edgeILAB
엣지아이랩 IT교육컨설팅

타이머와 카운터(8비트)

- 클럭과 카운터
- ATMega128A의 타이머/카운터
- 8비트 타이머/카운터
- 타이머를 활용한 예제 및 응용 실습

엣지아이랩

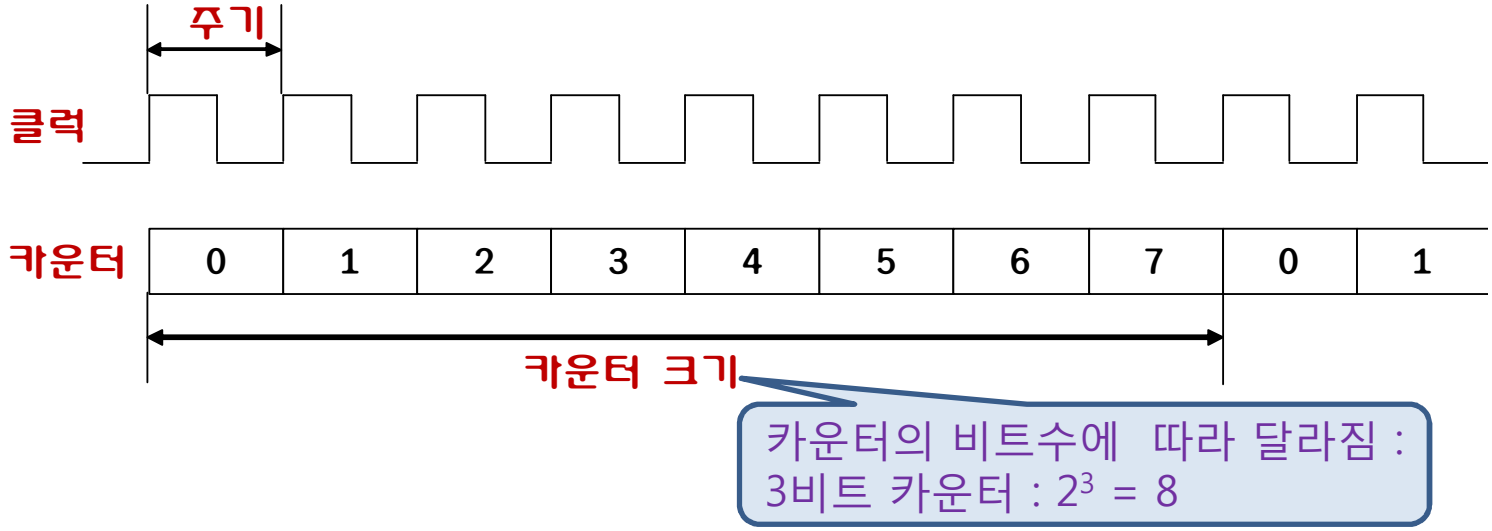
타이머/카운터

- 타이머와 카운터
  - 정확한 시간의 측정이 필요 (자명종과 스톱워치)
  - 임베디드 시스템에서 타이머와 카운터가 시간측정의 일을 담당
  - 타이머/카운터는 일정한 개수만큼의 클럭을 세어 시간을 측정하므로, 정확한 시간 재기가 가능
  - 타이머는 필요한 시간을 미리 레지스터에 설정하고, 다른 작업과 병행하게 타이머가 동작하고, 설정한 조건에서 인터럽트 발생하게 함으로써, MCU의 효율을 극대화

클럭과 카운터

- 클럭
  - 시계
  - 일정한 시간 간격으로 0과 1의 값이 번갈아 나타남
  - 주어진 일을 순서대로 정확한 시간에 처리하기 위해 사용

- 카운터
  - 클럭을 세는 장치



ATMega128A 타이머/카운터

- 타이머와 카운터
  - 타이머 : MCU 내부 클럭을 세는 장치
    - 동기모드
    - 타이머는 MCU의 내부클럭을 세어 일정시간 간격의 펄스를 만들어 내거나 일정시간 경과 후에 인터럽트를 발생
  - 카운터 : MCU의 외부에서 입력되는 클럭을 세는 장치
    - 비동기모드
    - 카운터는 외부 핀(TOSC1, TOSC2, T1, T2, T3)을 통해서 들어오는 펄스를 계수(Edge Detector)하여 Event Counter로서 동작

ATMega128A 타이머/카운터

edgeiLAB

- ATMega128A의 타이머 카운터
  - **타이머 0~3 모두 4개의 타이머/카운터를 보유**
    - 타이머 0,2 : 8비트 타이머로 서로 기능 유사
    - 타이머 1,3 : 16비트 타이머로 서로 기능 유사
  - 인터럽트 기능
    - **오버플로우 인터럽트** : 카운터의 값의 오버플로우에 의해 발생
    - 출력비교 인터럽트 : 카운터 값이 출력비교 레지스터의 값과 같게 되는 순간에 발생
    - 입력 캡쳐 인터럽트 : 외부로부터의 트리거 신호에 의해서 카운터의 초기값을 입력캡쳐
  - PWM 출력 기능
    - Pulse Width Modulation

8비트 타이머/카운터

- 8비트 타이머/카운터의 특징
  - 4개의 타이머/카운터 중 0번과 2번 타이머/카운터
  - PWM 및 비동기 동작 모드를 갖는 8비트 업/다운(Up/Down) 카운터
  - **8비트 카운터** : 2^8 = 256, 즉 0~255까지 셀 수 있음
  - 10비트의 프리스케일러(prescaler) 보유
  - 각종 인터럽트 기능
    - 오버플로우 인터럽트(overflow interrupt)
    - 출력비교 인터럽트(output compare match interrupt)
  - PWM 기능 제공
  - 타이머 0는 부가적으로 32.768KHz의 수정 진동자를 TOSC1,2 단자에 연결해 Real Time Clock으로 사용할 수 있는 기능 제공

8비트 타이머/카운터

- 8비트 타이머/카운터 레지스터
  - 타이머/카운터 제어 레지스터(TCCRn)
  - 타이머/카운터 레지스터(TCNTn)
  - 출력 비교 레지스터(OCRn)
  - 인터럽트 관련
    - 타이머/카운터 인터럽트 플래그 레지스터(TIFR)
    - 타이머/카운터 인터럽트 마스크 레지스터(TIMSK)

8비트 타이머/카운터

- 프리스케일러(Prescaler)
  - 고속의 클럭을 사용하여 타이머를 동작시킬 때 나타나는 문제를 해결하기 위해 클럭을 분주하여 더 느린 타이머 클럭을 만듦.
    - 4MHz 클럭을 사용하는 경우 그 클럭의 주기는 250ns
    - 이 클럭으로 256까지 센다고 해도 64us 이하를 세는 카운터 밖에는 만들 수가 없음
  - 10비트, 프리스케일러 보유(최대 2^{10} = 1024배 가능)
    - 프리스케일러를 1024로 하면 4MHz 클럭의 타이머 클럭은 주기가 250ns*1024=256us가 되고, 이 클럭을 256까지 센다면 256us*256=65.536ms 크기의 타이머를 만들 수 있음

