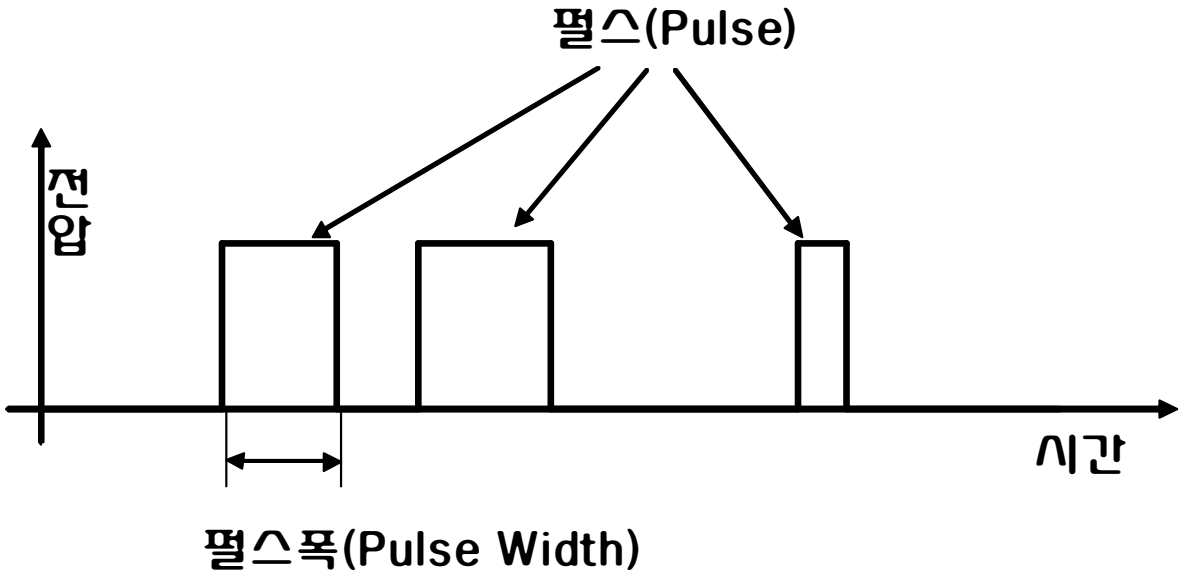
타이머와 PWM

- PWM(Pulse Width Modulation)
- 8비트 타이머/카운터의 동작모드
- 16비트 타이머/카운터
- 16비트 타이머/카운터 동작모드
- PWM으로 LED 밝기 조절하기
- 스위치 입력 값에 따라 LED 밝기 조절 하기
- PWM으로 PIEZO 울리기
- 스위치 입력 값에 따라 PIEZO 울리기

엣지아이랩

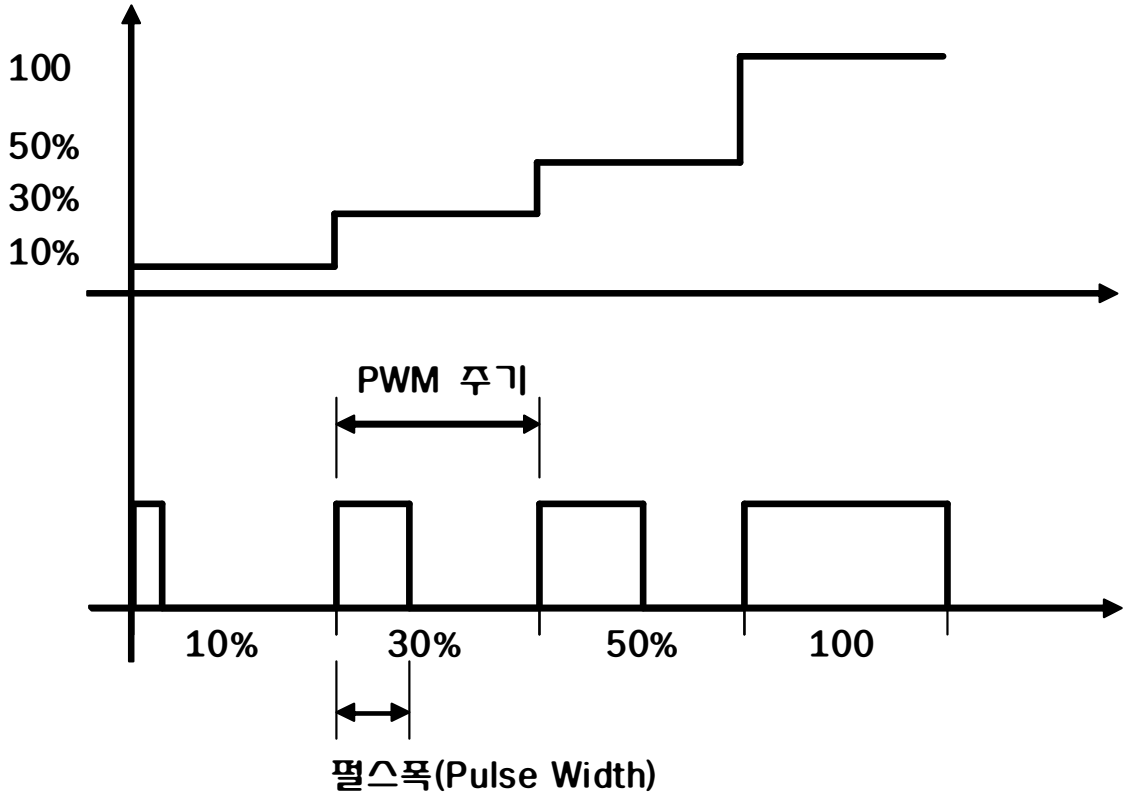
PWM(Pulse Width Modulation)

- 펄스(Pulse)와 펄스폭(Pulse Width)
  - 펄스 : 짧은 시간동안 생기는 진동 현상
  - 펄스폭 : 하나의 펄스가 가지는 폭



PWM(Pulse Width Modulation)

- PWM(펄스폭 변조)
  - 펄스 폭을 전송하고자 하는 신호에 따라 변화시키는 변조 방식
  - 모터 제어나 전압제어 등에 널리 사용

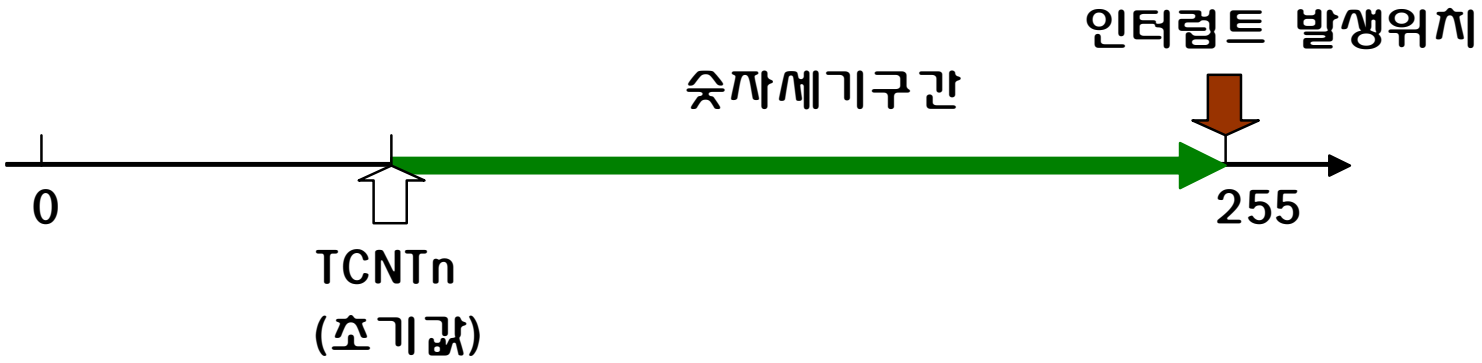


PWM주기 안에서의 펄스 폭으로 신호의 값을 나타냄

ATmega128A의 타이머는 펄스를 만드는데 매우 유용함

8비트 타이머/카운터의 동작모드

- Normal Mode(일반 동작모드)
  - 카운터는 업 카운터로서만 동작
  - MAX(0xFF)값이 되면, BOTTOM(0x00)값부터 다시 시작
  - MAX 위치에서 오버플로우 인터럽트 발생
  - TCNTn의 초기값을 설정하여 전체 타이머 주기를 결정
  - TCCRn레지스터의 WGMn1:n0 = 00으로 설정



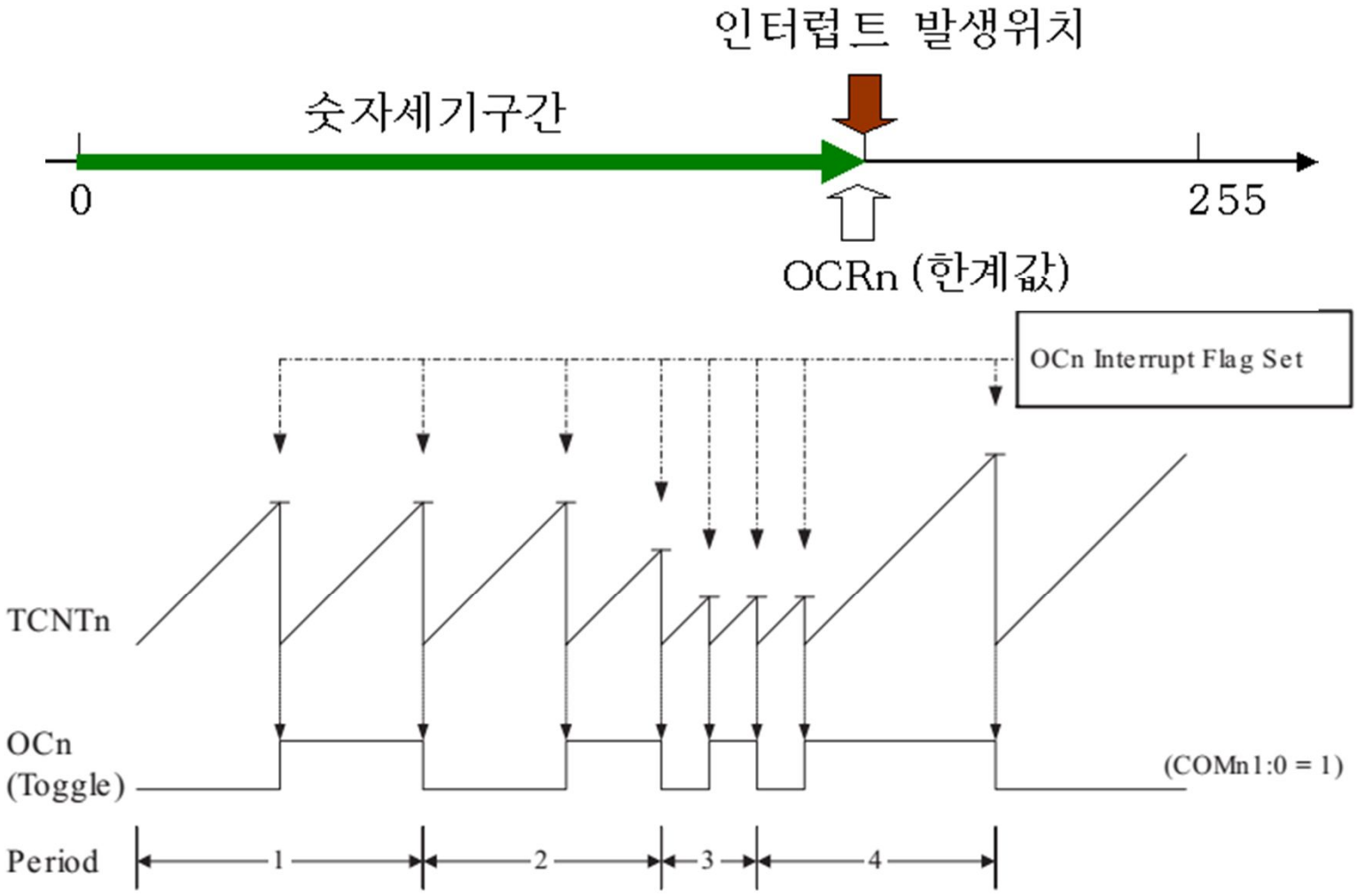
8비트 타이머/카운터의 동작모드

edgeILAB

- CTC(Clear Timer on Compare match) Mode
  - 카운트의 한계값(최대로 세는 수)을 설정
  - 카운터는 업 카운터로서만 동작
    - 0으로부터 설정된 한계값까지 세고 다시 0으로 클리어
    - TCNT 값이 증가하여, OCR값과 일치하면 출력 비교 인터럽트 발생
    - OCRn의 값을 바꾸면 그 다음 카운터 주기를 원하는 대로 변경 가능
  - OCn단자를 이용하여 출력파형 발생 가능
    - TCCRn의 COMn1~n0을 01로 설정
    - OCRn레지스터 값을 바꿔가면서 출력비교에 의해 OCn의 신호를 토글
    - 출력되는 파형의 주기는 foc = fclk / (2*N*(1+OCR0))로 계산
  - TCCRn레지스터의 WGMn1:n0를 "10"으로 설정

