인터럽트

- 폴링과 인터럽트 그리고 인터럽트 서비스루틴
- ATMega128A 인터럽트
- 인터럽트를 활용한 예제 및 응용 실습

엣지아이랩

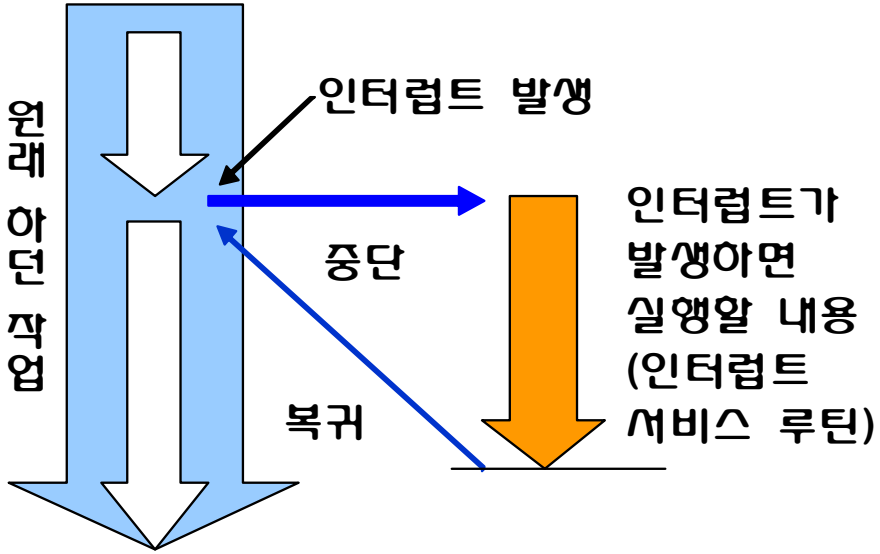
폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

edgeILAB

- 인터럽트(Interrupt)
  - "방해하다", "훼방놓다"
  - 어떤 작업을 진행하고 있다가 갑자기 다른 일이 발생하여 먼저 처리해야 하는 상황을 인터럽트 발생이라 함
  - 현재 수행중인 일을 잠시 중단하고 급한 일을 처리한 후, 원래의 일을 다시 이어서 수행
  - 이때, 그 급한 일을 해결하는 것이 인터럽트 서비스 루틴임
  - 발생 시기를 예측할 수 없는 경우에 더 효율적

폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

- 인터럽트와 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 실행



폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

- 마이크로 컨트롤러의 외부 입력 방법
  - 폴링(polling)
    - 사용자가 명령어를 사용하여 입력 핀의 값을 계속 읽어서 변화를 알아내는 방식
  - 인터럽트(interrupt)
    - MCU자체가 하드웨어적으로 그 변화를 체크하여 변화시에만 일정한 동작을 하는 방식

- 인터럽트의 구성요소
  - 발생원 : 누가 인터럽트를 요청했는가?
  - 우선순위 : 2개 이상의 요청시 누구를 먼저 서비스 할까?
  - (중요도 : Priority)
  - 인터럽트벡터 : 서비스루틴의 시작번지는 어디인가?

폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

- 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 Interrrupt)
    - 벡터형 인터럽트(Vectored Interrupt)

ATMega128A 인터럽트

- ATMega128A 인터럽트
  - 차단 가능한 외부 인터럽트
  - 리셋 포함 총 35개의 인터럽트 벡터를 가짐
    - 리셋 1개
    - 외부핀을 통한 외부 인터럽트 8개
    - 타이머 관련 14개
      - 타이머0(2개), 타이머1(5개), 타이머2(2개), 타이머3(5개)
    - UART 관련 6개
      - USART0(3개), USART1(3개)
    - 기타 6개

ATMega128A 인터럽트

- ATMega128A 인터럽트
  - 모든 인터럽트는 전역 인터럽트 Enable 비트인 SREG의 I 비트와 각각의 개별적인 인터럽트 플래그 비트가 할당됨
  - 인터럽트들과 개개의 리셋벡터는 각각 개별적인 프로그램 벡터를 프로그램 메모리 공간내에 존재
  - 모든 인터럽트들은 개별적인 인터럽트 허용 비트를 할당 받음
  - 특정 인터럽트를 가능하게 하려면 특정 인터럽트 가능 비트와 상태레지스터 (SREG)에 있는 전체 인터럽트 허용 비트(I 비트)가 모두 1로 세트되어 있어야함

ATMega128A 인터럽트

- 상태레지스터(SREG:Status REGister)
  - ALU의 연산 후 상태와 결과를 표시하는 레지스터

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

비트	설명
I	Global Interrupt Enable
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

ATMega128A 인터럽트

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

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

비트	설명
INT7~0	Int7~0 Interrupt Enable

ATMega128A 인터럽트

- ATMega128A 인터럽트 우선순위
  - 프로그램 메모리 공간에서 최하위 주소는 리셋과 인터럽트 벡터로 정의
  - 리스트는 서로 다른 인터럽트들의 우선순위를 결정
  - 최하위 주소에 있는 벡터는 최상위 주소에 있는 벡터에 비해 우선순위가 높음
    - RESET : 최우선 순위
    - INT0(External Interrupt Request 0) : 2순위
  - MCU 제어 레지스터(MCUCR: MCU Control Register)의 IVSEL(Interrupt Vector SELect) 비트를 세팅 함으로써 인터럽트 벡터의 배치 변경 가능