8비트 타이머/카운터

- 프리스케일러(Prescaler)
  - 타이머 0
    - 클럭소스 : TOSC1에 입력된 시스템 클럭 주파수의 1/4미만의 외부클럭, TOSC1/TOSC2단자에 연결된 수정 진동자에 의해 발생된 클럭, 내부 클럭
    - 분주비 : 1, 8, 32, 64, 128, 256, 1024
  - 타이머 2
    - 클럭소스 : T2핀으로 입력되는 외부 클럭, 내부 클럭
    - 분주비 : 1, 8, 64, 256, 1024

8비트 타이머/카운터

- 8비트 타이머/카운터의 일반 동작 모드
  - ATmega128A는 4가지의 타이머 동작모드를 가짐
  - 가장 기본적인 동작모드가 일반동작 모드임

- 8비트 타이머/카운터의 동작
  - 동작 모드 결정
  - 타이머에 사용할 클럭 소스와 프리스케일러 결정
  - 원하는 타이머의 주기 및 그 주기 동안의 시간을 정확히 세기 위한 타이머 클럭의 수 결정
  - 카운터 레지스터의 카운터 시작 값 설정(오버플로우 인터럽트 사용)

$$\text{Timer Period} = \frac{1}{\text{Clock Frequency} / \text{Prescaler}} \cdot \text{TCNT}$$

8비트 타이머/카운터

- 8비트 타이머/카운터의 동작 예
  - 동작 모드 결정 : 일반동작모드
  - 타이머에 사용할 클럭 소스와 프리스케일러 결정
    - 내부클럭 : 14.4756MHz
    - 프리스케일러 : 1024
    - 타이머 클럭 주파수 : 14745600/1024= 14.4KHz
    - 타이머 클럭 주기 : 약 69us (1/14400 = 0.000069444)
  - 원하는 타이머의 주기 및 그 주기 동안의 시간을 정확히 세기 위한 타이머 클럭의 수 결정
    - 1 주기당 타이머 클럭 개수 = 타이머 클럭 주파수 * 타이머 시간
    - 만약 10ms의 타이머를 만들기 원한다면 14400 * 0.01 = 144
  - 카운터 레지스터의 카운터 시작 값 설정
    - 카운터의 시작위치를 계산하여 카운터 레지스터를 설정
    - 카운터시작 위치 = 255 - 타이머 클럭 개수(255 – 144 = 112)
    - 카운터 레지스터(TCNT) 값을 112로 초기화 시키고 타이머를 동작시키면 144번 증가 후 255값을 넘어서 인터럽트가 걸리게 됨

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - 타이머/카운터 제어 레지스터 n(n=0 or 2)
  - 동작 모드, 프리스케일러등 타이머/카운터의 전반적인 동작 형태를 결정
  - 비트 7 : FOCn
  - 비트 6, 3 : WGM
  - 비트 5, 4 : COM
  - 비트 2, 1, 0 : CSn

7	6	5	4	3	2	1	0
FOCn	WGMn0	COMn1	COMn0	WGMn1	CSn2	CSn1	CSn0

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - FOC(Force Output Compare)
    - FoC가 1로 세트되면 출력 비교 실시(OC핀 작동)
    - FoC가 0로 세팅되면 출력 비교를 실시하지 않음
  - WGM(Waveform Generation Mode)
    - 타이머/카운터의 동작모드 설정
    - 카운터의 카운팅 방향, 최대 카운터 값, Waveform Generation방식 등 결정

모드	WGMn1 (CTCn)	WGMn0 (PWMn)	타이머/카운터 동작모드	TOP	OCRn의 업데이트	TOVn Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCRn	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - COM(Compare Output Mode)
    - OCn핀의 동작을 조정
    - COMn1/COMn0에 따른 Ocn 핀의 동작
    - Non-PWM 모드

COMn1	COMn0	내용
0	0	normal 포트 동작, OCn 연결을 끊다.
0	1	Toggle OCn on compare match
1	0	Clear OCn on compare match
1	1	Set OCn on compare match

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - Fast-PWM 모드

COMn1	COMn0	내용
0	0	normal 포트 동작, OCn 연결을 끊다.
0	1	예약
1	0	Clear OCn on compare match, set OCn at TOP
1	1	Set OCn on compare match, clear OCn at TOP

  - PC PWM 모드

COMn1	COMn0	내용
0	0	normal 포트 동작, OCn 연결을 끊다.
0	1	예약
1	0	up카운팅일때 Clear OCn on compare match, down카운팅일때 set Ocn
1	1	up카운팅일때 Set OCn on compare match, down카운팅일때 clear Ocn

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - CSn : 클럭 선택(Clock Select)
    - 타이머/카운터에 사용할 클럭과 프리스케일러를 선택

    - 타이머/카운터 0

CS22	CS21	CS20	설명
0	0	0	클럭소스가 없다.
0	0	1	No 프리스케일러
0	1	0	8분주
0	1	1	32분주
1	0	0	64분주
1	0	1	128분주
1	1	0	256분주
1	1	1	1024분주

8비트 타이머/카운터

- TCCRn(Timer/Counter Control Register n)
  - 타이머/카운터 2

CS22	CS21	CS20	설명
0	0	0	클럭소스가 없다.
0	0	1	No 프리스케일러
0	1	0	8분주
0	1	1	64분주
1	0	0	256분주
1	0	1	1024분주
1	1	0	외부클럭의 하강에지에서 카운터 2동작
1	1	1	외부클럭의 상승에지에서 카운터 2동작

8비트 타이머/카운터

- TCNTn(Timer/Counter Register n)
  - 타이머/카운터 레지스터 n(n은 0 or 2)
    - 타이머/카운터 n의 8비트 카운터 값을 저장하고 있는 레지스터
    - 이 레지스터는 쓸 수도 있고 읽을 수도 있음
    - 임의의 값을 써주면 타이머의 주기를 더 빠르게 할 수 있음
    - 레지스터의 값은 인터럽트가 걸리면 자동으로 0으로 클리어 되고 다시 분주된 주기마다 한 개씩 값이 늘어남

7	6	5	4	3	2	1	0
TCNT7	TCNT6	TCNT5	TCNT4	TCNT3	TCNT2	TCNT1	TCNT0

8비트 타이머/카운터

- OCRn(Output Compare Register n)
  - 출력 비교 레지스터 n(n은 0 or 2)
    - TCNTn 값과 비교하여 OCn 핀에 출력 펄스 결정
    - 8비트 값을 저장하는 레지스터

7	6	5	4	3	2	1	0
OCR7	OCR6	OCR5	OCR4	OCR3	OCR2	OCR1	OCR0

8비트 타이머/카운터

- TIMSK(Timer Interrupt Mask)
  - 타이머 인터럽트 마스크 레지스터
  - 타이머/카운터0, 타이머/카운터1, 타이머/카운터2가 발생하는 인터럽트를 개별적으로 enable하는 레지스터
  - 비트 1, 7 : OCIE0/OCIE2
    - 타이머/카운터0/2의 출력비교 인터럽트 Enable
  - 비트 0, 6 : TOIE0/TOIE2
    - 타이머/카운터0/2 오버플로우 인터럽트 Enable

