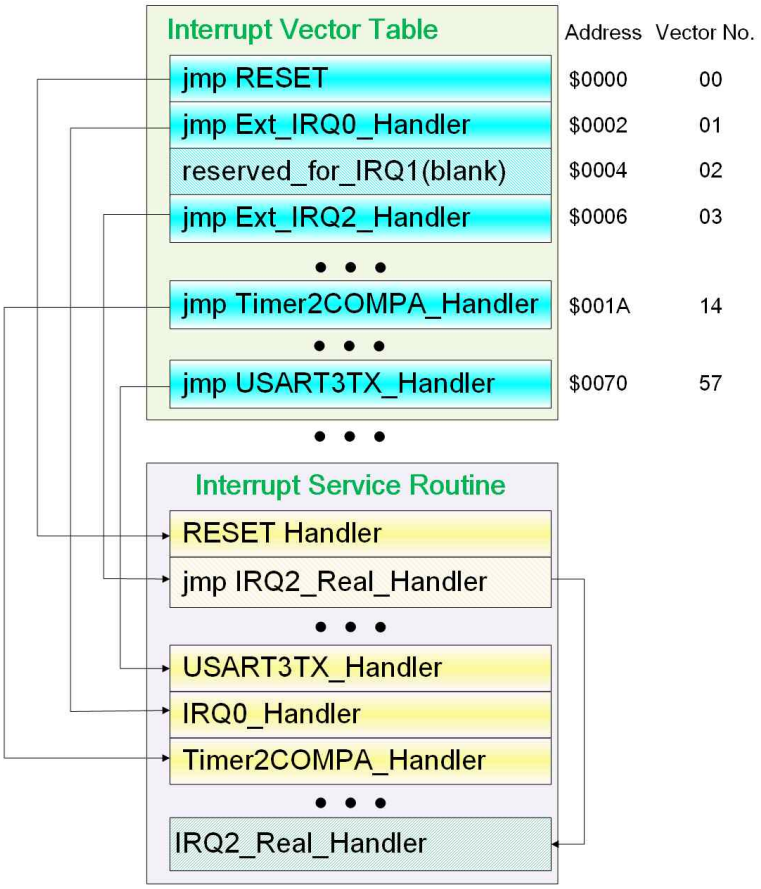


그림 14.2.4 ATmega2560 IVT, ISR과 벡터번호와의 관계

Atmega 2560은 외부 인터럽트 외에 다음과 같은 인터럽트를 제공한다.

■ 단자 상태 변화(pin change) 인터럽트(PCINT0~2)

다음과 같이 24개의 단자를 8개씩 3개 포트로 그룹지어 이 중 한 단자의 상태 변화가 다음 벡터 번호 10~12 중의 한 인터럽트 번호로 처리된다[8].

- PCINT7~PCINT0 : Pin Change Interrupt Request 0. 벡터 번호 10번

8) 아두이노에서는 이를 활용할만한 함수를 제공하고 있지 않다. 아두이노는 외부 인터럽만 지원한다.

- ▸ PCINT15~PCINT8 : Pin Change Interrupt Request 0. 벡터 번호 11번
- ▸ PCINT23~PCINT16 : Pin Change Interrupt Request 0. 벡터 번호 12번

■ Watchdog Time-out Interrupt-일정 시간이 경과하면 발생하는 인터럽트
■ 타이머 0~5 인터럽트 – capture/match
■ SPI, I2C 데이터 전송 관련 인터럽트
■ USART0~3 직렬통신 인터럽트
■ ADC 변환 인터럽트
■ EEPROM, FLASH 데이터 관련 인터럽트

14.3 아두이노 인터럽트 프로그래밍 개요

❑ 외부 인터럽트 사용법 개요

AVR 프로세서 자체는 여러 종류의 인터럽트를 보유하고 있지만[9] 아두이노에서는 일반 사용자에게는 단자로 제공되는 외부 인터럽트만 지원한다. 표 14.3.1에는 아두이노 모델의 일부 기종이 제공하는 인터럽트의 단자 번호와 인터럽트 번호를 보였다.

표 14.3.1 아두이노의 모델에 따른 단자 번호와 인터럽트 번호

모델 및 단자번호	인터럽트 번호[10]					
	INT0	INT1	INT2	INT3	INT4	INT5
Uno, Nano, Mini, other 328-based	2	3				
Mega, Mega2560, MegaADK	2	3	21	20	19	18
32u4 based (e.g Leonardo, Micro)	3	2	0	1	7	

아두이노 우노의 경우는 2개의 단자(2번과 3번)만 제공되며, Mega 2560은 6개의 단자(2, 3, 21~18번)를 지원한다. 우노의 경우 단자 2번은 인터럽트 0번으로 지원되며 3번 단자는 인터럽트 1번으로 제공된다[11]. 메가 2560과 우노는 단자 2번, 3번의 인터럽트 번호가 일치한다.