ATMega128A 인터럽트

- 외부 인터럽트의 트리거
  - 인터럽트를 발생시키기 위한 "방아쇠"
  - 인터럽트 발생의 유무를 판단하는 근거가 됨
  - 트리거 방법
    - Edge Trigger : 입력 신호가 변경되는 순간을 트리거로 사용하는 경우
      - 하강에지(Falling Edge) 트리거 : '1'에서 '0'로 변경되는 시점을 사용
      - 상승에지(Rising Edge) 트리거 : '0'에서 '1'로 변경되는 시점을 사용
      - ATMega128에서 에지 트리거는 50ns 이상의 펄스폭을 가져야 함
    - Level Trigger : 입력 신호가 일정 시간동안 원하는 레벨을 유지시 트리거로 사용하는 경우
      - 평상시 High(1)로 있다가 Low(0)로 변화되어 일정시간 유지되면 트리거 하게 됨
      - 레벨 트리거 인터럽트 신호는 워치독 오실레이터에 의해 2번 샘플링되며 이 기간이상의 펄스폭을 줘야 함

ATMega128A 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	INT의 Low level에서 인터럽트를 발생한다.
0	1	예약
1	0	INT의 하강 에지에서 인터럽트를 발생한다.
1	1	INT의 상승 에지에서 인터럽트를 발생한다.

ATMega128A 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	INTn1의 Low level에서 인터럽트를 발생한다.
0	1	INTn 핀에 논리적인 변화가 발생할 경우
1	0	INT의 하강 에지에서 인터럽트를 발생한다.
1	1	INT의 상승 에지에서 인터럽트를 발생한다.

ATMega128A 인터럽트

- EIFR(Interrupt Flag Register)
  - 외부 인터럽트 발생 여부를 알려주는 레지스터
  - 외부 인터럽트가 에지 트리거에 의해 요청된 경우 허용여부에 상관없이 1로 세트

7	6	5	4	3	2	1	0
INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0

비트	설명
INTF7~0	IntF7~0 Interrupt Flag

ATMega128A 인터럽트

- ATMega128A 인터럽트 프로그램
  - 인터럽트 프로그램의 틀

```
#include <io.h>
#include <interrupt.h>  // 인터럽트 관련 시스템 헤더 파일

SIGNAL(INT0_vect) // Int0에 대한 Interrupt Service Routine 선언
{
        // 인터럽트가 발생되었을 때 실행할 명령어 세트를 선언
}

int main(void)
{
  DDRD = 0xFE;          // 인터럽트 입력으로 D 포트를 사용
  EIMSK = 0x01;         // external interrupt 0 을 Enable
  sei();                // 전체 인터럽트 Enable
  for (;;) {            // main loop 명령어 세트
  }
}
```

1)예제 : 인터럽트로 LED 점멸

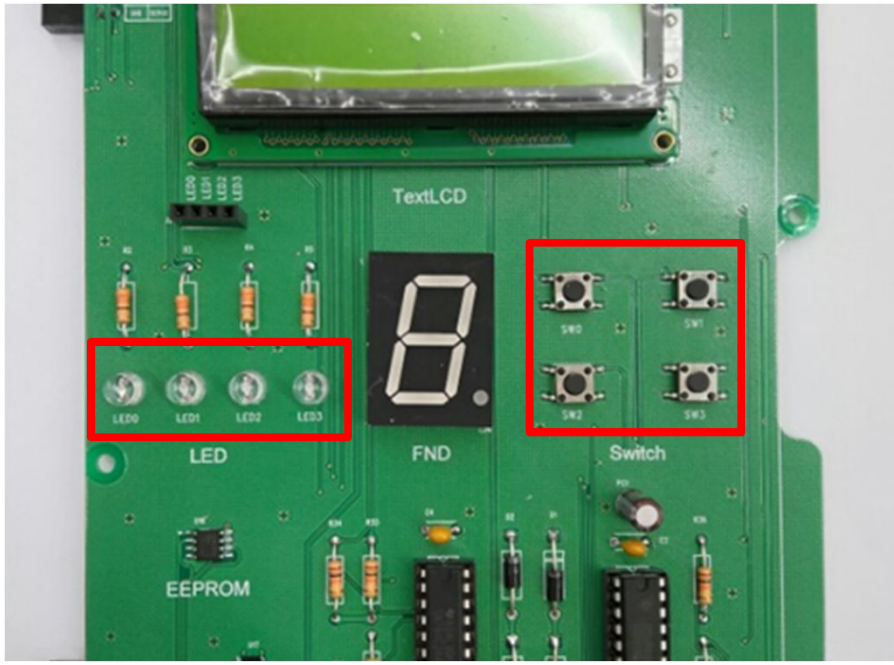
- 실습 개요
  - ATmega128A 마이크로컨트롤러의 인터럽트 기능을 이용하여 LED를 점멸시키기
  - 일정시간마다 LED가 순차적으로 켜지도록 하고, 버튼 스위치를 누르면 LED가 멈추었다가 다시 누르면 동작
  - 버튼 스위치가 눌려지면 인터럽트 발생
  - 입력포트 1개(인터럽트가 가능한 포트), 출력포트 1개 사용