7	6	5	4	3	2	1	0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

8비트 타이머/카운터

- TIFR(Timer Interrupt Flag Register)
  - 타이머 인터럽트 프래그 레지스터
  - 타이머/카운터0, 타이머/카운터1, 타이머/카운터2가 발생하는 인터럽트의 플래그를 저장하는 레지스터
  - 비트 1, 7 : OCF0/OCF2
    - TCNT0/2레지스터와 출력비교 레지스터 OCR0/2값을 비교해서, 같으면 이 비트가 "1"로 세트되어 출력 비교인터럽트가 발생
  - 비트 0, 6 : TOV0/TOV2
    - 타이머/카운터0/2에서 오버플로우가 발생하면 이 비트가 "1"로 세트되어 오버플로우 인터럽트가 발생

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

8비트 타이머/카운터

- ASSR(ASynchronous Status Register)
  - 비동기 상태 레지스터
  - 타이머/카운터0이 외부 클럭에 의하여 비동기 모드로 동작하는 경우 관련된 기능을 수행하는 레지스터
  - 비트 3(AS0 : ASynchronous timer 0)
  - 비트 2(TCN0UB : Timer/CouNter0 Update Busy)
  - 비트 1(OCR0UB : Output Compare Register 0 Update Busy)
  - 비트 0(TCR0UB : Timer Conter Register 0 Update Busy)

7	6	5	4	3	2	1	0
-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB

8비트 타이머/카운터

- SFIOR(Special Function I/O Register)
  - 특수 기능 I/O 레지스터
  - 타이머/카운터들을 동기화 하는데 관련된 기능을 담당하는 레지스터
  - 비트 7(TSM : Timer Synchronization Mode)
    - '0' : PSR0, PSR321비트를 하드웨어적으로 클리어
    - 이로 인해, 타이머들이 동시에 카운팅 동작을 시작
  - 비트 1(PSR0 : Prescaler Reset Timer 0)
    - '1' : 타이머0의 프리스케일러를 리셋
  - 비트 0(PSR321 : Prescaler Reset Timer 321)
    - '1' : 타이머1/2/3의 프리스케일러를 리셋

7	6	5	4	3	2	1	0
TSM	-	-	-	ACME	PUD	PSR0	PSR321

8비트 타이머/카운터

- ATMega128A 타이머 오버플로우 인터럽트 프로그램
  - 타이머 오버플로우 인터럽트 프로그램의 틀

```
#include <io.h>
#include <interrupt.h>  // 인터럽트 관련 시스템 헤더 파일
// Timer0의 Overflow Interrupt에 대한 ISR 선언

SIGNAL(TIMER0_OVF_vect) {
  cli();
  // 인터럽트가 발생되었을 때 실행할 명령어 세트를 선언
  sei();
}
int main(void) {
  TIMSK |= 1 << TOIE0;   // 오버플로우 인터럽트 허용
  TIFR |=1 << TOV0;      // TOV0 Timer/Counter0 overflow flag클리어
  sei();                 // 전체 인터럽트 Enable
  for (;;) {             // main loop 명령어 세트
  }
}
```

1)예제 : 타이머로 LED 점멸(1)

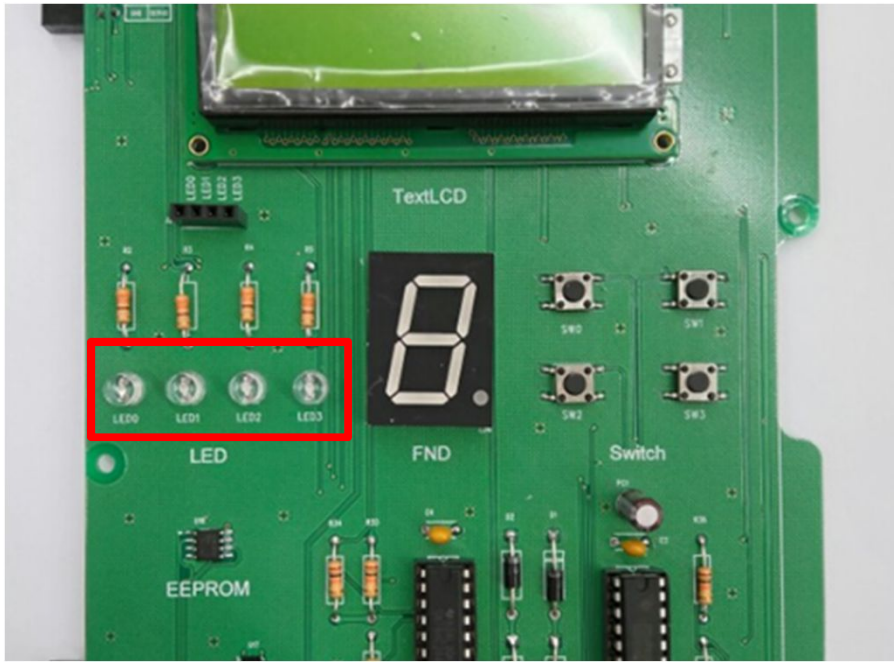
- 실습 개요
  - ATmega128A 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
  - 타이머를 이용하여 정확히 1초 마다 LED가 점멸하도록 함
  - 타이머는 타이머/카운터 0의 일반 동작 모드를 사용

- 실습 목표
  - 타이머0의 동작원리 이해
  - 타이머0 제어 방법의 습득(관련 레지스터 이해)
  - 오버플로우 인터럽트 제어 프로그램 방법 습득

1)예제 : 타이머로 LED 점멸(1)

- Edge-MCU AVR
  - LED는 C 포트에 연결

Sensor	GPIO
LED0	PC0
LED1	PC1
LED2	PC2
LED3	PC3



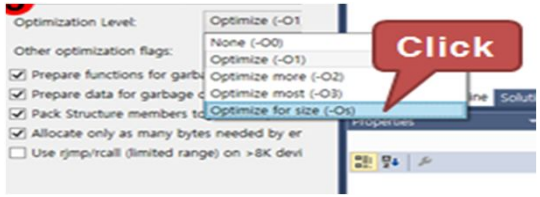
1)예제 : 타이머로 LED 점멸(1)

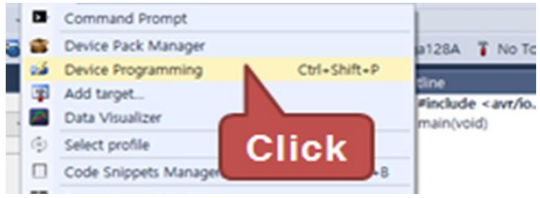
edgeILAB

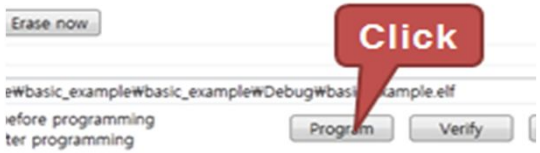
- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 8비트 타이머/카운터인 타이머/카운터 0를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCR 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러는 최대한 주기를 길게 하기 위해 1024를 사용
    - ASSR /SFIOR는 디폴트, TCCR0의 CS비트를 "111"로 설정
  - 타이머 주기 결정
    - 1초를 만들기 위해 10ms을 타이머 주기로 결정
    - 인터럽트는 10ms마다 발생, 인터럽트가 100번 발생하면 1초로 간주
    - 10ms를 맞추기 위해서 세어야 하는 카운터 개수를 계산하고 그 카운터 개수를 만족시키기 위한 TCNT 초기값을 TCNT0에 넣음(112)
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 오버플로우 인터럽트를 Enable하고, TIFR 레지스터의 Timer/Counter0 overflow flag를 클리어

