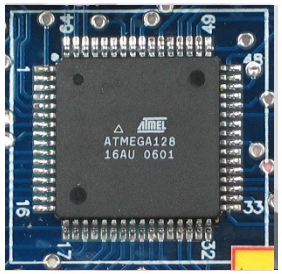
한국공학대학교
TECH UNIVERSITY OF KOREA



06 2022 May

ATMEGA128 인터럽트의 이해

• 학습 목표

- **인터럽트 동작 및 AVR에서의 인터럽트 메커니즘 이해**
  - 인터럽트와 메인 프로그램과의 관계
  - 인터럽트는 왜 사용하여야 하는가?
  - 일반적으로 인터럽트를 사용할 수 밖에 없는 상황은 어떠한 것인가?
- **AVR에서의 인터럽트 구동을 위한 처리 방법**
  (CodeVision에서의 처리 방법)
- **외부 인터럽트의 처리 및 활용 방법**
  - 외부에서 인가하는 인터럽트 신호는 전기적으로 어떻게 인가하는가?
  - 인가된 신호는 어떠한 방법으로 AVR은 인지하는가?

인터럽트(Interrupt) 개요

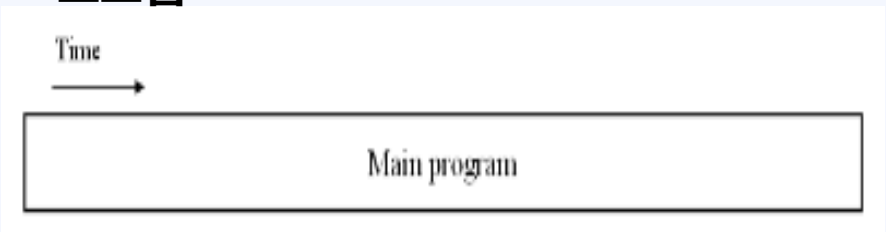
인터럽트 개념

- 프로그램이 수행되고 있는 동안에 어떤 조건이 발생하여 수행중인 프로그램을 일시적으로 중지 시키게 만드는 조건이나 사건의 발생
  - 비동기적으로 처리→다른 프로그램이 수행되는 동안 여러 개의 사건을 처리할 수 있는 메커니즘
  - 인터럽트가 발생하면 마이크로 컨트롤러는 현재 수행중인 프로그램을 일시 중단하고, 인터럽트 처리를 위한 프로그램을 수행한 후에 다시 원래의 프로그램으로 복귀
- ISR(Intertupt Service Routine) 또는 Interrupt handler : 인터럽트 처리하는 프로그램

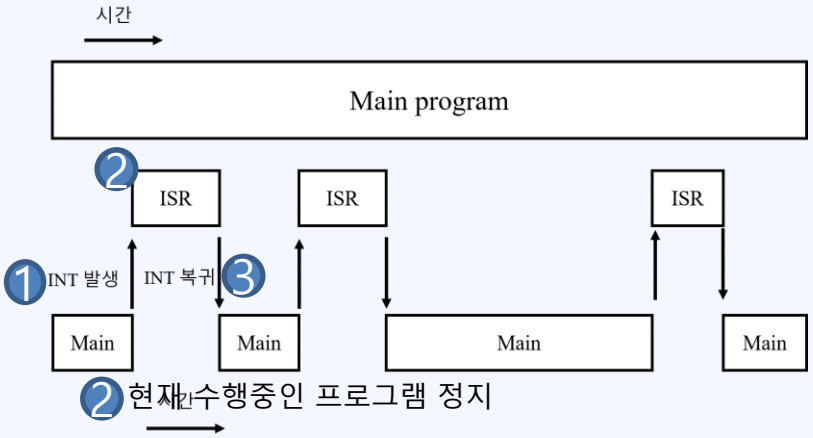
인터럽트(Interrupt) 개요

인터럽트 개념

- 프로그램이 수행되고 있는 동안에 어떤 조건이 발생하여 수행중인 프로그램을 일시적으로 중지 시키게 만드는 조건이나 사건의 발생
  - ➢ 비동기적으로 처리→다른 프로그램이 수행되는 동안 여러 개의 사건을 처리할 수 있는 메커니즘
    - 일반적 프로그램:Foreground Processs, 인터럽트 프로그램 : Background Process
  - ➢ 인터럽트가 발생하면 마이크로 컨트롤러는 현재 수행중인 프로그램을 일시 중단하고, 인터럽트 처리를 위한 프로그램을 수행한 후에 다시 원래의 프로그램으로 복귀
- ISR(Interrupt Service Routine) 또는 Interrupt handler : 인터럽트 처리하는 프로그램



< 인터럽트가 없는 프로그램 실행 >



< 인터럽트가 있는 프로그램 실행 >

인터럽트(Interrupt) 개요

인터럽트 의 종류

인터럽트 발생 원인에 의한 분류

- 하드웨어 인터럽트
  - 내부 인터럽트 : 마이크로 컨트롤러 내부의 기능에 의해 발생
  - 외부 인터럽트 : 마이크로 컨트롤러 외부에 부가된 소자에 의해 발생
- 소프트웨어 인터럽트

인터럽트 발생 원인에 의한 분류

- 일반적인 인터럽트($\overline{INT}$)
  - 프로그램에 의하여 인터럽트의 요청을 받아들이지 않고 무시할 수 있는 구조의 인터럽트(maskable interrupt)로, 일반적으로 우선처리 메커니즘 보유
- 차단 불가능 인터럽트($\overline{NMI}$)
  - 프로그램에 의해 어떤 방법으로도 인터럽트 요청이 차단될 수 없는 인터럽트(non-maskable interrupt)로, 전원 이상이나 비상 정지 등의 비상상황에 대비하기 위해 사용

인터럽트(Interrupt) 개요

인터럽트 처리 방식에 의한 분류

- **일반적인 인터럽트(/INT)**
  - 프로그램에 의하여 인터럽트의 요청을 받아들이지 않고 무시할 수 있는 구조의 인터럽트(maskable interrupt)를 의미
  - 우선처리 메커니즘
  - 우선처리 메커니즘의 경우 보통 인터럽트를 허용하는 방법은 인터럽트 마스크 레지스터 또는 인터럽트 허용 레지스터를 사용하여 각각의 인터럽트를 개별적 허용하고 이것들을 다시 전체적으로 허용함.

- **차단 불가능 인터럽트(/NMI)**
  - 프로그램에 의해 어떤 방법으로도 인터럽트 요청이 차단될 수 없는 인터럽트(non-maskable interrupt)를 의미
  - 전원 이상이나 비상 정지 스위치등과 같이 시스템에 치명적인 오류를 대비하기 위해 주로 사용