- 실습 목표
  - 인터럽트 발생 원리 이해
  - 인터럽트 제어 방법의 습득(관련 레지스터 이해)
  - 입출력 포트에 관한 이해(특히 인터럽트 관련 포트)

1)예제 : 인터럽트로 LED 점멸

- Edge-MCU AVR
  - LED는 C 포트에 연결
  - SWITCH는 E 포트에 연결

Sensor	GPIO
LED0	PC0
LED1	PC1
LED2	PC2
LED3	PC3
SW0	PE4(INT4)
SW1	PE5(INT5)
SW2	PE6(INT6)
SW3	PE7(INT7)

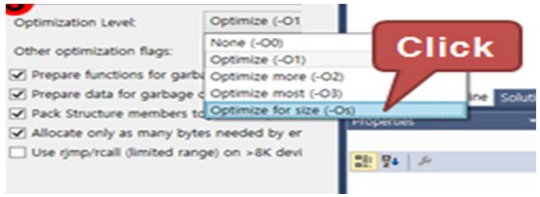


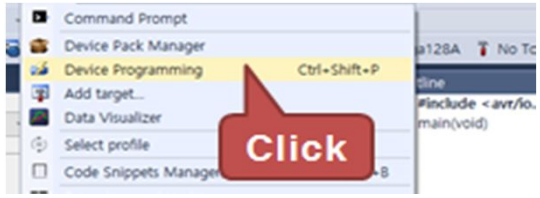
1)예제 : 인터럽트로 LED 점멸

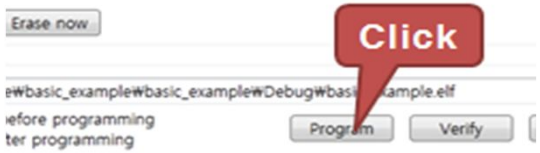
- 구동 프로그램 : 사전 지식
  - 인터럽트를 위한 입력 포트 선택
    - 포트 E의 4번 비트는 INT4로서, 인터럽트로 사용할 수 있는 포트
  - 외부 인터럽트 Enable
    - 상태 레지스터(SREG)의 전체 인터럽트 허용비트(I 비트)를 '1'로 세팅
    - INT4의 인터럽트 마스크 레지스터의 4번 비트를 1로 세팅
  - 인터럽트 트리거 방법 정의
    - 상승 에지에서 트리거하도록 EICRB 레지스터를 세팅
  - 전체 인터럽트 Enable
    - sei();
  - Main 루틴 기술
  - 인터럽트 서비스 루틴의 선언
    - SIGNAL(인터럽트소스명); 방식을 사용

1)예제 : 인터럽트로 LED 점멸

- 예제 프로그램 작성 및 구동 순서
  - Atmel Studio 실행
  - New Project 생성
    - Name : 05_INTLED_Example, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compier에서
    - Symbols -> F_CPU=14745600 추가
    - Optimization -> Optimize for size (-OS) 선택
    - 저장 (Ctrl+S)
  - 소스코드 작성
  - 프로젝트 빌드
    - Build 탭에서 "Build Solution" 클릭
  - 프로그래밍
    - Tool 탭에서 "Device Programming" 클릭
    - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
    - 인식 완료되면, Memories 탭 선택, "Program" 클릭







1)예제 : 인터럽트로 LED 점멸

- 구동 프로그램
  - main.c 코드 작성

```
#include <avr/io.h>      // AVR 입출력에 대한 헤더 파일
#include <avr/interrupt.h>       // AVR 인터럽트에 대한 헤더파일
#include <util/delay.h> // delay 함수사용을 위한 헤더파일

volatile unsigned char Time_STOP = 0;

int main(void) {
  unsigned  char LED_Data = 0x01;
  DDRC = 0x0F;            // 포트C 를 출력포트로 설정
  DDRE = 0x00;            // 포트E 를 입력포트로 설정
  EICRB = 0x03;           // 인터럽트 4를 상승엣지에서 동작하도록 설정
  EIMSK = 0x10;           // 인터럽트 4를 허용
  EIFR = 0x10;            // 인터럽트 4 플래그를 클리어
  sei();                  // 전체 인터럽트를 허용
```

1)예제 : 인터럽트로 LED 점멸

- 구동 프로그램
  - main.c 코드 작성

```
while (1)
  {
    PORTC = LED_Data;    // 포트C로 변수 LED_Data에 있는 데이타 출력
    if(Time_STOP == 0)   // Time_Stop이 0인 경우
    {
      if(LED_Data == 0x08) LED_Data = 0x01;
      // LED_Data 값이 0x08이면 LED_Data 값을 0x01로 함
      else LED_Data <<= 1;       // LED_Data 값을 왼쪽으로 쉬프트
    }
    _delay_ms(100);
  }
}
```

