10 장 : 버저로 노래 연주하기

JCnet 게임아카데미 신 상 석

목차

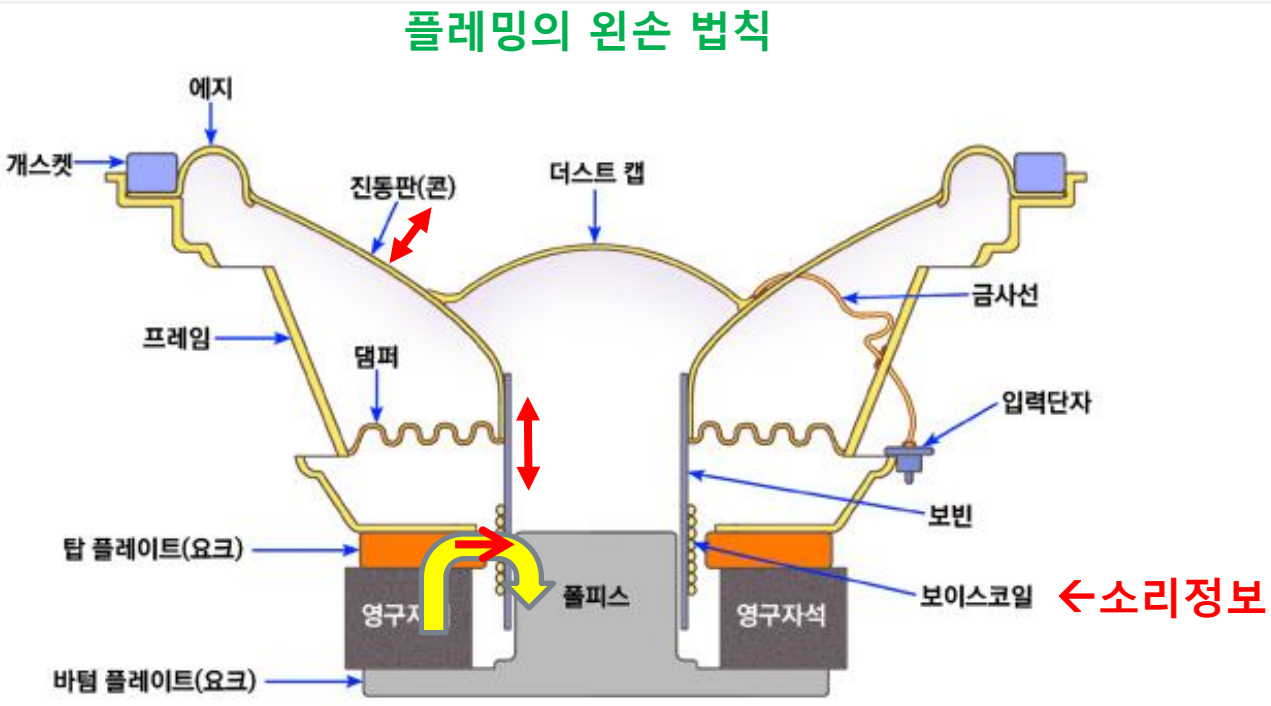
- 버저(Buzzer)
- JKIT-128-1에서의 버저 연결 설계
- 실습 BZ-1 : 버저로 다양한 소리내기
- 음계
- ATmega128 8비트 타이머/카운터
- 실습 BZ-2 : 버저로 노래 연주하기

버저 (Buzzer)

- 버저 (Buzzer)
  - 전자석의 코일에 단속적(斷續的)으로 전류를 보내어 철판 조각을 진동시켜 내는 신호 또는 그런 장치 (네이버 사전)
- 스피커 (Speaker)
  - 버저와 기본 원리를 동일하나, Diaphragm 이라 불리는 진동막을 진동시켜 소리를 구현하는 장치. 음의 재생 대역 및 음색이 다양해서, 사람의 소리 뿐 아니라, 사람이 들을수 있는 거의 모든 음을 재생시킬수 있는 음향 재생 소자

버저 (Buzzer)

- 버저/스피커의 원리



버저 (Buzzer)

- 버저 종류
  - Active Buzzer
    - 일정한 DC 전류에 반응
    - 내부에 Oscillator 회로가 있어서 일정한 주파수의 DC pulse 형태로 변환됨
    - 소리가 결정되어 있음
  - Passive Buzzer
    - DC pulse 형태의 전류에 반응
    - DC pulse의 주파수 형태에 따라 다른 소리 생성 가능