인터럽트(Interrupt) 개요

인터럽트의 제어 및 처리 절차

벡터형 인터럽트

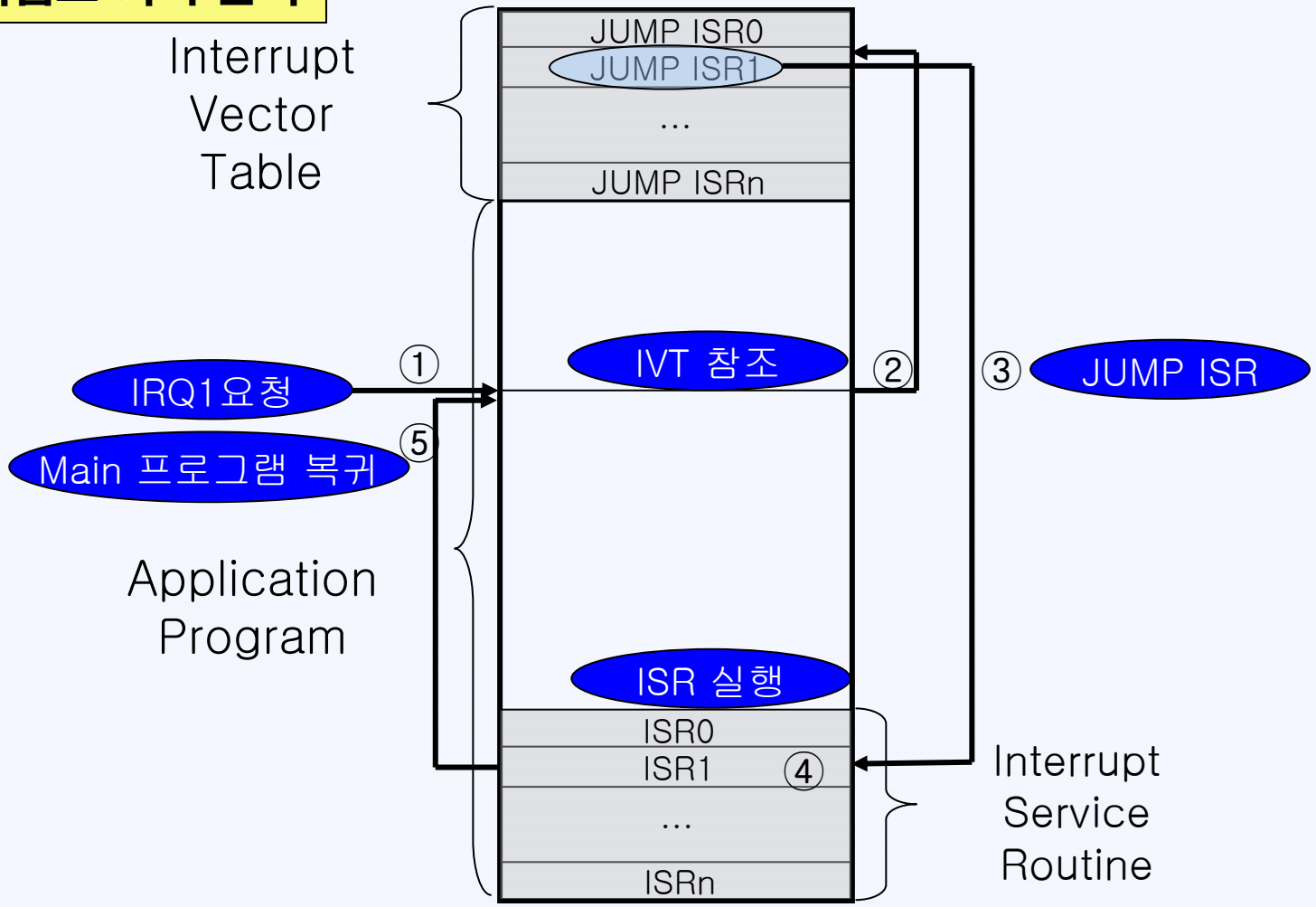
- 인터럽트가 발생할 때마다 인터럽트를 요청한 장치가 인터럽트 벡터를 마이크로 컨트롤러에게 전송하는 방식
- 각 주변장치가 각각의 인터럽트 신호선을 가지고 있고, 각 주변장치가 인터럽트를 요청하면 마이크로컨트롤러는 각각의 인터럽트에 따라 미리 지정된 인터럽트 벡터(벡터 테이블)를 가지고 있어 즉시 해당 인터럽트 서비스 루틴을 찾아가는 방식
  - 인터럽트 처리 응답시간이 빠르고, AVR에 구현된 방식

인터럽트의 우선순위

- 2개 이상의 주변장치가 동시에 마이크로 컨트롤러에 인터럽트를 요구하는 경우 우선 순위를 미리 정하여 한번에 하나의 인터럽트를 선택하여 처리
- 우선순위가 높은 인터럽트의 처리 중 -> 낮은 순위의 인터럽트는 대기상태
- 우선순위가 높은 인터럽트의 처리 끝 -> 낮은 순위의 인터럽트는 미리 지정된 우선순위에 의해 처리
  - AVR에서의 인터럽트 처리의 우선 순위는 벡터 테이블에 의해 정의된 순서임.

인터럽트(Interrupt) 개요

인터럽트 처리 순서



인터럽트의 종류

- 리셋을 포함하여 총 35종의 인터럽트 소스가 존재
  - 외부 인터럽트 8개,
  - 타이머/카운터0에 관련된 인터럽트 2개
  - 타이머/카운터1에 관련된 인터럽트 5개
  - 타이머/카운터2에 관련된 인터럽트 2개
  - 타이머/카운터3에 관련된 인터럽트 5개
  - USART0와 USART1에 관련된 인터럽트 각각 3개와 기타의 인터럽트 6개
- 인터럽트의 동작 형태
  - 인터럽트가 발생하면 관련 플래그 비트를 1로 세트하여 트리거 시키는 형태
    - ✓ 프로그램 카운터가 실제 인터럽트 벡터로 지정되어 인터럽트 처리 루틴을 수행
    - ✓ 해당 플래그는 하드웨어에 의해 자동으로 0으로 클리어됨.
    - ✓ 인터럽트 플래그는 해당 비트에 1을 써 넣음으로써 0으로 클리어할 수 있음.
    - ✓ 인터럽트 마스크 레지스터 또는 SREG 레지스터를 통해 금지 상태로 설정하여 놓았더라도 인터럽트가 발생하면 해당 인터럽트 플래그가 1로 설정되어 인터럽트 대기 상태로 되며, 나중에 인터럽트가 허가상태로 설정될 때 해당 인터럽트가 처리됨.
  - 인터럽트 조건이 발생한 동안에만 인터럽트를 트리거 하는 형태
    - 인터럽트 발생조건이 사라지면 인터럽트 요청도 없어지므로 나중에 인터럽트가 다시 허용 상태로 되더라도 인터럽트는 요청되지 않음.

ATmega128의 인터럽트 구성 (2)

인터럽트의 종류 및 인터럽트 벡터

높음

우선순위

낮음

벡터번호 (우선순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
1	0x0000	RESET	외부 핀, 전원 투입 리셋, 저전압 검출 리셋, 워치독 리셋, JTAG AVR 리셋
2	0x0002	INT0	외부 인터럽트 0
3	0x0004	INT1	외부 인터럽트 1
4	0x0006	INT2	외부 인터럽트 2
5	0x0008	INT3	외부 인터럽트 3
6	0x000A	INT4	외부 인터럽트 4
7	0x000C	INT5	외부 인터럽트 5
8	0x000E	INT6	외부 인터럽트 6
9	0x0010	INT7	외부 인터럽트 7
10	0x0012	TIMER2 COMP	타이머/카운터2 비교 일치
11	0x0014	TIMER2 OVF	타이머/카운터2 오버플로우
12	0x0016	TIMER1 CAPT	타이머/카운터1 입력 캡처
13	0x0018	TIMER1 COMPA	타이머/카운터1 비교 일치 A
14	0x001A	TIMER1 COMPB	타이머/카운터1 비교 일치 B
15	0x001C	TIMER1 OVF	타이머/카운터1 오버플로우
16	0x001E	TIMER0 COMP	타이머/카운터0 비교 일치
17	0x0020	TIMER0 OVF	타이머/카운터0 오버플로우

벡터번호 (우선순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
18	0x0022	SPI, STC	SPI 시리얼 통신 완료
19	0x0024	USART0, RX	USART0, 수신 완료
20	0x0026	USART0, UDRE	USART0, 데이터 레지스터 비움
21	0x0028	USART0, TX	USART0, 송신 완료
22	0x002A	ADC	ADC 변환 완료
23	0x002C	EE READY	EEPROM 준비
24	0x002E	ANALOG COMP	아날로그 비교기
25	0x0030	TIMER1 COMPC	타이머/카운터1 비교 일치 C
26	0x0032	TIMER3 CAPT	타이머/카운터3 입력 캡처
27	0x0034	TIMER3 COMPA	타이머/카운터3 비교 일치 A
28	0x0036	TIMER3 COMPB	타이머/카운터3 비교 일치 B
29	0x0038	TIMER3 COMPC	타이머/카운터3 비교 일치 C
30	0x003A	TIMER3 OVF	타이머/카운터3 오버플로우
31	0x003C	USART1, RX	USART1, 수신 완료
32	0x003E	USART1, UDRE	USART1, 데이터 레지스터 비움
33	0x0040	USART1, TX	USART1, 송신 완료
34	0x0042	TWI	I2C 통신 인터페이스
35	0x0044	SPM READY	저장 프로그램 메모리 준비

