

9 장 : 스위치로 1/100 스톱워치 만들기

ATmega128 마이크로컨트롤러를 이용한 임베디드시스템 구현



JCnet
제이씨넷

신 상 석

목차

1. 스위치
2. JKIT-128-1에서의 스위치 연결 설계
3. 실습 SW-1 : 스위치로 LED 제어
4. 인터럽트
5. ATmega128 인터럽트
6. 실습 SW-2 : 스위치로 입장객 숫자 세기
7. 실습 SW-3 : 스위치로 1/100초 스톱워치 만들기

스위치(Switch)

□ 스위치 : 전기 회로를 끊거나 잇는 기구



Tactile 스위치



Push Button
스위치



Slide 스위치



DIP 스위치



Toggle 스위치



Locker 스위치



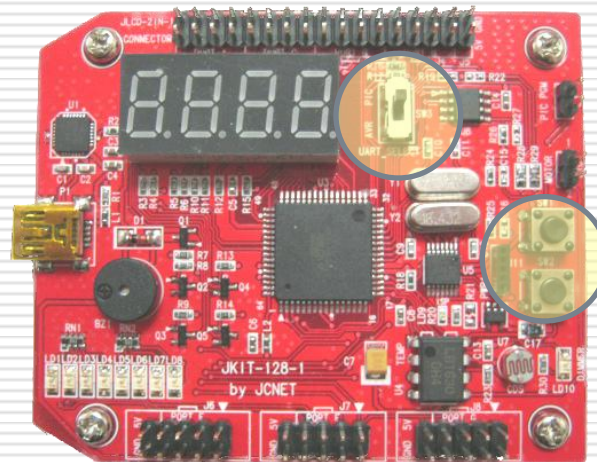
마이크로 스위치



로터리 스위치

스위치(Switch)

- 임베디드시스템에서 스위치의 이용
 - Tactile(Tact) 스위치 : 리셋, 인터럽트 등
 - Slide 스위치 : 전원 on/off, 모드 선택 등
 - Dip 스위치 : 다수의 모드 선택, 초기값 제공 등
 - Locker 스위치 : 전원 on/off 등