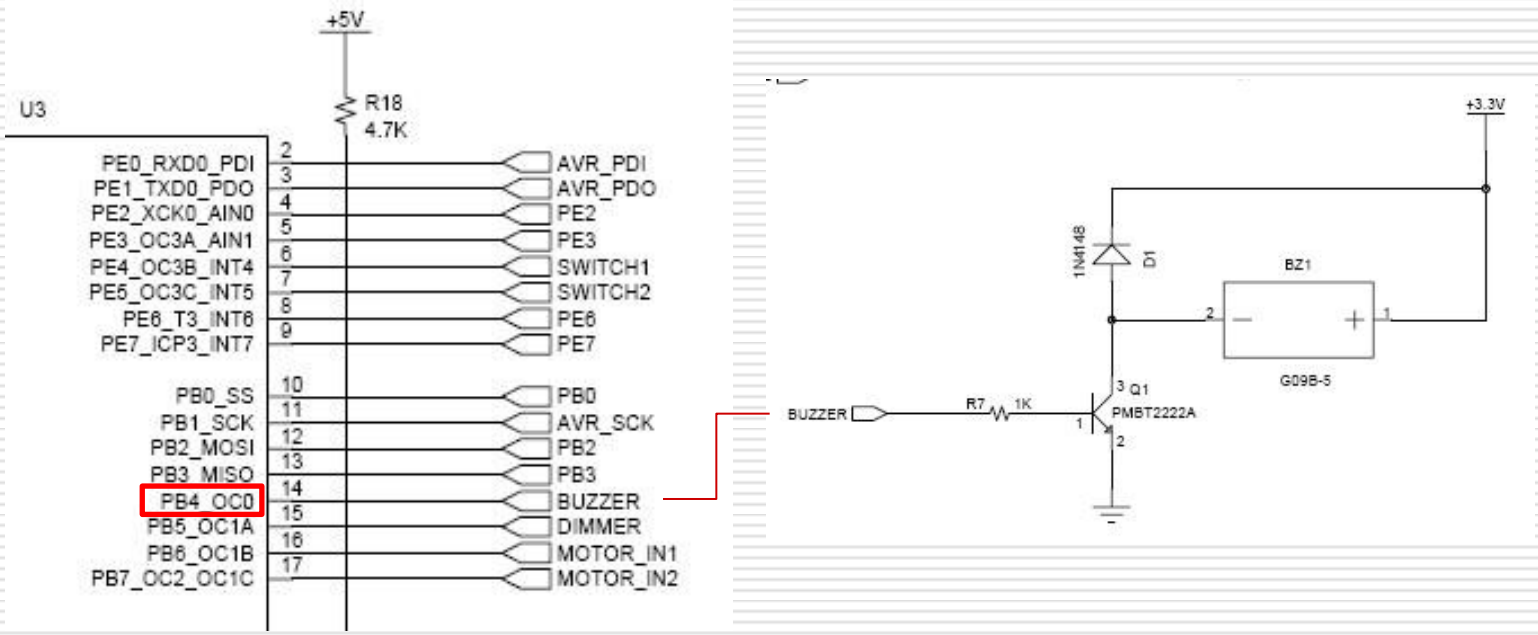
JKIT-128-1의 버저는 무슨 버저일까요?

JKIT-128-1 에서의 버저 연결 설계

- JKIT-128-1에서의 버저 연결 설계 개념
  - 버저는 음계를 만들 수 있어야하므로 Passive Buzzer 사용하며 1개만 제공
  - 버저를 제어하는 포트는 입출력 포트 중 PWM(Pulse Width Modulation)이 가능한 출력 포트인 PB(PB4)에 할당
  - 버저는 저전압(3.3V)용으로 설계
  - 버저는 전류를 많이 소모하므로 트랜지스터 구동회로를 연결하여 사용
  - 버저는 극성이 있으므로 이에 주의하여 연결하고 유도기전력에 의한 과전류로부터 보호하기 위하여 버저 양단에는 역방향으로 다이오드(Flywheel Diode)를 병렬로 삽입
  - 실습용이므로 구하기 쉽고 가격이 저렴한 것으로 설계