외부 인터럽트

내장 기능 인터럽트

- 타이머/카운터0에 관련된 인터럽트 2개
- 타이머/카운터1에 관련된 인터럽트 5개
- 타이머/카운터2에 관련된 인터럽트 2개
- 타이머/카운터3에 관련된 인터럽트 5개
- USART0와 USART1에 관련된 인터럽트 각각 3개와 기타의 인터럽트 6개

ATmega128의 인터럽트 구성 (3)

인터럽트의 종류 및 인터럽트 벡터

벡터 번호 (우선 순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
0	0x0000	RESET	외부 핀, 전원 투입 리셋, 저전압 검출 리셋, 워치독 리셋, JTAG AVR 리셋
1	0x0002	INT0	외부 인터럽트 0
2	0x0004	INT1	외부 인터럽트 1
3	0x0006	INT2	외부 인터럽트 2
4	0x0008	INT3	외부 인터럽트 3
5	0x000A	INT4	외부 인터럽트 4
6	0x000C	INT5	외부 인터럽트 5
7	0x000E	INT6	외부 인터럽트 6
8	0x0010	INT7	외부 인터럽트 7
9	0x0012	TIMER2 COMP	타이머/카운터2 비교 일치
10	0x0014	TIMER2 OVF	타이머/카운터2 오버플로우
11	0x0016	TIMER1 CAPT	타이머/카운터1 입력 캡처
12	0x0018	TIMER1 COMPA	타이머/카운터1 비교 일치 A
13	0x001A	TIMER1 COMPB	타이머/카운터1 비교 일치 B
14	0x001C	TIMER1 OVF	타이머/카운터1 오버플로우
15	0x001E	TIMER0 COMP	타이머/카운터0 비교 일치

ATmega128의 인터럽트 구성 (4)

인터럽트의 종류 및 인터럽트 벡터

벡터 번호 (우선 순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
16	0x0020	TIMER0 OVF	타이머/카운터0 오버플로우
17	0x0022	SPI, STC	SPI 시리얼 통신 완료
18	0x0024	USART0, RX	USART0, 수신 완료
19	0x0026	USART0, UDRE	USART0, 데이터 레지스터 비움
20	0x0028	USART0, TX	USART0, 송신 완료
21	0x002A	ADC	ADC 변환 완료
22	0x002C	EE READY	EEPROM 준비
23	0x002E	ANALOG COMP	아날로그 비교기
24	0x0030	TIMER1 COMPC	타이머/카운터1 비교 일치 C
25	0x0032	TIMER3 CAPT	타이머/카운터3 입력 캡처
26	0x0034	TIMER3 COMPA	타이머/카운터3 비교 일치 A
27	0x0036	TIMER3 COMPB	타이머/카운터3 비교 일치 B
28	0x0038	TIMER3 COMPC	타이머/카운터3 비교 일치 C
29	0x003A	TIMER3 OVF	타이머/카운터3 오버플로우
30	0x003C	USART1, RX	USART1, 수신 완료
31	0x003E	USART1, UDRE	USART1, 데이터 레지스터 비움
32	0x0040	USART1, TX	USART1, 송신 완료
33	0x0042	TWI	I2C 통신 인터페이스
34	0x0044	SPM READY	저장 프로그램 메모리 준비

ATmega128의 인터럽트의 동작

- 인터럽트가 발생하면 관련 플래그 비트를 1로 세트하여 트리거 시키는 형태
  - ✓ 프로그램 카운터는 실제 인터럽트 벡터로 지정되어 인터럽트 처리 루틴으로 변경되어 ISR을 수행
  - ✓ 해당 플래그는 하드웨어에 의해 자동으로 0으로 클리어됨.
  - ✓ 인터럽트 플래그는 해당 비트에 1을 써 넣음으로써 0으로 클리어할 수 있음.
  - ✓ 인터럽트 마스크 레지스터 또는 SREG 레지스터를 통해 금지 상태로 설정하여 놓았더라도 인터럽트가 발생하면 해당 인터럽트 플래그가 1로 설정되어 인터럽트 대기 상태로 되며, 나중에 인터럽트가 허가상태로 설정될 때 해당 인터럽트가 처리됨.
- 인터럽트 조건이 발생한 동안에만 인터럽트를 트리거 하는 형태
  - 인터럽트 발생조건이 사라지면 인터럽트 요청도 없어지므로 나중에 인터럽트가 다시 허용 상태로 되더라도 인터럽트는 요청되지 않음.

ATmega128의 인터럽트 구성

리셋 및 인터럽트 벡터의 배치

APP 영역

Boot 영역

BOOTRST	IVSEL	리셋 벡터 주소	인터럽트 벡터의 시작 주소
1	0	0x0000	0x0002
1	1	0x0000	부트 리셋 주소 + 0x0002
0	0	부트 리셋 주소	0x0002
0	1	부트 리셋 주소	부트 리셋 주소 + 0x0002

부트리셋주소:
부트사이즈(BOOTSZ1-0) 비트에 의해 설정되며, 이에 따라 부트로더영역 주소가 달라짐.

- **리셋 및 인터럽트 벡터 : BOOTRST와 IVSEL 비트의 조합에 의해 가변적임.**
- **교육용 보드에서는 응용 영역에서 동작하여야 하므로 BOOTRST 비트는 1로 설정되고, IVSEL은 0으로 설정**
  - **BOOTRST 비트는 퓨즈비트에서 설정하고, IVSEL은 MCUCR에서 설정함.**

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	

  - **IVSEL (인터럽트 벡터 선택, Interrupt Vector Select)**
    - ✓ **IVSEL = 0 : 인터럽트 벡터를 플래시 메모리의 응용 프로그램 섹션에 배치**
    - ✓ **IVSEL = 1 : 인터럽트 벡터를 플래시 메모리의 부트 로더 섹션에 배치**
  - **IVCE (인터럽트 벡터 변경 허가, Interrupt Vector Change Enable)**
    - ✓ **IVSEL 비트의 변경을 허가하기 위해서 IVCE 비트는 1로 설정되어 있어야 함.**

ATmega128의 인터럽트 구성

인터럽트 벡터의 배치

- 인터럽트 벡터를 응용 프로그램 섹션과 부트 로더 섹션 사이에서 이동하기 위해서는 MCU 컨트롤 레지스터(MCUCR, MCU Control Register)를 사용
- MCUCR 레지스터의 비트 구성은 아래와 같으며, 여기서 IVSEL과 IVCE 비트가 이러한 목적으로 사용되고, 나머지는 외부 인터럽트를 개별적으로 허가하는 용도로 사용

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	

  - IVSEL과 IVCE 비트를 사용
  - IVSEL (인터럽트 벡터 선택, Interrupt Vector Select)
    - IVSEL = 0 : 인터럽트 벡터는 응용 프로그램 섹션인 플래시 메모리의 시작 부분에 위치
    - IVSEL = 1 : 인터럽트 벡터는 부트 로더 섹션의 시작 부분에 위치
  - IVCE (인터럽트 벡터 변경 허가, Interrupt Vector Change Enable)
    - IVSEL 비트의 변경을 허가하기 위해서 IVCE 비트는 1로 설정되어 있어야 함.