1)예제 : 타이머로 LED 점멸(1)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERLED_Example_01, 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 : 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

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

volatile unsigned char LED_Data = 0x00;
unsigned char timer0Cnt=0;
SIGNAL(TIMER0_OVF_vect);            // Timer0 Overflow0 ISP

int main(void) {
  DDRC = 0x0F;                      // 포트C 를 출력포트로 설정
  TCCR0 = 0x07;
  TCNT0 = 112;
  // 256-144=112 -> 0.01초 마다 한번씩 인터럽트 발생
  TIMSK = 0x01;
  TIFR |=1 << TOV0;
  sei();
```

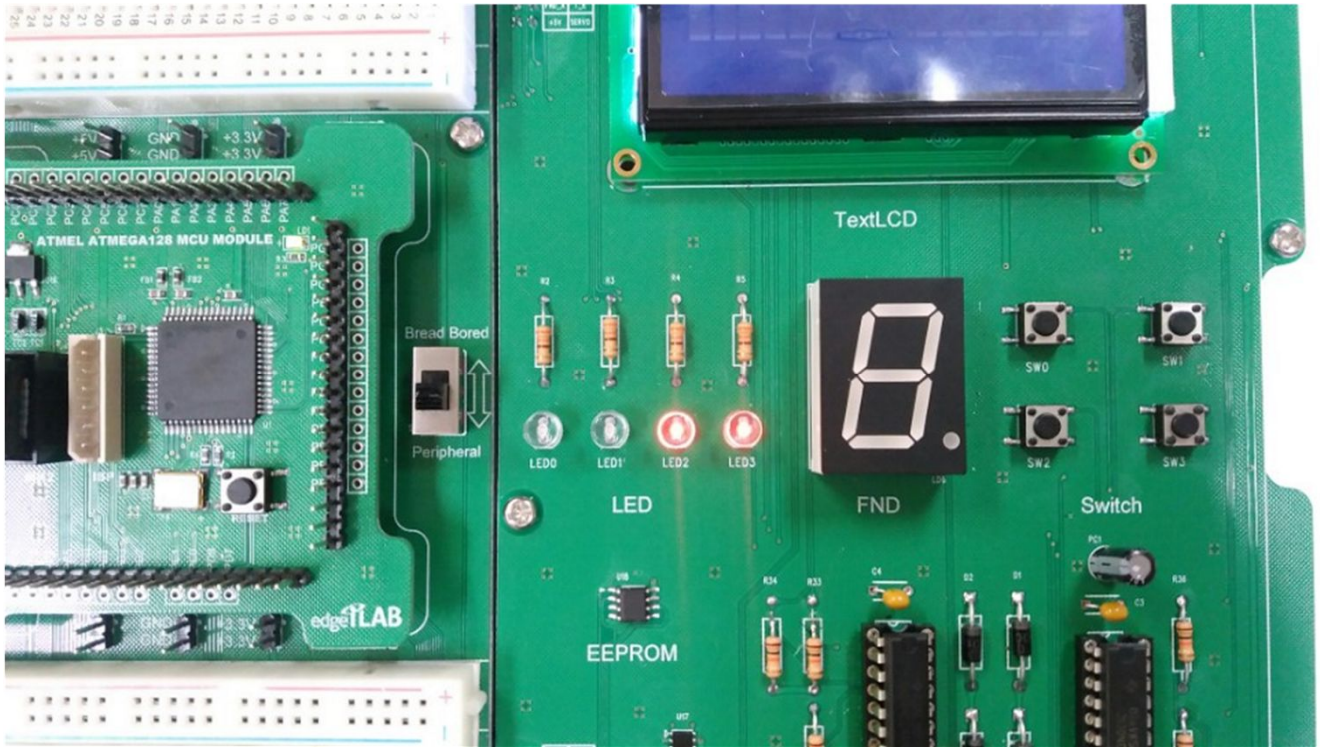
1)예제 : 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

```
  while (1) {
    PORTC = LED_Data;    // 포트C로 변수 LED_Data에 있는 데이타 출력
  }
}
SIGNAL(TIMER0_OVF_vect) {
  cli();
  TCNT0 = 112;  // 256-144=112 -> 0.01초 마다 한번씩 인터럽트 발생
  timer0Cnt++;  // timer0Cnt 변수를 1 증가
  if(timer0Cnt == 100) { // 0.01s*100=1s를 얻기 위한 카운트 횟수
    LED_Data++;           // LED_Data 변수를 1 증가
    if(LED_Data>0x0F) LED_Data = 0;
    timer0Cnt = 0;
  }
  sei();
}
```

1)예제 : 타이머로 LED 점멸(1)

- 실행 결과
  - LED의 불이 1초 마다 순차적으로 점멸



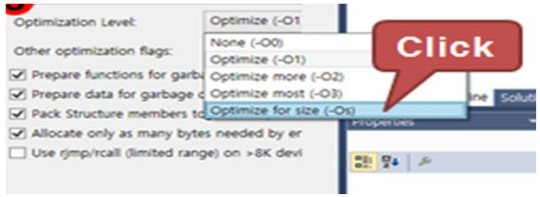
2)예제 : 타이머로 LED 점멸(2)

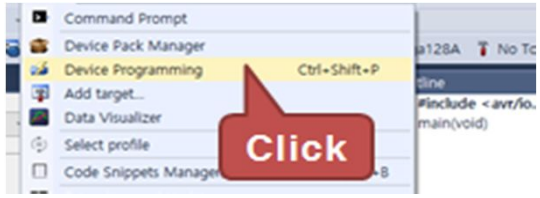
- 실습 개요
  - ATmega128A 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
  - 타이머를 이용하여 정확히 0.5초 마다 LED가 점멸하도록 함
  - 타이머는 **타이머/카운터 2의 일반 동작 모드**를 사용
  - LED0~3의 점멸을 좌우이동하여 반복

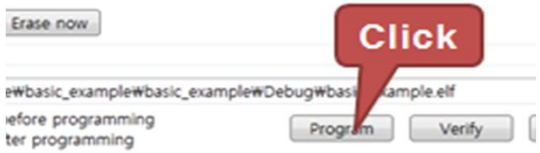
- 실습 목표
  - 타이머2의 동작원리 이해
  - 타이머2 제어 방법의 습득(관련 레지스터 이해)
  - 오버플로우 인터럽트 제어 프로그램 방법 습득

2)예제 : 타이머로 LED 점멸(2)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERLED_Example_02, 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)예제 : 타이머로 LED 점멸(2)

- 구동 프로그램
  - main.c 코드 작성

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

volatile unsigned char LED_Data = 0x01;
unsigned char timer2Cnt=0, Shift_Flag = 0;

SIGNAL(TIMER2_OVF_vect);               // Timer2 Overflow0 ISP

int main(void)
{
  DDRC = 0x0F;                         // 포트C 를 출력포트로 설정

  TCCR2 = 0x05;                        // 프리스케일러 1024로 설정

  // 256-144=112 -> 0.01초 마다 한번씩 인터럽트 발생
  TCNT0 = 112; TIMSK = 0x40;
  TIFR |=1 << TOV2;

  sei();
```