8비트 타이머/카운터의 동작모드

- CTC(Clear Timer on Compare match) Mode



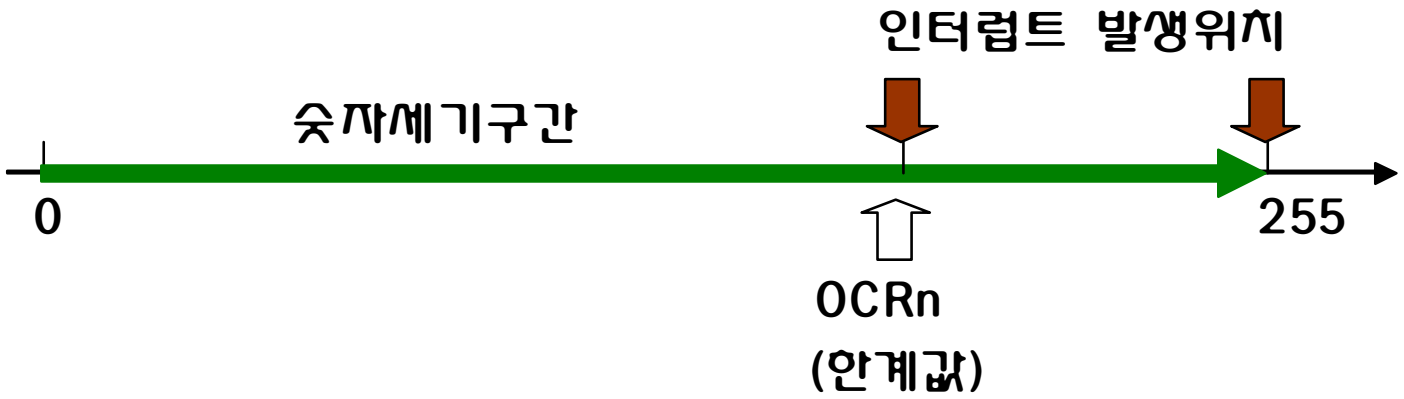
8비트 타이머/카운터의 동작모드

edgeILAB

- Fast PWM Mode
  - 0에서 255까지 세는 동안 두 번의 인터럽트 발생 가능
  - 카운터는 업 카운터로서만 동작
    - TCNT 값이 증가하여, OCR값과 일치하면 출력 비교 인터럽트 발생
    - TCNTn는 업카운팅을 계속하여 255까지 증가했다가 0으로 바뀌는 순간 오버플로우 인터럽트 발생
    - OCRn의 값을 바꾸면 그 다음 카운터 주기를 원하는 대로 변경 가능
  - 두가지 모드로 OC0핀에 구형파 출력 가능
  - 비반전 비교 출력 모드
    - TCCRn 레지스터의 COM 비트를 "10"로 설정
    - TCNT0가 OCR0와 일치하면, OC0 핀에 0를 출력하고 TCNT0가 0이 되면 OC0 핀에 1을 출력
  - 반전 비교 출력 모드
    - TCCRn 레지스터의 COM 비트를 "11"로 설정
    - TCNT0가 OCR0와 일치하면, OC0 핀에 1을 출력하고, TCNT0가 0이 되면 OC0 핀에 0을 출력
  - 출력 파형 주파수 foc = fclk / (N*256)

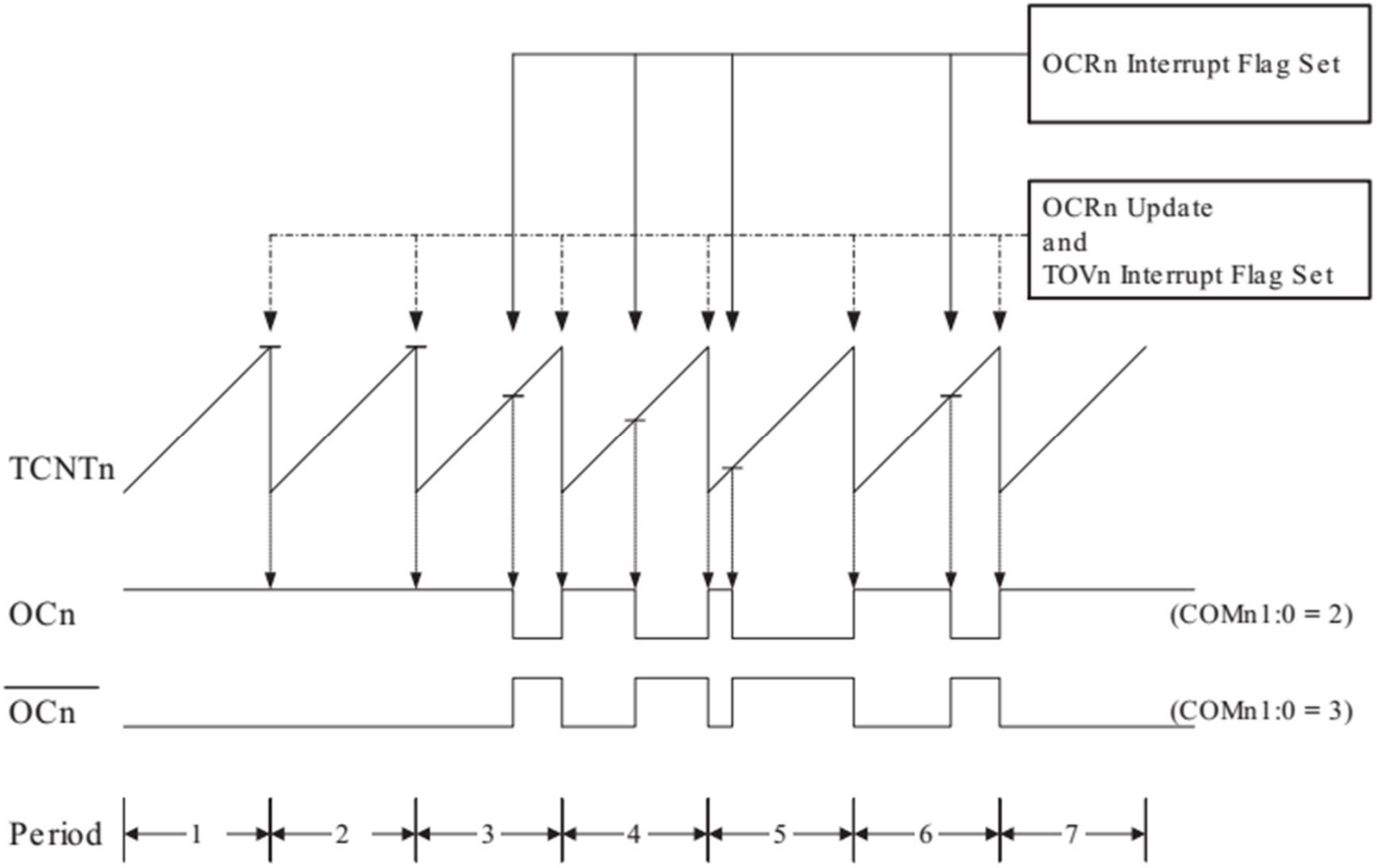
8비트 타이머/카운터의 동작모드

- Fast PWM Mode
  - TCCRn레지스터의 WGMn1:n0를 11 으로 설정
  - 높은 주파수의 PWM 파형 발생시 유용



8비트 타이머/카운터의 동작모드

- Fast PWM Mode



8비트 타이머/카운터의 동작모드

edgeILAB

- PCPWM(Phase Correct Pulse Width Modulation) Mode
  - Fast PWM과 유사
  - 업카운팅과 다운카운팅이 번갈아 일어남
    - TCNT 값이 증가하여, OCR값과 일치하면 출력 비교 인터럽트 발생
    - TCNTn는 업카운팅을 계속하여 TCNTn는 255에 도달하면 다운카운팅 시작
    - 다운카운팅을 하다가 0에 도달하면 오버플로우 인터럽트가 발생
    - OCRn의 값을 바꾸면 그 다음 카운터 주기를 원하는 대로 변경 가능
  - PWM 주기를 변경하기 위해 OCRn 레지스터에 새로운 값을 기록하더라도 즉시 변경되지 않고 TCNTn이 255에 도달하면 갱신됨

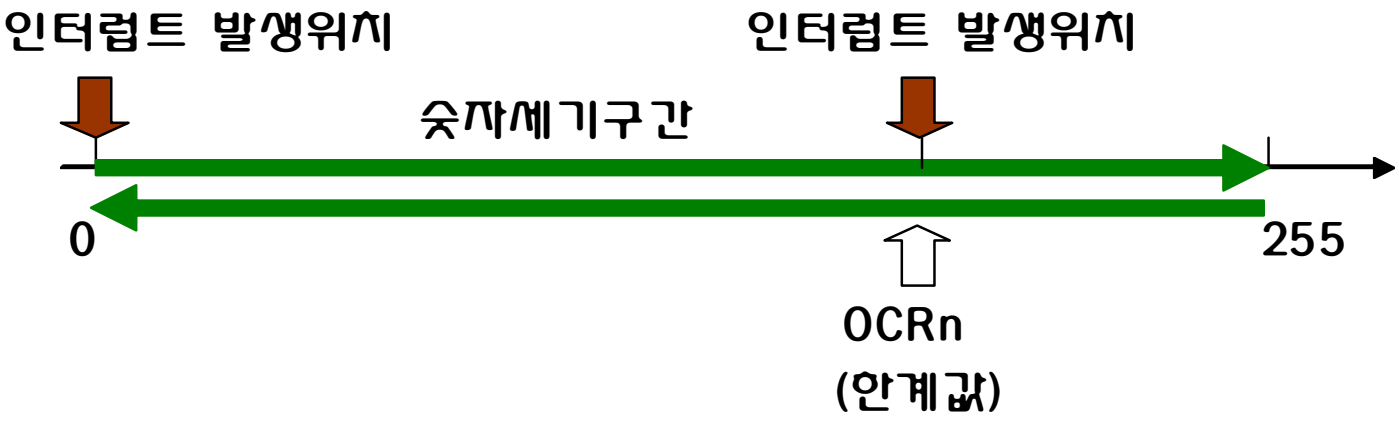
8비트 타이머/카운터의 동작모드

edgeILAB

- PCPWM(Phase Correct Pulse Width Modulation) Mode
  - 두 가지 모드로 OC0핀에 구형파 출력 가능
    - 비반전 비교 출력 모드(TCCRn 레지스터의 COM 비트를 “10”로 설정)
      - 업 카운트 중에 TCNT0와 OCR0가 일치하면, OC0핀에 0를 출력
      - 다운 카운트 중에 일치하면, OC0핀에 1을 출력
    - 반전 비교 출력 모드(TCCRn 레지스터의 COM 비트를 “11”로 설정)
      - 업 카운트 중에 TCNT0와 OCR0가 일치하면, OC0핀에 1을 출력
      - 다운 카운트 중에 일치하면, OC0핀에 0를 출력
    - 출력 파형 주파수 foc = fclk / (N*256)