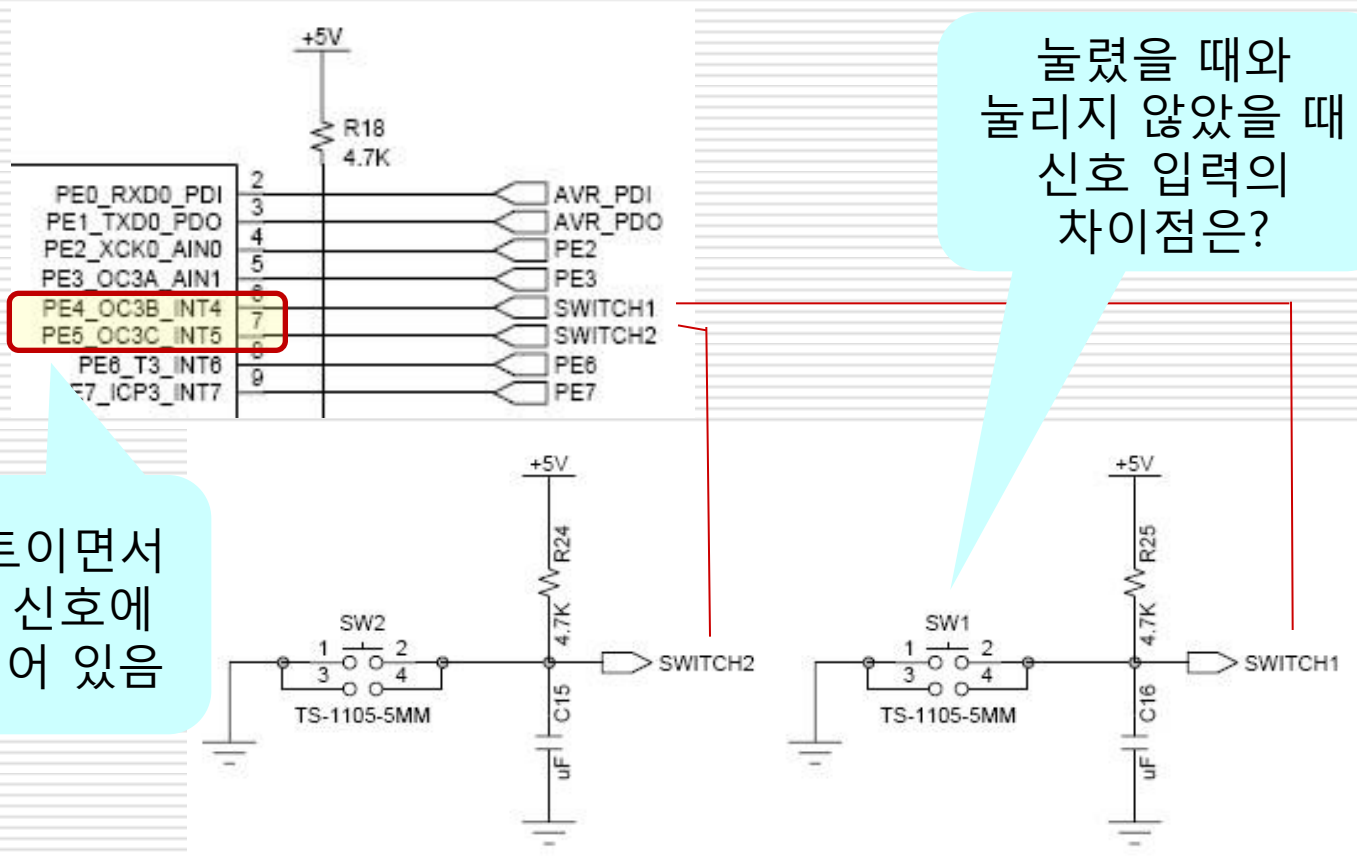
JKIT-128-1 에서의 스위치 연결 설계

□ JKIT-128-1에서의 스위치 연결 설계 개념

- 스위치는 공간의 제약으로 최소한의 개수인 2개만 배정하고 동일한 입출력포트에 할당
- 스위치는 인터럽트 시험용으로도 사용 가능하여야 하므로 ATmega128의 외부인터럽트(External Interrupt)를 사용할 수 있는 입출력 포트 PE(PE4, PE5)에 할당
- 스위치는 tactile 스위치를 사용하고, 크기가 작고 가격이 저렴한 것으로 하며, 스위치가 눌러지지 않았을 때는 '1', 눌러졌을 때는 '0'이 되도록 설계
- 실습용이므로 스위치 bouncing 현상을 방지하기 위한 대비 회로는 따로 고려하지 않음

JKIT-128-1 에서의 스위치 연결 설계

□ JKIT-128-1에서의 스위치 연결 설계



PE 포트이면서
INT인 신호에
연결되어 있음

눌렀을 때와
눌리지 않았을 때
신호 입력의
차이점은?

Switch Debouncing

□ Switch Bouncing

- 기계 스위치의 경우, 원하는 곳에 최종 접속되기 이전에 짧게 여러 번 연결되었다 끊어졌다를 반복하는 현상
- Chattering이라고 하며, 수 ms 이내에 사라짐

□ Switch Debouncing

- Bouncing 현상을 막아주는 회로 또는 방법
- Debouncing 처리 방법
 - 스위치의 양단에 커패시터를 달아주는 방법
 - RS 플립플롭을 사용하는 방법
 - 슈미트트리거 NAND 회로를 2개 연속 달아주는 방법
 - 소프트웨어로 처리하는 방법