2)예제 : 타이머로 LED 점멸(2)

- 구동 프로그램
  - main.c 코드 작성

```
  while (1)
  {
    PORTC = LED_Data;      // 포트C로 변수 LED_Data에 있는 데이타 출력
  }
  return 0;
}

// 타이머 오버플로우 인터럽트
SIGNAL(TIMER2_OVF_vect)
{
  cli();
  TCNT2 = 112;      // 256-144=112 -> 0.01초 마다 한번씩 인터럽트 발생
  timer2Cnt++;      // timer2Cnt 변수를 1 증가

  // 0.01s * 50 = 0.5s, 0.5초를 얻기 위한 카운트 횟수
  if(timer2Cnt == 50)
  {
```

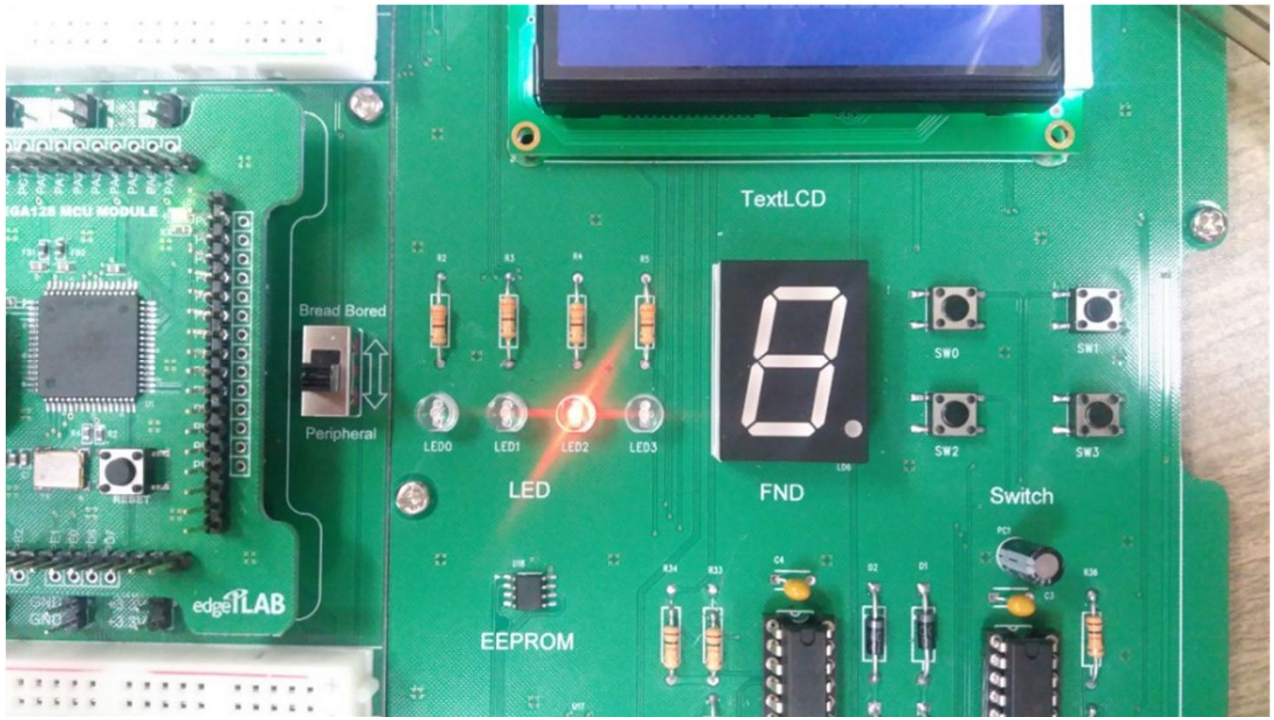
2)예제 : 타이머로 LED 점멸(2)

- 구동 프로그램
  - main.c 코드 작성

```
    if(Shift_Flag == 0)
    {
      // LED0 ~ LED3을 이동
      LED_Data <<= 1;          // LED_Data 변수를 좌측으로 쉬프트
      if(LED_Data == 0x08)     // LED3으로 이동하면
        Shift_Flag = 1;        // 우측으로 쉬프트하도록 설정
    }
    else
    {
      // LED3 ~ LED0으로 이동
      LED_Data >>= 1;          // LED_Data 변수를 우측으로 쉬프트
      if(LED_Data == 0x01)     // LED0으로 이동하면
        Shift_Flag = 0;        // 좌측으로 쉬프트하도록 설정
    }
    timer2Cnt=0;
  }
  sei();
}
```

2)예제 : 타이머로 LED 점멸(2)

- 실행 결과
  - LED의 불이 0.5초마다 좌우로 이동



3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

edgeiLAB

- 실습 개요
  - ATmega128A 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
  - 타이머를 이용하여 정확히 1초 마다 LED가 점멸하도록 함
  - 프리스케일러를 256으로 변경
  - 타이머는 **타이머/카운터 0의 일반 동작 모드**를 사용

- 실습 목표
  - 프리스케일러에 따른 타이머0의 동작원리 이해

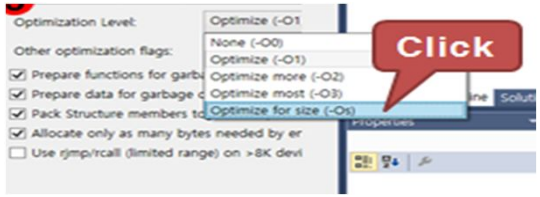
3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

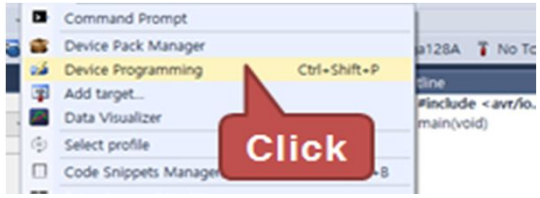
edgeILAB

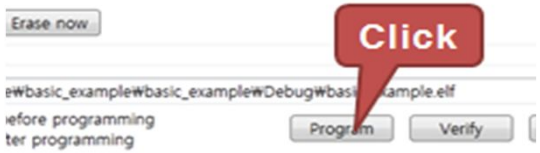
- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 8비트 타이머/카운터인 타이머/카운터 0를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCR 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러는 최대한 주기를 길게 하기 위해 **256**를 사용
    - ASSR /SFIOR는 디폴트, TCCR0의 CS비트를 "111"로 설정
  - 타이머 주기 결정
    - 1초를 만들기 위해 **2.5ms을 타이머 주기로 결정**
    - 인터럽트는 10ms마다 발생, 인터럽트가 **400번 발생하면 1초로 간주**
    - 10ms를 맞추기 위해서 세어야 하는 카운터 개수를 계산하고 그 카운터 개수를 만족시키기 위한 TCNT 초기값을 TCNT0에 넣음(112)
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 오버플로우 인터럽트를 Enable하고, TIFR 레지스터의 Timer/Counter0 overflow flag를 클리어