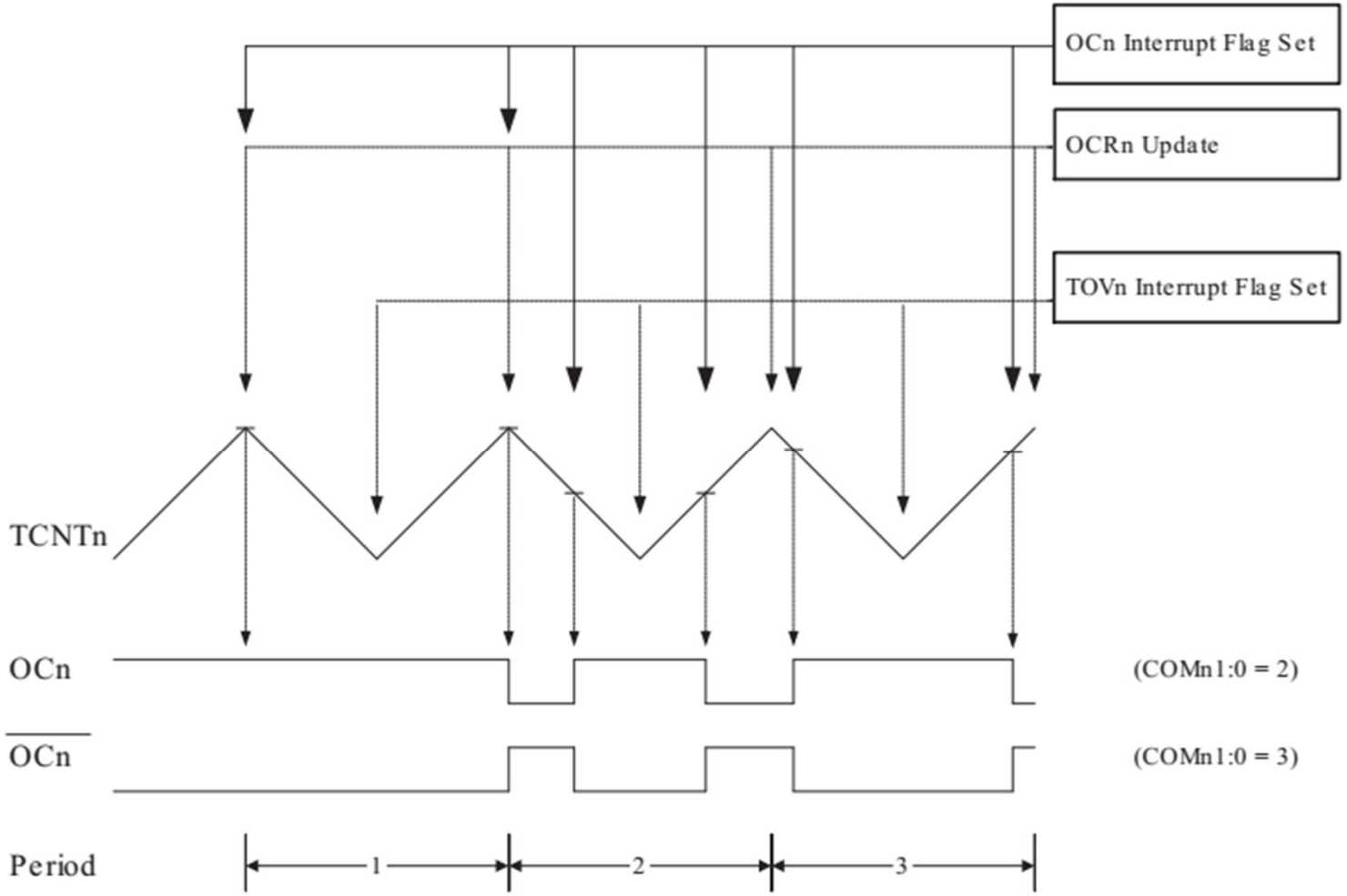
8비트 타이머/카운터의 동작모드

- PCPWM(Phase Correct Pulse Width Modulation) Mode
  - TCCRn레지스터의 WGMn1:n0를 01으로 설정
  - 높은 분해능의 PWM 파형 발생시 유용



8비트 타이머/카운터의 동작모드

- PCPWM(Phase Correct Pulse Width Modulation) Mode



16비트 타이머/카운터

- 16비트 타이머/카운터
  - 타이머/카운터 1,3
  - 16비트의 카운터를 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
  - TCCRxA~C
    - 타이머x의 동작 방식 설정
  - TCNTx(Timer/CouNTer x)
    - 타이머x의 16비트 카운터 값을 저장
  - OCRxA~C(Output Compare Resister x A~C)
    - TCNTx의 값과 출력 비교되기 위한 16비트 데이터 값을 저장
  - ICRx(Input Capture Register x)
    - 입력캡쳐시 TCNTx의 카운터 값을 저장
  - 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(ICPn)의 입력을 필터링
    - 4개의 오실레이터 사이클 만큼 지연
  - 비트 6 : ICESn
    - ICPn에 해당되는 에지의 형태를 선택
    - 1로 설정하면 상승에지에서 검출, 0으로 설정하면 하강에지에서 검출
    - 카운터 값은 ICRn에 저장되고, 입력 캡처 플래그(ICPn)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)
  - 입력캡쳐레지스터
  - 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

edgeILAB

- ETIMSK(Extended Timer Interrupt MaSK)
  - TICIE3 r 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

16비트 타이머/카운터 동작모드

- 16비트 타이머/카운터 동작모드
  - 크게 나누어 5가지 동작 모드
    - 8비트 타이머/카운터 4가지 모드 + PFC PWM(Phase and Frequency Correct PWM) 모드
  - 세분하면 총 15가지 동작모드로 구분

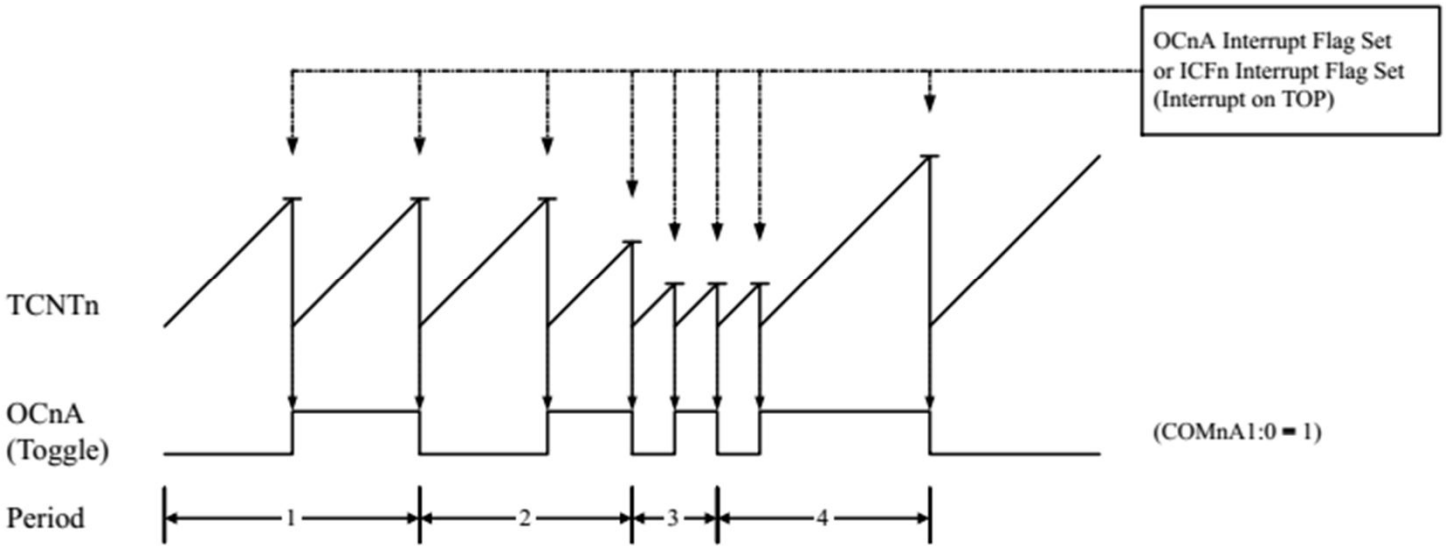
- Normal Mode(일반 동작모드)
  - 타이머n는 항상 상향 카운터로만 동작
  - TCNTn의 값이 0xFFFF에서 0으로 바뀌는 순간 TOVn비트가 세트되며 오버플로우 인터럽트 발생
  - TCCRxA~B의 WGMx3~0 = 00으로 설정
  - 카운터가 16비트라는 것을 제외하고는 8비트 타이머와 동일

16비트 타이머/카운터 동작모드

- CTC(Clear Timer on Compare match) 모드
  - 4번 모드
    - 인터럽트 : TCNTn의 값이 OCRnA에 설정한 값과 일치되면 그 다음 클럭 사이클에서 TCNTn의 값이 0으로 클리어되고, OCFnA비트가 세트되며 출력비교 인터럽트 발생
    - 출력파형 : COMnA/COMnB/COMnC1~0을 01로 설정하고 OCRnA레지스터 값을 바꿔가면서 출력비교에 의해 OCnA/OCnB/OCnC 의 신호를 토글
  - 12번 모드
    - 인터럽트 : TCNTn의 값이 ICRn에 설정한 값과 일치되면 그 다음 클럭 사이클에서 TCNTn의 값이 0으로 클리어되고, ICFn비트가 세트되며 입력캡쳐 인터럽트 발생
    - 출력파형 : COMnA/COMnB/COMnC1~0을 01로 설정하고 OCRnA/OCRnB/OCRnC 레지스터 값을 바꿔가면서 출력비교에 의해 OCnA/OCnB/OCnC의 신호를 토글
  - TCCRnB 레지스터의 WGMx3~0을 4 또는 12로 설정

16비트 타이머/카운터 동작모드

- CTC(Clear Timer on Compare match) 모드



16비트 타이머/카운터 동작모드

- Fast PWM(Fast Pulse Width Modulation) 모드
  - TCNTn는 반복적으로 업카운팅하며 항상 0x0000~TOP의 값을 가짐
  - TCNTn의 값이 TOP 값과 일치되면 그 다음 사이클에서 0으로 클리어
  - 모드별 TOP 값

동작모드	5	6	7	14	15
TOP 값	0x00FF	0x01FF	0x03FF	ICR1	OCR1A

  - 인터럽트
    - OCFnA비트 세트, 출력비교 인터럽트 발생
    - ICFn비트 세트, 입력캡춰 인터럽트 발생
    - TOVn비트 세트, 오버플로우 인터럽트 발생

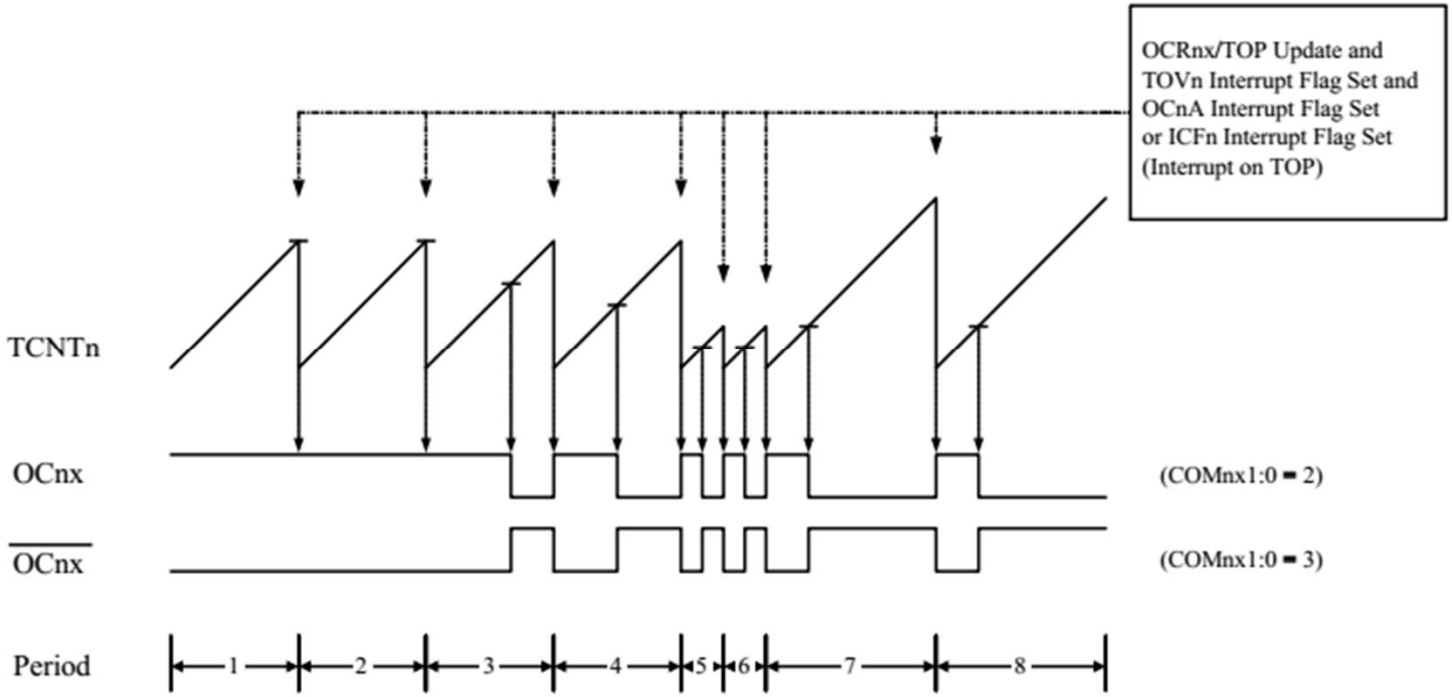
  - TCCRxB레지스터의 WGMx3~0 비트를 5,6,7,14,15로 설정