레오나르도 모델의 경우에는 2번 단자는 인터럽트 1번으로 지원된다. 이렇듯 선택한 모델에 따라 단자의 번호와 해당 인터럽트 번호가 달라질 수 있다.

---

9) Atmega 프로세서의 내부 레지스터를 사용한다면 이를 기능을 모두 이용할 수 있겠지만 스케치 함수로 제공되는 인터럽트는 단자를 사용하는 이른바 "외부인터럽트" 뿐이다.

10) 아두이노에서는 인터럽트 타입 번호를 인터럽트 번호라고 부른다.

11) 단자 번호 말고도 인터럽트 번호가 따로 관리되는 이유는 실제로 인터럽트는 인터럽트 번호(즉, 인터럽트 타입번호)가 IVT 내의 실제 ISR을 지정하는 인덱스로 쓰이기 때문이다.

아두이노 스케치 프로그램에서는 아래와 같이 인터럽트 번호에 따라 ISR 루틴을 연결짓는 함수(attachInterrupt)를 제공한다. 또한 단자 번호를 인터럽트 번호로 변환하는 함수(digitalPinToInterrupt)도 함께 제공하고 있다.

```
void attachInterrupt(digitalPinToInterrupt(pinNum), FuncPtr ISR, int mode)
```

이 함수는 pinNum으로 지정된 단자에 인터럽트가 발생하면 ISR() 함수를 수행하도록 지정한다[12].

사례 1) 예를 들어 아두이노 우노보드에서 아래와 같이 설정한 인터럽트는 아두이노 3번 단자의 신호가 RISING(Low에서 High로 신호가 바뀜)이면 인터럽트 1[13]이 발생하여 함수 myISR()이 수행되도록 설정한다.

```
attachInterrupt(digitalPinToInterrupt(3), myISR, RISING)
```

사례 2) 보드가 메가 2560이라면 18~21번 단자를 사용할 수 있다. 만약 아래와 같이 설정하였다면 아두이노 19번 단자의 상태가 High에서 Low로 신호가 바뀌면 인터럽트 4가 발생하면 yourIsr() 함수가 수행되도록 설정한다.

```
attachInterrupt(digitalPinToInterrupt(19), yourIsr, FALLING)
```

위 함수를 이용해 인터럽트 서비스 루틴을 IVT에 링크를 걸면 특별한 조작이 없어도 해당 단자로 조건에 맞는 신호가 유입될 때마다 해당 함수가 계속 호출되게 된다.

이때 인터럽트를 발생하기 위한 단자의 신호 조건은 다음과 같다.

---

12) 즉 타입 번호(혹은 벡터 번호)에 맞는 IVT의 초기화 작업을 수행하며, IVT의 해당 위치에 ISR() 함수로 분기하게 하는 명령어를 삽입한다.
13) 우노의 경우는 단자 3의 인터럽트 번호는 1이다.

❑ 인터럽트 발생의 조건

attachInterrupt()에서 설정하는 인터럽트 조건은 위에서 제시한 사례를 포함해서 다음과 같이 총 4가지의 모드를 지원한다.

```
LOW :  해당 단자가 LOW 상태일 때 인터럽트 발생
RISING : LOW→HIGH로 바뀔 때 인터럽트 발생
FALLING : HIGH→LOW로 바뀔 때 인터럽트 발생
CHANGE : 상태가 바뀔 때 인터럽트 발생(LOW→HIGH 혹은 HIGH→LOW로 변할 때 중 어떤 경우든 발생)
         RISING+FALLING의 OR 조건이라 볼 수 있다.
```

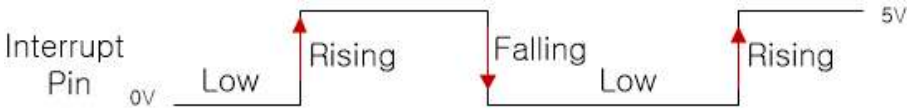


그림 14.2.1 인터럽트 입력 단자의 상태

예를 들어 아래와 같이 아두이노 2번 단자를 사용하여 인터럽트 조건을 CHANGE로 설정하였다면 단자가 HIGH로 되었을 때와 LOW로 되었을 때 모두 인터럽트가 발생한다. 이 경우 조건을 만족할 때마다 인터럽트가 발생하여 myISR()에 함수를 호출하게 된다.

```
attachInterrupt(digitalPinToInterrupt(2), myISR, CHANGE)
```

단자의 상태는 외부의 입력에 의해 제공되는 것이 일반적이지만, digitalWrite() 등의 함수로 출력한 결과에 의해 상태가 바뀌어도 인터럽트가 발생한다.