외부 인터럽트의 개요

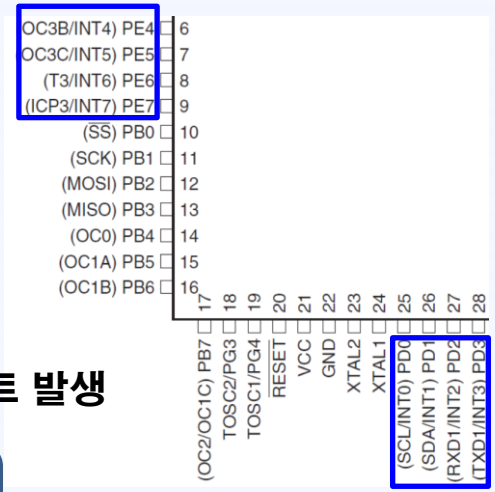
- 8개의 외부 인터럽트 입력 : INT0~7
  - 상승/하강 에지 및 Low 상태에 의해 인터럽트 발생
  - 외부 인터럽트의 트리거 방법은 EICRA(INT3~0)와 EICRB(INT7~4)에 의해 결정
- 레벨 변화 방식 : 해당 핀에 Low상태가 유지되는 동안에 인터럽트 발생(비동기적)
- 에지 트리거 방식 : I/O클럭에 동기를 맞추어 인터럽트가 발생
- INT7~4 인터럽트가 에지 트리거 방식에 의해 설정되면 이는 I/O 클럭을 필요로 하므로 이것들은 I/O 클럭이 차단되는 휴면 모드 이외의 슬립 모드에서는 슬립 모드를 해제하는 수단으로 사용할 수 없음.
- 모든 Low 레벨 인터럽트와 INT3~0 인터럽트가 하강 또는 상승 에지 트리거 방식으로 설정된 경우에는 인터럽트가 클럭에 상관없이 비동기적으로 검출되므로 슬립모드를 해제하는 수단으로 사용할 수 있음.
- 레벨 변화 방식으로 사용되는 인터럽트가 전원 차단 모드의 해제 수단으로 사용되는 경우에는 좀 더 긴 인터럽트 신호가 요구됨. -> 슬립 모드를 해제하고 인터럽트가 발생되려면 충분히 긴 시간동안 인터럽트 신호가 L 상태로 입력되어야 함.

ATmega128의 외부 인터럽트

한국공학대학교 TECH UNIVERSITY OF KOREA

외부 인터럽트 핀의 구성

- 8개의 외부 인터럽트 입력 : INT0~7



외부 인터럽트의 개요

- 외부 인터럽트(INT0~7) 입력의 동작 :
  - 상승/하강 에지 및 Low 상태에 의해 인터럽트 발생

- 에지 트리거 방식(I/O 클럭에 동기)

Low Level • 레벨 트리거 방식(비동기)

- INT0~INT3과 INT4~INT7의 차이점(슬립모드)
  - ➢ INT0~INT3 : (슬립모드에서의 차이)
    - 인터럽트의 모든 트리거 방식으로 설정되더라도 클럭에 관계없이 비동기적으로 검출됨. 따라서 슬립모드를 해제하는 수단으로 사용할 수 있음.
  - ➢ INT4~INT7 :
    - 에지 트리거 방식으로 설정되면 이는 I/O 클럭을 필요로 하므로, I/O 클럭이 차단되는 휴면 모드 이외의 슬립 모드에서는 슬립 모드를 해제하는 수단으로 사용할 수 없음.
    - 레벨 변화 방식으로 사용되는 인터럽트가 전원 차단 모드의 해제 수단으로 사용되는 경우에는 좀 더 긴 인터럽트 신호가 요구됨. -> 충분히 긴 시간동안 인터럽트 신호가 L 상태로 입력되어야 함.

ATmega128의 인터럽트 구성

한국공학대학교 TECH UNIVERSITY OF KOREA

외부 인터럽트 제어 레지스터

외부 인터럽트 레지스터	설 명	
EIMSK	외부 인터럽트 마스크 레지스터	인터럽트 허가
EIFR	외부 인터럽트 플래그 레지스터	인터럽트 발생여부
EICRA	외부 인터럽트 트리거 방식 설정 레지스터(INT0~3)	트리거방식
EICRB	외부 인터럽트 트리거 방식 설정 레지스터(INT4~7)	트리거방식

EIMSK 제어 레지스터

- **외부 인터럽트 마스크 레지스터 EIMSK(External Interrupt Mask Regiter)는 인터럽트 INT7~0을 개별적으로 허가하는데 사용, 1로 설정 시 인터럽트 허용, 0으로 설정 시 인터럽트 금지 <SREG레지스터의 글로벌 인터럽트 허용 비트 I가 1로 되어야만 실제 허용가능>**

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	

Group B Group A

- 외부 인터럽트 제어용 레지스터 -

전체 인터럽트의 허가 및 금지 방법

- 인터럽트를 사용하기 위해서는 전체 인터럽트 허가 비트(SREG의 I 비트)를 1로 설정 → SREG 레지스터의 의 I를 이용하여 전체 인터럽트를 허가함.
  - SREG |= 0x80; // SREG의 7비트를 1로 설정하여 전체 인터럽트를 허가함.
  - 그러나, Atmel Studio(ver 7.xx 이상) 부터 해당 방법 보다 sei(), cli() 함수를 이용한 전체 인터럽트 허용 및 금지 방법을 추천하고 있음.
  - Interrupt 기능을 사용하기 위해서는 다음과 같이 헤더를 추가 해야함.

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

- sei()와 cli() 함수의 사용 목적
  - sei(); // 전체 인터럽트 허가, 전역 인터럽트 플래그 비트 셋
  - cli(); // 전체 인터럽트 금지, 전역 인터럽트 플래그 비트 클리어

상태 레지스터(SREG)

- 최근에 실행된 산술 연산의 명령어 처리 결과를 알려 주는 레지스터
  - 조건부 처리 명령에 의해 프로그램의 흐름을 변경하는데 사용
- 인터럽트의 허가/금지 기능 비트(I-bit) : Global Interrupt Enable

비트	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

- Bit 7 (I) : 전체 인터럽트 허가 (Global Interrupt Enable)
- Bit 6 (T) : 비트 복사 저장 (Bit Copy Storage)
- Bit 5 (H) : 보조 캐리 플래그 (Half Carry Flag)
- Bit 4 (S) : 부호 비트 (Sign Bit)
- Bit 3 (V) : 2의 보수 오버플로우 비트 (two's complement overflow bit)
- Bit 2 (N) : 음의 플래그(Negative Flag)
- Bit 1 (Z) : 영 플래그(Zero Flag)
- Bit 0 (C) : 캐리 플래그 (Carry Flag)

Microchip Studio를 이용한 ISR의 작성

인터럽트 서비스 루틴(ISR) 작성 방법

- ISR 이란 ? 인터럽트가 발생하면 현재 수행 중인 프로그램을 정지하고 수행 해야 되는 일(기능) 이를 ISR(Interrupt Service Routine) 이라고 부름.
- ATmega128내 인터럽트 시스템을 사용하기 위한 ISR 함수를 작성은 아래와 같음.

- 인터럽트 함수에는 입력 및 출력 인자를 사용할 수 없음.

```
#include <avr/interrupt.h>
ISR(PCINT1_vect)
{
        .....          // ISR의 프로그램 코드
}
```

- 인터럽트 벡터명의 벡터번호 매크로는 iom128.h에서 정의됨.

- ISR : interrupt 서비스 함수.
- 인터럽트 벡터 ➡ 인터럽트 주소 생성, n=1~35

- 컴파일러에서 인터럽트 벡터와 목록(entry)과 탈출코드(exit code)를 자동으로 생성함.

ATmega128의 인터럽트 구성

한국공학대학교 TECH UNIVERSITY OF KOREA

EIFR 제어 레지스터 External Interrupt Flag Register

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	

- 외부 인터럽트 플래그 레지스터 EIFR(External Interrupt Flag Register)는 INT7~0핀에 인터럽트 신호가 입력되어, 해당 인터럽트가 트리거 되었음을 표시하는데 사용한다.
- 이 비트들은 인터럽트 처리가 시작되고 마이크로 콘트롤러가 인터럽트 벡터를 인출하여 인터럽트 서비스 루틴으로 점프하게 되면 다시 0으로 클리어 된다.
- 강제로 0을 클리어 하려면, 해당 비트에 1을 라이트 하면 된다.