16비트 타이머/카운터 동작모드

- Fast PWM(Fast Pulse Width Modulation) 모드
  - 출력 파형
    - COM1:0 가 "10"일때
      - TCNTn의 값이 ICRn이나 OCRnA/OCRnB/OCRnC 값과 같아지면 OCnA/OCnB/OCnC의 값은 0으로 클리어
      - TOP까지 증가했다가 0으로 바뀌는 순간 OCnA/OCnB/OCnC 는 1로 세트
    - COM1:0 가 "11"일때
      - TCNTn의 값이 ICRn이나 OCRnA/OCRnB/OCRnC 값과 같아지면 OCnA/OCnB/OCnC의 값은 1로 세트
      - TOP까지 증가했다가 0으로 바뀌는 순간 OCnA/OCnB/OCnC 는 0으로 클리어
    - COM1:0 가 "00"or "01"일때
      - OCnA/OCnB/OCnC는 신호 출력되지 않음

16비트 타이머/카운터 동작모드

- Fast PWM(Fast Pulse Width Modulation) 모드



16비트 타이머/카운터 동작모드

- PC PWM(Phase Correct PWM) 모드
  - TCNTn는 업카운팅하여 TOP으로 증가하다가 다운카운팅으로 0x0000으로 감소를 반복
  - 모드별 TOP 값

동작모드	1	2	3	10	11
TOP 값	0x00FF	0x01FF	0x03FF	ICR1	OCR1A

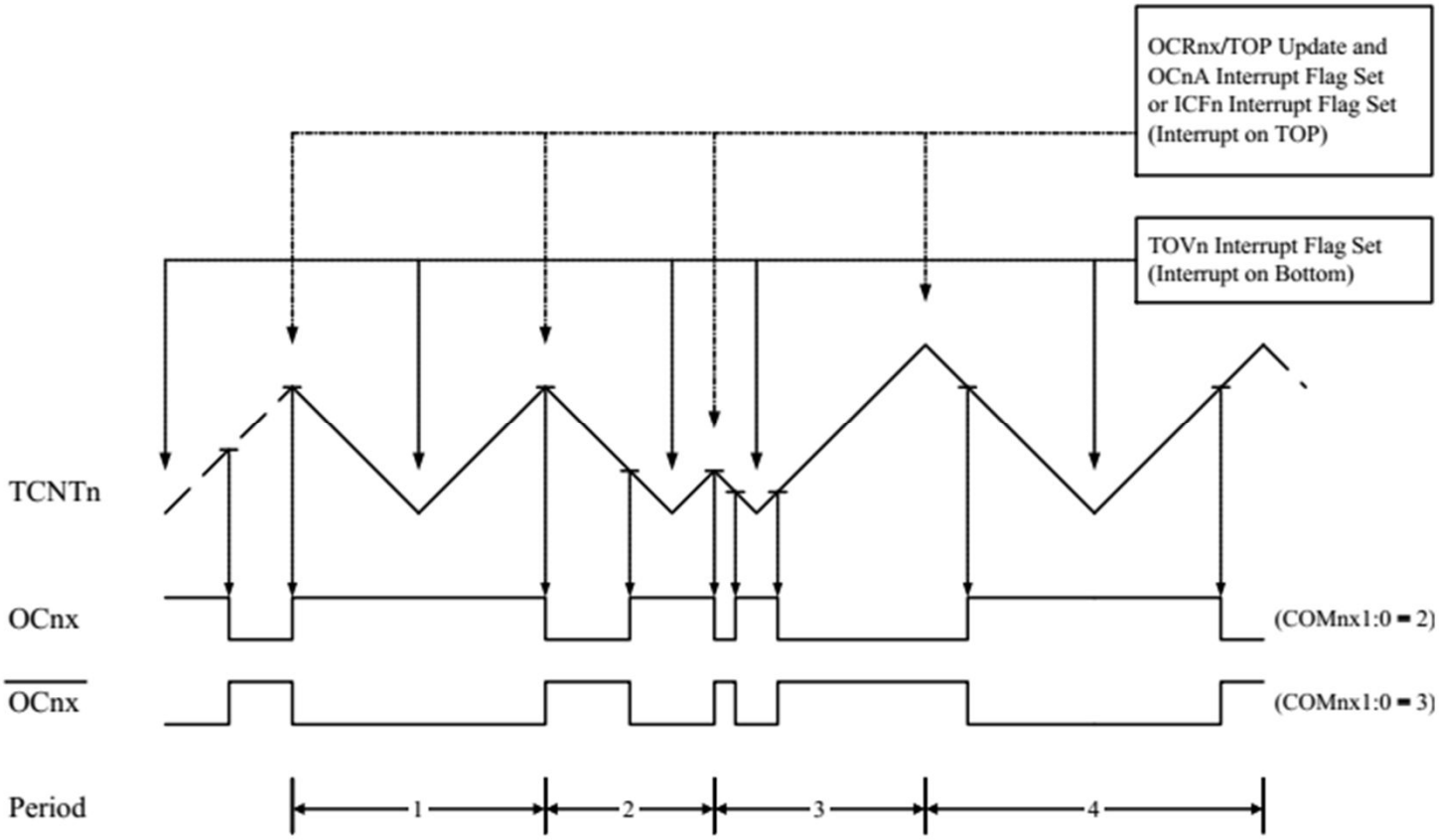
  - 인터럽트
    - OCFnA비트 세트, 출력비교 인터럽트 발생
    - ICFn비트 세트, 입력캡춰 인터럽트 발생
    - TOVn비트 세트, 오버플로우 인터럽트 발생
  - TCCRxB레지스터의 WGMx3~0 비트를 1,2,3,10,11로 설정

16비트 타이머/카운터 동작모드

- PC PWM(Phase Correct PWM) 모드
  - 출력 파형
    - COM1:0 가 "10"일때
      - TCNTn의 값이 OCRnA/OCRnB/OCRnC 레지스터에 설정한 값과 일치되면 OCnA/OCnB/OCnC 의 값은 상향 카운팅의 경우에는 0으로, 하향의 경우에는 1로 세트
    - COM1:0 가 "11"일때
      - TCNTn의 값이 OCRnA/OCRnB/OCRnC 레지스터에 설정한 값과 일치되면 OCnA/OCnB/OCnC 의 값은 상향 카운팅의 경우에는 1로, 하향의 경우에는 0로 세트
    - COM1:0 가 "00"or "01"일때
      - OCnA/OCnB/OCnC는 신호 출력되지 않음
  - PWM 주기를 변경하기 위해 OCRnA/OCRnB/OCRnC 레지스터에 새로운 값을 기록하더라도 즉시 변경되지 않고 TCNTn이 TOP에 도달하면 비로소 값이 갱신

16비트 타이머/카운터 동작모드

- PC PWM(Phase Correct PWM) 모드



16비트 타이머/카운터 동작모드

edgeILAB

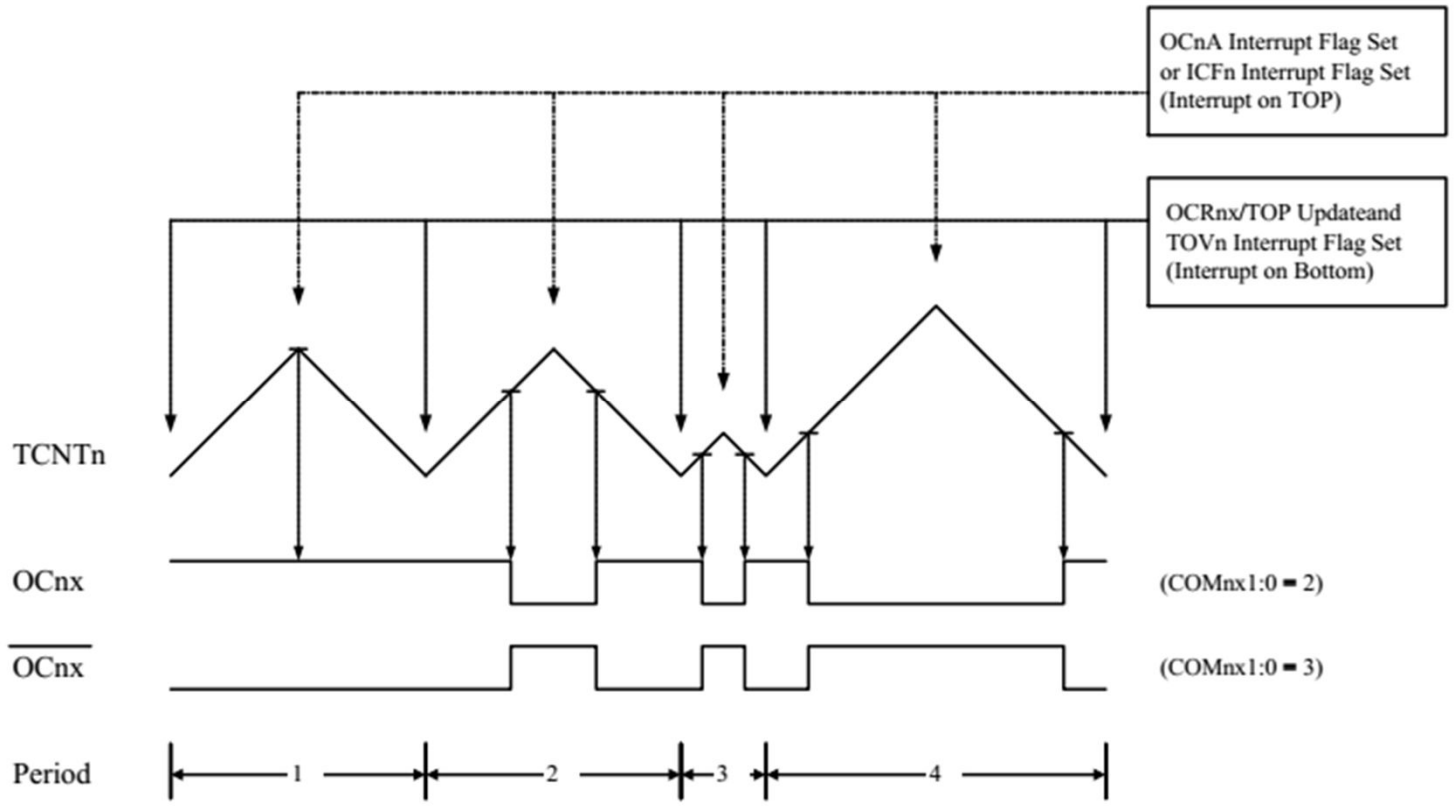
- PFC PWM(Phase & Frequency Correct PWM) 모드
  - PC PWM 모드와 거의 유사
  - 모드별 TOP 값
    - 모드 8 : ICR1
    - 모드 9 : OCRnA
  - TCNTn는 업카운팅하여 TOP으로 증가하다가 다운카운팅으로 0x0000으로 감소를 반복
  - 인터럽트
    - OCFnA비트 세트, 출력비교 인터럽트 발생
    - ICFn비트 세트, 입력캡춰 인터럽트 발생
    - TOVn비트 세트, 오버플로우 인터럽트 발생
  - 새로운 값으로의 갱신이 TCNTn가 0일 때 이루어짐
  - TCCRxB레지스터의 WGMx3~0 비트를 8,9로 설정

16비트 타이머/카운터 동작모드

- PFC PWM(Phase & Frequency Correct PWM) 모드
  - 출력 파형
    - COM1:0 가 "10"일때
      - TCNTn의 값이 OCRnA/OCRnB/OCRnC 레지스터에 설정한 값과 일치되면 OCnA/OCnB/OCnC 의 값은 상향 카운팅의 경우에는 0으로, 하향의 경우에는 1로 세트
    - COM1:0 가 "11"일때
      - TCNTn의 값이 OCRnA/OCRnB/OCRnC 레지스터에 설정한 값과 일치되면 OCnA/OCnB/OCnC 의 값은 상향 카운팅의 경우에는 1로, 하향의 경우에는 0로 세트
    - COM1:0 가 "00"or "01"일때
      - OCnA/OCnB/OCnC는 신호 출력되지 않음