❑ 인터럽트 프로그래밍 주의사항

1) ISR 함수 안에서는 delay() 함수를 사용할 수 없다. delay() 함수가 사실은 인터럽트를 사용하는데 ISR 안에서는 다른 인터럽트가 금지되기 때문이다.
2) millis() 함수 값이 증가하지 않는다. delayMicroseconds()는 정상 가동한다[14].
3) 이 함수 안에서 받은 직렬 데이터는 손실될 수 있다.
4) 함수 내에서 변경하는 변수에 대해서는 '*volatile*'로 선언해야 한다[15].
5) ISR() 함수는 입력 파라미터를 사용할 수 없고, 어떤 값도 반환할 수 없다.
6) ISR 안에서는 가급적 빨리 수행루틴을 마무리하도록 작성한다.
7) ISR을 동시에 여러 개 수행할 수는 없다. 한 순간에는 우선 순위가 높은 그 중 하나만 수행된다.
8) 현재의 아두이노 지원체계에서는 기본적으로는 ISR 수행 중에 다른 인터럽트가 발생하였을 때 이를 처리하지는 않는다[16]. 만약 ISR 중에 인터럽트를 받고자 한다면 ISR 시작 당시에 interrupt() 함수를 수행하면 된다[17].

---

14) 설명을 보면 카운터를 쓰지 않기 때문이라고 했는데 '인터럽트를 사용하지 않아서' 라고 의역할 수 있다. 이렇게 되려면 타이머 카운트의 값이 64비트여야 하는데 타이머 카운트는 실제로는 16비트가 최대라서 의문이 든다(??).
15) volatile은 컴파일러에게 변수를 어떻게 다루어야 하는지를 알려주는 일종의 directive로서 variable qualifier라고 한다. 이 지시어는 컴파일러는 이들 변수의 값을 실제로 읽어보지 않고 CPU 내의 레지스터에 저장시켜 놓고 이를 재활용하여 사용하는 최적화를 시행하는 것을 금지한다. 변수의 값이 값이 바뀌었는데도 컴파일러는 나름 최적화를 꾀하기 위해 변하지 않은 레지스터의 값을 읽어 사용하는 문제를 방지하기 위한 조치이다. 참조 URL
16) 사실 여기에는 상당한 복잡한 메카니즘의 문제점이 도사리고 있다. 경험으로 관찰한 바에 따르면 Atmega는 nested interrupt를 지원하지 않는다. 이 구조가 허용된다면 우선순위가 높은 인터럽트가 발생하면 일단 그 인터럽트를 허가하고 그 다음 순위의 인터럽트로 돌아와야 한다. 일반적으로 ISR 내부에서는 모든 인터럽트는 금지된다. 따라서 별도로 추가 인터럽트를 허가하는 명령을 수행해야 한다. 아두이노도 이를 지원한다. 그러나 이 인터럽트는 무조건 모두 허용하기 때문에 우선순위가 높은 인터럽트의 동작의 수행을 보장하지 않는다.
17) 인터럽트에 대한 자세한 고찰은 다음 URL을 참고하기 바란다.
참조 URL: http://gammon.com.au/interrupts

14.4 인터럽트 관련 아두이노 표준 함수

1) 인터럽트 설치 함수– attatchInterrupt()

