타이머와 카운터(16비트)

- 16비트 타이머/카운터
- 타이머를 활용한 예제 및 응용 실습

엣지아이랩

16비트 타이머/카운터

- 16비트 타이머/카운터
  - 4개의 타이머/카운터 중 1번과 3번 타이머/카운터
  - **16비트의 카운터**를 보유
    - 2^{16} = 65536까지 셀 수 있음
  - 10비트 프리스케일러 내장
  - 입력 캡쳐 유닛 내장
  - 비교 매치에서 타이머 클리어(오토 리로드)
  - 3개의 PWM 출력
    - 가변 PWM 주기 파형 출력
    - 3개의 출력 비교 유닛 내장
  - T1, T3핀에 의한 카운터 동작
  - 10개의 인터럽트 소스
    - 오버플로우, 출력 비교 매치 A,B,C, 입력캡쳐

16비트 타이머/카운터

- 16비트 타이머/카운터 레지스터
  - 타이머/카운터 제어 레지스터(TCCRxA~C)
    - 타이머x의 동작 방식 설정
  - 타이머/카운터 레지스터(TCNTx)
    - 타이머x의 16비트 카운터 값을 저장
  - 출력 비교 레지스터(OCRxA~C)
    - TCNTx의 값과 출력 비교되기 위한 16비트 데이터 값을 저장
  - 입력 캡쳐 레지스터(ICRx)
    - 입력캡쳐시 TCNTx의 카운터 값을 저장
  - 인터럽트 관련
    - TIMSK(Timer Interrupt MaSK)
    - ETIMSK(Extended TIMSK)
    - TIFR(Timer Interrupt Flag Register)
    - ETIFR(Extended TIFR)

16비트 타이머/카운터

- TCCRnA(Timer/Counter Control Register nA)
  - 타이머/카운터 제어 레지스터 nA (n=1 or 3)
  - 타이머/카운터 1,3 의 동작을 설정
  - 비트 7:6 : COMnA1:0
  - 비트 5:4 : COMnB1:0
  - 비트 3:2 : COMnC1:0
  - 비트 1:0 : WGMn1:0

7	6	5	4	3	2	1	0
COMnA1	COMnA0	COMnB1	COMnB0	COMnC1	COMnC0	WGMn1	WGMn0

16비트 타이머/카운터

- TCCRnA(Timer/Counter Control Register nA)
  - COMnA1:0, COMnB1:0, COMnC1:0
    - 출력비교 핀 OCnA와 OCnB, OCnC를 제어

출력모드 비교, non-PWM

COMnA1/COMnB1/COMnC1	COMnA0/COMnB0/COMnC0	설 명
0	0	normal포트동작, OCnA/OCnB/OCnC분리
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (low level에서 출력)
1	1	Set OCnA/OCnB/OCnC on compare match (high level에서 출력)

16비트 타이머/카운터

- TCCRnA(Timer/Counter Control Register nA)

  출력모드 비교, Fast-PWM

COMnA1/COMnB1/COMnC1	COMnA0/COMnB0/COMnC0	설 명
0	0	normal포트 동작, OCnA/OCnB/OCnC분리
0	1	WGMn3:0=15:Toggle OCnA on compare match, OCnB disconnected(normal포트동작) For all other WGMn3:0 settings, normal 포트동작, OCnA/OCnB/OCnC disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA / OCnB / OCnC at TOP
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at TOP

16비트 타이머/카운터

- TCCRnA(Timer/Counter Control Register nA)

출력모드 비교, Phase correct and Phase and Frequency Correct PWM

COMnA1/COMnB1/COMnC1	COMnA0/COMnB0/COMnC0	설 명
0	0	normal포트 동작, OCnA/OCnB/OCnC분리
0	1	WGMn3:0=9 or 14:Toggle OCnA on compare match, OCnB disconnected(normal포트동작) For all other WGMn3:0 settings, normal 포트동작, OCnA/OCnB/OCnC disconnected
1	0	업 카운팅일때 Clear OCnA/OCnB/OCnC on compare match, 다운 카운팅일때 set OCnA / OCnB / OCnC at TOP
1	1	업 카운팅일때 Set OCnA/OCnB/OCnC on compare match, 다운 카운팅일때clear OCnA/OCnB/OCnC at TOP

16비트 타이머/카운터

- TCCRnA(Timer/Counter Control Register nA)
  - 비트1:0 : WGMn1:0 : 15가지의 동작 모드를 결정

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	타이머/카운터의 1,3 동작 모드	TOP	OCR1x 업데이트	TOV1플래그 Set 시점
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03Ff	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM 10-bit	0x03Ff	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

16비트 타이머/카운터

- TCCRnB(Timer/Counter Control Register nB)
  - 타이머/카운터 제어 레지스터 nB (n=1 or 3)
  - 타이머/카운터1, 3의 프리스케일러 등을 설정하는 기능 수행
  - 비트 7 : ICNCn
  - 비트 6 : ICESn
  - 비트 4:3 : WGMn3:2
  - 비트 2:0 : CSn2:0, -클럭 선택

7	6	5	4	3	2	1	0
ICNCn	ICESn	-	WGMn3	WGMn2	CSn2	CSn1	CSn0

16비트 타이머/카운터

- TCCRnB(Timer/Counter Control Register nB)
  - 비트 7 : ICNCn
    - 1로 세트하면 Input Capture Noise Canceler 설정
    - 입력 캡처 핀(ICn)의 입력을 필터링
    - 4개의 오실레이터 사이클 만큼 지연
  - 비트 6 : ICESn
    - ICn에 해당되는 에지의 형태를 선택
    - 1로 설정하면 상승에지에서 검출, 0으로 설정하면 하강에지에서 검출
    - 카운터 값은 ICRn에 저장되고, 입력 캡처 플래그(ICFn)가 설정된 경우 입력 캡처 인터럽트가 발생
  - 비트 4:3 : WGMn3:2
    - TCCRxA의 비트1~0(WGMx1~0)와 결합하여 동작모드를 설정

16비트 타이머/카운터

- TCCRnB(Timer/Counter Control Register nB)
  - 비트 2:0 : CSn2:0
  - 분주비와 클럭소스를 선택