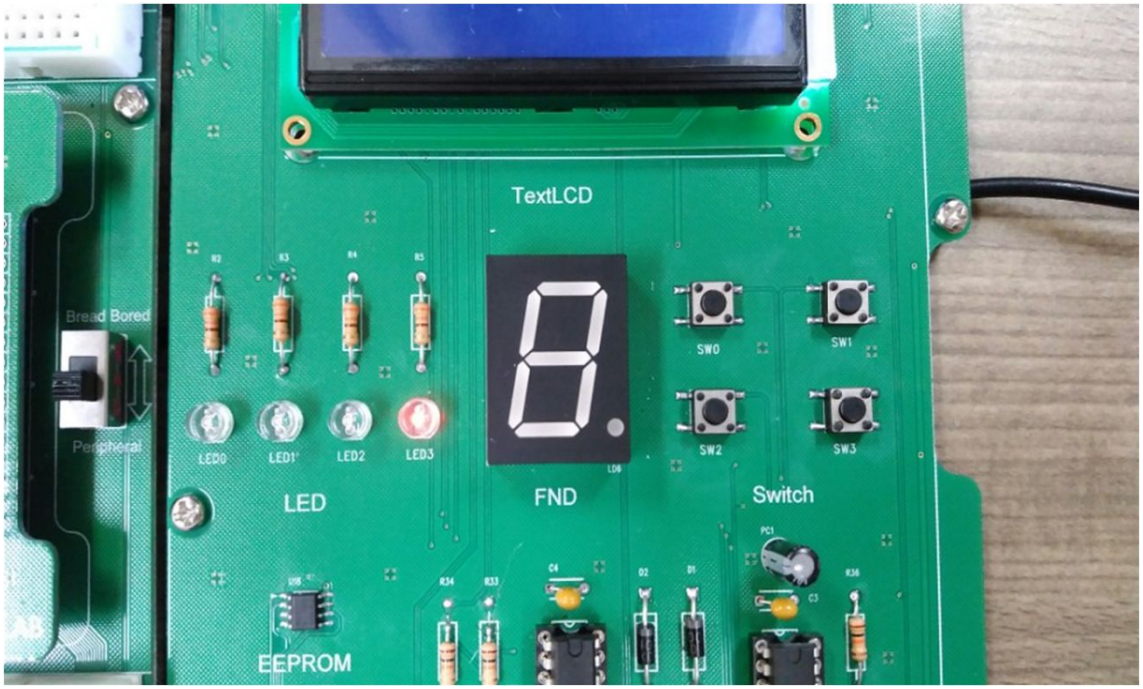
1)예제 : 인터럽트로 LED 점멸

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT4_vect)          // 인터럽트 서비스 루틴
{
  cli();                   // 전체 인터럽트를 금지
  if(Time_STOP == 0)       // Time_Stop이 0인 경우
  {
    Time_STOP = 1;         // Time_Stop에 1을 입력
  }
  else                     // Time_Stop이 1인 경우
  {
    Time_STOP = 0;         // Time_Stop에 0을 입력
  }
  sei();                   // 전체 인터럽트를 허용
}
```

1)예제 : 인터럽트로 LED 점멸

- 실행 결과
  - LED의 불이 100ms마다 우측으로 쉬프트 되면서 켜짐
  - 0번 버튼을 누르면 LED가 멈추었다가 한번 더 누르면 다시 동작



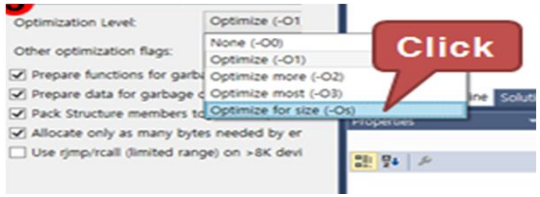
2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

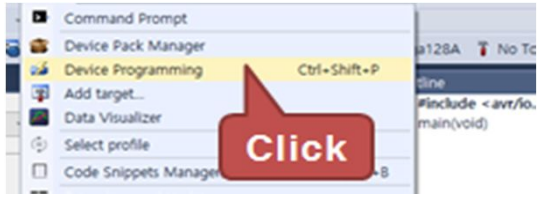
- 실습 개요
  - 인터럽트 기능을 이용하여 LED점멸 방향 변경
  - 버튼 스위치의 트리거(하강, 상승)에 따라 인터럽트 발생
  - 입력포트 2개(인터럽트가 가능한 포트), 출력포트 1개 사용

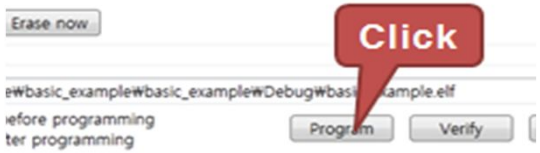
- 실습 목표
  - 인터럽트 발생 원리 이해
  - 인터럽트 제어 방법의 습득(관련 레지스터 이해)
  - 입출력 포트에 관한 이해(특히 인터럽트 관련 포트)

2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

- 예제 프로그램 작성 및 구동 순서
  - Atmel Studio 실행
  - New Project 생성
    - Name : 05_INTLED_Application, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compier에서
    - Symbols -> F_CPU=14745600 추가
    - Optimization -> Optimize for size (-OS) 선택
    - 저장 (Ctrl+S)
  - 소스코드 작성
  - 프로젝트 빌드
    - Build 탭에서 "Build Solution" 클릭
  - 프로그래밍
    - Tool 탭에서 "Device Programming" 클릭
    - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
    - 인식 완료되면, Memories 탭 선택, "Program" 클릭







2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

- 구동 프로그램
  - main.c 코드 작성