3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERLED_Application_01, 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)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

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

volatile unsigned char LED_Data = 0x00;
unsigned int timer0Cnt=0;

SIGNAL(TIMER0_OVF_vect);               // Timer0 Overflow0 ISP

int main(void)
{
  DDRC = 0x0F;                         // 포트C 를 출력포트로 설정

  TCCR0 = 0x06;                        // 프리스케일러 256

  // 256-144=112 -> 0.0025초 마다 한번씩 인터럽트 발생
  TCNT0 = 112;
  TIMSK = 0x01;
  TIFR |=1 << TOV0;
```

3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

```
  sei();

  while (1)
  {
    PORTC = LED_Data;    // 포트C로 변수 LED_Data에 있는 데이타를 출력
  }
  return 0;
}
```

3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

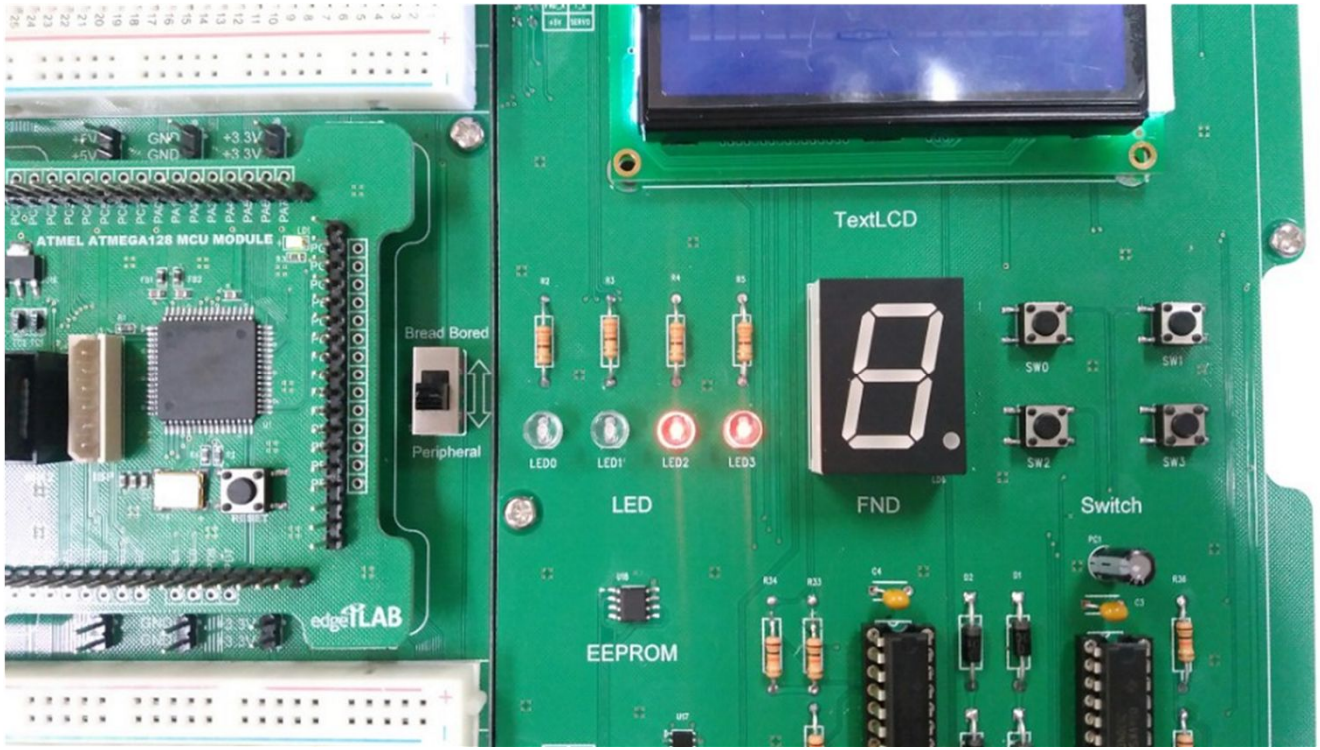
```
SIGNAL(TIMER0_OVF_vect)
{
  cli();

  // 256-144=112 -> 0.0025초 마다 한번씩 인터럽트 발생
  TCNT0 = 112;
  timer0Cnt++;                        // timer0Cnt 변수 1 증가