CSn2	CSn1	CSn0	설명
0	0	0	클럭소스 없음(타이머/카운터가 멈춤)
0	0	1	클럭소스 존재(프리스케일링이 없음)
0	1	0	8분주
0	1	1	64분주
1	0	0	256분주
1	0	1	1024분주
1	1	0	T1핀에서 외부클럭소스, 하강에지에서 클럭 발생
1	1	1	T1핀에서 외부클럭소스, 상승에지에서 클럭 발생

16비트 타이머/카운터

- TCCRnC(Timer/Counter Control Register nC)
  - 타이머/카운터 제어 레지스터 nC (n=1 or 3)
  - 타이머/카운터 1, 3의 Force Output Compare를 설정
  - non-PWM모드일 경우에만 활성화
  - 1로 설정하면 compare match가 파형 발생 장치로 되어, OCnA/OCnB/OCnC에 출력비교가 일치할 때 출력되는 값과 동일한 출력을 내보냄
  - 비트 7 : FOCnA
  - 비트 6 : FOCnB
  - 비트 5 : FOCnC

7	6	5	4	3	2	1	0
FOCnA	FOCnB	FOCnC	-	-	-	-	-

16비트 타이머/카운터

- TCNTn (Timer Counter Rgister n)
  - 타이머 카운터 레지스터 n (n=1 or 3)
  - 타이머/카운터1, 3의 16비트 카운터 값을 저장하고 있는 레지스터
    - 읽기 및 쓰기가 가능한 카운터로 동작하며 자동으로 증가
    - 16비트 레지스터 값을 저장
    - TCNTnH와 TCNTnL로 구성

15	14	13	12	11	10	9	8
TCNTn15	TCNTn14	TCNTn13	TCNTn12	TCNTn11	TCNTn10	TCNTn9	TCNTn8
7	6	5	4	3	2	1	0
TCNTn7	TCNTn6	TCNTn5	TCNTn4	TCNTn3	TCNTn2	TCNTn1	TCNTn0

16비트 타이머/카운터

- OCRnA, OCRnB, OCRnC (Output Compare Register)
  - 출력 비교 레지스터(TCNT1/3와 계속적으로 비교)
  - 16비트
  - 두 레지스터의 값이 일치했을 때, OCnA, OCnB, OCnC 핀을 통하여 설정된 값이 출력되거나 출력 비교 인터럽트가 발생

- ICRn(Input Capture Register)
  - 입력캡쳐 레지스터
  - 입력캡쳐핀 ICx으로 들어오는 신호변화를 검출하여 일어나는 입력캡쳐시 TCNTx의 카운터 값을 저장하는 16비트 레지스터
  - 이때 ICFx 플래그가 세트되고 입력캡쳐 인터럽트가 요청
  - 어떤 신호의 주기 측정에 응용

16비트 타이머/카운터

- TIMSK(Timer Interrupt MaSK)
  - 타이머 인터럽트 마스크 레지스터
  - 타이머/카운터0, 타이머/카운터1, 타이머/카운터2가 발생하는 인터럽트를 개별적으로 enable하는 레지스터
  - 비트 5 : TICIE1
  - 비트 4:3 : OCIE1A~B
  - 비트 2 : TOIE1

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

16비트 타이머/카운터

- TIMSK(Timer Interrupt MaSK)
  - TICIE1 : Timer Input Capture Interrupt Enable 1
    - 1로 설정되면 타이머1의 입력캡쳐 인터럽트를 개별적으로 Enable
    - IC1 핀에서 캡쳐 트리거 이벤트가 발생했을 경우, TIFR.ICF1 플래그가 세트되면서 인터럽트 서비스 루틴이 실행
  - OCIE1A~B : Output Compare match Interrupt Enable timer 1 A~B
    - 1로 설정되면 타이머1의 출력비교 인터럽트 A, B를 개별적으로 Enable
    - TCNT1과 OCR1A/B의 값이 일치하면, TIFR.OCF1A/B 플래그가 세트되면서 인터럽트 서비스 루틴이 실행
  - TOIE1 : Timer Overflow Interrupt Enable for timer 1
    - 1로 설정되면 타이머1의 오버플로우 인터럽트를 개별적으로 Enable
    - 타이머/카운터1의 오버플로우가 발생시 TIFR.TOV1 플래그가 세트되면서 인터럽트 서비스 루틴이 실행

16비트 타이머/카운터

- ETIMSK(Extended Timer Interrupt MaSK)
  - 확장 타이머 인터럽트 마스크 레지스터
  - 타이머1, 3이 발생하는 인터럽트를 개별적으로 Enable 제어하는 레지스터
  - 비트 5 : TICIE3
  - 비트 4:3 : OCIE3A~B
  - 비트 2 : TOIE3
  - 비트 1:0 : OCIExC

7	6	5	4	3	2	1	0
-	-	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C

16비트 타이머/카운터

- ETIMSK(Extended Timer Interrupt MaSK)
  - TICIE3 : Input Capture Interrupt Enable 3
    - 1로 설정되면 타이머3의 입력캡쳐 인터럽트를 개별적으로 Enable
    - IC3 핀에서 캡쳐 트리거 이벤트가 발생했을 경우, TIFR.EICF3 플래그가 세트되면서 인터럽트 서비스 루틴 실행
  - OCIE3A~B : Output Compare match Interrupt Enable timer 3 A~B
    - 1로 설정되면 타이머3의 출력비교 인터럽트 A, B를 개별적으로 Enable
    - TCNT3과 OCR3A/B의 값이 일치하면, ETIFR.OCF3A/B 플래그가 세트되면서 인터럽트 서비스 루틴 실행
  - TOIE3 : Timer Overflow Interrupt Enable for timer 1
    - 1로 설정되면 타이머3의 오버플로우 인터럽트를 개별적으로 Enable.
    - 타이머/카운터3의 오버플로우가 발생시 ETIFR.TOV3 플래그가 세트되면서 인터럽트 서비스 루틴이 실행
  - OCIExC : Output Compare match Interrupt Enable timer x C
    - 1로 설정되면 타이머x의 출력비교 인터럽트 C를 개별적으로 Enable.
    - TCNT3/1과 OCR3C/OCR1C의 값이 일치하면, ETIFR.OCF3C/ETIFR.OCF1C 플래그가 세트되면서 인터럽트 서비스 루틴이 실행