실습 SW-1 : 스위치로 LED 제어

□ 실습 내용

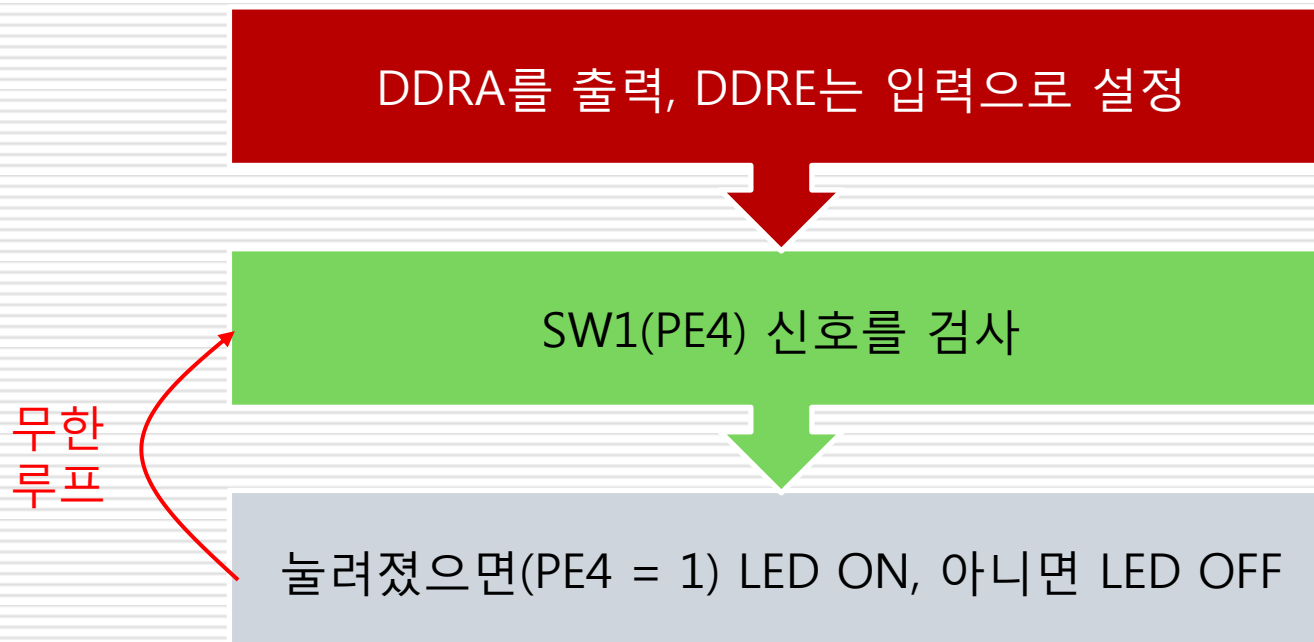
1. SW1로 모든 LED를 ON/OFF 제어하기 (SW1을 누르고 있는 동안만 ON)
2. SW1로 모든 LED를 ON/OFF 제어하기 (SW1 1번 누르면 ON, 다시 누르면 OFF - toggle) (함께 해보기)

실습 SW-1 : 스위치로 LED 제어

- 구동 프로그램 설계 : LED ON/OFF (sw_1_1.c)
 - SW1이 눌러졌는지를 체크하기 위하여는 SW이 연결되어 있는 포트를 입력으로 하고 이 신호의 레벨을 검사하면 됨
 - SW1이 눌러졌으면 입력값은 '0'이고 눌러지지 않았으면 입력값은 '1'임
 - SW1이 '0'이면 LED를 ON시키고, '1'이면 LED를 OFF하되, 이를 무한루프로 실행
 - 인터럽트는 아직 사용하지 않음 (세팅하지 않음)

실습 SW-1 : 스위치로 LED 제어

□ 구동 프로그램 설계 : LED ON/OFF (sw_1_1.c)



실습 SW-1 : 스위치로 LED 제어

□ 구동 프로그램 코딩 : LED ON/OFF (sw_1_1.c)

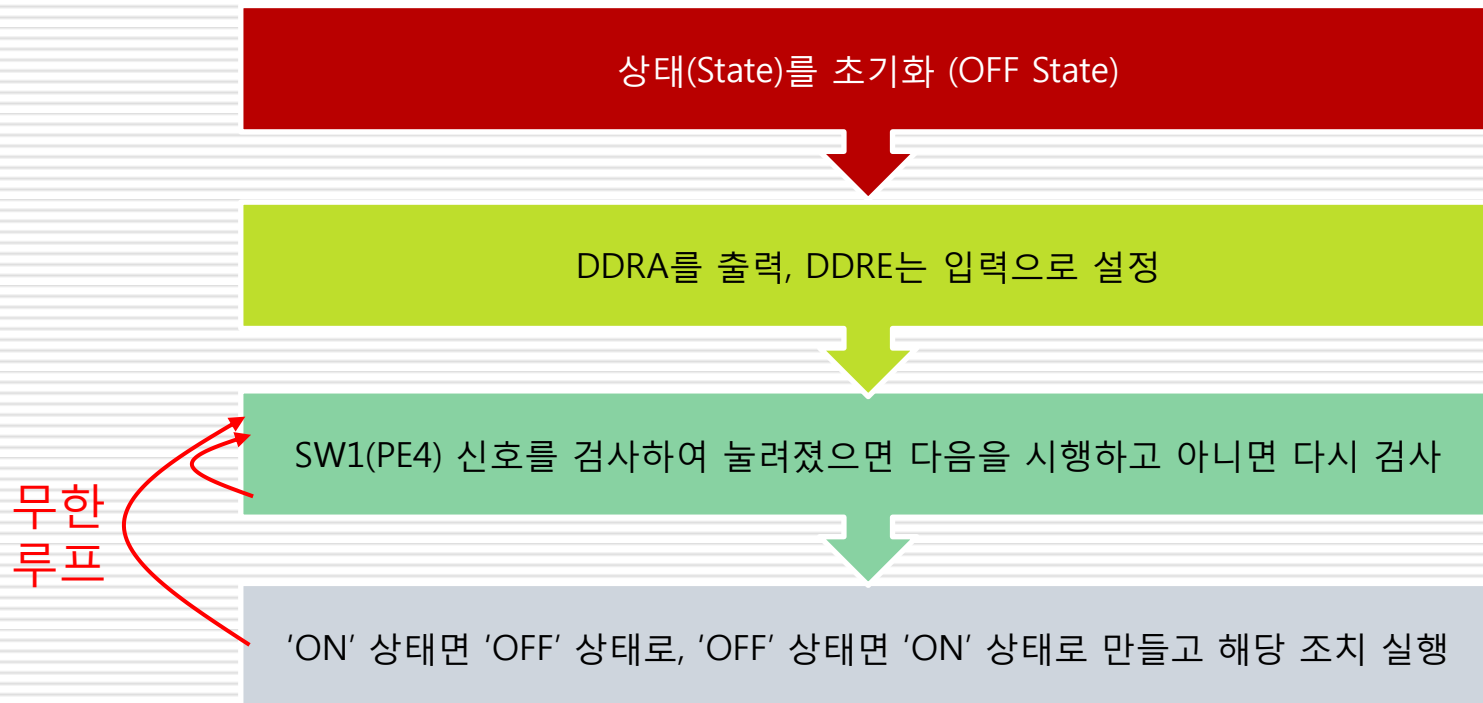
```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
int main(void)
{
    DDRA = 0xff;
    DDRE = 0x00;
    while (1)
    {
        if ((PINE & 0x10) == 0x00)           // SW1 = PE bit4
            PORTA = 0xff;                     // LED = 'ON'
        else
            PORTA = 0x00;                     // LED = 'OFF'
            _delay_ms(100);                   // debounce 무시
    }
}
```