<table>
<tr><td colspan="3">void attachInterrupt(digitalPinToInterrupt(pinNum), FuncPtr ISR, int mode) 추천
<s>void attachInterrupt(uint8_t intNo, FuncPtr ISR, int mode) 비추천. 설명생략</s>
<s>void attachInterrupt(uint8_t pinNo, FuncPtr ISR, int mode) 비추천. 설명 생략</s></td></tr>
<tr><td>기능</td><td colspan="2">인터럽트 발생 시 처리할 ISR을 지정(연결).
주의! : 아두이노는 외부 인터럽트만 지원한다.</td></tr>
<tr><td rowspan="3">매개변수</td><td>digitalPinToInterrupt(pinNum)</td><td>digitalPinToInterrupt(pinNum) 함수는
단자번호(pinNum)를 지정하면 다음과 같이 외부 인터럽트 번호(INT0~INT5)로 변환해 준다.
<table>
<tr><th>인터럽트 번호</th><th>단자 번호</th><th>인터럽트 번호</th><th>단자 번호</th></tr>
<tr><td>0</td><td>2</td><td>1</td><td>3</td></tr>
<tr><td>2</td><td>21</td><td>3</td><td>20</td></tr>
<tr><td>4</td><td>19</td><td>5</td><td>18</td></tr>
</table></td></tr>
<tr><td>FuncPtr ISR</td><td>인터럽트 발생 시 분기하여 수행할 ISR 함수.
callback 함수라고도 부른다.</td></tr>
<tr><td>int mode</td><td>인터럽트가 언제 발생할 것인지 지정
- LOW : 해당 단자가 LOW 상태일 때 발생[18)]
- RISING : LOW→HIGH로 바뀔 때 발생
- FALLING :HIGH→LOW로 바뀔 때 발생
- CHANGE : 상태가 바뀔 때 발생(LOW→HIGH 혹은 HIGH→LOW로 변할 때 중 어떤 경우든 발생)</td></tr>
<tr><td>리턴값</td><td>void</td><td>없음</td></tr>
</table>

18) 실험에 의하면 ISR을 빠져 나오면 단자가 LOW를 계속 유지하는 이상 계속 인터럽트가 발생하는 것으로 관찰되었다.

attachInterrupt() 함수는 외부 인터럽트 번호, 혹은 단자와 이곳에 인터럽트가 발생할 경우 호출될 ISR 함수를 연결하는 작업을 수행한다. 인터럽트를 사용하기 전 이와 같은 초기화 설정을 통해 인터럽트가 발생되면 해당 ISR로 분기할 수 있도록 IVT를 초기화하는 작업을 수행하도록 미리 ISR 함수를 정의해두는 작업이 필요하다.

이 함수의 첫 번째 인자는 인터럽트 번호 혹은 인터럽트 단자 번호를 첫 번째로 지정하게 되어 있는데 이와 같이 인터럽트 번호 혹은 단자 번호 숫자를 직접 지정하는 것은 입력하는 값이 단자 번호인지 인터럽트 번호인지가 애매하기 때문에 권장하지 않는다.

이보다는 **digitalPinToInterrupt()** 함수를 사용하여 **단자 번호를 지정**하여 이를 인터럽트 번호로 치환하여 사용하는 방법을 추천하고 있다. 아래에 스케치 통합환경에 digitalPinToInterrupt() 함수의 실제로 정의된 내용을 보였다.

```
Arduino₩Arduez_sketch_v16₩hardware₩arduino₩variants₩mega₩pins_arduino.h
#define digitalPinToInterrupt(p) ((p) == 2 ? 0 : ₩
((p) == 3 ? 1 : ((p) >= 18 && (p) <= 21 ? 23 - (p) : NOT_AN_INTERRUPT)))
```

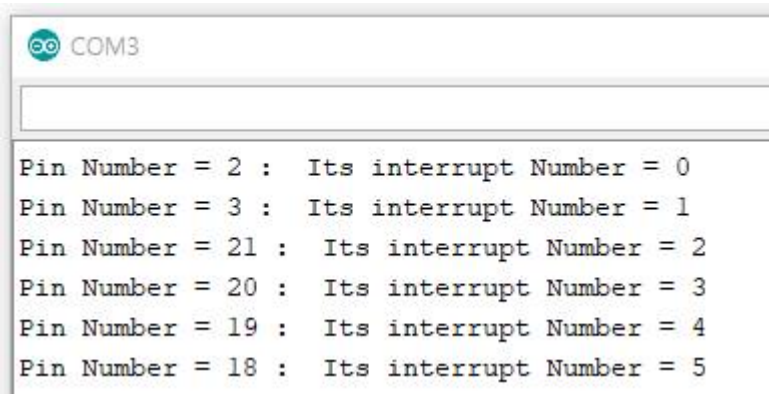
이를 관찰하여 보면 단자번호와 INT 번호와의 관계는 다음과 같음을 알 수 있다.

digitalPinToInterrupt(2) -> 0을 반환. 단자번호=2. INT번호=0.
digitalPinToInterrupt(3) -> 1을 반환. 단자번호=3. INT번호=1.
digitalPinToInterrupt(21) -> 2를 반환. 단자번호=21. INT번호=2.
digitalPinToInterrupt(20) -> 3을 반환. 단자번호=20. INT번호=3.
digitalPinToInterrupt(19) -> 4를 반환. 단자번호=19. INT번호=4.
digitalPinToInterrupt(18) -> 5을 반환. 단자번호=18. INT번호=5.