16비트 타이머/카운터

- TIFR(Timer Interrupt Frag Register)
  - 타이머 인터럽트 플래그 레지스터
  - 타이머 0~2가 발생하는 인터럽트 플래그를 저장하는 레지스터
  - 비트 5 : ICF1(Input Capture Flag 1)
  - 비트 4:3 : OCF1A~B(Output Compare match Flag 1 A~B)
  - 비트 2 : TOV1 (Timer Overflow Flag 1)

7	6	5	4	3	2	1	0
-	-	ICF1	OCF1A	OCF1B	TOV1	-	-

16비트 타이머/카운터

- TIFR(Timer Interrupt Frag Register)
  - ICF1 : Input Capture Flag 1
    - 입력캡쳐신호 또는 아날로그 비교기로부터의 신호에 의해 캡쳐 동작이 수행될 때 1로 세트되고, 입력캡쳐 인터럽트가 발생
    - ICR1 레지스터가 TOP 값으로 사용되는 동작 모드에서 TCNT1의 값이 TOP과 같아질 때 1로 세트되고 인터럽트가 발생
  - OCF1A~B : Output Compare match Flag 1 A~B
    - TCNT1레지스터와 출력비교 레지스터 OCR1A~B의 값을 비교하여 같으면 OCF1A~B는 1로 세트되고 출력비교 인터럽트가 발생
  - TOV1 : Timer Overflow Flag 1
    - 오버플로우가 발생하면(0xFFFF까지 세고 0x0000으로 넘어가게 되면) 이 TOV1는 1로 세트되면서 오버플로우 인터럽트가 발생
    - Phase correct PWM 모드에서는 타이머1이 0x00에서 계수방향을 바꿀 때 TOV1가 세트됨

16비트 타이머/카운터

- ETIFR(Extended Timer Interrupt Frag Register)
  - 타이머 인터럽트 플래그 레지스터
  - 타이머 1,3이 발생하는 인터럽트 플래그를 저장하는 레지스터
  - 비트 5 : ICF3(Input Capture Flag 3)
  - 비트 4,3,1 : OCF3A,B,C (Output Compare match Flag 3 A,B,C)
  - 비트 2 : TOV3 (Timer Overflow Flag 3)
  - 비트 0 : OCF1C: Output Compare match Flag 1 C

7	6	5	4	3	2	1	0
-	-	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C

16비트 타이머/카운터

- ETIFR(Extended Timer Interrupt Frag Register)
  - ICF3 : Input Capture Flag 3
    - 입력캡쳐신호 또는 아날로그 비교기로부터의 신호에 의해 캡쳐 동작이 수행될 때 1로 세트되고 입력캡쳐 인터럽트 발생
    - ICR3 레지스터가 TOP 값으로 사용되는 동작 모드에서 TCNT3의 값이 TOP과 같아질 때 1로 세트되고 인터럽트 발생
  - OCF3A,B,C : Output Compare match Flag 3 A,B,C
    - TCNT3레지스터와 출력비교 레지스터 OCR3A~C의 값을 비교하여 같으면 OCF3A~C는 1로 세트되고 출력비교 인터럽트 요청
  - TOV3 : Timer Overflow Flag 3
    - 오버플로우가 발생하면 TOV3는 1로 세트되면서 오버플로우 인터럽트 발생
    - PC PWM 모드에서는 타이머1이 0x00에서 계수방향을 바꿀 때 TOV3가 세트
  - OCF1C : Output Compare match Flag 1 C
    - TCNT1레지스터와 출력비교 레지스터 OCR1C의 값을 비교하여 같으면 OCF1C는 1로 세트되고 출력비교 인터럽트 발생

1)예제 : 타이머로 LED 점멸(3)

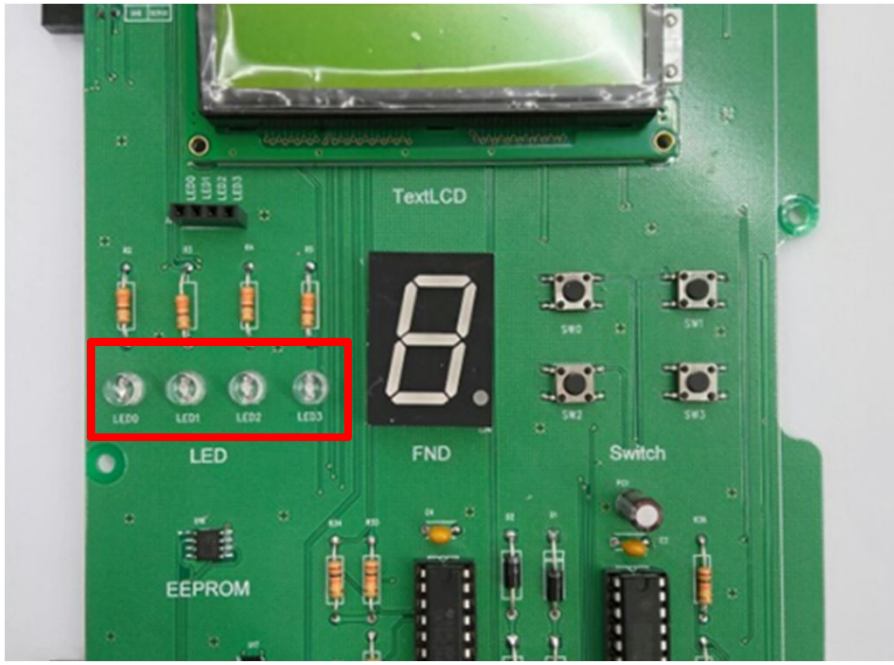
- 실습 개요
  - ATmega128A 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
  - 타이머를 이용하여 정확히 1초 마다 LED가 점멸하도록 함
  - 타이머는 타이머/카운터 1의 일반 동작 모드를 사용