실습 SW-1 : 스위치로 LED 제어

- 구동 프로그램 설계 : LED ON/OFF Toggle (sw_1_2.c)
 - SW1이 눌러졌는지를 체크하기 위하여는 SW이 연결되어 있는 포트를 입력으로 하고 이 신호의 레벨을 검사하면 됨
 - SW1이 눌러졌으면 입력값은 '0'이고 눌러지지 않았으면 입력값은 '1'임
 - SW1이 '0'이면 LED를 ON/OFF Toggle 시키고, '1'이면 현 상태를 유지하며, 이를 무한루프로 실행
 - Toggle 수행을 위하여 상태(State)의 개념 이해 필요
 - 여기서는 'ON' State와 'OFF' State로 정의
 - 인터럽트는 아직 사용하지 않음 (세팅하지 않음)

실습 SW-1 : 스위치로 LED 제어

□ 구동 프로그램 설계 : LED ON/OFF Toggle (sw_1_2.c)



실습 SW-1 : 스위치로 LED 제어

□ 구동 프로그램 코딩 : LED ON/OFF (led_1_2.c)

```
#include <avr/io.h>                // ATmega128 register 정의
#include <util/delay.h>

unsigned char data = 0x00;          // 초기 data값 = 0x00
DDRA = 0xff;                        // Port A 출력
DDRE = 0x00;                        // Port E 입력
while (1)
{
    if((PINE & 0x10) == 0x00) // SW1 검사
    {
        PORTA = ~data; // 데이터 Toggle
        _delay_ms(100); // bouncing 무시
    }
}
}
```

실습 SW-1 : 스위치로 LED 제어

□ 구동 프로그램 코딩 : LED ON/OFF (led_1_2.c)

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#define ON 1
#define OFF 0
int main( )
{
    int state = OFF;
    DDRA = 0xff; // Port A 출력
    DDRE = 0x00; // Port E 입력
    while (1)
    {
        // SW1 검사
        if ((PINE & 0x10) == 0x00)
```