JKIT-128-1 에서의 버저 연결 설계

- JKIT-128-1 에서의 buzzer 연결 설계



실습 BZ-1 : 버저로 다양한 소리내기

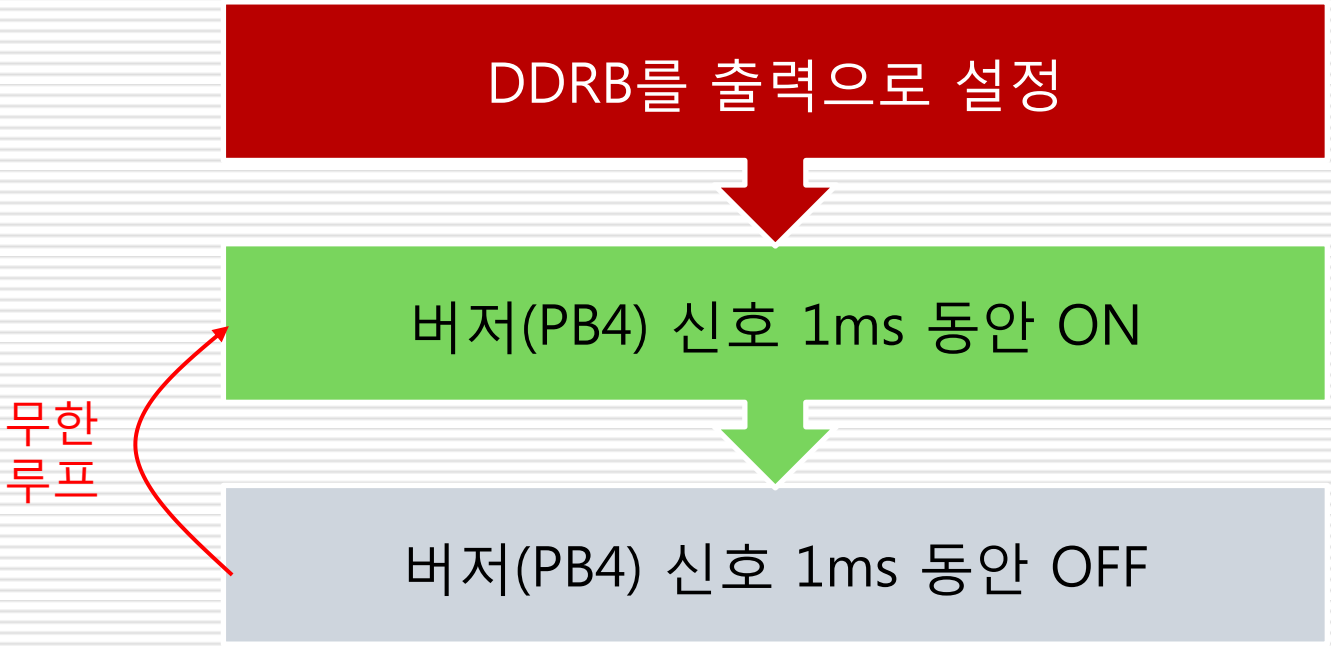
- 실습 내용
  - 버저를 이용하여 삐~ 소리 내기
  - 버저를 이용하여 전화벨 소리 내기
  - 버저를 이용하여 경찰차 사이렌 소리 내기 (혼자 해보기)

실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 설계 : 삐~ 소리 (buzzer_1_1.c)
  - 버저를 이용하여 소리를 생성할 수 있도록 버저와 연결된 포트를 제어
  - Passive 버저이므로 'On'과 'Off'이 교대로 일어나도록 하며, 일단 알기 쉽게 500 Hz로 동작하도록 프로그램 작성
  - 버저는 B 포트 bit4에 연결되어 있으므로 이것을 1ms 동안 'On', 다음 1ms 동안 'Off'이 계속적으로 반복되도록 프로그램하면 됨

실습 BZ-1 : 버저로 다양한 소리내기

- 구동 프로그램 설계 : 삐~ 소리 (buzzer_1_1.c)