- 실습 목표
  - 타이머1의 동작원리 이해
  - 타이머1 제어 방법의 습득(관련 레지스터 이해)
  - 오버플로우 인터럽트 제어 프로그램 방법 습득

1)예제 : 타이머로 LED 점멸(3)

- Edge-MCU AVR
  - LED는 C 포트에 연결

Sensor	GPIO
LED0	PC0
LED1	PC1
LED2	PC2
LED3	PC3

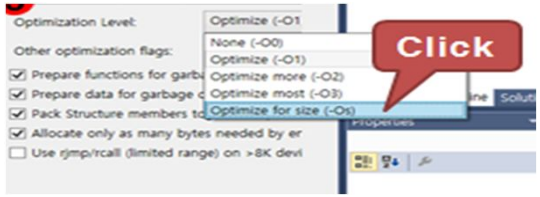


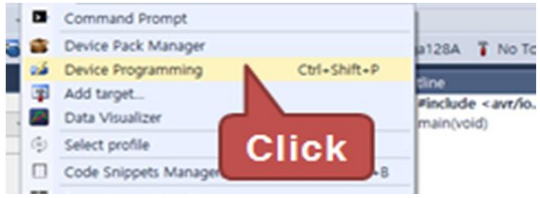
1)예제 : 타이머로 LED 점멸(3)

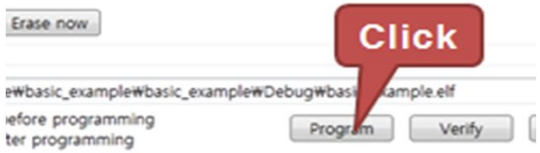
- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 16비트 타이머/카운터인 타이머/카운터 1를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCRxA~C 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러를 "1"로 설정
    - TCCRxB의 CS비트를 "001"로 설정
  - 타이머 주기 결정
    - 65536개의 카운터 개수를 계산할 수 있으므로, 65536/14745600 = 1/255 초마다 인터럽트 발생
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 오버플로우 인터럽트를 Enable하고, TIFR 레지스터의 Timer/Counter1 overflow flag를 클리어

1)예제 : 타이머로 LED 점멸(3)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERLED_Example_03, 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 timer1Cnt=0;

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

  TCCR1A = 0x00;
  TCCR1B = 0x01;                       // 프리스케일러 1

  TCNT1 = 0;                  // 0 -> 1/225초 마다 한번씩 인터럽트 발생
  TIMSK = 0x04;
  TIFR |=1 << TOV1;

  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
  }
  return 0;
}
```

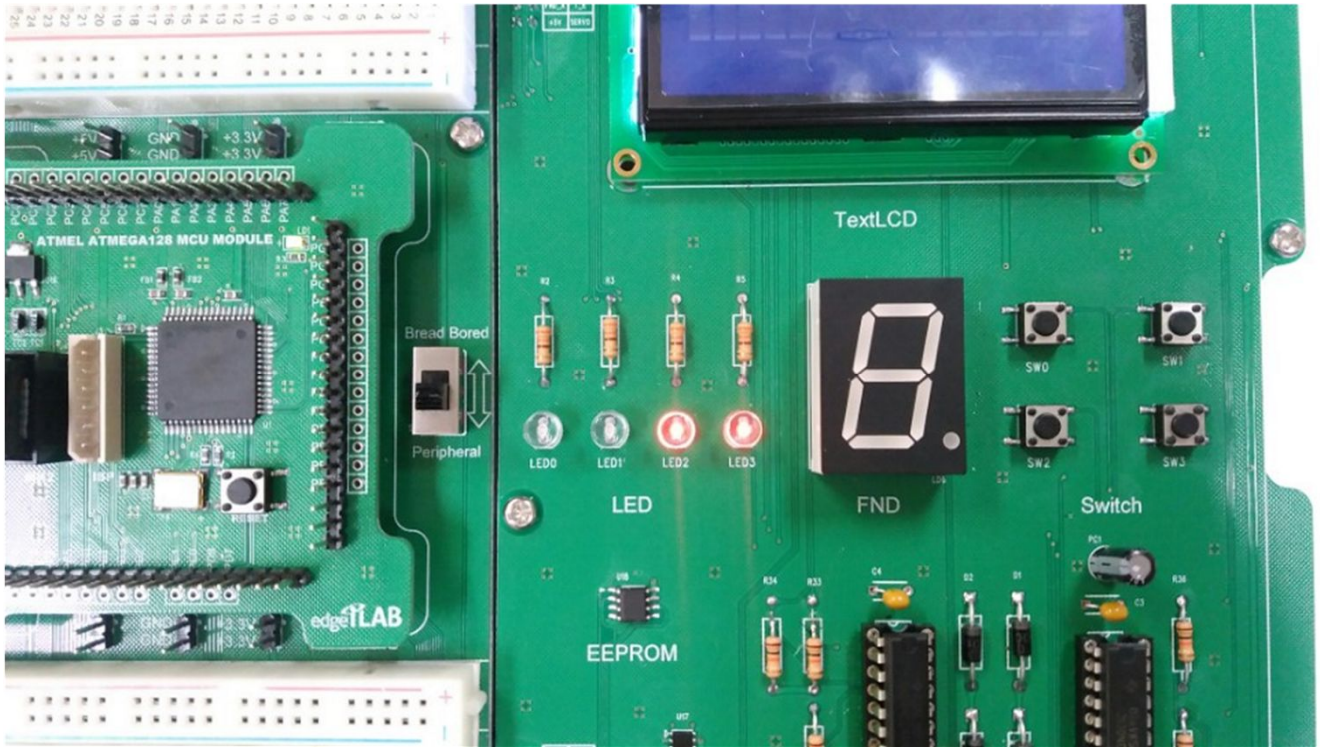
1)예제 : 타이머로 LED 점멸(1)

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(TIMER1_OVF_vect)
{
  cli();

  timer1Cnt++;                              // timer0Cnt 변수 1 증가

  // 1/255s * 255 = 1s, 1초를 얻기 위한 카운트 횟수
  if(timer1Cnt == 225)
  {
    LED_Data++;                            // LED_Data 변수 1 증가

    if(LED_Data>0x0F)
      LED_Data = 0;

    timer1Cnt=0;
  }
  sei();
}
```