본 자료의 정확도를 판단하기 위해서는 **digitalPinToInterrupt()** 함수를 사용해 입력한 단자가 몇 번째의 INT 번호로 할당되는지 실험 프로그램을 작성

하여 출력해 보면 알 수 있다.

❒ 실험 4.1 : digitalPinToInterrupt()의 역할을 살펴본다. 단자번호를 입력하면 해당 인터럽트 번호를 반환하는 기능을 수행한다.	
digitalPinToInterrupt.ino : 아두이노 표준 함수, digitalPinToInterrupt()	
01	int pinNumList[6] = {2, 3, 21, 20, 19, 18}; // 인터럽트 단자번호 목록
02	void setup() {
03	Serial.begin(9600);
04	for (int i=0; i<6; i++) {
05	Serial.print("Pin Number = ");
06	int pinNum = pinNumList[i];
07	Serial.print(pinNum);
08	Serial.print(" : Its interrupt Number = ");
09	Serial.println(digitalPinToInterrupt(pinNum));
10	}
11	}
12	void loop() {
13	}



digitalPinToInterrupt.ino의 수행결과(메가 2560)

```
COM5

Pin Number = 2 :  Its interrupt Number = 0
Pin Number = 3 :  Its interrupt Number = 1
Pin Number = 21 :  Its interrupt Number = -1
Pin Number = 20 :  Its interrupt Number = -1
Pin Number = 19 :  Its interrupt Number = -1
Pin Number = 18 :  Its interrupt Number = -1
```

digitalPinToInterrupt.ino의 수행결과(아두이노 우노)

이상의 결과를 종합하여 보면 아두이노 우노는 인터럽트를 2개만 지원하며 메가 2560과 인터럽트INT0와 INT1와 단자번호까지 같음을 알 수 있다. 실험 결과를 정리하여 보면 표 14.4.1과 같다.

표 14.4.1 인터럽트 번호와 아두이노 논리 단자 번호 매핑 관계

지원하는 모델	인터럽트 번호	단자번호	인터럽트 번호	단자번호
Uno, Mega 2560	**INT0**	**2**	**INT1**	**3**
Mega 2560	**INT2**	**21**	**INT3**	**20**
	INT4	**19**	**INT5**	**18**

2) 인터럽트 해제 함수– detachInterrupt()

detachInterrupt(digitalPinToInterrupt(pinNum)) 추천 ~~detachInterrupt(interrupt) 비추천. 설명 생략.~~ ~~detachInterrupt(pin) 비추전. 설명생략.~~		
기능	지정한 외부 인터럽트의 사용을 중지한다. 해당 단자(pinNum)에 인터럽트를 발생시킬 수 있는 조건(신호 유입 등)이 되어도 인터럽트가 발생되지 않는다. 따라서 ISR 함수도 수행되지 않는다. 프로세서 내부의 인터럽트제어기의 마스크 레지스터 해당 비트를 설정하는 방식으로 수행된다.	
매개변수	digitalPinToInterrupt(pinNum)	없음
리턴 값	void	없음

3) 인터럽트 금지, 허가 함수 – noInterrupts(), interrupts()

attatchInterrupt()와 detachInterrupt() 함수가 지정한 특정 단자의 인터럽트를 금지하는데 반해 noInterrupts() 함수는 모든[19] HW 인터럽트를 금지한다. 여러 종류의 인터럽트를 접수하는 맨 마지막 단계에서 금지하는 명령어라고 할 수 있다. 인터럽트에 관련된 초기화 작업 이전에 이 명령어를 사용하면 준비되지 않는 상태에서 인터럽트가 발생하여 오류가 발생하는 일을 막을 수 있다.

그림 14.4.1에는 여러 종류의 인터럽트를 입력하는 개념도를 보였다. noInterrupt() 함수는 CPU 코어가 받아들이는 최종 단계에서 인터럽트를 금

19) 14.2절에 나온 H/W 기반의 모든 인터럽트(8종)를 금지한다.

지할 수 있는 함수이다. 이는 CPU의 제어 플래그(flag), IEF를 해제함으로써 이루어진다.