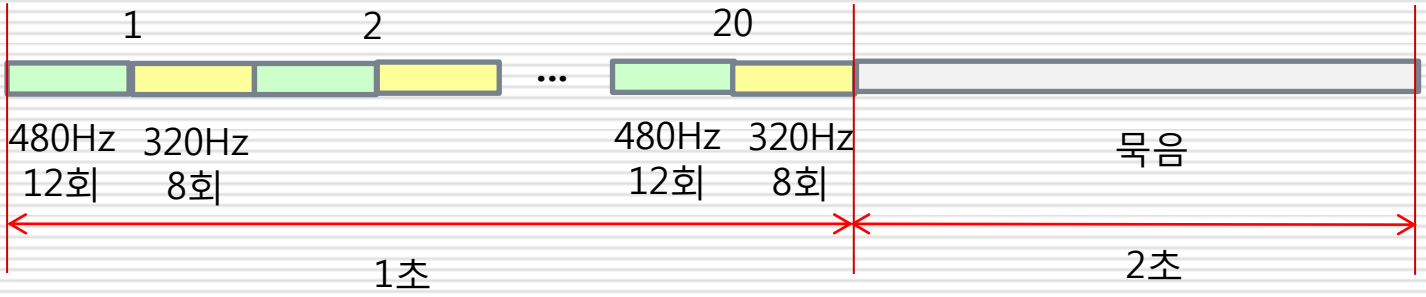
실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 코딩 : 삐~ 소리 (buzzer_1_1.c)

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
int main()
{
        DDRB = 0x10;                // 포트 B의 bit4 를 출력 상태로 세팅
        while(1)                    // 500 Hz로 동작
        {
                PORTB = 0x10;       // 1ms 동안 'On' 상태 유지
                _delay_ms(1);
                PORTB = 0x00;       // 1ms 동안 'Off' 상태 유지
                _delay_ms(1);
        }
}
```

실습 BZ-1 : 버저로 다양한 소리내기

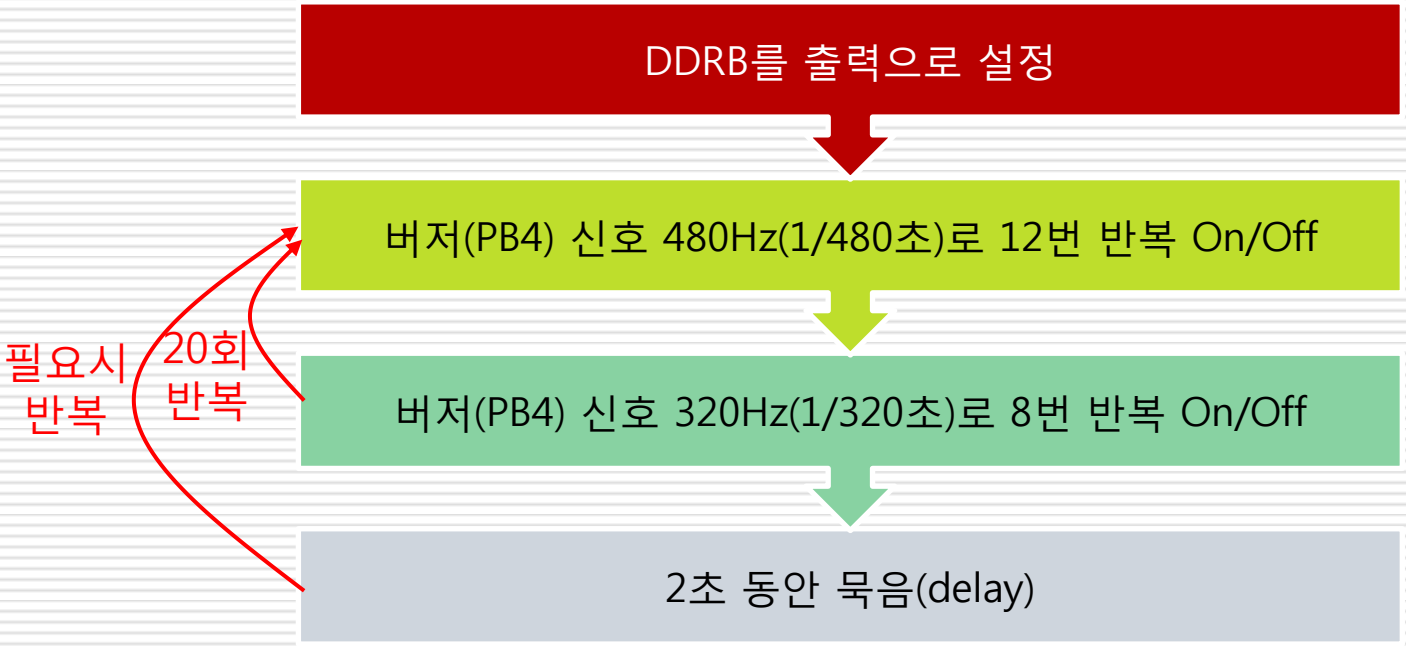
- 구동프로그램 설계 : 전화벨 소리 (buzzer_1_2.c)
  - 여러가지 주파수의 소리를 혼합하여 반복하면 다양한 소리 발생
  - 소리 내는 방법은 buzzer_1_1.c 와 동일
  - 전화벨 소리 구성



  - 반복해서 처리해야 하므로 buzzer(cycle_time, count) 함수를 정의하여 프로그램하는 것이 편리

실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 설계 : 전화벨 소리 (buzzer_1_2.c)



실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 코딩 : 전화벨 소리 (buzzer_1_2.c)