1)예제 : 타이머로 LED 점멸(1)

- 실행 결과
  - LED의 불이 1초 마다 순차적으로 점멸



2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

edgeILAB

- 실습 개요
  - ATmega128A 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
  - 타이머를 이용하여 정확히 1초 마다 LED가 점멸하도록 함
  - 프리스케일러를 1024으로 변경
  - 타이머는 **타이머/카운터 3의 일반 동작 모드**를 사용

- 실습 목표
  - 프리스케일러에 따른 타이머3의 동작원리 이해

2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

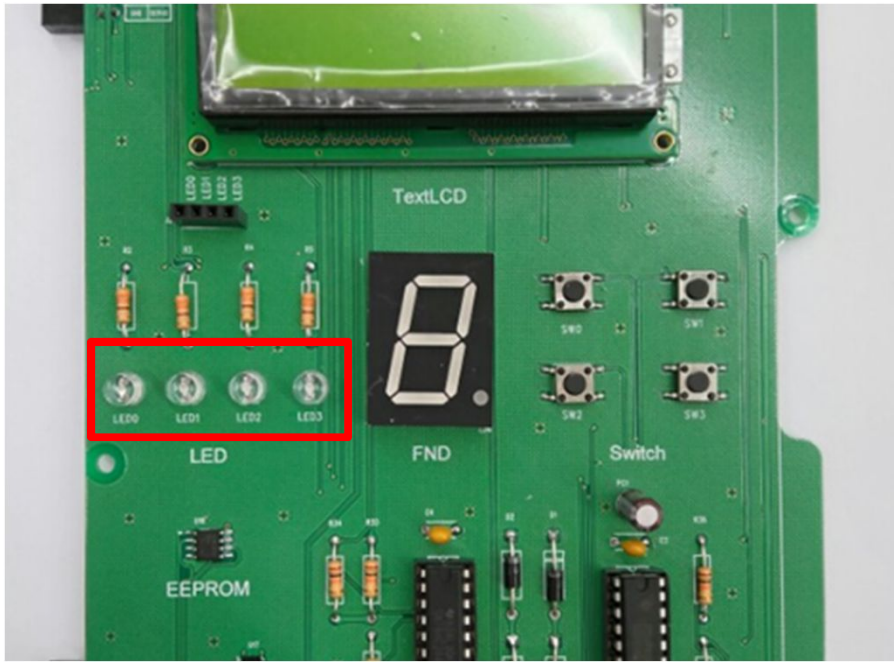
edgeILAB

- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 16비트 타이머/카운터인 타이머/카운터 3를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCRxA~C 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러를 "1024"로 설정
    - TCCRxB의 CS비트를 "101"로 설정
  - 타이머 주기 결정
    - 타이머 클럭 주파수가 14400Hz(=14745600/1024)이므로, 카운터 레지스터의 시작위치를 51136(=65536-14400)로 설정하면 인터럽트 발생
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 오버플로우 인터럽트를 Enable하고, TIFR 레지스터의 Timer/Counter3 overflow flag를 클리어

2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

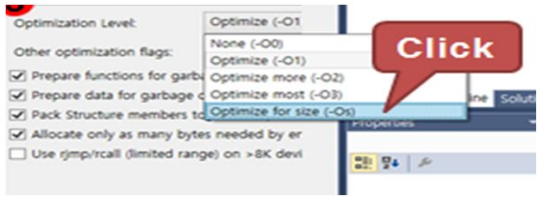
- Edge-MCU AVR
  - LED는 C 포트에 연결

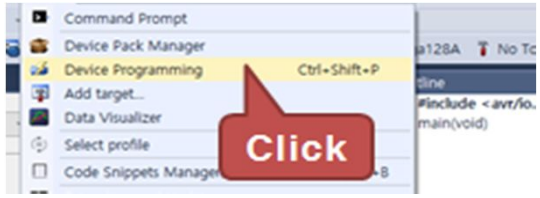
Sensor	GPIO
LED0	PC0
LED1	PC1
LED2	PC2
LED3	PC3

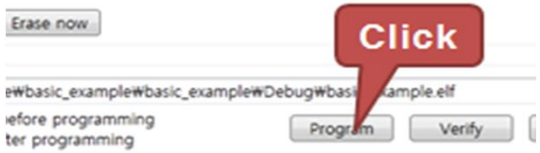


2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERLED_Application_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)응용 : 프리스케일러를 변경하여 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 = 0x00;

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

  TCCR3A = 0x00;
  TCCR3B = 0x05;                      // 프리스케일러 1024;

  // 65536 - 14400 = 51136-> 1초 마다 한번씩 인터럽트 발생
  TCNT3 = 51136;
  ETIMSK = 0x04;
  ETIFR |=1 << TOV3;