```
    {
        if (state == OFF)
        {
            PORTA = 0xff;
            state = ON;
        }
        else
        {
            PORTA = 0x00;
            state = OFF;
        }
        _delay_ms(100);
    }
}
```

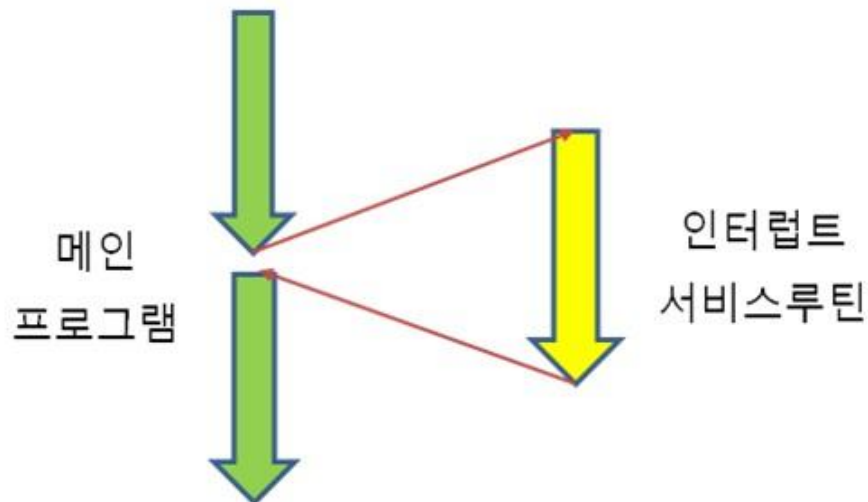
인터럽트(Interrupt)

- 인터럽트(interrupt)란?
 - <방해하다> < 훼방놓다>의 뜻
 - MCU에서의 인터럽트 사용 이유
 - 긴급하거나 불규칙적인 사건의 처리
 - 외부와의 인터페이스
 - 폴링(polling)으로 처리하기에는 프로세싱 타임 낭비
 - 어떻게 하나?
 - 현재 수행중인 일을 잠시 중단하고 급한 일을 처리
 - 일이 끝나면 본래의 일을 다시 이어서 수행
 - 이때, 급한 일을 해결하는 작업을 **인터럽트 서비스 루틴**이라 하는데, 각 인터럽트마다 고유의 인터럽트 서비스 루틴 존재
 - 인터럽트가 여러 개 동시에 걸리면 우선 순위에 따라 조치
-

인터럽트(Interrupt)

□ 인터럽트 서비스 루틴

- 인터럽트가 발생하면 프로세서는 현재 수행중인 프로그램을 멈추고
- 상태 레지스터와 PC(Program Counter) 등을 스택에 잠시 저장한 후 인터럽트 서비스 루틴으로 점프
- 인터럽트 서비스 루틴을 실행한 후에는 이전의 프로그램으로 복귀하여 정상적인 절차를 실행한다.



인터럽트(Interrupt)

□ 인터럽트의 종류

- 발생원인에 따른 인터럽트 분류
 - 내부 인터럽트
 - 외부 인터럽트
- 차단가능성에 의한 인터럽트 분류
 - 차단(마스크) 불가능(Non maskable, NMI)인터럽트
 - 차단(마스크) 가능(Maskable)인터럽트
- 인터럽트 조사 방식에 따른 분류
 - 조사형 인터럽트(Polled Interrupt)
 - 벡터형 인터럽트(Vectored Interrupt)

ATmega128 인터럽트

□ ATmega128 인터럽트

- 차단 가능한 외부 인터럽트
 - 리셋 포함 총 35개의 인터럽트 벡터를 가짐
 - 리셋 1개
 - 외부핀을 통한 외부 인터럽트 8개
 - 타이머 관련 14개
 - 타이머 0(2개), 타이머 1(5개), 타이머 2(2개), 타이머 3(5개),
 - UART 관련 6개
 - USART0(3개), USART1(3개),
 - 기타 6개
-

ATmega128 인터럽트

□ ATmega128 인터럽트

- 특정 인터럽트를 가능하게 하려면 상태레지스터(SREG)에 있는 전체 인터럽트 허용 비트(I 비트)와 특정 인터럽트 가능 비트와 모두 1로 세트되어 있어야 함
 - 인터럽트는 각각 개별적인 프로그램 벡터를 프로그램 메모리 공간내에 가짐
 - 최하위 주소는 리셋과 인터럽트 벡터 순으로 정의되어 있는데, 최하위 주소에 있는 벡터는 최상위 주소에 있는 벡터에 비해 우선순위가 높음
 - RESET : 최우선 1순위
 - INT0(External Interrupt Request 0) : 2순위
 - MCU 제어 레지스터(MCUCR: MCU Control Register)의 IVSEL(Interrupt Vector SElect) 비트를 세팅함으로써 인터럽트 벡터의 배치 변경 가능
-

ATmega128 인터럽트

□ 상태레지스터(SREG : Status REGister) I 비트

- ALU의 연산 후 상태와 결과를 표시하는 레지스터인데, 인터럽트가 경우는 제어 레지스터의 의미를 가짐

SREG

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

비트	설명