16비트 타이머/카운터 동작모드

- PFC PWM(Phase & Frequency Correct PWM) 모드



실습 1 : PWM으로 LED 밝기 조절 하기

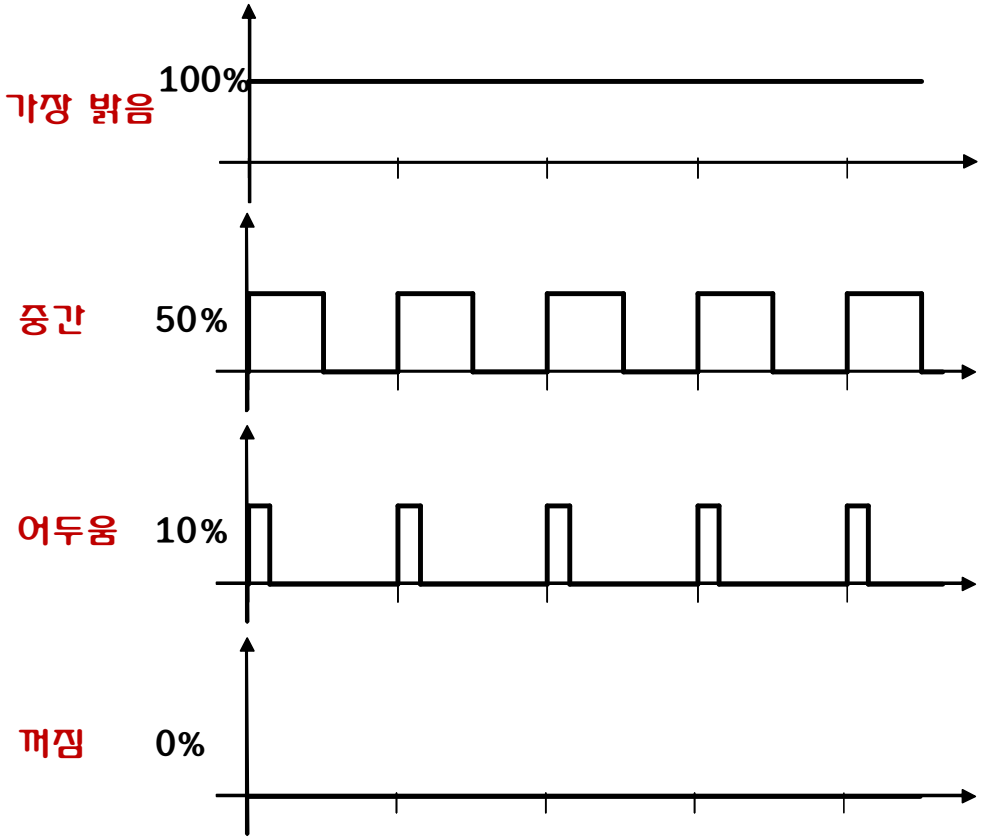
edgeILAB

- 실습 개요
  - 타이머 0의 PC PWM 동작 모드를 이용하여 LED의 밝기를 조절하기
  - PWM 동작 모드에서 OC0 핀을 통해 PWM신호를 만들어 출력함으로써, LED의 밝기를 조절하도록 함
  - 밝기는 PWM 신호의 듀티비(Pulse Duty)에 의해 좌우됨

- 실습 목표
  - 타이머0의 PWM 기능 동작원리 이해
  - 타이머0의 PCPWM 모드 제어 방법의 습득(관련 레지스터 이해)
  - PWM 신호 출력 제어 방법 습득

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램 : 사전 지식
  - PWM 신호에 의한 LED의 밝기 조절 방법
    - 출력 비교용 레지스터(OCR)값을 조절하여 OC0로 출력되는 PWM 신호의 듀티비를 원하는 대로 변경



듀티비 100%인 경우
LED는 가장 밝을 것이며,
듀티비 0%인 경우
LED는 꺼짐

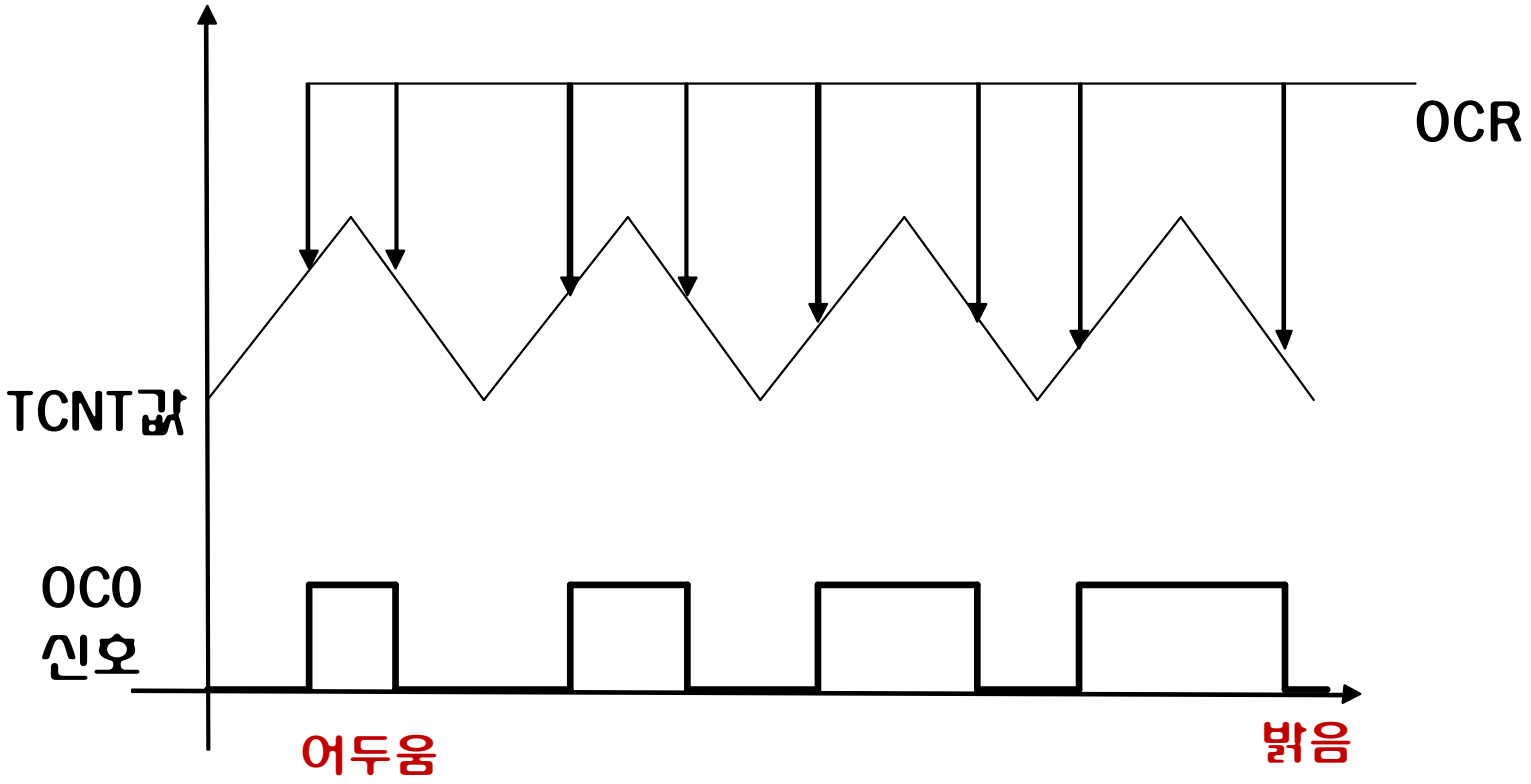
실습 1 : PWM으로 LED 밝기 조절 하기

edgeILAB

- 구동 프로그램 : 사전 지식(PWM 동작 모드 설정)
  - 타이머/카운터 0 사용
  - Phase Correct PWM 동작 모드 사용
  - TCCR0 설정
    - CS 비트는 111로 세팅하여 Prescaler의 분주비를 1024로 설정
    - FOC 비트는 0로 설정
    - WGM 비트는 PC PWM 모드인 01로 설정
    - COM 비트는 11로 세팅하여 업카운팅의 경우 OC0를 1로 세트하고, 다운카운팅의 경우 OC0를 클리어 시키도록 설정
    - PWM 클럭 = 메인클럭/N*510, (N = 클럭분주)

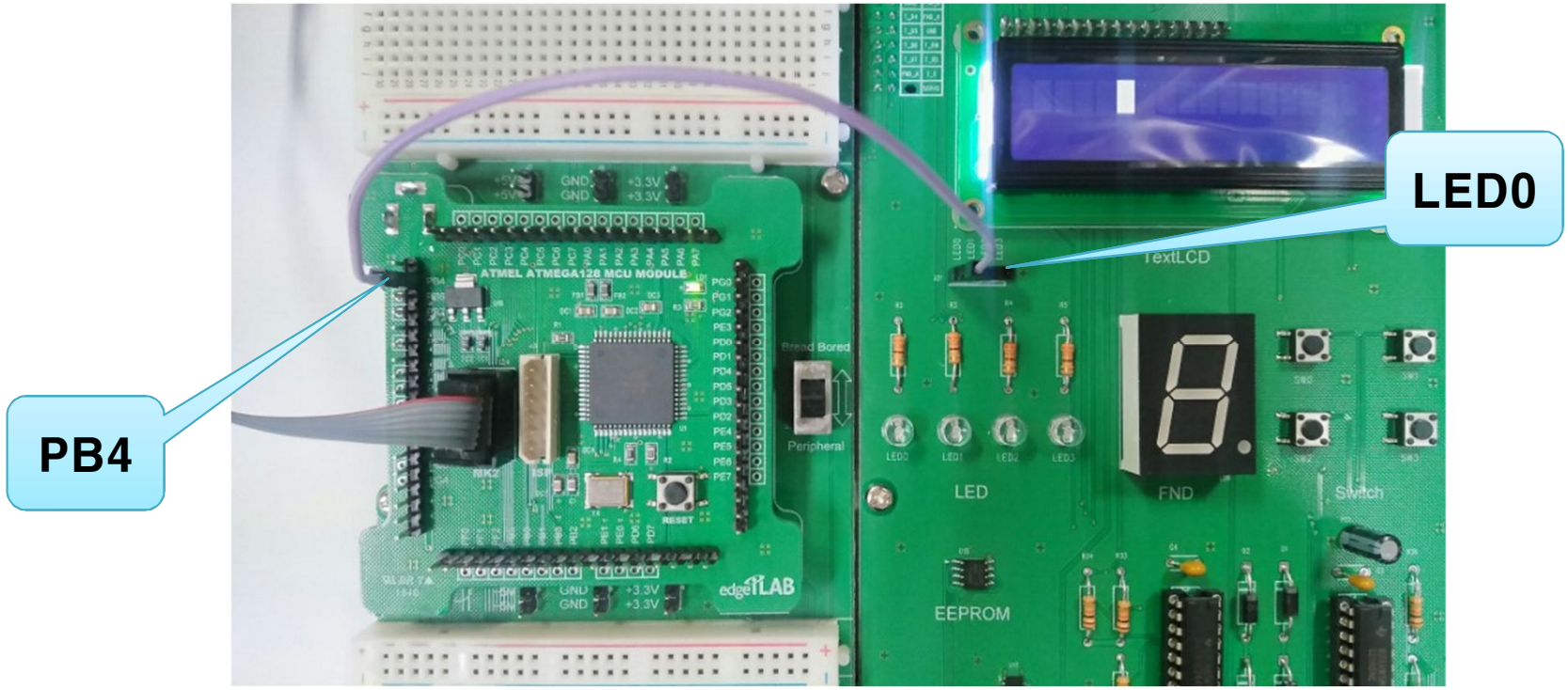
실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램 : 사전 지식(PWM 동작 모드 설정)



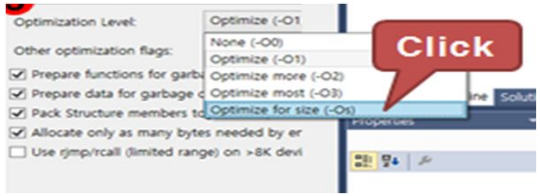
실습 1 : PWM으로 LED 밝기 조절 하기

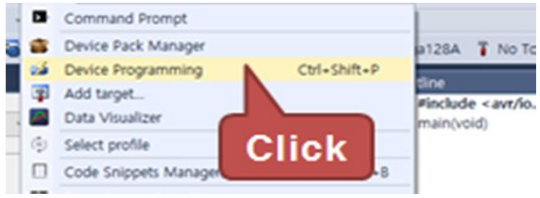
- 실습 준비
  - 다음 그림과 같이 연결
    - Edge-MCU보드의 PB4 → Edge-Peri 보드의 LED0(LED 상단)