  sei();
```

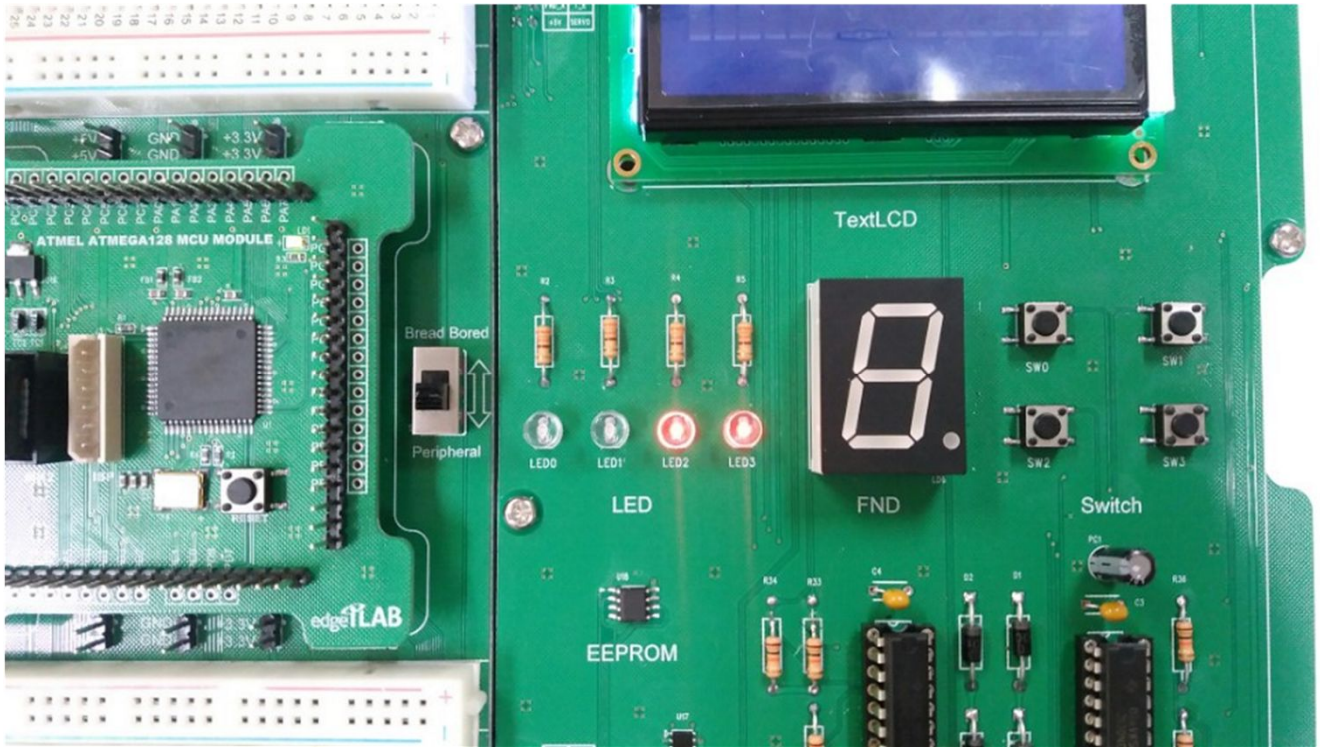
2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

- 구동 프로그램
  - main.c 코드 작성

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

SIGNAL(TIMER3_OVF_vect)
{
  cli();

   // 65536 - 14400 = 51136-> 1초 마다 한번씩 인터럽트 발생
  TCNT3 = 51136;
  LED_Data++;              // LED_Data 변수1 증가

  if(LED_Data>0x0F)
    LED_Data = 0;
  sei();
}
```

2)응용 : 프리스케일러를 변경하여 타이머로 LED 점멸(2)

- 실행 결과
  - LED의 불이 1초 마다 순차적으로 점멸



3)예제 : 타이머를 이용한 디지털 시계(2)

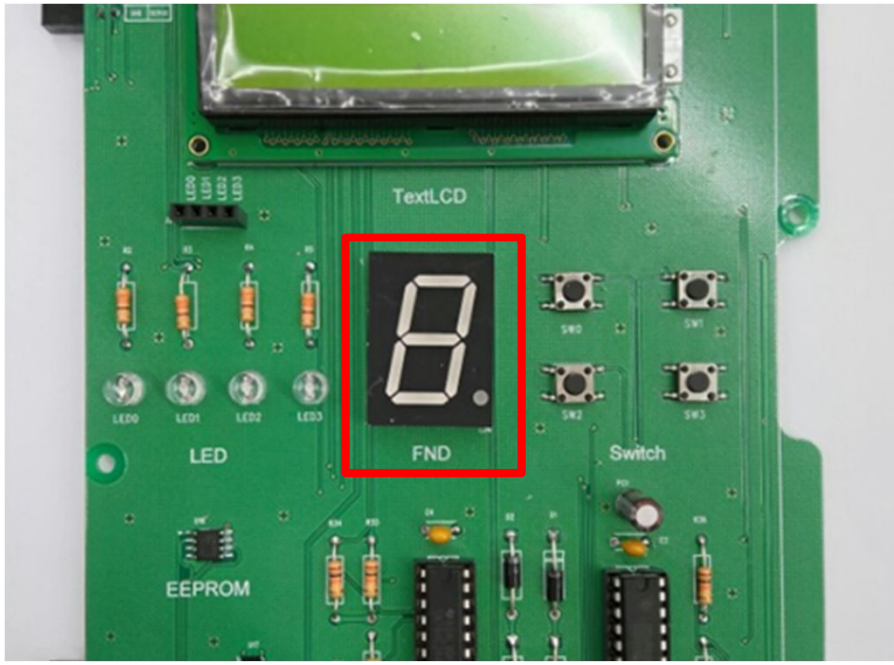
- 실습 개요
  - 타이머를 이용하여 디지털 시계의 기능을 설계
  - Array-FND 모듈에 마이크로 컨트롤러 출력 포트를 연결하고, 클럭을 이용하여 일정 카운트 기능을 수행
  - 타이머/카운터 1의 일반 모드 동작 사용

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

3)예제 : 타이머를 이용한 디지털 시계(2)

- 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



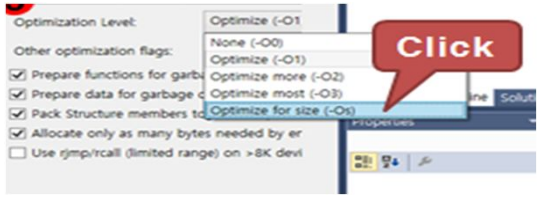
3)예제 : 타이머를 이용한 디지털 시계(2)