I	Global Interrupt Enable : sei(), cli())
T	Bit Copy Storage
H	Half Carry Flag
S	Sign Bit
V	2's Complement Overflow Flag
N	Negative Flag
Z	Zero Flag
C	Carry Flag

ATmega128 인터럽트

- 인터럽트 마스크 레지스터 (EIMSK : External Interrupt MaSK register)
 - 외부 인터럽트의 개별적인 허용 제어 레지스터
 - INTn이 1로 세트되면 외부 인터럽트 인에이블

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

비트	설명
INT7~0	Int7~0 Interrupt Enable

ATmega128 인터럽트

□ 외부 인터럽트의 트리거

- 인터럽트 발생의 유무를 판단하는 근거로 Edge Trigger와 Level Trigger가 있음
 - Edge Trigger : 입력 신호가 변경되는 순간을 인터럽트 트리거로 사용하는 경우
 - 하강 에지(Falling Edge) 트리거 : '1' → '0' 변경 시점
 - 상승에지(Rising Edge) 트리거 : '0' → '1' 변경 시점
 - ATmega128 에지 트리거는 50ns 이상의 펄스폭을 가져야 함
 - Level Trigger : 입력 신호가 일정 시간 동안 원하는 레벨을 유지되면 트리거하는 경우
 - 평상시 High(1)로 있다가 Low(0)로 변화되어 일정시간 유지되면 트리거 함
 - 레벨 트리거 인터럽트 신호는 워치독 오실레이터에 의해 2번 샘플링되며 이 기간 이상의 펄스폭을 주어야 함
-

ATmega128 인터럽트

□ EICRA(External Interrupt Control Register A)

- 외부 인터럽트 0~3의 트리거 설정에 사용

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

ISCn1	ISCn0	설명
0	0	INTn의 Low level에서 인터럽트 발생
0	1	예약
1	0	INTn의 하강 에지에서 인터럽트 발생
1	1	INTn의 상승 에지에서 인터럽트 발생

ATmega128 인터럽트

□ EICRB(External Interrupt Control Register B)

- 외부 인터럽트 4~7의 트리거 설정에 사용

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

ISCn1	ISCn0	설명
0	0	INTn의 Low level에서 인터럽트 발생
0	1	INTn 핀에 논리적인 변화가 발생할 경우
1	0	INT의 하강 에지에서 인터럽트 발생
1	1	INT의 상승 에지에서 인터럽트 발생

실습 SW-2 : 스위치로 입장객 숫자 세기

□ 실습 내용

1. SW1을 누를 때마다 FND의 숫자가 0부터 시작하여 1씩 증가(인터럽트로 처리)하기
2. 1번과 함께 SW2를 누르면 FND 숫자가 다시 0으로 리셋되도록 제어하기 (혼자 해보기)

실습 SW-2 : 스위치로 입장객 숫자 세기

□ 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)

- 스위치가 연결되어 있는 신호를 포트 입력으로 설정
- 스위치(SW1)가 눌러지면 인터럽트가 발생하도록 관련 레지스터 초기화
- 인터럽트 서비스루틴에서는 count 값을 1 증가시키고, 메인루틴에서는 count 수에 해당하는 값을 FND에 디스플레이
- 인터럽트 서비스루틴과 메인루틴에서 함께 사용하는 변수는 volatile을 선언하여 처리하는 것이 안전함
- 인터럽트 처리 루틴 기본 형식 사용

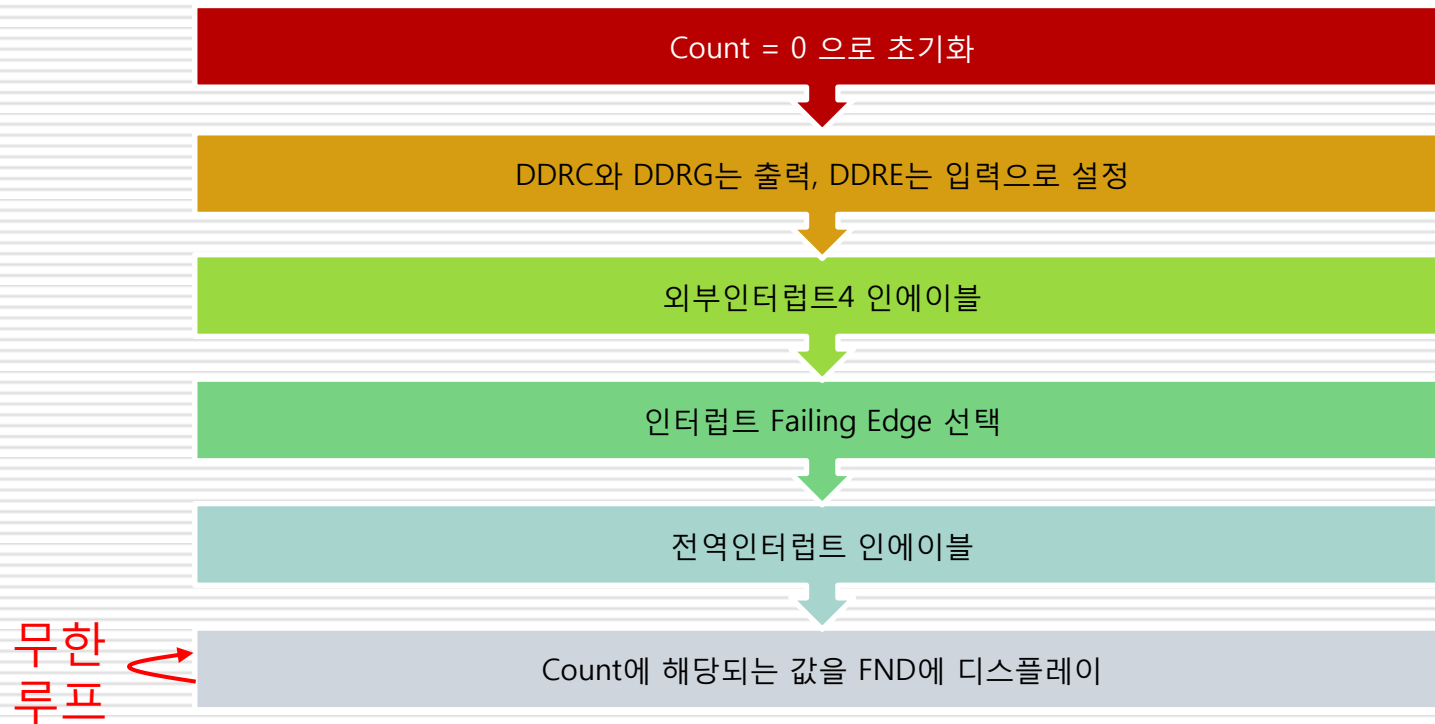
```
SIGNAL(인터럽트 종류)
{
    인터럽트 서비스 루틴
}
```