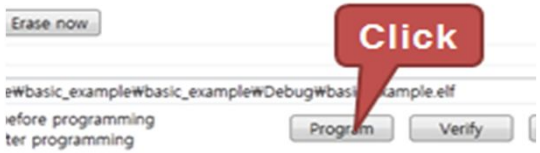


실습 1 : PWM으로 LED 밝기 조절 하기

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 07_PwmLed_Example, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compiler에서
    - 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







실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램
  - main.c 코드 작성

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

int main(void)
{
  unsigned char Light=0;

  DDRB = 0x10;           // 포트B 를 출력포트로 설정

  TCCR0 = 0x77;          // PC PWM 모드, 1024 분주 ==> 14.4 KHz :
  TCNT0 = 0x00;          // 타이머0 카운터를 초기화
```

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램
  - main.c 코드 작성

```
int main(void) {
  //...
  while (1)
  {
    for(Light=0;Light<255;Light++)
    {
      OCR0 = Light;
      _delay_ms(10);
    }
    for(Light=255;0<Light;Light--)
    {
      OCR0 = Light;
      _delay_ms(10);
    }
  }
}
```

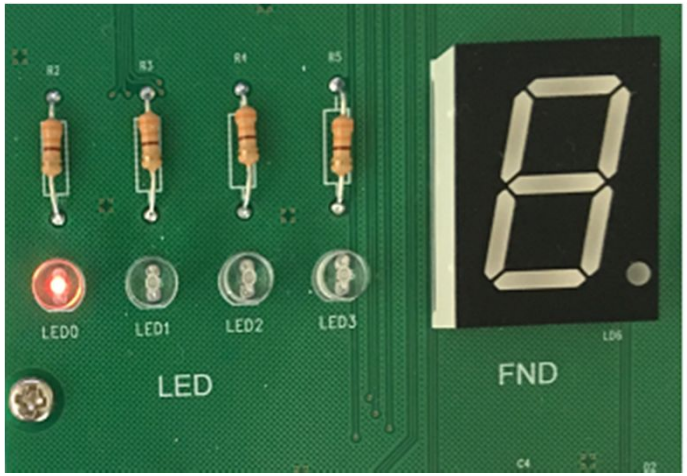
실습 1 : PWM으로 LED 밝기 조절 하기

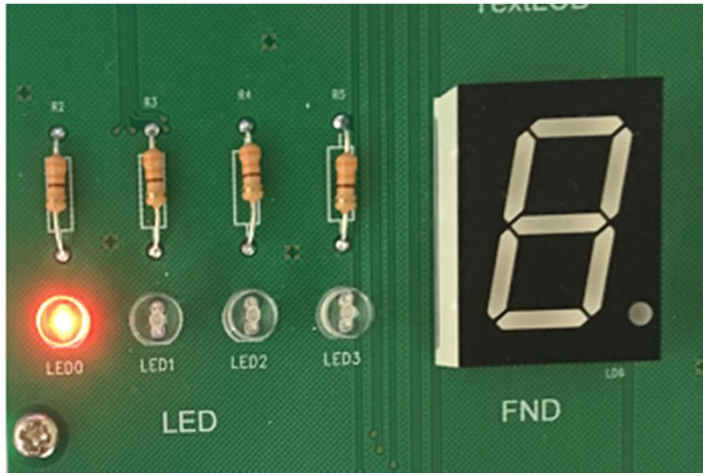
- 구동 프로그램 : 소스 분석
  - TCCR0 = 0x77;
  - CS는 "111" 이므로 Prescaler의 분주비가 1024
  - WGM(1:0)는 "01" 이므로 PC PWM 모드
  - COM(1:0)는 "11" 로 세팅
  - FOC는 '0'로 세팅

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

실습 1 : PWM으로 LED 밝기 조절 하기

- 실행 결과
  - LED 가 어두워졌다 밝아짐





응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

edgeILAB

- 실습 개요
  - 타이머 2의 PC PWM 동작 모드를 이용하여 LED의 밝기를 조절하기
  - 스위치 입력 값에 따라 OC2 핀을 통해 나오는 PWM 신호의 듀티비를 조절함
  - PWM 동작 모드에서 OC핀을 통해 PWM신호를 만들어 출력함으로써, LED의 밝기를 조절하도록 함
  - 밝기는 PWM 신호의 듀티비(Pulse Duty)에 의해 좌우됨

- 실습 목표
  - 타이머0의 PWM 기능 동작원리 이해
  - 타이머0의 PCPWM 모드 제어 방법의 습득(관련 레지스터 이해)
  - PWM 신호 출력 제어 방법 습득

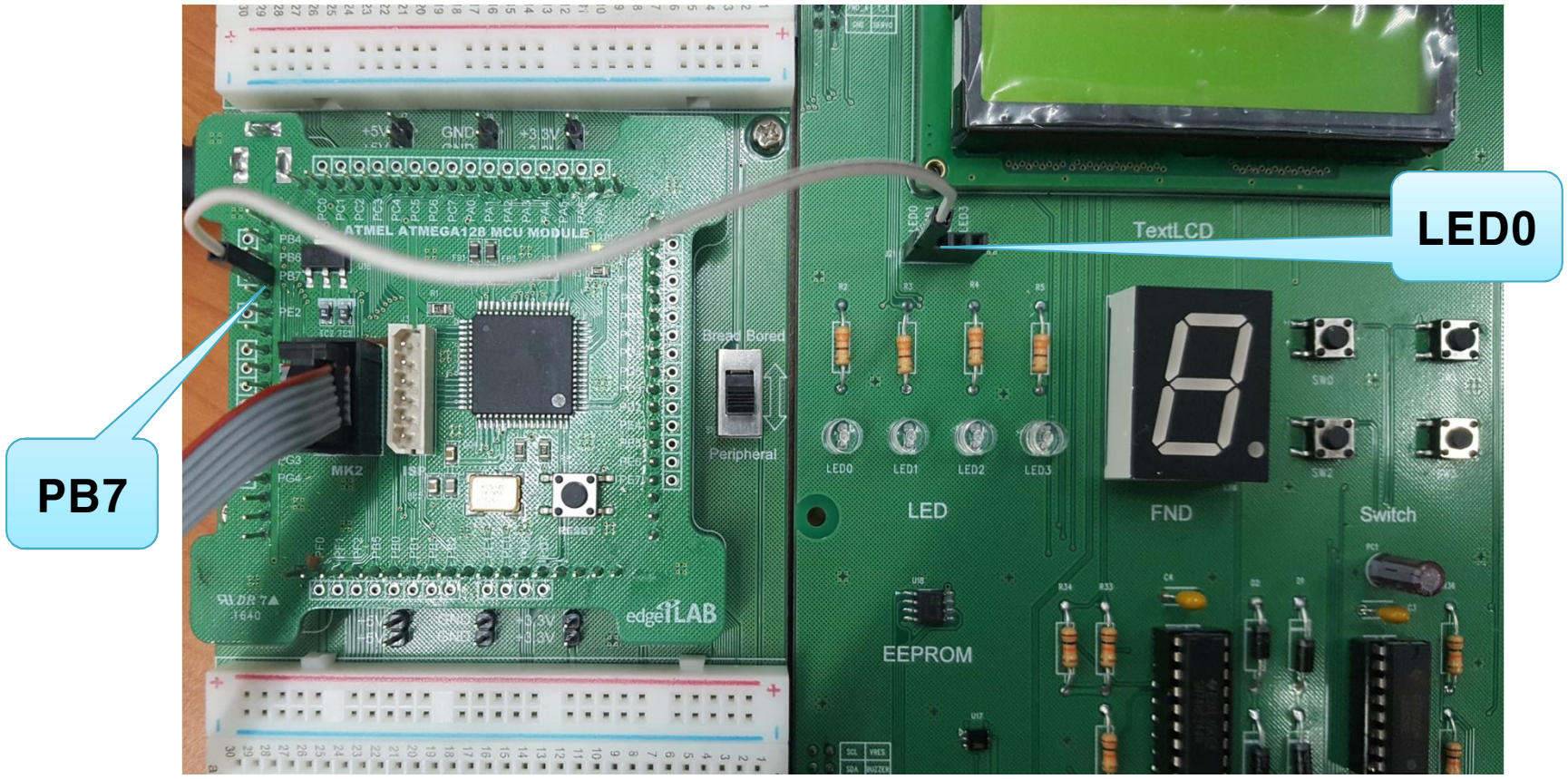
응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

edgeILAB

- 구동 프로그램 : 사전 지식(PWM 동작 모드 설정)
  - 타이머/카운터 2 사용
  - Phase Correct PWM 동작 모드 사용
  - TCCR2 설정
    - CS비트는 010로 세팅하여 분주비를 8로 설정
    - FOC 비트는 0로 설정
    - WGM 비트는 PC PWM 모드인 01로 설정
    - COM 비트는 10로 세팅하여 업카운팅의 경우 OC2를 0으로 클리어 하고, 다운카운팅의 경우 OC2를 1로 세트하도록 설정
    - PWM 클럭 = 메인클럭/N*510, (N = 클럭분주)

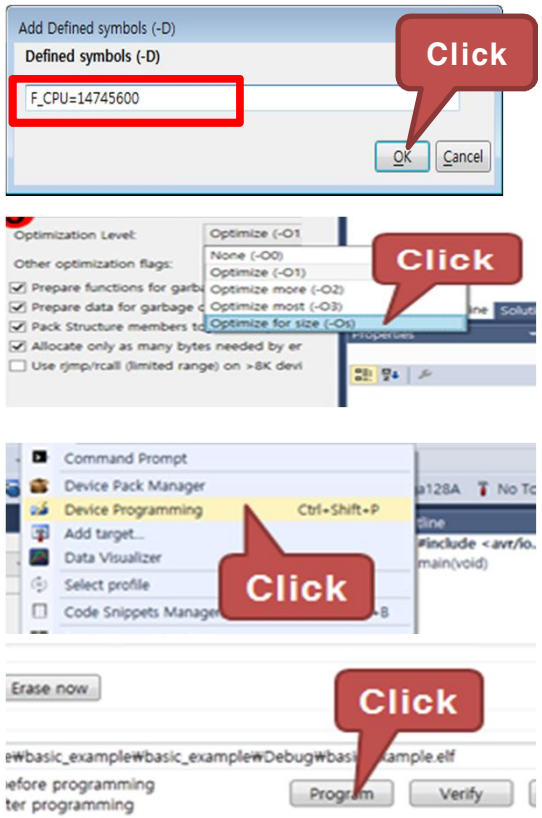
응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 실습 준비
  - 다음 그림과 같이 연결
    - Edge-MCU 보드의 PB7 → Edge-Peri 보드의 LED0(LED 상단)



응용 1 : 스위치의 입력 값에 따라 LED 밝기 조절 하기

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 07_PwmLed_Application, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compiler에서
    - 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



응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

edgeILAB

- 구동 프로그램
  - Main.c 코드 작성

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

volatile int Light = 0;
volatile unsigned char Light_flag = 1;

int main(void)
{
  DDRB = 0x80;  // 포트B 를 출력포트로 설정한다.
                // LED0을 PB7과 케이블로 연결
  DDRE = 0x00;  // 포트E 를 입력포트로 설정한다.
```

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 구동 프로그램
  - Main.c 코드 작성

```
  TCCR2 = 0x62; // PC PWM 모드, 8 분주
                // PWM 주기 : F_CPU/256/2/8 = 3.6 KHz
                // 업카운트시 Clear, 다운카운트시 Set 으로 설정
  TCNT2 = 0x00; // 타이머2 카운터를 초기화 한다.

  EICRB = 0xFF; // 인터럽트 4, 5, 6, 7을 상승엣지에서 동작하도록 설정
한다.
  EIMSK = 0xF0; // 인터럽트 4, 5, 6, 7을 허용
  EIFR = 0xF0;  // 인터럽트 4, 5, 6, 7 플래그를 클리어

  sei();        // 전체 인터럽트를 허용
```