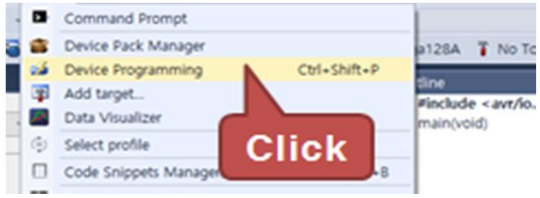
edgeiLAB

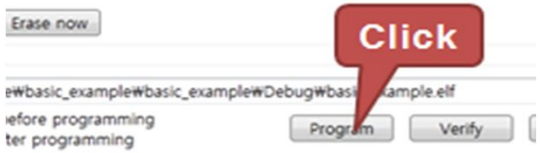
- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 16비트 타이머/카운터인 타이머/카운터 1를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCRxA~C 레지스터의 CS를 제외한 모든 비트들을 0으로 세트
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러를 "1024"로 설정
    - TCCRxB의 CS비트를 "101"로 설정
  - 타이머 주기 결정
    - 타이머 클럭 주파수가 14400Hz(=14745600/1024)이므로, 카운터 레지스터의 시작위치를 51136(=65536-14400)로 설정하면 인터럽트 발생
  - 인터럽트 인에이블
    - TIMSK 레지스터를 설정하여 **출력비교 인터럽트를 Enable**하고, TIFR 레지스터의 Timer/Counter1 Output Compare flag를 클리어

3)예제 : 타이머를 이용한 디지털 시계(2)

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







3)예제 : 타이머를 이용한 디지털 시계(2)

- 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

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

  TCCR1A = 0x00;
  TCCR1B = 0x05;                       // 프리스케일러 1024

  OCR1A = 14400;                      // 1초 마다 한번씩 인터럽트 발생
  TIMSK = 0x10;
  TIFR |=1 << OCF1A;
```

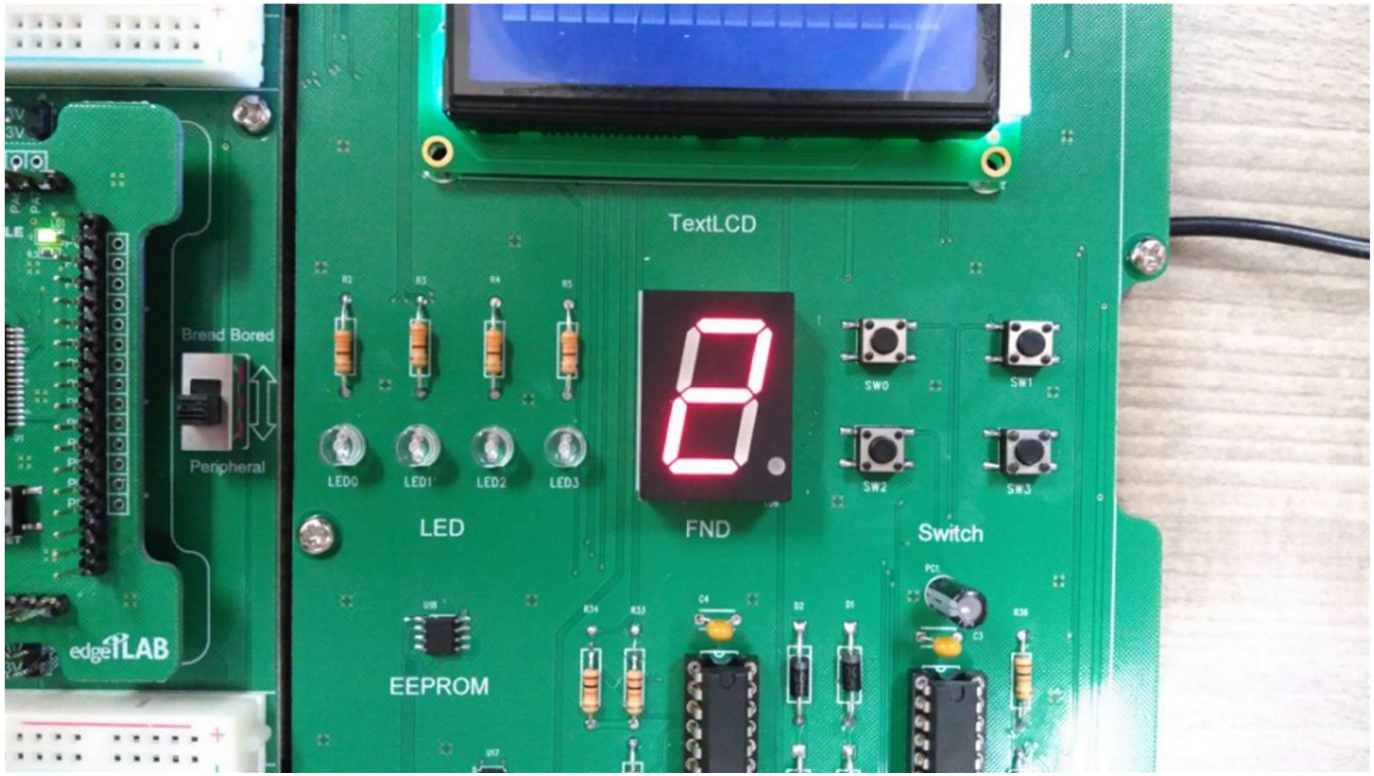
3)예제 : 타이머를 이용한 디지털 시계(2)

- 구동 프로그램
  - main.c 코드 작성

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

SIGNAL(TIMER1_COMPA_vect) {
  cli();

  OCR1A += 14400;                 // 1초 후에 인터럽트 발생
  if(time_s >= 17)
    time_s =0;                    // time_s 변수는 17까지만 증가
  else
    time_s++;                     // 17되면 0으로 초기화

  sei();
}
```

3)예제 : 타이머를 이용한 디지털 시계(2)

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



4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 실습 개요
  - **타이머3의 입력캡처 인터럽트**를 이용하여 스위치를 누른 시간을 FND에 출력
  - 타이머는 **타이머/카운터 3의 일반 동작 모드**를 사용