ATmega128의 인터럽트 구성

한국공학대학교 TECH UNIVERSITY OF KOREA

EICRA 제어 레지스터 — External Interrupt Control Register A

- 외부 인터럽트 INT3~0 핀으로 입력되는 신호에 대한 인터럽트 트리거 방법을 설정
- 모든 레벨 트리거 인터럽트와 INT3~00이 하강 또는 상승 에지 트리거 방식으로 설정시 인터럽트가 클럭 신호와 관계없이 비동기적으로 검출, 슬립모드를 해제하는 수단으로 사용 가능.

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	

ISCn1	ISCn0	인터럽트 발생 방식
0	0	INTn 핀의 L상태 입력이 인터럽트를 트리거 한다.
0	1	사용하지 않음(Reserved)
1	0	INTn 핀에 하강 에지의 신호가 입력 시 비동기적으로 트리거
1	1	INTn 핀에 상승 에지의 신호가 입력 시 비동기적으로 트리거

ATmega128의 인터럽트 구성

한국공학대학교 TECH UNIVERSITY OF KOREA

EICRB 제어 레지스터 — External Interrupt Control Register B

- **외부 인터럽트 INT7~4 핀으로 입력되는 신호에 대한 인터럽트 트리거 방법을 설정**
- **INT7~40이 에지 트리거 방식으로 설정 시, I/O클럭이 필요하게 되므로 I/O 클럭이 차단 되는 IDLE 모드 이외의 슬립모드에서는 슬립모드를 해제하는 수단으로 사용 불가**

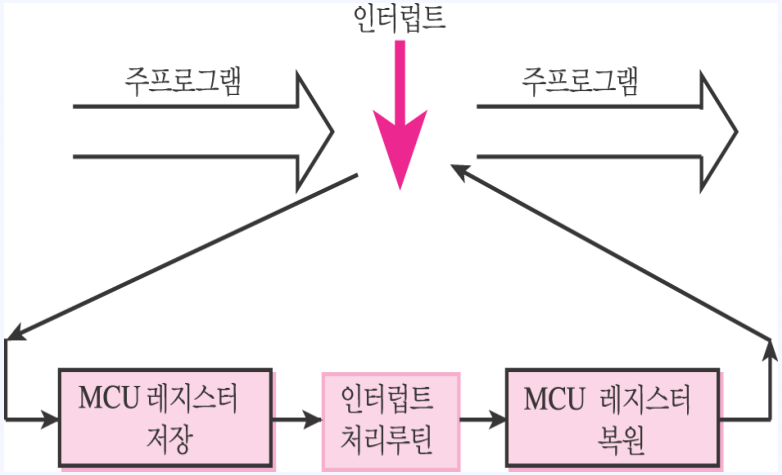
Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISCn1	ISCn0	인터럽트 발생 방식
0	0	INTn 핀의 L상태 입력이 인터럽트를 트리거 한다.
0	1	INTn 핀의 하강 에지 또는 상승 에지가 인터럽트를 트리거한다.
1	0	INTn 핀에 하강 에지의 신호가 인터럽트를 트리거
1	1	INTn 핀에 상승 에지의 신호가 인터럽트를 트리거

기호	파라메터	Min.	Typ.	Max.	Units
t_{INT}	비동기 외부 인터럽트에 대한 최소의 펄스 폭		50		ns

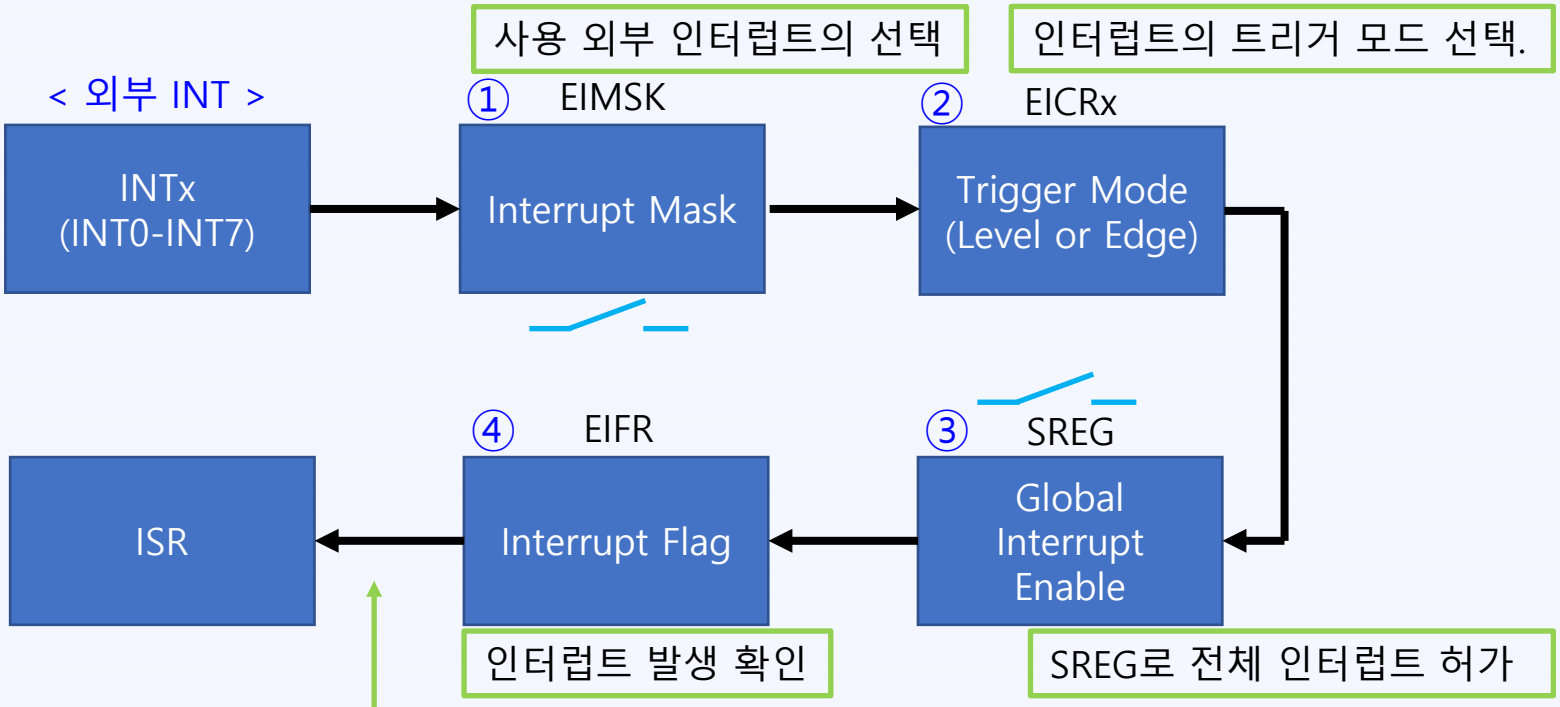
- **에지 트리거로 사용되는 인터럽트 신호의 최소 신호 폭은 50ns이상 이어야 함.**

ATmega128의 인터럽트 처리 메카니즘



- ATmega128에서의 인터럽트 처리는 정해진 우선순위에 의해 처리
- ATmega128에서는 여러 인터럽트기 동시에 발생하였을 때 우선순위가 높은 인터럽트가 먼저 처리됨. 우선순위는 벡터 no.가 고정되어 변경이 불가능
- 인터럽트가 발생
  - 해당 인터럽트 플래그 비트를 설정-> interrupt request
  - SREG=1로 설정되어 있으면, interrupt 발생하여 벡터 주소를 참조하여 ISR 함수 수행함.
- ISR이 수행되면, SREG=0으로 클리어하여 다른 모든 인터럽트의 요청을 불허함. -> ISR이 종료되면서 인터럽트를 허가함.
- ATmega128이 ISR 함수를 수행하고 난후에 주프로그램으로 복귀하는 과정은 최소 4 클럭 사이클이 소요됨. -> 이 시간동안에 PC의 값이 스택으로부터 복구됨

ATmega128의 인터럽트 처리 순서



- ①~③ 과정을 통해 설정된 외부 인터럽트 소스에 대해 인터럽트가 발생하면, EIFR의 해당 비트가 1로 설정되어 ISR을 실행하게 됨.