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 구동 프로그램
  - Main.c 코드 작성

```
int main(void)
{
//...
  while (1)
  {
    if(Light_flag)
    {
      OCR2 = Light;    // Light 값에 따라 밝기 제어
      Light_flag = 0;
    }
  }
}
```

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 구동 프로그램
  - Main.c 코드 작성

```
SIGNAL(INT4_vect)        // 인터럽트 서비스 루틴
{
  cli();          // 전체 인터럽트를 금지
  Light = 0;   // LED OFF;
  Light_flag = 1;
  sei();          // 전체 인터럽트를 허용
}
SIGNAL(INT5_vect)        // 인터럽트 서비스 루틴
{
  cli();          // 전체 인터럽트를 금지
  Light -= 51;   // LED 밝기 감소
  if(Light < 0)
  Light = 0;
  Light_flag = 1;
  sei();          // 전체 인터럽트를 허용
}
```

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 구동 프로그램
  - Main.c 코드 작성

```
SIGNAL(INT6_vect)        // 인터럽트 서비스 루틴
{
  cli();          // 전체 인터럽트를 금지
  Light += 51;    // LED 밝기 증가;
  if(Light > 255)
  Light = 255;
  Light_flag = 1;
  sei();          // 전체 인터럽트를 허용
}
SIGNAL(INT7_vect)        // 인터럽트 서비스 루틴
{
  cli();          // 전체 인터럽트를 금지
  Light = 255;    // LED 밝기 100%
  Light_flag = 1;
  sei();          // 전체 인터럽트를 허용
}
```

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

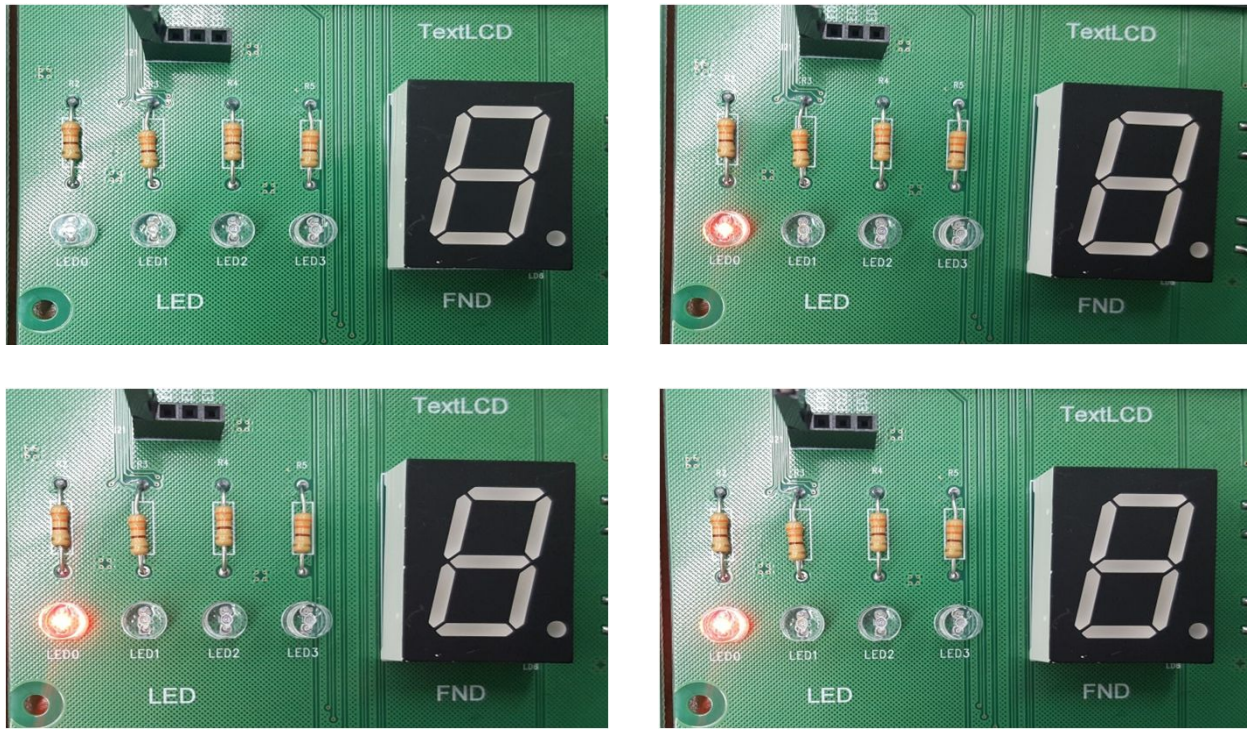
edgeILAB

- 구동 프로그램 : 소스 분석
  - TCCR0 = 0x62;
  - CS는 "010" 이므로 Prescaler의 분주비가 8
  - WGM(1:0)는 "01" 이므로 PC PWM 모드
  - COM(1:0)는 "10" 로 세팅
  - FOC는 '0'로 세팅

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

응용 1 : 스위치 입력 값에 따라 LED 밝기 조절 하기

- 실행 결과
  - 스위치에 따라 LED 밝기가 변함



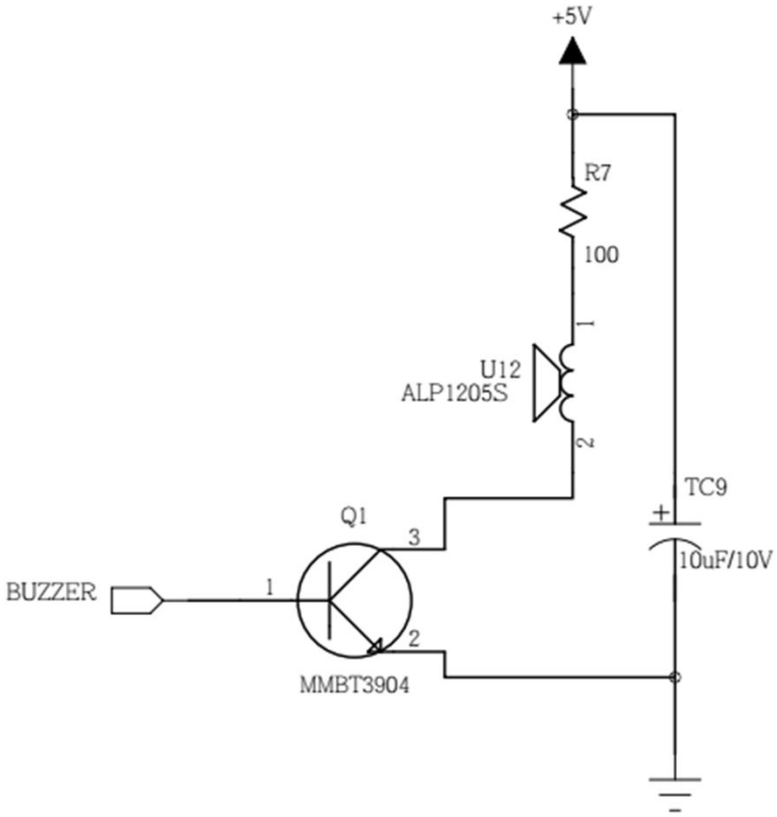
실습 2 : PWM으로 PIEZO 울리기

- 실습 개요
  - 타이머를 이용하여 원하는 주파수의 신호를 만들고, 이를 Piezo에 입력하여 여러 가지 소리를 내도록 설계
  - 16비트 타이머인 타이머/카운터 3을 사용하며, 동작모드는 CTC 모드을 사용
  - 각기 다른 스위치 입력에 따라, 다른 주파수의 신호를 만듦

- 실습 목표
  - 16비트 타이머/카운터 활용 방법의 습득(관련 레지스터 이해)
  - Piezo의 동작원리 이해

실습 2 : PWM으로 PIEZO 울리기

- 사용 모듈 : Audio 모듈 회로
  - 압전 Piezo 관련 회로

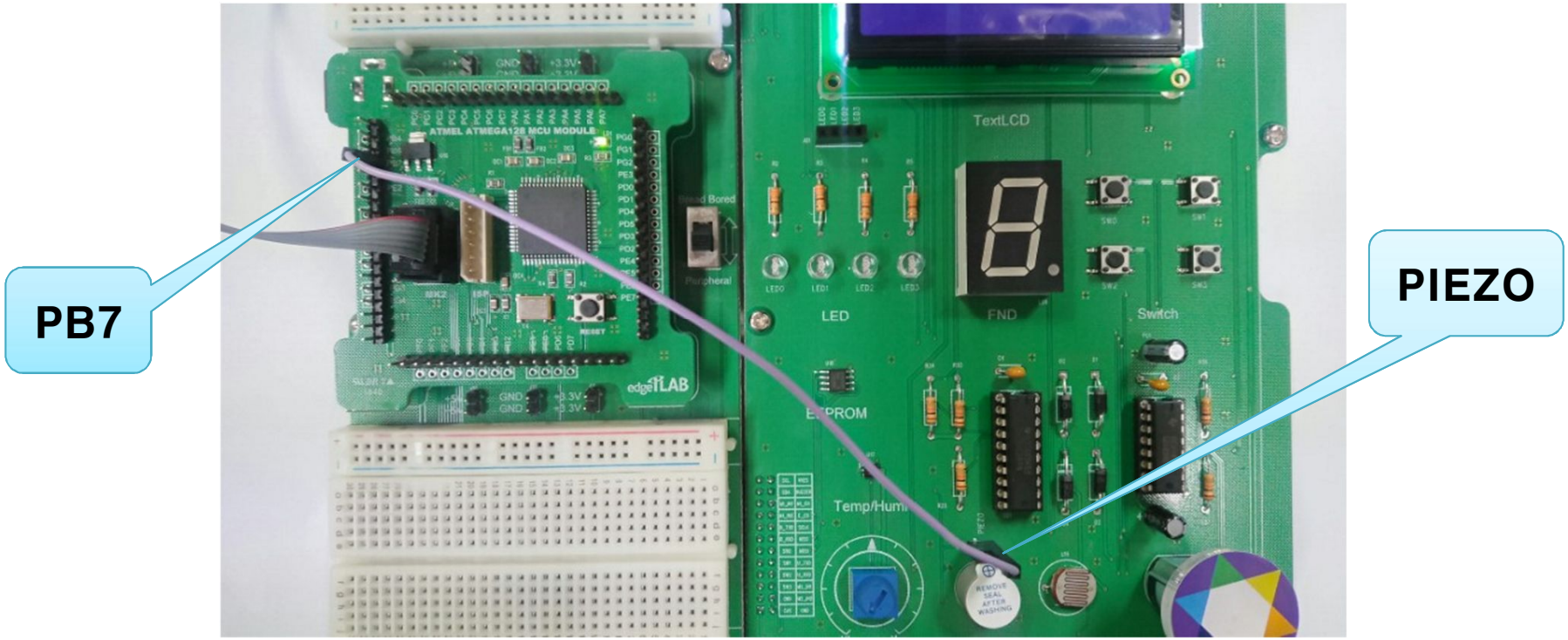


실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램 : 사전 지식
  - 타이머를 이용하여 Piezo를 여러가지 소리가 나도록 울리도록 함
  - 16비트 타이머/카운터 1을 이용/동작 모드는 14번 Fast PWM모드를 사용
  - TCCR1A/TCCR1B/ TCCR1C 레지스터들을 적절히 세팅
    - Piezo에 입력될 신호는 OC1C핀을 사용
    - WGM(3:0)은 "1110"으로 세팅
    - COM1C(1:0)비트를 "01"로 설정하여 non-inverting 모드를 사용
    - 프리스케일러는 CS(2:0)비트에 "001"을 세팅하여 1분주로 함
  - OC1C 핀 출력신호의 주파수
    - OC1C 주파수 = 메인클럭 / (분주비 * (1 + ICR1))