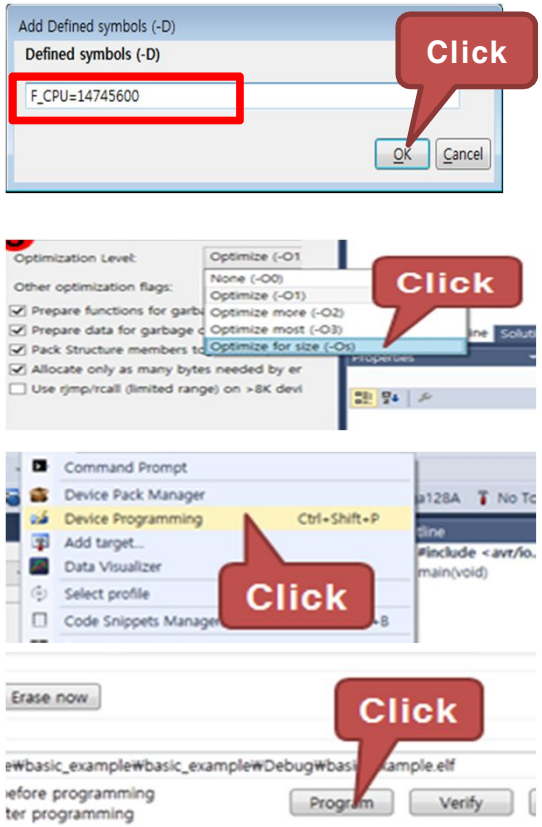
- 실습 목표
  - 프리스케일러에 따른 타이머3의 동작원리 이해

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
  - 사용할 타이머/카운터 결정
    - 여기서는 16비트 타이머/카운터인 타이머/카운터 3를 사용
  - 동작 모드 결정
    - 여기서는 일반 동작 모드 사용
    - TCCRxB 레지스터 중 ICx(입력캡쳐 핀)의 트리거를 설정하고 CS를 이용해 프리스케일러 설정
  - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
    - 여기서는 내부 클럭(14.7456MHz)을 사용
    - 프리스케일러를 "1024"로 설정
    - TCCRxB의 CS비트를 "101"로 설정
  - 인터럽트 인에이블
    - ETIMSK 레지스터를 설정하여 입력캡처 인터럽트를 Enable하고, ETIFR 레지스터의 Timer/Counter3 Input Capture flag를 클리어

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

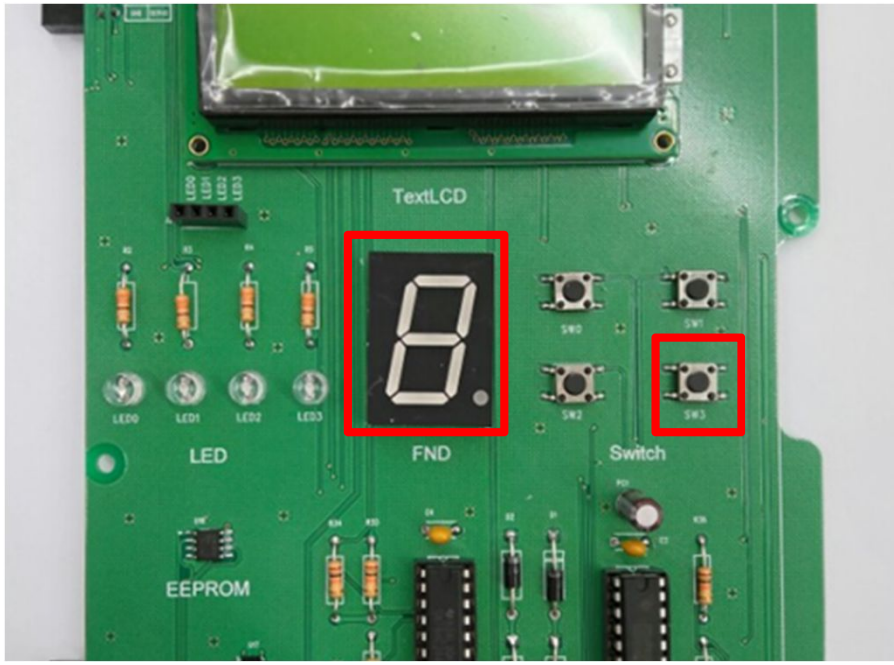
- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 06_TIMERCLOCK_Application, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compier에서
    - Symbols -> F_CPU=14745600 추가
    - Optimization -> 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 : 타이머를 이용한 스위치 누른시간 구하기

- 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
SW3	PE7(INT7)



4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 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
volatile unsigned char FND_flag=0, edge_flag = 0;

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

  TCCR3A = 0x00;
```

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 구동 프로그램
  - main.c 코드 작성

```
  // 프리스케일러 1024, 상승(양)에지 캡처 트리거 설정
  TCCR3B = 0x45;
  ETIMSK = 0x20;                        // 입력캡처 인터럽트3 활성화
  ETIFR |=1 << ICF3;

  sei();
  PORTA = FND_DATA_TBL[0]; //포트A에 FND Table의 값을 출력
  while (1) {
    if(FND_flag) {
      if(time_s > 15)                   // 최대 1.5초까지 표시(F)
        time_s = 15;

      PORTA = FND_DATA_TBL[time_s];  // 포트A에 FND Table 값 출력
      FND_flag = 0;
    }
  }
  return 0;
}
```

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(TIMER3_CAPT_vect) {
  cli();

  unsigned int count_check;

  // 스위치가 눌릴 시간 측정을 위해
  //상승에지에서 하강에지까지의 시간을 계산

  if(edge_flag == 0) { // 상승 에지(스위치를 누르면)

    // 프리스케일러 1024, 하강(음)에지 캡처 트리거 설정
    TCCR3B = 0x05;
    TCNT3 = 0;                          // TCNT3 레지스터를 0으로 초기화
    ICR3 = 0;                           // ICR3 레지스터를 0으로 초기화
    edge_flag = 1;
  }
```

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 구동 프로그램
  - main.c 코드 작성

```
else { // 하강 에지(스위치를 떼면)
    // 프리스케일러 1024, 상승(양)에지 캡처 트리거 설정
    TCCR3B = 0x45;
    count_check = ICR3;

    // 14745600/1024 = 14400 Hz, 1초동안 TCNT의 값은 14400
    time_s = count_check/1440;     // 누를 시간을 0.1초 단위로 변경

    // 측정 시간 FND로 출력 0 ~ 1.5초 까지 측정 가능
    FND_flag = 1;
    edge_flag = 0;
  }
  sei();
}
```

4)응용 : 타이머를 이용한 스위치 누른시간 구하기

- 실행 결과
  - SW3을 "0~1.5초"까지 눌렀다 뗐을 때, FND에는 "0~F"까지 표시