그림에 보인 바와 같이 모든 인터럽트는 인터럽트 제어기에 의해 의해 관리된다. 아두이노에서 지원하는 외부 인터럽트(그림에서 Ext. INTs)도 여타의 인터럽트와 같이 인터럽트 제어기에 접수되어 CPU 코어에게 전달된다. noInterrupt() 함수는 사실상 IEF를 클리어[20]하는 역할을 담당하기 때문에 다른 여타의 인터럽트들도 모두 접수가 차단된다. 이렇게 금지할 수 있는 인터럽트를 통틀어 maskable 인터럽트라고 칭한다.

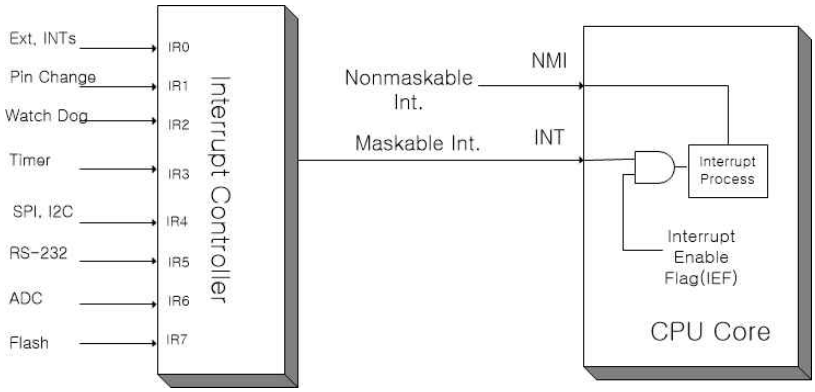


그림 14.4.1 인터럽트 입력 개념도

20) AND 게이트의 한쪽 입력에 IEF에 연결된다. 이 비트가 0이면 인터럽트에서 요구되는 모든 인터럽트 신호가 making되어 CPU 코어의 인터럽트 처리 과정으로 전달되지 않는다.

void noInterrupts()		
기능	메인 프로그램이 시작할 때는 인터럽트는 허용하는 것이 본래 기본값으로 설정되어 있다. 본 함수는 CPU 차원에서 이를 불허하도록 CPU의 상태 레지스터 중 IE(Interrupt Enable) 비트를 0으로 해제한다. IE가 0이면 외부에서 인터럽트가 발생해도 CPU는 이를 무시한다. 외부 인터럽트를 설정하는 작업이 마무리되지 못하였거나, 시간을 다투는 중요한 루틴을 수행하기 전에 호출할 수 있다[21]. IE가 0이면 직렬통신의 송신 작업이 진행되지 않으며 너무 오래 이 상태를 유지하면 타이머 동작 등이 제대로 진행되지 않을 수 있다. "cli() // clear interrupt flag." 명령으로 대신할 수 있다.	
매개변수	void	없음
리턴 값	void	없음

인터럽트를 금지 후 attachInterrupt() 함수로 인터럽트 서비스 루틴을 설치하고 나면 다시 인터럽트를 허가할 때 interrupts() 함수를 수행한다. 이 함수는 CPU가 인터럽트를 받는 일을 허가한다.

void interrupts()		
기능	인터럽트 요청에 따른 ISR 처리를 허용한다. noInterrupts()함수로 인터럽트 요청을 불허했던 것을 해제한다. CPU의 상태 레지스터 중에서 인터럽트를 받아들이도록 하는 비트 IEF가 1로 설정된다. "sei() // set interrupt flag." 명령으로 대신할 수 있다.	
매개변수	void	없음
리턴 값	void	없음

21) CPU의 상태 레지스터에는 보통 외부 인터럽트를 무시하는 제어 비트를 갖고 있다. 아마도 이 비트(Global Interrupt Enable)를 제어하는 것으로 추정된다.

14.5 외부 인터럽트 프로그래밍 예제

본 절에서는 INT 단자를 통해 인입되는 외부 인터럽트를 처리하는 사례를 연습해보기로 한다.

예제(5.1) - GPIO 출력에 의한 INT 단자의 인터럽트 요구 발생 - 인터럽트 기반의 LED 점등(LED 회로 결선 필요)

❒ 예제 5.1 : 메인 루틴에서 1초마다 행하는 출력 동작이 인터럽트에 의해 토글된 state의 값을 출력한다. 아두이노 우노/메가 2560 점검 완료.	
INT_LED.ino: 아두이노 표준 함수, attachInterrupt(), digitalPinToInterrupt()	
01	int pinINT = 2; // pin number of INT0. connected to LED
02	volatile int state = HIGH;
03	void setup() {
04	pinMode(pinINT, OUTPUT);
05	digitalWrite(pinINT, LOW);
06	attachInterrupt(digitalPinToInterrupt(pinINT), blink, CHANGE);
07	}
08	void loop() {
09	digitalWrite(pinINT, state); // Int occurs because of pin state change
10	delay(1000);
11	}
12	void blink() { // Interrupt Service Routine
13	state = !state; // Toggle the state
14	}

본 예제 프로그램의 실습을 위해서는 그림 14.5.1에 보인 바와 같이 단자 2번을 LED에 연결하여 접지시켜야 한다. 예제 프로그램을 수행하면 LED가 점등된 이후 1초 주기로 소등과 점등을 무한 반복한다.