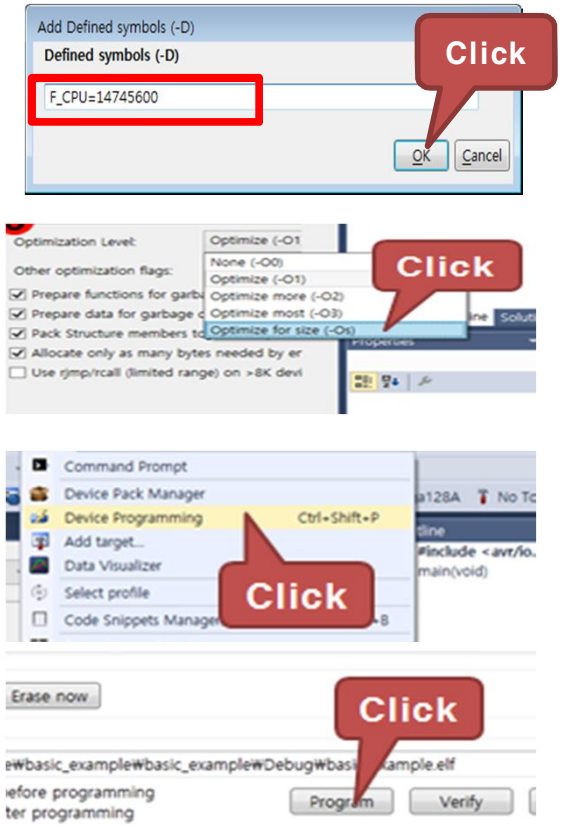
실습 2 : PWM으로 PIEZO 울리기

- 실습 준비
  - 다음 그림과 같이 연결
    - Edge-MCU보드의 PB7 → Edge-Peri 보드의 PIEZO(PIEZO상단우측)



실습 2 : PWM으로 PIEZO 울리기

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 07_PwmPiezo_Example, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compiler에서
    - 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



실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램
  - main.c 코드 작성

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

// 피아노 음계에 해당하는 PWM 주파수
unsigned int DoReMi[8] = {523,587, 659, 698, 783,880, 987, 1046};

int main(void)
{
  unsigned char piano=0;

  DDRB = 0x80;  // PWM 출력, OCR1C

  TCCR1A |= 0x0A; // COM1C(1:0)="10", OC1C핀사용, WGM3(1:0)="10"
  TCCR1B |= 0x19; // WGM3(3:2)="11", CS3(2:0)="001" 1분주 사용
  TCCR1C = 0x00;  // WGM3(3:0)="1110", Fast PWM, 모드 14
  TCNT1 = 0x0000; // 타이머1 카운터 초기화
```

실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램
  - main.c 코드 작성

```
int main(void) {
  //...
  while(1)
  {
    ICR1 = 14745600/DoReMi[piano];        // 버튼에 맞는 음향을 연주
    OCR1C = ICR1/2;                       // 50% 듀티비
    piano++;                              // piano 변수 1증가
    if(8 < piano) piano = 0;              // piano가 9가 되면 초기화
    _delay_ms(1000);
  }
}
```

실습 2 : PWM으로 PIEZO 울리기

- 실행 결과
  - PIEZO에서 '도레미파솔라시도' 소리가 출력

응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 실습 개요
  - 타이머를 이용하여 원하는 주파수의 신호를 만들고, 이를 Piezo에 입력하여 여러 가지 소리를 내도록 설계
  - 16비트 타이머인 타이머/카운터 3을 사용하며, 동작모드는 CTC 모드을 사용
  - 각기 다른 스위치 입력에 따라, 다른 주파수의 신호를 만듦

- 실습 목표
  - 16비트 타이머/카운터 활용 방법의 습득(관련 레지스터 이해)
  - Piezo의 동작원리 이해

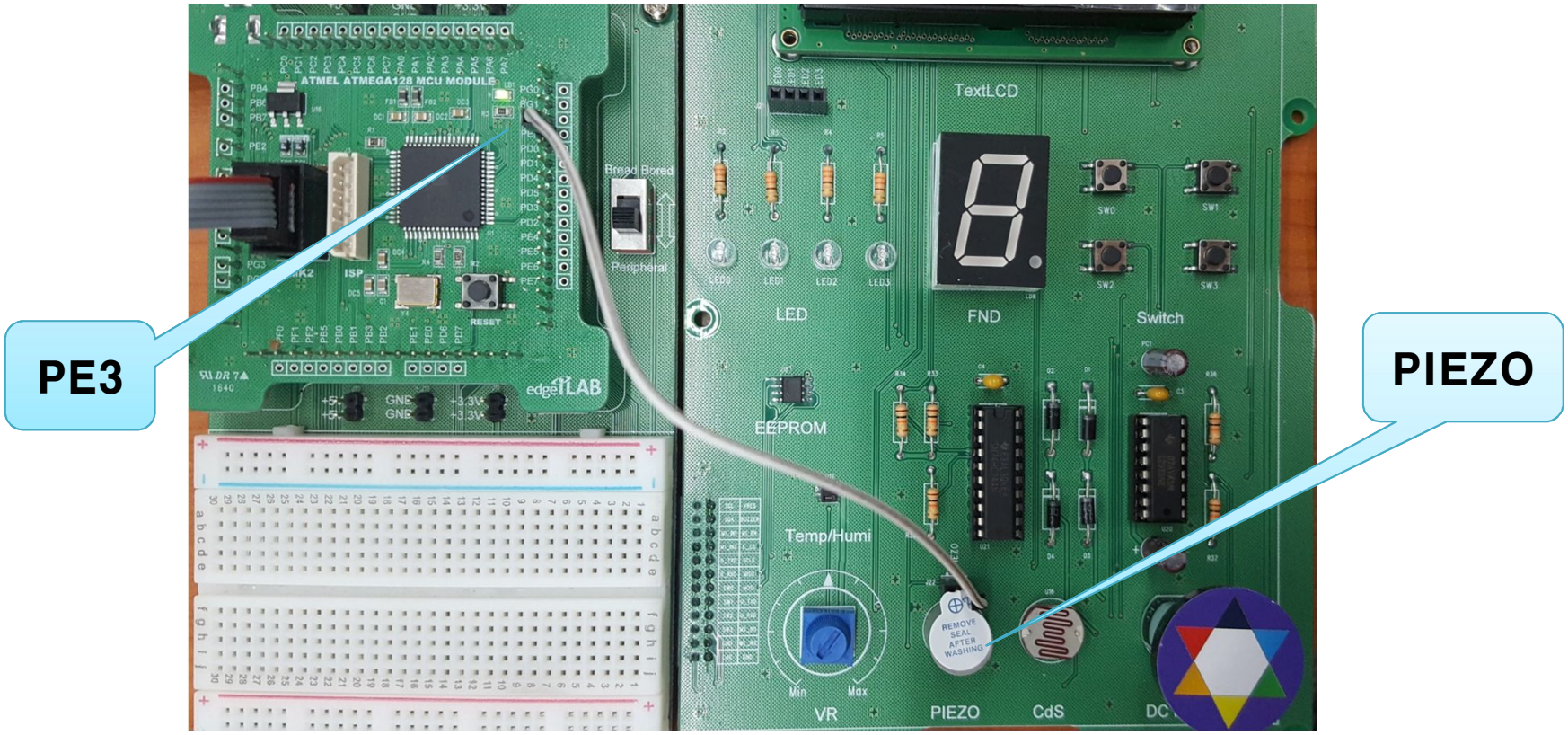
응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

edgeILAB

- 구동 프로그램 : 사전 지식
  - 타이머를 이용하여 Piezo를 여려 가지 소리가 나도록 함
  - 16비트 타이머/카운터 3을 이용/동작 모드는 14번 Fast PWM 모드를 사용
  - TCCR3A/TCCR3B/TCCT3C 레지스터들을 적절히 세팅
    - Piezo에 입력될 신호는 OC3A핀을 사용
    - WGM(3:0)은 "1110"으로 세팅
    - COM3C(1:0)비트를 "00"로 설정하여 normal 모드를 사용
  - OC3A핀 출력신호의 주파수
    - OC3A 주파수 = 메인클럭 / (분주비 * (1 + ICR3))

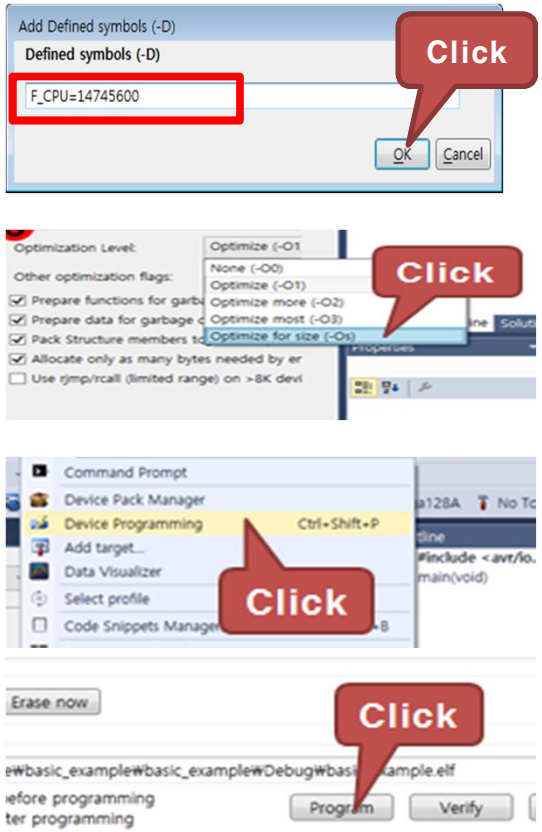
응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 실습 준비
  - 다음 그림과 같이 연결
    - Edge-MCU보드의 PE03 → Edge-Peri 보드의 PIEZO(PIEZO상단우측)



응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 예제 프로그램 작성 및 구동
  - Atmel Studio 실행
  - New Project 생성
    - Name : 07_PwmPiezo_Application, Location : D:₩AVR_Example
    - Device Selection : ATmega128A
  - 프로젝트 설정
    - Project 탭에서 "... Properties..." 선택
    - Toolchain -> AVR/GNU C Compiler에서
    - 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



응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 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
//피아노 음계에 해당하는 PWM 주파수
unsigned int DoReMi[8] = {523, 587, 659, 698, 783,880, 987, 1046};
volatile unsigned char sound_flag = 1;

int main(void)
{
  DDRE = 0x08;  // 포트E PE3를 출력 나머지는 입력포트로 설정한다.
                // Buzzer를 PE3에 연결
  TCCR3A = 0x00;          // WGM3(1:0) = “00”
  TCCR3B = 0x19;          // WGM3(3:2) = “11”   , CS3(2:0) = “001”  1
분주 사용
  TCCR3C = 0x00;  // WGM3(3:0) = “1110”, Fast PWM, 모드 14
  TCNT3 = 0x0000; // 타이머3 카운터 초기화
```

응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 구동 프로그램
  - main.c 코드 작성

```
  EICRB = 0xFF; // 인터럽트 4, 5, 6, 7을 상승엣지에서 동작하도록 설정
한다.
  EIMSK = 0xF0; // 인터럽트 4, 5, 6, 7을 허용
  EIFR = 0xF0;  // 인터럽트 4, 5, 6, 7 플래그를 클리어
  sei();
  while(1)
  {
    if(sound_flag)
    {
      _delay_ms(2000);  // 2초 지연
      TCCR3A = 0x00;   // 부저 소리를 끈다.
      sound_flag = 0;
    }
  }
}
```

응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT4_vect)          // 인터럽트 서비스 루틴
{
  cli();                   // 전체 인터럽트를 금지
  ICR3 = 14745600/DoReMi[0]/2; // 도의 음향을 연주한다
  TCCR3A = 0x40;           // PE4로 출력
  sound_flag = 1;          // 부저 음이 발생하도록 설정
  sei();                   // 전체 인터럽트를 허용
}
SIGNAL(INT5_vect)          // 인터럽트 서비스 루틴
{
  cli();                   // 전체 인터럽트를 금지
  ICR3 = 14745600/DoReMi[1]/2; // 래의 음향을 연주한다
  TCCR3A = 0x40;           // PE4로 출력
  sound_flag = 1;          // 부저 음이 발생하도록 설정
  sei();                   // 전체 인터럽트를 허용
}
```

응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 구동 프로그램
  - main.c 코드 작성

```
SIGNAL(INT6_vect)  // 인터럽트 서비스 루틴
{
  cli();            // 전체 인터럽트를 금지
  ICR3 = 14745600/DoReMi[2]/2; // 미의 음향을 연주한다
  TCCR3A = 0x40;    // PE4로 출력
  sound_flag = 1; // 부저 음이 발생하도록 설정
  sei();           // 전체 인터럽트를 허용
}
SIGNAL(INT7_vect) // 인터럽트 서비스 루틴
{
  cli();           // 전체 인터럽트를 금지
  ICR3 = 14745600/DoReMi[3]/2; // 파의 음향을 연주한다
  TCCR3A = 0x40;  // PE4로 출력
  sound_flag = 1; // 부저 음이 발생하도록 설정
  sei();           // 전체 인터럽트를 허용
}
```

응용 2 : 스위치 입력 값에 따라 PIEZO 울리기

- 실행 결과
  - 스위치에 따라 '도', '레', '미', '파' 출력