ATmega 128의 인터럽트 처리 메카니즘

- ATmega128에서의 인터럽트 처리는 정해진 우선순위에 의해 처리
- ATmega128에서는 여러 인터럽트기 동시에 발생하였을 때 우선순위가 높은 인터럽트가 먼저 처리되고, 이 우선순위는 변경이 불가능
- 인터럽트가 발생하면 인터럽트에 해당하는 플래그 비트가 세트, 이 플래그 비트에 의해 인터럽트가 요청, 전체 인터럽트 허가 비트 I와 해당 인터럽트 허가 비트가 모두 1로 설정되어 있으면, 인터럽트가 요청되어 인터럽트 벡터의 주소를 찾아가 인터럽트 서비스 루틴(ISR)을 수행하게됨
- ISR이 수행되고 있을 때, ATmega128은 자동적으로 전체 인터럽트 허가 비트(SREG의 I 비트)를 클리어 모든 인터럽트의 발생을 금지 서비스 루틴의 종료와 함께 인터럽트를 허용
- ATmega1280이 RETI 명령에 의하여 ISR의 실행을 마치고 주프로그램으로 복귀하는 데에도 4 클럭 사이클이 소요된다. 이 시간동안에 PC의 값이 스택으로부터 복구됨

Code Vision을 이용한 ISR의 작성

외부 인터럽트의 ISR의 작성

- 인터럽트의 서비스는 벡터 주소라는 고유 번지에서 시작(iom128.h에 정의됨.)
- 인터럽트 벡터에는 인터럽트 기능을 서비스 하기 위한 프로그램의 시작 번지가 저장되어 있음.
- ISR은 인터럽트 함수로 선언하면 컴파일러에서 자동으로 인터럽트 번호를 참고하여 인터럽트 벡터에 주소를 저장함.

인터럽트 서비스 함수의 선언 : ISR(Interrupt_N){ /*… Interrupt Code …*/ }

```
#define INT0_vect          1  // External Interrupt Request 0, _VECTOR(1)
#define INT1_vect          2  // External Interrupt Request 1, _VECTOR(2)
#define INT2_vect          3  // External Interrupt Request 2, _VECTOR(3)
#define INT3_vect          4  // External Interrupt Request 3, _VECTOR(4)
#define INT4_vect          5  // External Interrupt Request 4, _VECTOR(5)
#define INT5_vect          6  // External Interrupt Request 5, _VECTOR(6)
#define INT6_vect          7  // External Interrupt Request 6, _VECTOR(7)
#define INT7_vect          8  // External Interrupt Request 7, _VECTOR(8)
#define TIMER2_COMP_vect   9  // Timer/Counter2 Compare Match, VECTOR(9)
#define TIMER2_OVF_vect    10 // Timer/Counter2 Overflow, VECTOR(10)
#define TIMER1_CAPT_vect   11 // Timer/Counter1 Capture Even, _VECTOR(11)
#define TIMER1_COMPA_vect  12 // Timer/Counter1 Compare Match A, _VECTOR(12)
#define TIMER1_COMPB_vect  13 // Timer/Counter1 Compare Match B, _VECTOR(13)
```

인터럽트 벡터 번호 정의(계속)

```
#define TIMER1_OVF_vect      14  // Timer/Counter1 Overflow, _VECTOR(14)
#define TIMER0_COMP_vect     15  // Timer/Counter0 Compare Match, _VECTOR(15)
#define TIMER0_OVF_vect      16  // Timer/Counter0 Overflow, _VECTOR(16)
#define SPI_STC_vect         17  // SPI Serial Transfer Complete, _VECTOR(17)
#define USART0_RX_vect       18  // USART0, Rx Complete, _VECTOR(18)
#define USART0_UDRE_vect     19  // USART0 Data Register Empty, _VECTOR(19)
#define USART0_TX_vect       20  // USART0, Tx Complete, _VECTOR(20)
#define ADC_vect             21  // ADC Conversion Complete, _VECTOR(21)
#define EE_READY_vect        22  // EEPROM Ready, _VECTOR(22)
#define ANALOG_COMP_vect     23  // Analog Comparator, _VECTOR(23)
#define TIMER1_COMPC_vect    24  // Timer/Counter1 Compare Match C, _VECTOR(24)
#define TIMER3_CAPT_vect     25  // Timer/Counter3 Capture Event, _VECTOR(25)
#define TIMER3_COMPA_vect    26  // Timer/Counter3 Compare Match A, _VECTOR(26)
#define TIMER3_COMPB_vect    27  // Timer/Counter3 Compare Match B, _VECTOR(27)
#define TIMER3_COMPC_vect    28  // Timer/Counter3 Compare Match C, _VECTOR(28)
#define TIMER3_OVF_vect      29  // Timer/Counter3 Overflow, _VECTOR(29)
#define USART1_RX_vect       30  // USART1, Rx Complete, _VECTOR(30)
#define USART1_UDRE_vect     31  // USART1 Data Register Empty, _VECTOR(31)
#define USART1_TX_vect       32  // USART1, Tx Complete, _VECTOR(32)
#define TWI_vect             33  // 2-wire Serial Interface, _VECTOR(33)
#define SPM_READY_vect       34  // Store Program Memory Read, _VECTOR(34)
```

ISR의 작성

- 인터럽트의 서비스는 벡터 주소라는 교유 번지에서 시작
- 인터럽트 벡터에는 인터럽트 기능을 서비스 하기 위한 프로그램이 위치해 있어야 함
- 인터럽트 서비스 루틴이 호출되기 위해서는 C 언어에서 인터럽트 서비스 루틴이 올바르게 선언되어 있어야 한다.

  인터럽트 서비스 루틴의 선언 : interrupt [n] void int_func_name (void)

- 타이머 0의 오버플로우 인터럽트에 대한 인터럽트 서비스 루틴의 작성 예

```
// Called automatically on TIMER0 overflow
unsigned int interrupt_cnt;
unsigned char second;
interrupt [0x0020] void timer0_overflow(void)
{
        if (++interrupt_cnt == 4000){           // count to 4000
                second++;                       // second counter
                Interrupt_cnt = 0;   // clear int counter
        }
}
```

인터럽트를 이용한 실험

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	

- EICRA 레지스터는 그림과 같은 비트 구성을 가지고 있으므로 각 비트 명에 대해 다음과 같이 선언

```
#define ISC00 0
#define ISC01 1
#define ISC10 2
#define ISC11 3
.....
```

필요시, 개발자가 편한 명칭으로 매크로 정의

avr/iom128.h 에 정의된 형태를 써도 됨.