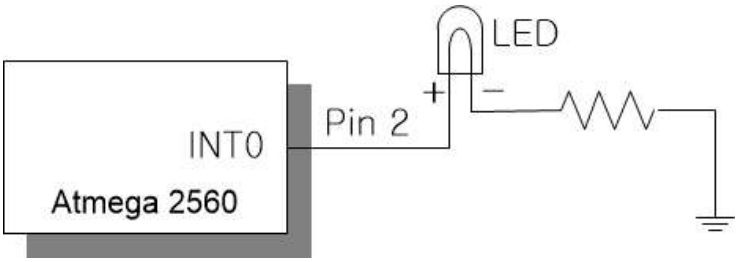


그림 14.5.1 실험을 위한 LED 연결도

그림 14.5.2에는 예제 프로그램의 흐름도를 보였다. 단자 2번에 변수 state를 쓰는 동작(소스의 11번 줄)에서 초기 0->1로 상태가 변하면서 인터럽트가 발생한다. 인터럽트가 발생하면 ISR로 분기하여 그곳에서 state를 토글시키고 돌아온다. 이후 delay함수를 1초가 수행시킨 후 digitalWrite() 함수를 수행하면 LED는 소등되면서 다시 단자 2의 상태가 바뀌어 인터럽트가 발생한다. 이는 다시 ISR을 수행하게 만들고 변수 state를 토글하게 하여 이 같은 동작을 무한 수행하게 된다.

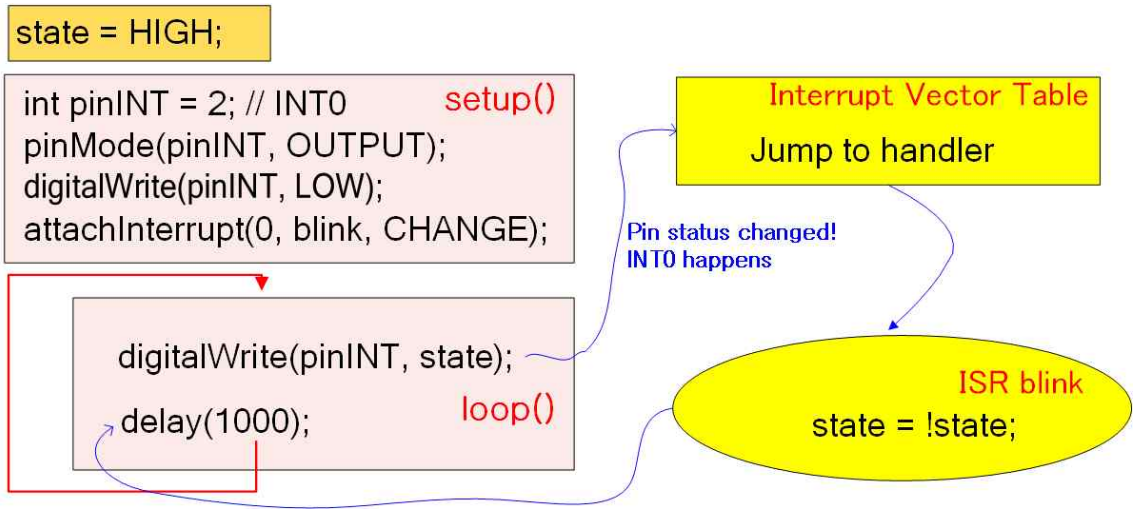


그림 14.5.2 프로그램의 흐름도

예제(5.2) - GPIO 출력에 의한 INT 단자의 인터럽트 요구 발생 - 인터럽트 기반의 LED 점등(내장 LED 활용)