```
#include <avr/io.h>                    // AVR 입출력에 대한 헤더 파일
#include <avr/interrupt.h>             // AVR 인터럽트에 대한 헤더파일
#include <util/delay.h>                // delay 함수사용을 위한 헤더파일

volatile unsigned char Shift_flag = 1;

int main(void) {
  unsigned  char LED_Data = 0x01;

  DDRC = 0x0F;                         // 포트C 를 출력포트로 설정
  DDRE = 0x00;                         // 포트E 를 입력포트로 설정

  // 인터럽트 5를 하강에지, 7을 상승에지에서 동작하도록 설정
  EICRB = 0xC8;

  EIMSK = 0xA0;                       // 인터럽트 5, 7을 허용
  EIFR = 0xA0;                        // 인터럽트 5, 7 플래그를 클리어

  sei();                              // 전체 인터럽트를 허용
```

2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

- 구동 프로그램
  - main.c 코드 작성

```
  while (1) {

    //포트C로 변수 LED_Data에 있는 데이타를 출력
    PORTC = LED_Data;
    if(Shift_flag == 1)           // LED0 ~ LED3으로 이동
    {
      if(LED_Data == 0x08) LED_Data = 0x01;

      // LED_Data 값을 왼쪽으로 쉬프트
      else LED_Data <<= 1;
    }
    else if(Shift_flag == 2)      // LED3 ~ LED0으로 이동
    {
      if(LED_Data == 0x01) LED_Data = 0x08;
      else LED_Data >>= 1;        // LED_Data 값을 오른쪽으로 쉬프트
    }
    _delay_ms(100);
  }
}
```

2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

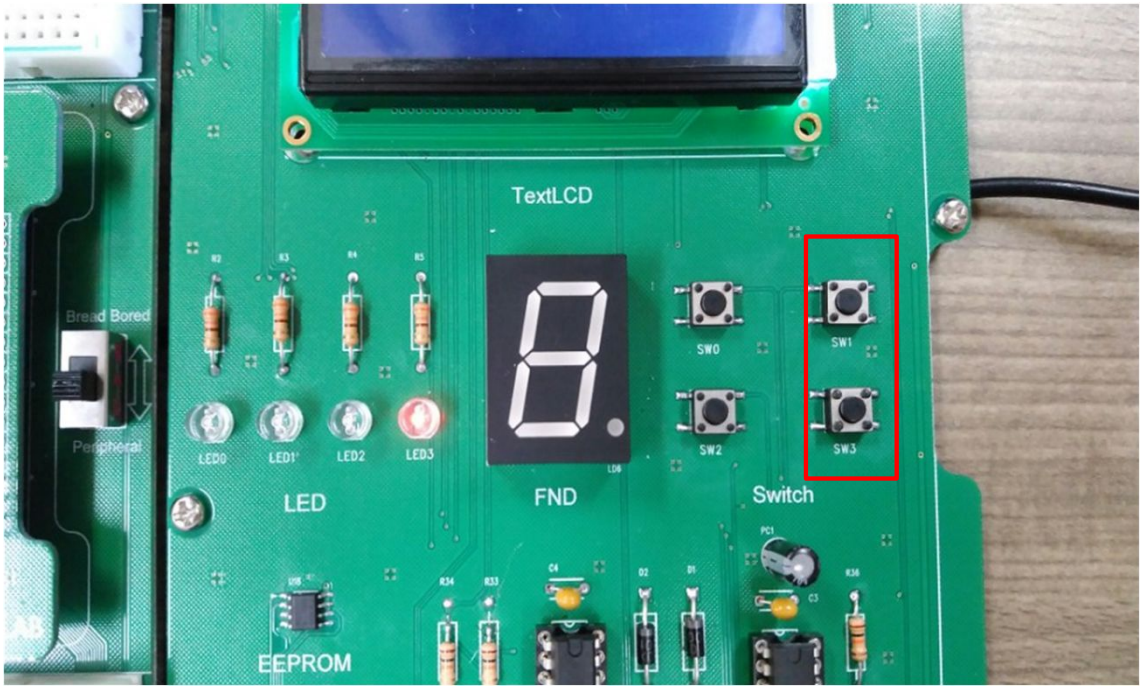
- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT5_vect)                    // 인터럽트 서비스 루틴
{
  cli();                             // 전체 인터럽트를 금지
  Shift_flag = 1;                    // Shift_flag에 1을 입력
  sei();                             // 전체 인터럽트를 허용
}

SIGNAL(INT7_vect)                    // 인터럽트 서비스 루틴
{
  cli();                             // 전체 인터럽트를 금지
  Shift_flag = 2;                    // Shift_flag에 2를 입력
  sei();                             // 전체 인터럽트를 허용
}
```

2)응용 : 인터럽트로 LED의 좌우 이동 방향 변경하기

- 실행 결과
  - LED의 불이 100ms마다 우측 방향으로 점멸
  - 1번 버튼을 눌렀다 뗐을 때 LED가 좌에서 우로 이동하고, 3번 버튼을 눌렀을 때 우에서 좌로 이동