- ISC01 비트만 1로 설정하기 위한 과정 → 왼쪽 시프트연산자 “<<” 를 사용
  1<<ISC01
  - ISC01은 1이므로 1<<1 // 0b00000001을 1비트 시프트 → 0b00000010
  - 최종적으로 MCUCR의 ISC01 비트가 세트됨

인터럽트를 이용한 실험

- EICRA 레지스터의 ISC01과 ISC00을 세트하는 프로그램

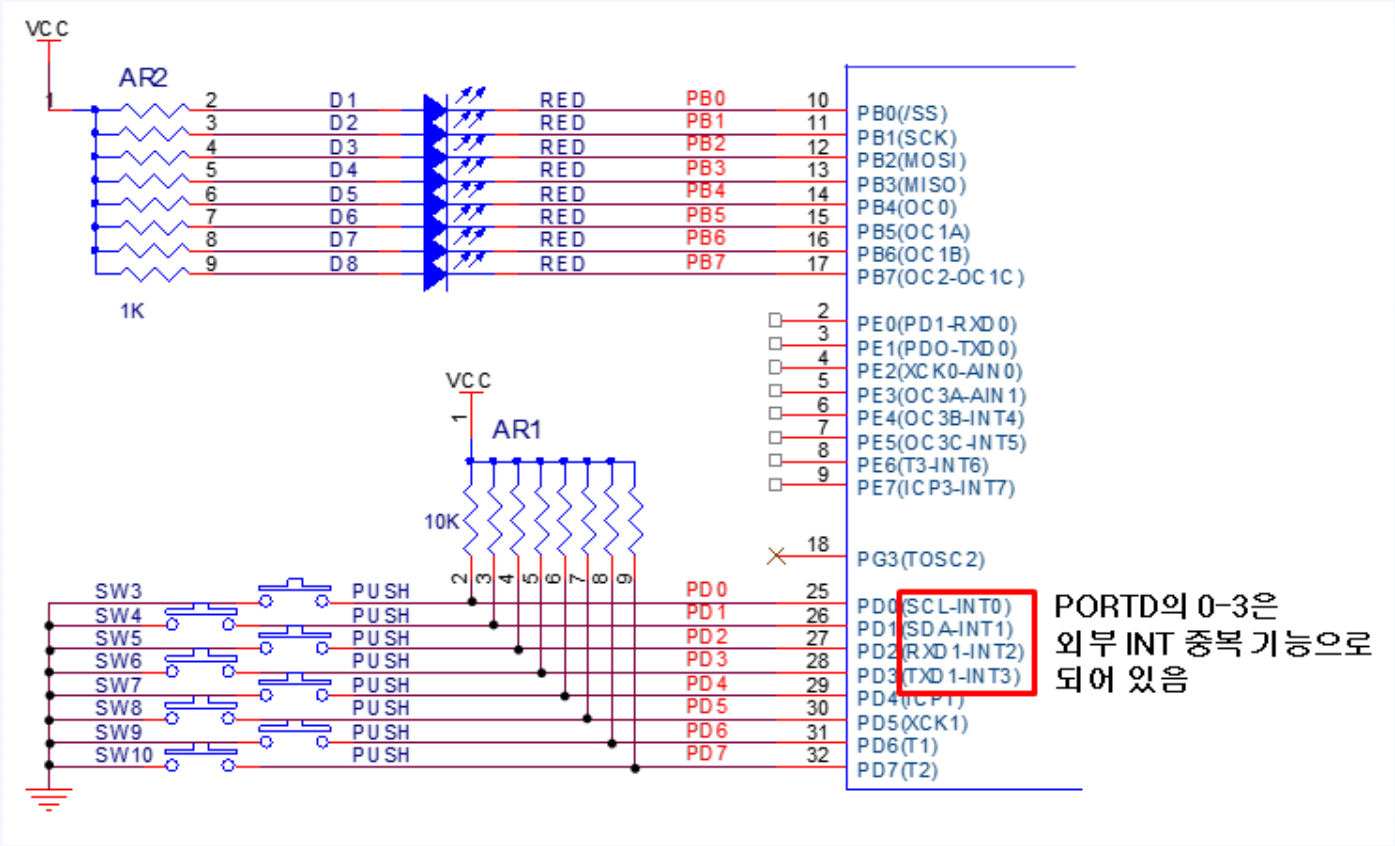
  EICRA = 1<<ISC01 | 1<<ISC00; // rising edge

- 이상의 비트 제어를 통해 인터럽트 초기화 함수 Interrupt_init()의 프로그램을 재작성

```
void Interrupt_init(void)
{
        //cli();

        EIMSK = 0x01;
        EICRA = 1<<ISC01 | 1<<ISC00;
        sei();
}
```

인터럽트를 이용한 실험

외부 인터럽트의 실험



< 실험 회로 >

// 프로그램 예 : INT0을 이용하여 PORTB의 LED를 On/Off 프로그램 작성
// 한번 눌리면 전체 LED On, 다시 눌리면 전체 LED Off
// PORTB : LED 8개 연결 , PORTD0-PORTD3 : INT0-INT3 연결

```
#define F_CPU 14.7456E6
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>


unsigned char Key_sts = 0;

ISR[INT0_vect]
{
    Key_sts = ~Key_sts;
    // Key_sts 변수를 이용해 눌릴 때 마다 반전함
}

void Interrupt_init(void)
{
    DDRB = 0xff;      // PORTB를 출력으로 설정
    PORTB = 0xff;     //  모든 I/O 출력 상태 'H' LED 꺼짐.
    EIMSK = 0x01;     // ① 외부인터럽트 종류(INT0) 선택
    EICRA = 0x02;     // ② 인터럽트 트리거 방식 선택 :
                      // INT0 하강에지 트리거
    SREG |= 0x80;     // ③ 전체 인터럽트 허가

    DDRD = 0x00;  // PORTD는 I/O로 사용시 입력으로
                  // 설정되어야 하나, 초기값이 0으로 되어 있어 생략하여도 됨.
}
```

```
void main()
{
    Interrupt_init();      // Init. INT & PORTB

    while(1)
    {
        if(Key_sts) { PORTB = 0x00; }    // LED all on
            else { PORTB = 0xff; }       // LED all off
    }
}
```

인터럽트를 이용한 실험

폴링

1. 폴링 방식의 프로그램과 인터럽트 프로그램의 차이점

- PORTD의 스위치의 누름을 계수하여 PORTB의 LED에 카운트된 값을 출력하는 프로그램 작성

```
#define F_CPU 14.7456E6
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

typedef unsigned char Byte;

bit key7=0;
Byte count=0, change=0;

Byte Exch(void)
{
      while(1)
      {
          key7 = PIND.0;
                if(key7 == 0)
                  {
                     count++;
                     delay_ms(1000);
                     return 1;
                   }
      }
}
```

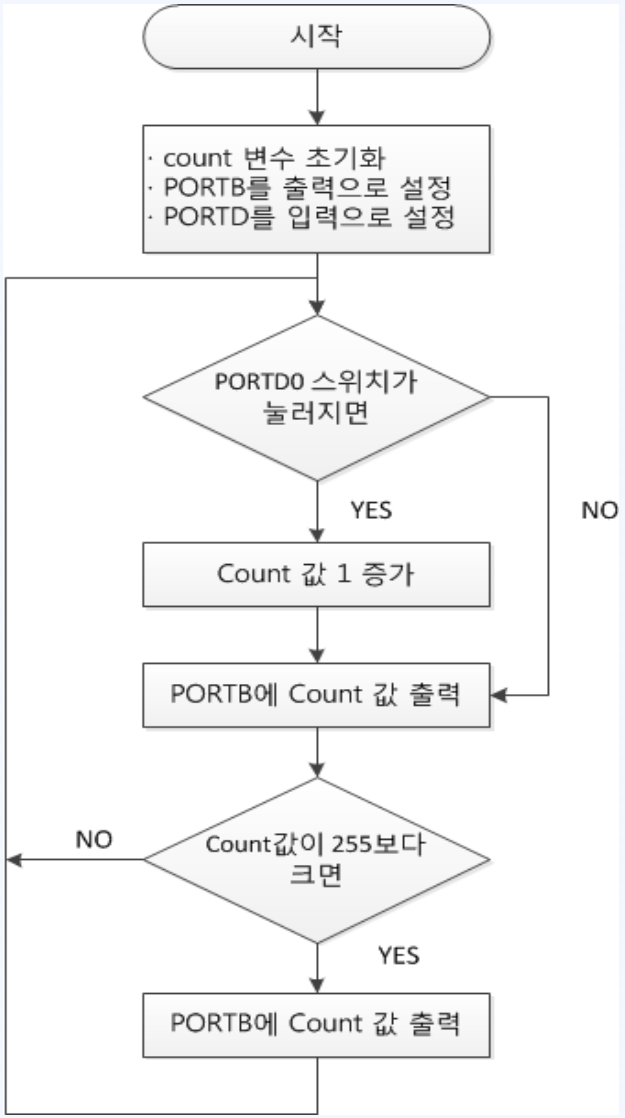
```
void main(void)
{
          // count 변수의 선언 및 초기화
          Count = 0; change=0;

          // 포트 D를 출력으로 설정
          // 포트 B의 LED를 모두 OFF
          DDRB = 0xFF;
          DDRD = 0x00;
          PORTB = 0xFF;

   while(1)
   {
            key7 = PIND.0;

                if(change == 0 && key7 == 0)
                        change = Exch();
                else if(change == 1 && key7 == 1)
                        change = 0;
                else;

                   PORTB = ~(count&0xFF);
                   If(count>= 255) count = 0;
   }
}
```