```
#include <avr/io.h>
#define F_CPU 16000000UL
#define
__DELAY_BACKWARD_COMPATIBLE_
_          // for Atmel Studio 6
#include <util/delay.h>

void buzzer(int hz, int count);

int main()
{
    int i;
    DDRB = 0x10;
    while(1)
    {
        for(i=0; i<20; i++)
        {
            buzzer(480, 12);
            buzzer(320, 8);
        }
        _delay_ms(2000);
    }
}
```

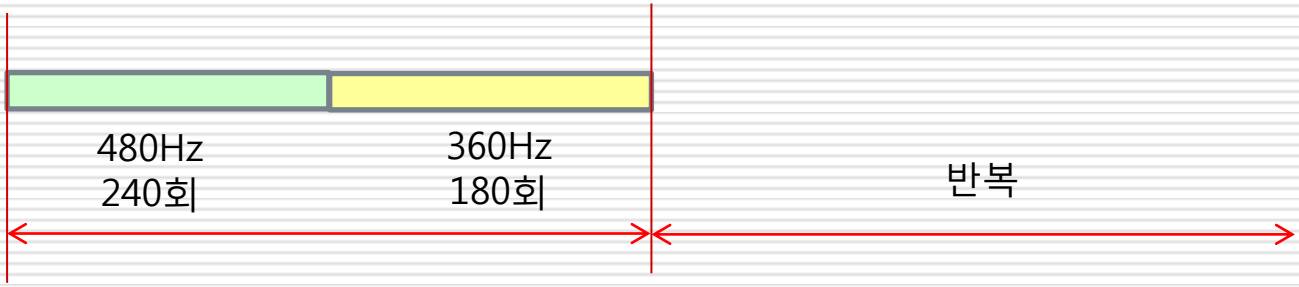
실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 코딩 : 전화벨 소리 (buzzer_1_2.c)