3)예제 : 인터럽트로 FND 점멸

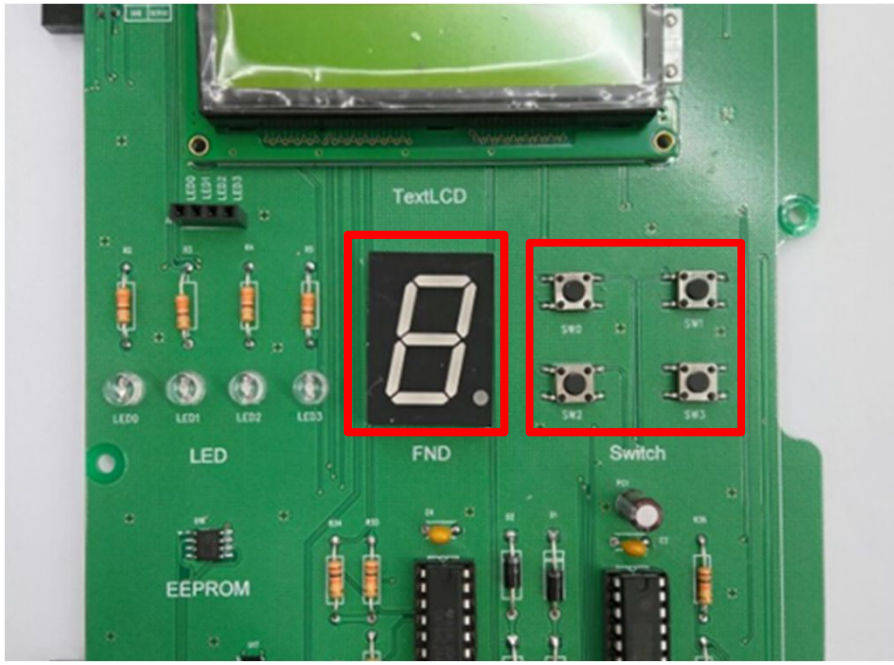
- 실습 개요
  - 스위치 모듈과 FND 모듈에 연결하여 FND 점멸 실습
  - 일정 시간마다 클럭에 의해 FND에 숫자와 문자가 디스플레이 되도록 하고, 스위치를 누르면 FND 디스플레이가 초기화되도록 하며, 또한 다른 버튼을 누르면 잠시 멈추었다가 다시 이어서 동작을 함
  - 포트 E의 5번 비트를 INT5의 인터럽트로 사용
    - INT5에 의해서 FND 점멸 동작의 Stop/Start 기능을 구현

- 실습 목표
  - 인터럽트 활용 방법의 습득(관련 레지스터 이해)
  - Array FND 동작 원리 이해

3)예제 : 인터럽트로 FND 점멸

- Edge-MCU AVR
  - FND는 A 포트에 연결
  - SWITCH는 E 포트에 연결

Sensor	GPIO
DATA_A	PA0
DATA_B	PA1
DATA_C	PA2
DATA_D	PA3
DATA_E	PA4
DATA_F	PA5
DATA_G	PA6
DATA_H	PA7
SW0	PE4(INT4)
SW1	PE5(INT5)
SW2	PE6(INT6)
SW3	PE7(INT7)

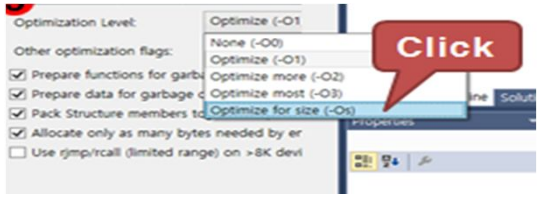


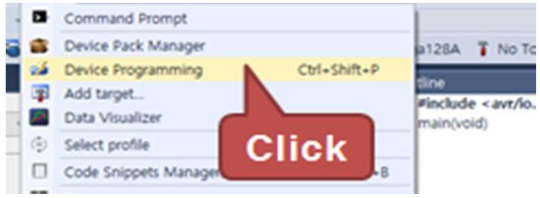
3)예제 : 인터럽트로 FND 점멸

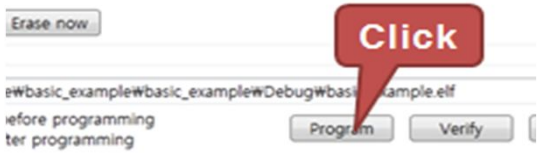
- 구동 프로그램 : 사전 지식
  - 인터럽트를 위한 입력 포트 선택
    - 포트 E의 5번 비트는 INT5로서, 인터럽트로 사용할 수 있는 포트
  - 외부 인터럽트 Enable
    - 상태 레지스터(SREG)의 전체 인터럽트 허용비트(I 비트) 를 '1'로 세팅
    - INT5의 인터럽트 마스크 레지스터의 5번 비트를 1로 세팅
  - 인터럽트 트리거 방법 정의
    - 상승 에지에서 트리거하도록 EICRB 레지스터를 세팅
  - 전체 인터럽트 Enable
    - sei();
  - Main 루틴 기술
  - 인터럽트 서비스 루틴의 선언
    - SIGNAL(인터럽트소스명); 방식을 사용