❒ 예제 5.2 : 5.1의 예제에 내장 LED를 점등하는 기능을 추가한다. 아두이노 우노 점검 완료.	
INT_LED_BUILTIN.ino: 표준 함수, attachInterrupt(), digitalPinToInterrupt()	
01	int pinINT = 2; // pin number of INT0. connected to LED
02	volatile int state = HIGH;
03	void setup() {
04	pinMode(LED_BUILTIN, OUTPUT);
05	pinMode(pinINT, OUTPUT);
06	digitalWrite(pinINT, LOW);
07	attachInterrupt(digitalPinToInterrupt(pinINT), blink, CHANGE);
08	}
09	void loop() {
10	digitalWrite(pinINT, state); // Int occurs because of pin state change
11	digitalWrite(LED_BUILTIN, state); // 외장 LED와 반대의 동작을 한다.
12	delay(1000);
13	}
14	void blink() { // Interrupt Service Routine
15	state = !state; // Toggle the state
16	}

본 예제는 외장 LED와 내장 LED가 서로 반대로 LED가 점등되는 것을 관찰할 수 있다. 10번 라인에서 외장 LED를 점등하는 순간 인터럽트가 발생하여 14번의 blink() 함수를 수행하여 state 상태를 반전해 놓고 난 후 11번 라인으로 귀환하기 때문에 내장 LED는 반대로 동작하는 것이다[22].

☺ 미션 1: 본 실험을 기반으로 state 값을 시리얼 모니터로 출력해 보자.

22) 그런데 내장 LED만 활용하려고 외장 LED를 제거하면 동작이 잘 안되는 것을 관찰하였다. 여러분들은 어떤지 확인해보고 그 이유와 대책을 토론해 보도록 하자.

예제(5.3) - 푸시 버튼에 의한 인터럽트 발생 실험(메가 2560)

그림 14.5.3과 같이 LED 단자(2번)와 인터럽트 발생 단자(21번)를 분리시켜서 푸시 버튼에 의해 인터럽트를 발생시킬 수도 있다.

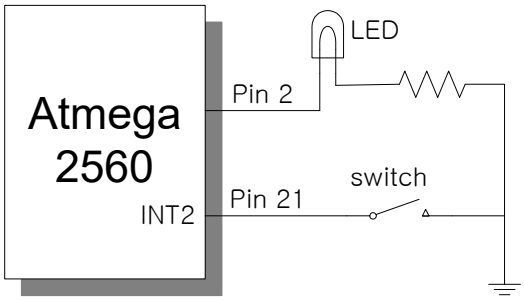


그림 14.5.3 인터럽트 실험 결선도

INT2(단자 21번)에 푸시 버튼을 연결해서 INT2 단자의 상태가 LOW로 바뀌는 경우 blink 함수를 수행하게 하여 LED를 점등 혹은 소등한다.

버튼을 누를 때마다 LED를 On/Off 하는 동작을 확인할 수 있는데 스위치의 회로 결성에 따라 항상 정확하게 토글하지는 않고 가끔 스위치의 변화를 무시할 수도 있다. 이는 스위치의 채터링 잡음 때문에 인터럽트가 짝수 번 발생하여 토글이 무시되는 것과 같은 효과를 낼 수도 있기 때문이다. 일반적으로 디지털 회로에서는 입력단자를 아무런 연결을 취하지 않는 floating 상태로 두면 H로 인식된다. 단자를 아무 곳에도 연결되지 않으면 그 단자는 H 값을 갖는다고 볼 수 있다. 따라서 단자 21을 접지에 살짝 접촉시키는 것만으로 접촉하는 순간 L가 되므로 FALLING 펄스 입력을 인가한 것과 같은 효과를 낼 수 있다.

☺ 미션 2: 본 실험은 메가 2560용으로 설계한 것이다. 여러분들이 그동안 배운 지식을 활용하여 아두이노 용으로 바꾸어 실험해 보기 바란다.

❒ 예제 5.3 : 단자 21번(INT2) 핀을 접지에 푸시 버튼을 이용해 연결한다. LOW가 되는 순간 인터럽트 요구를 발생시켜 LED 점등을 제어한다. 메가 2560.	
INT_2.ino : 아두이노 표준 함수 – attachInterrupt()	
01	#define LED_PIN 2 // connected to LED
02	#define INT_PIN 21 // connected to push button switch. =INT2.
03	volatile int state = HIGH;
04	void setup(){
05	pinMode(LED_PIN, OUTPUT);
06	pinMode(INT_PIN, INPUT_PULLUP);
07	attachInterrupt(digitalPinToInterrupt(INT_PIN), blink, FALLING);
08	}
09	void loop() {
10	digitalWrite(LED_PIN, state);
11	}
12	void blink() {
13	state = !state;
14	}