  // 0.0025s * 400 = 1s, 1초를 얻기 위한 카운트 횟수
  if(timer0Cnt == 400)
  {
    LED_Data++;                       // LED_Data 변수 1 증가
    if(LED_Data>0x0F) LED_Data = 0;
      timer0Cnt=0;
  }
  sei();
}
```

3)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(1)

- 실행 결과
  - LED의 불이 1초 마다 순차적으로 점멸



4)예제 : 타이머를 이용한 디지털 시계(1)

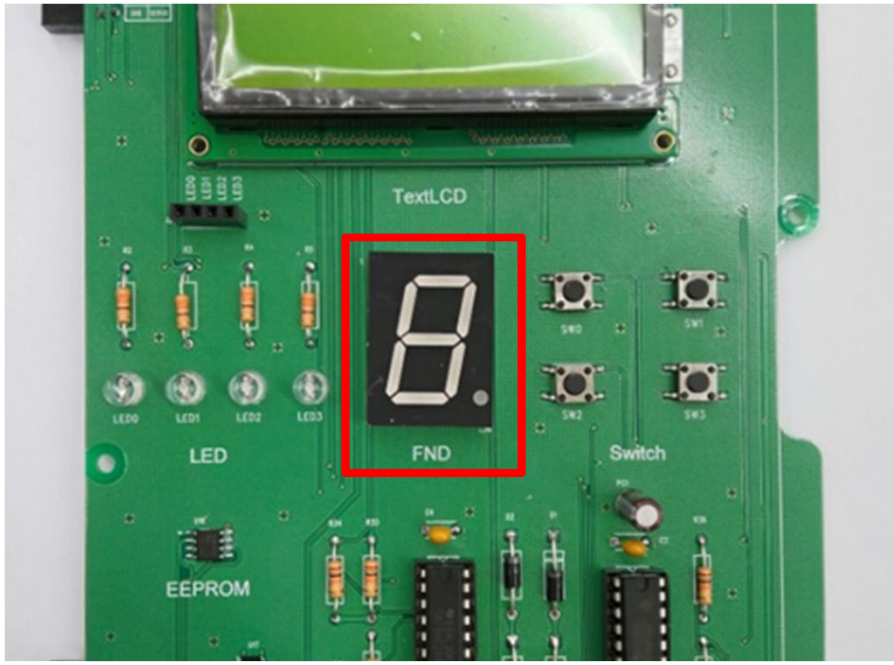
- 실습 개요
  - 타이머를 이용하여 디지털 시계의 기능을 설계
  - Array-FND 모듈에 마이크로 컨트롤러 출력 포트를 연결하고, 클럭을 이용하여 일정 카운트 기능을 수행
  - 타이머/카운터 0의 일반 모드 동작 사용

- 실습 목표
  - 타이머/카운터 활용 방법의 습득(관련 레지스터 이해)
  - 디지털 시계(초/분) 구현 방법 이해
  - **출력 비교** 인터럽트 제어 프로그램 방법 습득

4)예제 : 타이머를 이용한 디지털 시계(1)

- Edge-MCU AVR
  - FND는 A 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



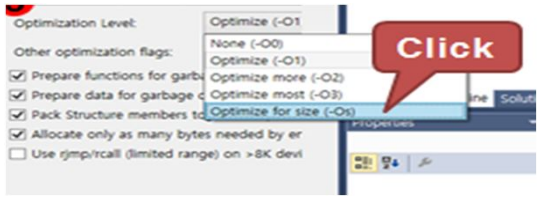
4)예제 : 타이머를 이용한 디지털 시계(1)

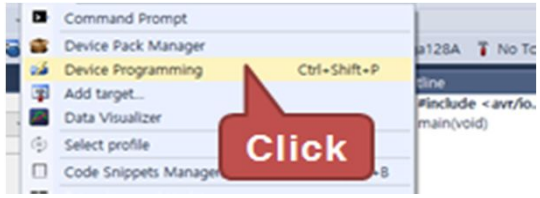
edgeiLAB

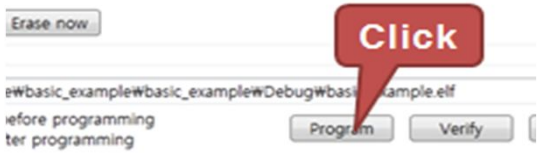
- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 8비트 타이머/카운터인 타이머/카운터 0를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCR 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러는 최대한 주기를 길게 하기 위해 1024를 사용
    - ASSR /SFIOR는 디폴트, TCCR0의 CS비트를 "111"로 설정
  - 타이머 주기 결정
    - 1초를 만들기 위해 10ms을 타이머 주기로 결정
    - 인터럽트는 10ms마다 발생, 인터럽트가 100번 발생하면 1초로 간주
    - 10ms를 맞추기 위해서 세어야 하는 카운터 개수를 계산하고 그 카운터 개수를 만족시키기 위한 TCNT 초기값을 TCNT0에 넣음(112)
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 **출력비교 인터럽트**를 Enable하고, TIFR 레지스터의 Timer/Counter0 Output Compare flag를 클리어

4)예제 : 타이머를 이용한 디지털 시계(1)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERFND_Example_01, 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)예제 : 타이머를 이용한 디지털 시계(1)

- 구동 프로그램
  - main.c 코드 작성

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

// 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};

volatile unsigned char time_s=0;        // 초를 세는 변수
unsigned char timer0Cnt=0;

int main(void)
{
  DDRA = 0xFF;                          // 포트A 를 출력포트로 설정

  TCCR0 = 0x07;                         // 프리스케일러 1024
  OCR0 = 144;                           // 0.01초 마다 한번씩 인터럽트 발생
  TIMSK = 0x02;                         // 출력비교 인터럽트 활성화
  TIFR |=1 << OCF0;
```

4)예제 : 타이머를 이용한 디지털 시계(1)

- 구동 프로그램
  - main.c 코드 작성

```
  sei();

  while (1)
  {
    PORTA = FND_DATA_TBL[time_s]; //포트A에 FND Table 값을 출력
  }
  return 0;
}
```

4)예제 : 타이머를 이용한 디지털 시계(1)

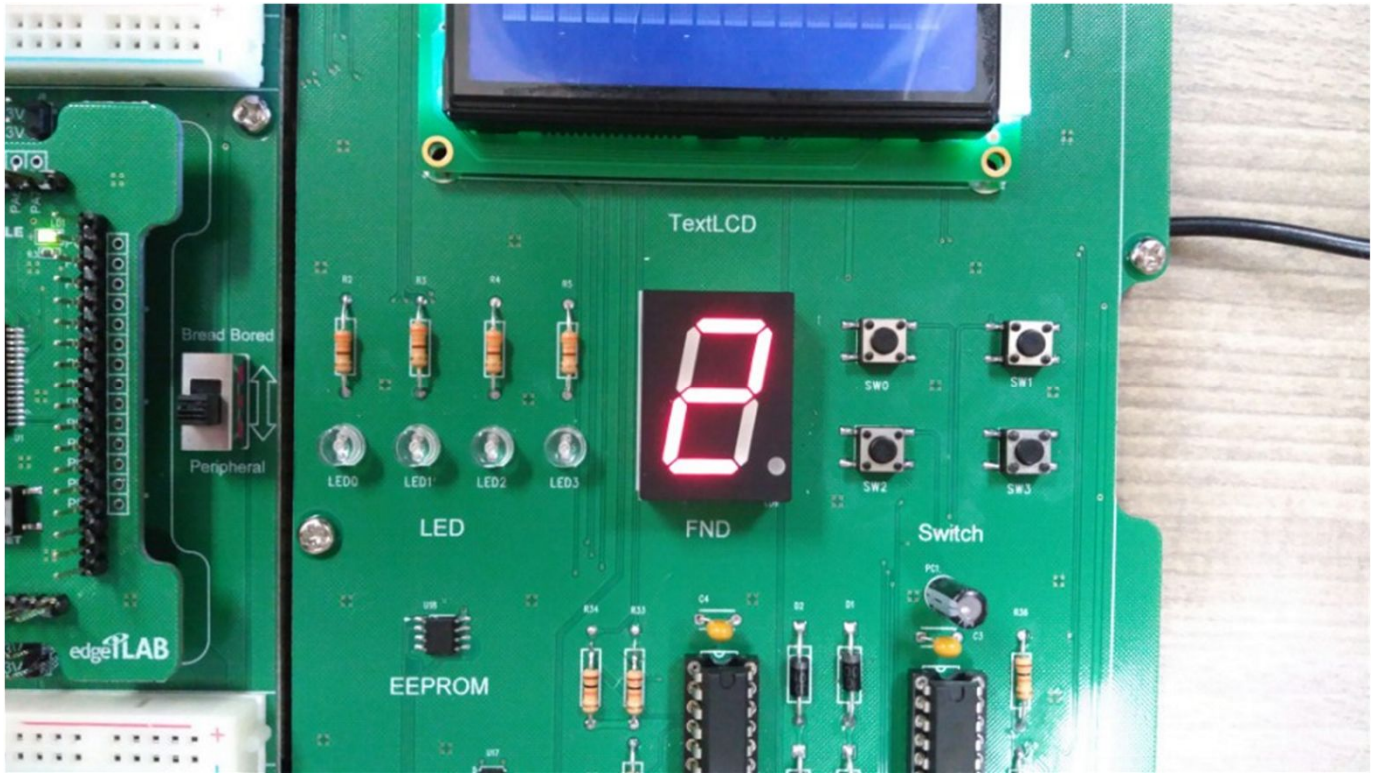
- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(TIMER0_COMP_vect)
{
  cli();
  OCR0 += 144;                          // 0.01초 후에 인터럽트 발생
  timer0Cnt++;                          // timer0Cnt 변수 1 증가