3)예제 : 인터럽트로 FND 점멸

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 05_INTFND_Example, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compier에서
    - Symbols -> F_CPU=14745600 추가
    - Optimization -> Optimize for size (-OS) 선택
    - 저장 (Ctrl+S)
  - 소스코드 작성
  - 프로젝트 빌드
    - Build 탭에서 "Build Solution" 클릭
  - 프로그래밍
    - Tool 탭에서 "Device Programming" 클릭
    - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
    - 인식 완료되면, Memories 탭 선택, "Program" 클릭







3)예제 : 인터럽트로 FND 점멸

- 구동 프로그램
  - main.c 코드 작성

```
#include <avr/io.h>                 // AVR 입출력에 대한 헤더 파일
#include <avr/interrupt.h>          // AVR 인터럽트에 대한 헤더파일
#include <util/delay.h>             // delay 함수사용을 위한 헤더파일

volatile unsigned char Time_STOP = 0;

int main(void) {

  // 7-Segment에 표시할 글자의 입력 데이터를 저장
  unsigned char FND_DATA_TBL []={0x3F,0X06,0X5B,0X4F,0X66,0X6D,
                                 0X7C,0X07,0X7F,0X67,0X77,0X7C,
                                 0X39,0X5E,0X79,0X71,0X08,0X80};
  unsigned char cnt=0;   // FND Table 카운터 변수
  DDRA = 0xFF;           // 포트A 를 출력포트로 설정
  DDRE = 0x00;           // 포트E 를 입력포트로 설정
```

3)예제 : 인터럽트로 FND 점멸

- 구동 프로그램
  - main.c 코드 작성

```
  EICRB = 0x0C;           // 인터럽트 5를 상승엣지에서 동작하도록 설정
  EIMSK = 0x20;           // 인터럽트 5를 허용
  EIFR = 0x20;            // 인터럽트 5 플래그를 클리어
  sei();                  // 전체 인터럽트를 허용

  while (1) {
    PORTA = FND_DATA_TBL[cnt];   // 포트A에 FND Table 값을 출력
    if(Time_STOP == 0)           // Time_Stop이 0인 경우
    {
      cnt++;                     // FND Table 카운터 변수를 증가
      if(cnt>17) cnt=0;          // 테이블 크기를 초과하는 경우 방지
    }
    _delay_ms(200);
  }
}
```