```
SIG_INTERRUPT4
SIG_INTERRUPT5
...
```

실습 SW-2 : 스위치로 입장객 숫자 세기

□ 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)

■ 메인프로그램



실습 SW-2 : 스위치로 입장객 숫자 세기

- 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)
 - 인터럽트4 서비스루틴

count 값을 1 증가

실습 SW-2 : 스위치로 입장객 숫자 세기

□ 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)

■ 인터럽트4 서비스루틴

```
#include <avr/io.h>                // ATmega128 register 정의
#include <avr/interrupt.h>          // interrupt 관련
#define F_CPU 16000000UL
#include <util/delay.h>
unsigned char digit[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7c, 0x07,
0x7f, 0x67};
unsigned char fnd_sel[4] = {0x01, 0x02, 0x04, 0x08};
volatile int      count = 0;        // 전역변수(Global Variable)
SIGNAL(SIG_INTERRUPT4)
{
    count++;
    _delay_ms(1);
}
```

실습 SW-2 : 스위치로 입장객 숫자 세기

- 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)
 - 메인 프로그램(FND 디스플레이 함수)

```
void display_fnd(int count)
{
    int i, fnd[4];
    fnd[3] = (count/1000)%10;           // 천 자리
    fnd[2] = (count/100)%10;           // 백 자리
    fnd[1] = (count/10)%10;            // 십 자리
    fnd[0] = count%10;                 // 일 자리
    for (i=0; i<4; i++)
    {
        PORTC = digit[fnd[i]];
        PORTG = fnd_sel[i];
        _delay_ms(2+i%2);
    }
}
```

실습 SW-2 : 스위치로 입장객 숫자 세기

□ 구동 프로그램 설계 : SW Counting 1 (sw_2_1.c)

■ 메인 프로그램