  // 0.01s * 100 = 1s                   // 1초를 얻기 위한 카운트 횟수
  if(timer0Cnt == 100)
  {
    if(time_s >= 16)
      time_s = 0;                       // time_s 변수는 16까지만 증가
    else time_s++;                      // 16되면 0으로 초기화
      timer0Cnt=0;
  }
  sei();
}
```

4)예제 : 타이머를 이용한 디지털 시계(1)

- 실행 결과
  - 1초마다 FND의 숫자가 증가('0' ~ '_' 까지)



5)응용 : 타이머와 인터럽트로 FND 점멸(1)

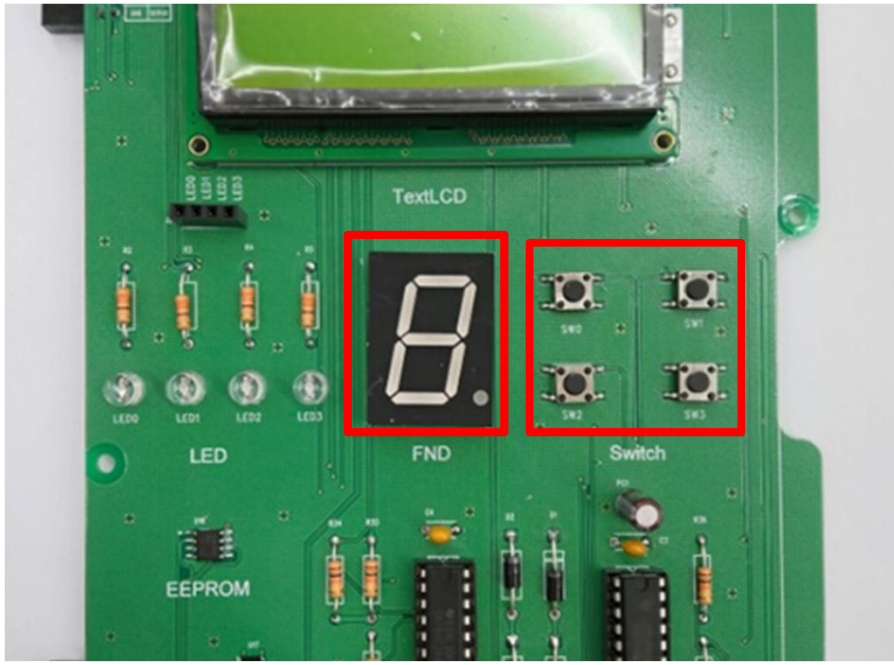
- 실습 개요
  - 타이머와 외부 인터럽트를 이용하여 디지털 시계의 기능을 설계
  - 타이머/카운터 2의 일반 모드 동작 사용
  - 출력 비교 인터럽트로 타이머 측정
  - 외부 인터럽트 4를 이용하여 동작 시작, 정지시키기

- 실습 목표
  - 타이머/카운터 활용 방법의 습득(관련 레지스터 이해)
  - 디지털 시계(초/분) 구현 방법 이해

5)응용 : 타이머와 인터럽트로 FND 점멸(1)

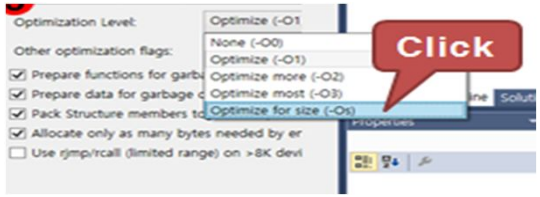
- Edge-MCU AVR
  - FND는 A 포트에 연결
  - SWITCH는 E 포트에 연결

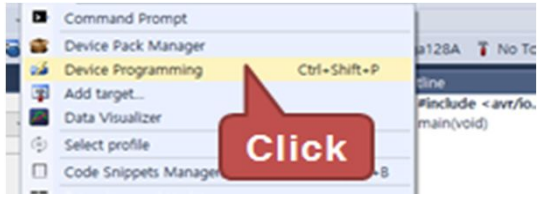
Sensor	GPIO
DATA_A	PA0
DATA_B	PA1
DATA_C	PA2
DATA_D	PA3
DATA_E	PA4
DATA_F	PA5
DATA_G	PA6
DATA_H	PA7
SW0	PE4(INT4)
SW1	PE5(INT5)
SW2	PE6(INT6)
SW3	PE7(INT7)

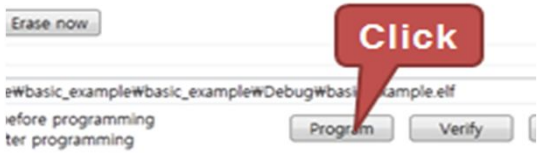


5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERINTFND_Application_01, 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







5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

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

// 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};

volatile unsigned char time_s=0;     // 0.5초를 세는 변수
volatile unsigned char Time_STOP=0; // 숫자 증가 및 정지시키는 변수

unsigned char timer0Cnt=0;

int main(void)
{
  DDRA = 0xFF;                      // 포트A를 출력포트로 설정
  DDRE = 0x00;                      // 포트E 를 입력포트로 설정
```

5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

```
  TCCR2 = 0x05;             // 프리스케일러 1024
  OCR2 = 144;               // 0.01초 마다 한번씩 인터럽트 발생
  TIMSK = 0x80;
  TIFR |=1 << OCF2;

  EICRB = 0x03;             // 인터럽트 4를 상승엣지에서 동작하도록 설정
  EIMSK = 0x10;             // 인터럽트 4를 허용
  EIFR = 0x10;              // 인터럽트 4 플래그를 클리어

  sei();

  while (1)
  {
    PORTA = FND_DATA_TBL[time_s]; // 포트A에 FND Table 값을 출력
  }
  return 0;
}
```

5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(TIMER2_COMP_vect)
{
  cli();                    // 전체 인터럽트를 금지
  OCR2 += 144;              // 0.01초 후에 인터럽트 발생
  timer0Cnt++;              // timer0Cnt 변수를 1 증가
  if(timer0Cnt == 50)
  {
    // 0.01s * 50 = 0.5s, 0.5초를 얻기 위한 카운트 횟수
    if(Time_STOP == 0)    // Time_Stop이 0인경우에만
    {
      if(time_s >= 16)
        time_s =0;          // time_s 변수는 16까지만 증가
      else
        time_s++;           // 16되면 0으로 초기화
    }
    timer0Cnt=0;
  }
  sei(); // 전체 인터럽트를 허용
}
```

5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

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

  if(Time_STOP == 0)       // Time_Stop이 0인 경우에만
    Time_STOP = 1;         // Time_Stop에 1을 입력
  else                     // Time_Stop이 1인경우에만
    Time_STOP = 0;         // Time_Stop에 0을 입력

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

5)응용 : 타이머와 인터럽트로 FND 점멸(1)

- 실행 결과
  - FND의 숫자가 500ms마다 증가하면서 켬
  - 1번 버튼을 누르면 FND가 멈추었다가 한번 더 누르면 다시 동작