```
void buzzer(int hz, int count)
{
    int i, j, ms;
    ms = 500/hz;
    for(i=0; i<count; i++)
    {
        PORTB = 0x10;
            _delay_ms(ms);
        PORTB = 0x00;
            _delay_ms(ms);
    }
}
```

실습 BZ-1 : 버저로 다양한 소리내기

- 구동프로그램 설계 : 앰블런스 소리 (buzzer_1_3.c)
  - 소리 내는 방법은 전화벨 소리 내는 방법과 비슷
  - 앰블런스 소리 구성



  - 반복해서 처리해야 하므로 buzzer(hz, count) 함수를 정의하여 프로그램하는 것이 편리

음계

- 음의 구성
  - 음의 높이(음계, tone), 길이(박자, beat), 강약, 색깔로 구분
- 음계
  - 반음까지 포함하여 한 옥타브에는 12개의 음이 존재
  - 이웃하는 음계 사이에는 주파수가 1: $2^{1/12}$ 의 비율이 성립하며, 한 옥타브 간 같은 음계의 차이는 2배 차이가 남
  - 음계는 이미 표준 주파수가 정해져 있음
  - 음계표 (6 옥타브)

음계	도	레	미	파	솔	라	시	도
주파수	1046.6	1174.6	1318.6	1397.0	1568.0	1760.0	1795.6	2093.2

ATmega128 8비트 타이머/카운터

- 타이머
  - MCU 내부 클럭을 세는 장치
  - MCU의 내부 클럭을 세어 일정시간 간격의 펄스를 만들어 내거나 일정시간 경과 후에 인터럽트를 발생
  - 보통 동기모드로 사용
- 카운터
  - MCU의 외부에서 입력되는 클럭을 세는 장치
  - 외부 핀(TOSC1, TOSC2, T1, T2, T3)을 통해서 들어오는 펄스를 세어(Edge Detector) 특정 값이 되거나 Overflow 등이 생기면 인터럽트를 발생
  - 보통 비동기모드로 동작

ATmega128 8비트 타이머/카운터

- ATmega128의 타이머/카운터
  - 모두 4개의 타이머/카운터를 보유
    - **타이머 0, 2** : **8비트 타이머**로 서로 기능 유사
    - **타이머 1, 3** : **16비트 타이머**로 서로 기능 유사
  - 인터럽트 기능
    - 오버플로우(Overflow) 인터럽트 : 카운터의 값이 오버플로우되는 경우에 발생, 즉, 8비트 타이머의 경우 값이 0xff 에서 0x00으로 넘어갈 때 발생
    - 출력비교(Output Compare) 인터럽트 : 카운터 값이 출력비교 레지스터의 값과 같게 되는 순간에 발생

ATmega128 8비트 타이머/카운터

- 8비트 타이머/카운터의 특징
  - 4개의 타이머/카운터 중 0번과 2번 타이머/카운터
  - 비동기 동작 모드를 갖는 8비트 업/다운(Up/Down) 카운터
  - 8비트 카운터 : 0~255까지 셀 수 있음
  - 10비트의 프리스케일러(prescaler) 보유
  - 인터럽트 기능
    - 오버플로우 인터럽트(Overflow Interrupt)