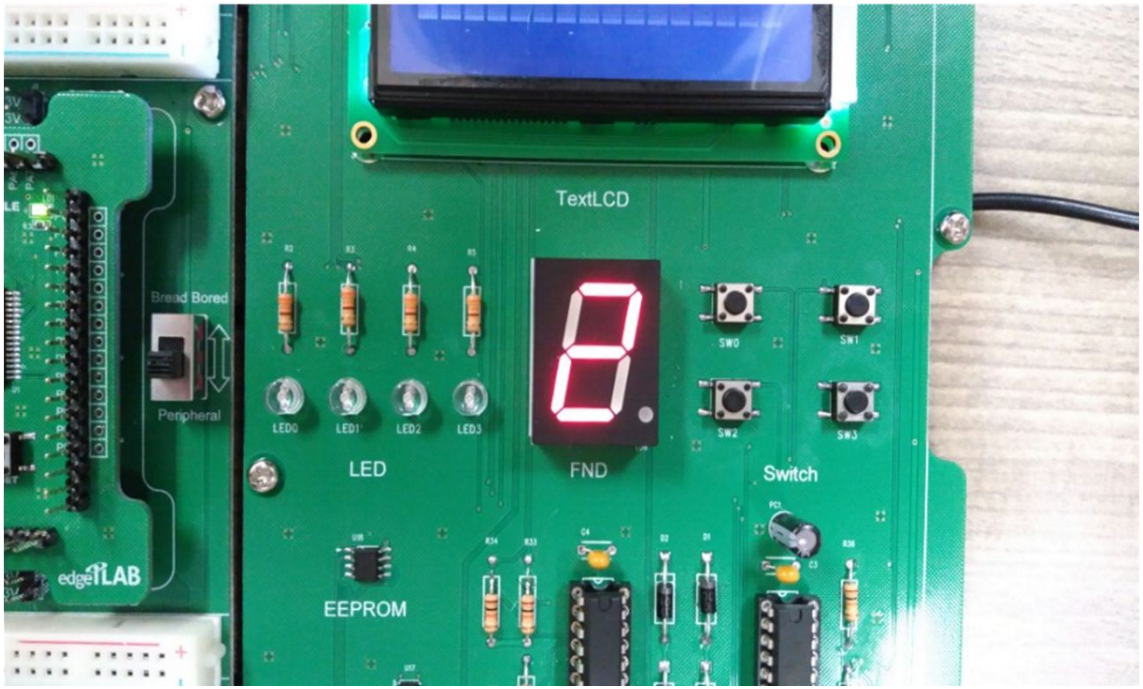
3)예제 : 인터럽트로 FND 점멸

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT5_vect)          // 인터럽트 서비스 루틴
{
    cli();                 // 전체 인터럽트를 금지
    if(Time_STOP == 0)     // Time_Stop이 0인 경우
    {
      Time_STOP = 1;       // Time_Stop에 1을 입력
    }
    else                   // Time_Stop이 1인 경우
    {
      Time_STOP = 0;       // Time_Stop에 0을 입력
    }
    sei();                 // 전체 인터럽트를 허용
}
```

3)예제 : 인터럽트로 FND 점멸

- 실행 결과
  - FND의 숫자가 200ms마다 증가하면서 켬
  - 1번 버튼을 누르면 FND가 멈추었다가 한번 더 누르면 다시 동작



4)응용 : 인터럽트로 FND LED 값 변경하기

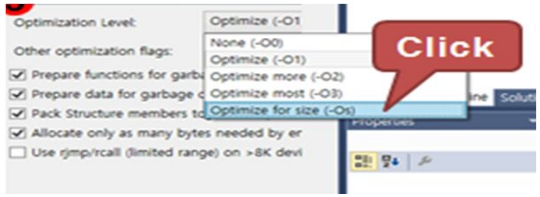
edgeILAB

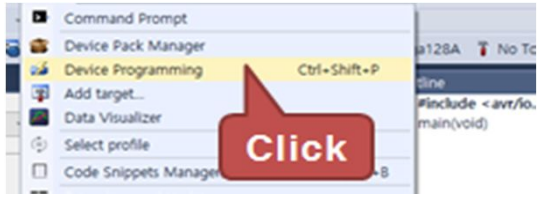
- 실습 개요
  - 스위치 모듈과 FND 모듈에 연결하여 FND의 값 변경하기
  - 0~F까지의 숫자를 FND에 표시하고, 스위치0을 누르면 0으로 초기화, 스위치2를 누르면 다시 동작
  - 포트 E의 4번, 6번 비트를 INT4, INT6의 인터럽트로 사용

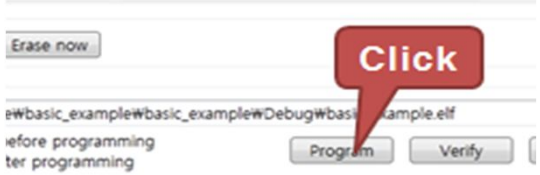
- 실습 목표
  - 인터럽트 활용 방법의 습득(관련 레지스터 이해)
  - Array FND 동작 원리 이해

4)응용 : 인터럽트로 FND LED 값 변경하기

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 05_INTFND_Application, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compier에서
    - Symbols -> F_CPU=14745600 추가
    - Optimization -> Optimize for size (-OS) 선택
    - 저장 (Ctrl+S)
  - 소스코드 작성
  - 프로젝트 빌드
    - Build 탭에서 "Build Solution" 클릭
  - 프로그래밍
    - Tool 탭에서 "Device Programming" 클릭
    - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
    - 인식 완료되면, Memories 탭 선택, "Program" 클릭







4)응용 : 인터럽트로 FND LED 값 변경하기

- 구동 프로그램
  - main.c 코드 작성

```
#include <avr/io.h>                 // AVR 입출력에 대한 헤더 파일
#include <avr/interrupt.h>          // AVR 인터럽트에 대한 헤더파일
#include <util/delay.h>             // delay 함수사용을 위한 헤더파일

volatile unsigned char Time_STOP = 0;
volatile unsigned char cnt = 0; // FND Table 카운터 변수

int main(void) {
  // 7-Segment에 표시할 글자의 입력 데이터를 저장
  unsigned char FND_DATA_TBL []={0x3F,0X06,0X5B,0X4F,0X66,0X6D,
                                 0X7C,0X07,0X7F,0X67,0X77,0X7C,
                                 0X39,0X5E,0X79,0X71,0X08,0X80};

  DDRA = 0xFF;              // 포트A 를 출력포트로 설정
  DDRE = 0x00;              // 포트E 를 입력포트로 설정
```

4)응용 : 인터럽트로 FND LED 값 변경하기

- 구동 프로그램
  - main.c 코드 작성

```
  // 인터럽트 4를 하강엣지, 6을 상승엣지에서 동작하도록 설정
  EICRB = 0x32;
  EIMSK = 0x50;          // 인터럽트 4, 6을 허용
  EIFR = 0x50;           // 인터럽트 4, 6 플래그를 클리어
  sei();                 // 전체 인터럽트를 허용

  while (1) {
    PORTA = FND_DATA_TBL[cnt];   // 포트A에 FND Table 값을 출력
    if(Time_STOP == 0)           // Time_Stop이 0인 경우
    {
      cnt++;                     // FND Table 카운터 변수를 증가
      if(cnt>17) cnt=0;          // 테이블 크기를 초과하는 경우 방지
    }
    _delay_ms(200);
  }
}
```

4)응용 : 인터럽트로 FND LED 값 변경하기

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT4_vect)                    // 인터럽트 서비스 루틴
{
  cli();                             // 전체 인터럽트를 금지

  Time_STOP = 1;                     // Time_Stop에 1을 입력
  cnt = 0;                           // FND Table 카운터 변수 값을 초기화

  sei();                             // 전체 인터럽트를 허용
}

SIGNAL(INT6_vect)                    // 인터럽트 서비스 루틴
{
  cli();                             // 전체 인터럽트를 금지

  Time_STOP = 0;                     // Time_Stop에 0을 입력

  sei();                             // 전체 인터럽트를 허용
}
```

4)응용 : 인터럽트로 FND LED 값 변경하기

- 실행 결과
  - FND의 숫자가 200ms마다 증가
  - 0번 버튼을 누르면 0으로 초기화 및 정지, 2번 버튼을 누르면 다시 동작