```
int main()
{
    DDRC = 0xff;           // C 포트는 FND 데이터 신호
    DDRG = 0x0f;           // G 포트는 FND 선택 신호
    DDRE = 0xef;           // 0b11101111, PE4(switch1)는 입력
    EICRB = 0x02;          // INT4 = falling edge
    EIMSK = 0x10;          // INT4 interrupt enable
    SREG |= 1<<7;          // SREG bit7 = I (Interrupt Enable)
    while (1)
    {
        display_fnd(count); // FND Display
    }
```

sei()와
동일

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 실습 내용

1. SW1을 한 번 누르면 Start 하고 다시 누르면 Stop을 반복하며, SW2를 누르면 초기화되는 1/100 스톱워치 만들기
(단, Stop한 후에도 스톱워치는 시간은 계속 카운트하고 있어 다시 Start를 하면 현재 시간부터 디스플레이되어야 함)

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 설계 : Stopwatch (sw_3_1.c)

■ 메인 프로그램

□ 초기 설정

- 포트 A는 출력 포트, 포트 E의 PE4, PE5는 입력 포트 설정
- 초기 FND 값 세팅 : 0000
- 초기 state 변수 값 세팅 : STOP
- PE4, PE5에 해당하는 INT4 인터럽트 활성화
 - 인터럽트 타입 결정 : INT4, INT5 모두 Falling Edge Trigger
 - 해당 INT4, INT5 인터럽트 활성화 : EIMSK 레지스터의 INT4, INT5 비트를 세트(enable)
 - 전역 인터럽트 활성화 : SREG 레지스터의 I 비트를 세트(enable)

- state를 검사하여 STOP 상태면 복사된 10초, 초, 1/10초, 1/100초 변수를 디스플레이하고, GO 상태면 현재의 10초, 초, 1/10초, 1/100초 변수를 디스플레이

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

- 구동 프로그램 설계 : Stopwatch (sw_3_1.c)
 - 인터럽트 처리 프로그램 INT4
 - state 값이 'STOP'이면 state 값을 'GO'시킴
 - state 값이 'GO'이면 state 값을 'STOP'시키고, 이 때의 10초, 1초, 1/10초, 1/100초 값을 복사하여 저장
 - 인터럽트 처리 프로그램 INT5
 - 10초, 1초, 1/10초, 1/100초 변수 및 복사된 10초, 1초, 1/10초, 1/100초 변수를 모두 0으로 초기화
 - state 값도 'STOP'으로 초기화

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 설계 : Stopwatch (sw_3_1.c)

■ 메인프로그램



실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 설계 : Stopwatch (sw_3_1.c)

■ 인터럽트4 서비스루틴

State가 GO이면 STOP으로 변경하고 현재시간 복사,
State가 STOP이면 GO로 변경

■ 인터럽트5 서비스루틴

State를 STOP으로 변경하고 현재시간 및 복사시간
변수를 모두 '0'으로 초기화

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 코딩 : Stopwatch (sw_3_1.c)

```
#include <avr/io.h>                // ATmega128 register 정의
#include <avr/interrupt.h>          // interrupt 관련
#define F_CPU 16000000UL
#include <util/delay.h>

#define      STOP    0
#define      GO      1

volatile int cur_time = 0;
volatile int stop_time = 0;
volatile int state = STOP; // 전역변수(Global Variable)
unsigned char digit[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7c, 0x07,
0x7f, 0x67};
unsigned char fnd_sel[4] = {0x01, 0x02, 0x04, 0x08};
```

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 코딩 : Stopwatch (sw_3_1.c)

```
SIGNAL(SIG_INTERRUPT4)
{
    if (state == STOP)
        state = GO;                // STOP → GO
    else
    {
        state = STOP;              // GO → STOP
        stop_time = cur_time;      // 현재시간 복사
    }
}

SIGNAL(SIG_INTERRUPT5)
{
    state = STOP;                  // 초기화, 리셋
    cur_time = 0;
    stop_time = 0;
}
```

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 코딩 : Stopwatch (sw_3_1.c)

```
void init( )
{
    DDRC = 0xff;    // FND Data
    DDRG = 0x0f;    // FND Select
    DDRE = 0xcf;    // INT4, 5
    PORTC = digit[0];
    PORTG = 0x0f;
    EICRB = 0x0a;    //falling edge
    EIMSK = 0x30;    //interrupt en
    sei();
}
```


실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 코딩 : Stopwatch (sw_3_1.c)

```
void display_fnd(int count)                // 수행시간 = 약 10ms
{
    int i, fnd[4];
    fnd[3] = (count/1000)%10;              // 천 자리
    fnd[2] = (count/100)%10;              // 백 자리
    fnd[1] = (count/10)%10;               // 십 자리
    fnd[0] = count%10;                    // 일 자리
    for (i=0; i<4; i++)
    {
        PORTC = digit[fnd[i]];
        PORTG = fnd_sel[i];
        _delay_ms(2+i%2);
    }
}
```

실습 SW-3 : 스위치로 1/100 스톱워치 만들기

□ 구동 프로그램 코딩 : Stopwatch (sw_3_1.c)

```
int main()
{
    init( );
    while(1)
    {
        if (state == STOP)
            display_fnd(stop_time);
        else
            display_fnd(cur_time);
        cur_time++;
    }
}
```

문고 답하기

Q & A