    - 출력비교 인터럽트(Output Compare Match Interrupt)
  - PWM(Pulse Width Modulation) 기능 제공
  - 타이머 0는 부가적으로 32.768KHz의 수정 진동자를 TOSC1,2 단자에 연결해 Real Time Clock으로 사용할 수 있는 기능도 제공

ATmega128 8비트 타이머/카운터

- 8비트 타이머/카운터 0 관련 레지스터
  - 타이머/카운터 레지스터(TCNT0)
  - 타이머/카운터 제어 레지스터(TCCR0)
  - 타이머/카운터 인터럽트 마스크 레지스터 (TIMSK)

ATmega128 8비트 타이머/카운터

- TCNT0(Timer/Counter Register 0)
  - 타이머/카운터 0의 8비트 값을 저장하고 있는 레지스터
  - Read/Write 가능
  - 레지스터의 값은 인터럽트가 걸리면 자동으로 0으로 클리어 되고 다시 분주된 주기마다 1씩 값 증가

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

ATmega128 8비트 타이머/카운터

- TCCR0(Timer/Counter Control Register 0)
  - 타이머/카운터 제어 레지스터 0
  - 동작 모드, 프리스케일러등 타이머/카운터의 전반적인 동작 형태를 결정

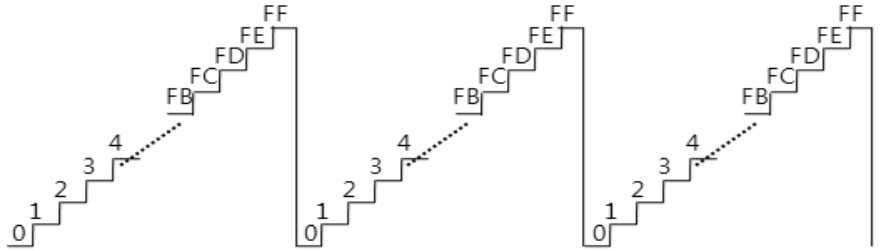
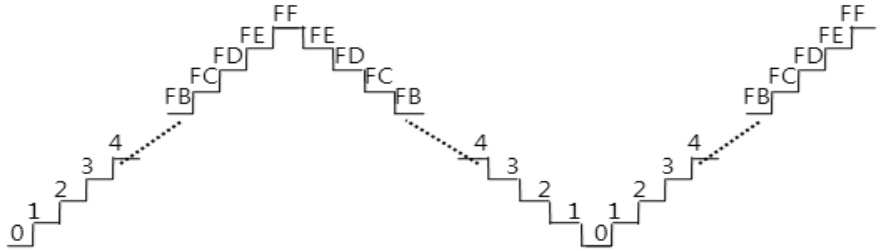
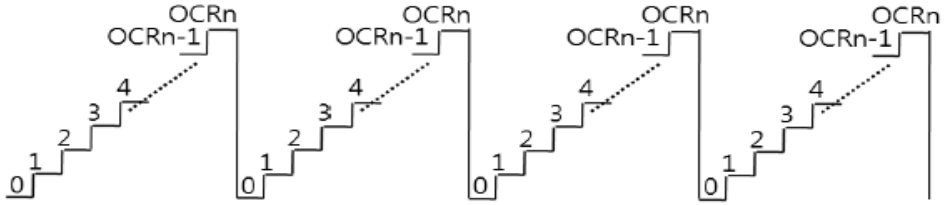
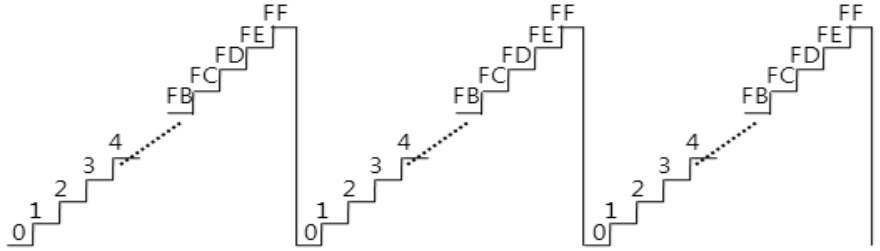
7	6	5	4	3	2	1	0
FOCn	WGMn0	COMn1	COMn0	WGMn1	CSn2	CSn1	CSn0

0 0 0 0 0 0 1 1 = 0x03

- 비트7 : FOCn (Force Output Compare) = 0 (비교 안함)
- 비트 6,3 : WGM (Waveform Generation Mode) = 00 (Normal)
- 비트 5,4 : COM (Compare Output Mode) = 00 (Normal)
- 비트 2, 1, 0 : CSn (Clock Select) = 011 (32분주)

ATmega128 8비트 타이머/카운터

- 타이머/카운터 동작 모드 (WGM)
  - 일반 모드(Normal Mode)
    - 업 카운터로 동작, 0x00-0xFF(TOP)
    - 0xFF -> 0x00, 오버플로우 인터럽트 (TOV0) 발생
  - CTC 모드 (Clear Time on Compare Match Mode)
    - 지정된 값(OCR0) 까지 업 카운트, 0x00-OCR0
    - OCR0 -> 0x00, 출력비교 인터럽트 (OCF0) 발생
  - Fast PWM 모드(Fast PWM Mode)
    - 업 카운터로 동작, 0x00-0xFF(TOP)
    - 오버플로우 인터럽트 (TOV0)와 출력비교 인터럽트 (OCF0) 발생
  - PC PWM 모드 (Phase Correct PWM Mode)