인터럽트를 이용한 실험

- **PORTD.0(INT0핀)의 스위치의 신호를 계수하여 PORTB의 LED에 카운트된 값을 출력하는 프로그램 작성**

- 폴링 방식(Polling 방식)
  - ✓프로그램에서 입력되는 값을 계속 읽어서 변화되는 상황을 감지
  - ✓모든 경우의 입력(신호의 변화)에 대응하여 처리 가능
  - ✓주기적으로 계속 검사

- 인터럽트 방식
  - ✓MCU가 스스로 하드웨어의 변화를 확인하여 변화시에만 대응하여 처리 가능
  - ✓하드웨어적으로 지원되는 제한된 개수의 입력에만 대응하여 처리



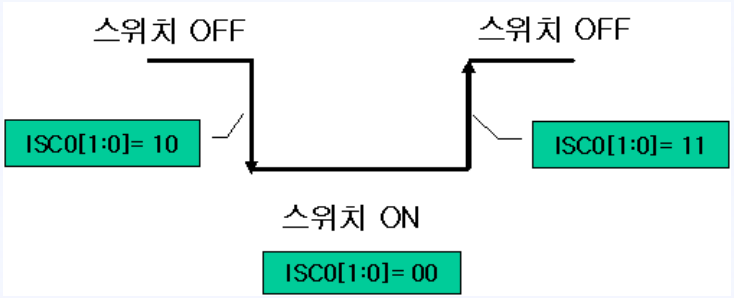
인터럽트를 이용한 실험

2. 외부 인터럽트 0 서비스 루틴의 작성

- **PORTD에 연결된 스위치를 폴링 방식이 아닌 인터럽트(INT0)로 계수하여 PORTB의 LED에 출력하는 프로그램 작성**

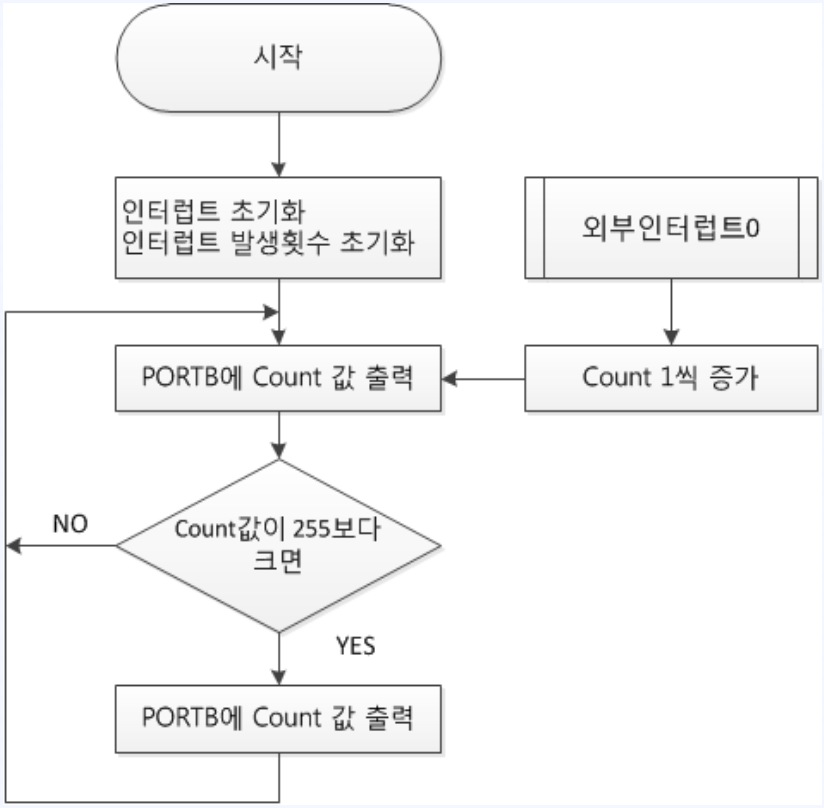
```
#define F_CPU 14.7456E6
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

unsigned char count;
ISR[INT0_vect]
{
    //외부인터럽트 발생 횟수를 계수
    count++;
}
void Interrupt_init(void)
{
    // ① 외부인터럽트 종류 선택
    EIMSK = 0x01;
    // ② 인터럽트 제어방법선택
    // (INT0 하강에지트리거)
    EICRA = 0x02;
    // ② 포트 B를 출력으로 선택
    DDRB = 0xff;
    // ② 포트 D를 입력으로 선택
    // DDRD = 0x00;
    // ③인터럽트 동작시작(전체인터럽트허가)
    SREG |= 0x80;
}
```

```
void main(void)
{
        // 외부인터럽트 발생 횟수 초기화
         count = 0;
         Interrupt_init();
         while(1){
             // 포트B에 출력(Active Low 동작하므로 반전)
             PORTB = ~count;
             if(count >= 255) count = 0;
         }
}
```



< 스위치 동작에 따른 인터럽트 모드 설정 >

인터럽트를 이용한 실험



- **예제 1과 예제 2와의 차이점**
- <예제 1>의 경우는 주프로그램에서 스위치의 입력 상태를 확인하기 위해 대부분의 마이크로컨트롤러의 시간을 소모함.
- <예제 2>의 경우에는 스위치의 입력 상태는 비동기적으로 스위치의 입력이 발생할 때마다 자동적으로 감지되고, 감지된 결과에 대한 처리를 모두 ISR내에서 수행하여 주프로그램에서는 포트 출력만 하면 됨.
- 따라서, 인터럽트를 사용하면 대부분의 마이크로컨트롤러의 수행 시간을 주프로그램에서 처리하고, 비동기적으로 일어나는 사건에 대해서는 사건이 발생할 때마다 처리할 수 있으므로 인터럽트를 사용하는 것이 효율적인 것을 알 수 있음.

Report

- **INT0~INT3을 이용하여 다음의 동작을 수행하는 프로그램을 작성하시오.**

INT 핀	동 작	트리거 방식	비고
INT0	PORTB의 하위 니블의 LED를 PORTB.3→PORTB.0의 순서로 우로 시프트하는 동작	상승 에지	
INT1	PORTB의 하위 니블의 LED를 PORTB.0→PORTB.3의 순서로 좌로 시프트하는 동작	하강 에지	
INT2	4digit FND에 문자가 모두 표기 되고 있는 도중 해당 버튼을 누를 때 마다 표기 숫자 1씩 증가	하강 에지	
INT3	4digit FND에 문자가 모두 표기 되고 있는 도중 해당 버튼을 누를 때 마다 표기 숫자 1씩 감소	상승 에지	

- **외부 인터럽트 기능과 폴링 방식으로 작성된 같은 동작기능의 코드 작성 원리 비교**

  외부 인터럽트를 사용 하지 않고 Port D의 2번 3번 I/O를 폴링으로 작성하여 상기 숫자 증가, 감소 프로그램을 작성 하여 코드 작성 원리를 비교 하시오.

Q & A