    - 업 다운 카운터로 동작 , 0x00-0xFF-0x00 .....
    - 오버플로우 인터럽트 (TOV0)와 출력비교 인터럽트 (OCF0) 발생
  - **모든 경우 인터럽트 발생시마다 OCn 출력 신호는 Toggle**

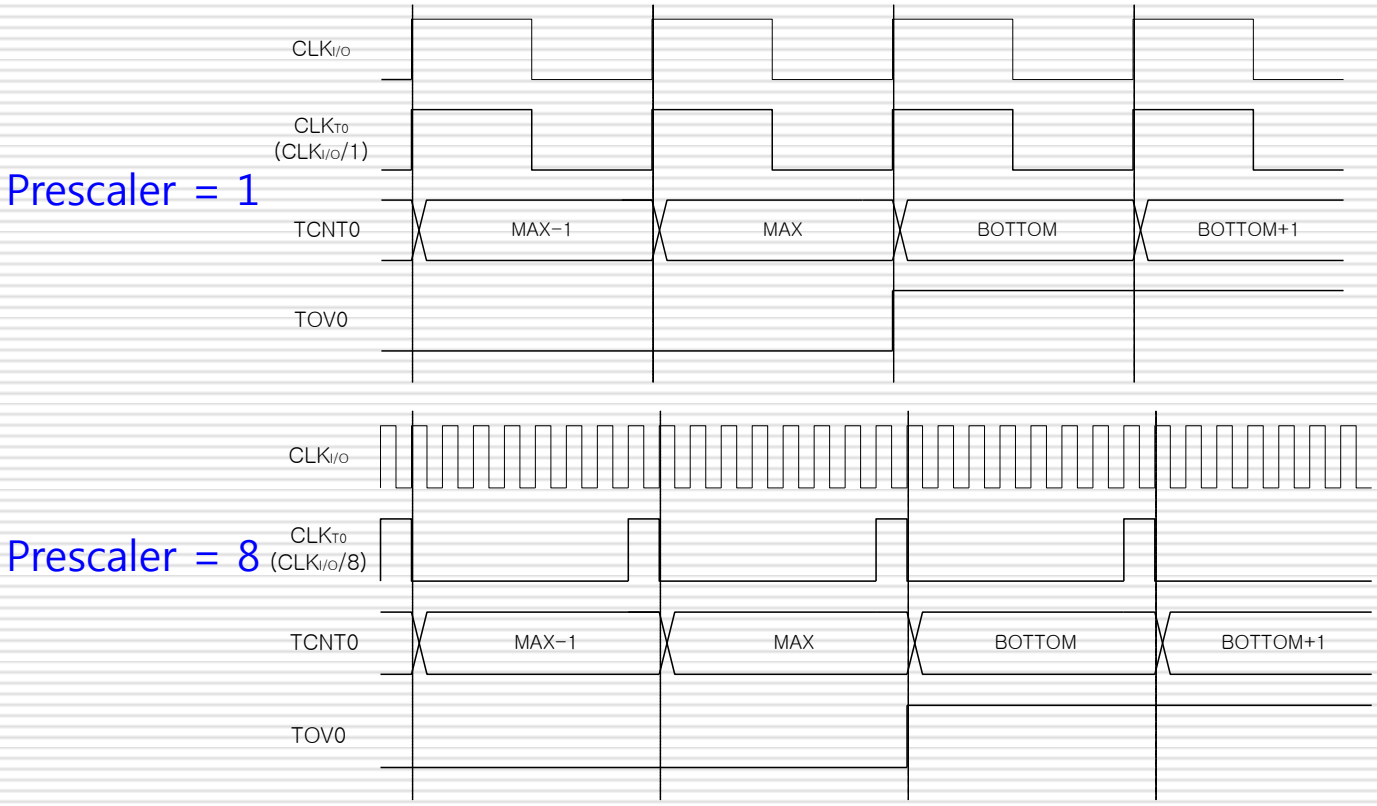
파형 발생모드	TCCRn 레지스터		최댓값 (TOP)	TCNTn 카운터의 16진 계수 순서
	WGMn1	WGMn0		
일반 모드	0	0	FF	
위상정정 PWM 모드	0	1	FF	
CTC 모드	1	0	OCRn	
고속 PWM 모드	1	1	FF	

ATmega128 8비트 타이머/카운터

- 프리스케일러(Prescaler)
  - 고속의 클럭을 사용하여 타이머를 동작시킬 때 나타나는 문제를 해결하기 위해 클럭을 분주하여 더 느린 타이머 클럭을 생성
  - 16MHz 클럭을 사용하는 경우 그 클럭의 주기는 62.5 ns로 이 클럭으로 256까지 센다고 해도 16 us 까지만 카운트 가능
  - 10비트 프리스케일러를 보유한 경우, 최대 2의 10제곱 = 1024배 가능
  - 프리스케일러를 1024로 하면 16MHz 클럭의 타이머 클럭은 주기가 62.5 ns * 1024 = 64 us가 되고, 이 클럭을 256까지 센다면 64 us * 256 = 16.384 ms 크기의 타이머를 만들 수 있음.

ATmega128 8비트 타이머/카운터

- 프리스케일러(Prescaler)



ATmega128 8비트 타이머/카운터

- TCCR0(Timer/Counter Control Register 0)
  - CS2-0 (Clock Select) : 클럭 및 프리스케일러 선택

CS02	CS01	CS00	설명
0	0	0	클록 입력 차단
0	0	1	No 프리스케일러, 1분주
0	1	0	8분주
0	1	1	32분주
1	0	0	64분주
1	0	1	128분주
1	1	0	256분주
1	1	1	1024분주

ATmega128 8비트 타이머/카운터

- TIMSK(Timer Interrupt Mask)
  - 타이머 인터럽트 마스크 레지스터
  - 타이머/카운터0, 타이머/카운터1, 타이머/카운터2가 발생하는 인터럽트를 개별적으로 Enable하는 레지스터

7	6	5	4	3	2	1	0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
0	0	0	0	0	0	0	1 = 0x01

  - 비트 0 (TOIE0) : 타이머/카운터 0의 오버플로우 인터럽트 인에이블
  - 비트 1 (OCIE0) : 타이머/카운터 0의 출력비교 인터럽트 인에이블

ATmega128 8비트 타이머/카운터

- 타이머로 원하는 시간을 세팅하는 방법 (예 100us)
  - Prescaler의 값을 정하여 타이머의 기본 주기를 계산
    - 예를 들어, 16MHz 클럭, Prescaler=32이면, 타이머 클럭의 주기는 (1/(16*1000000)) *32 = 2 (us)
  - 필요한 지연시간을 타이머의 기본 주기로 나누어 클록 사이클을 계산
    - 예를 들어, 100us의 지연시간을 얻으려면 100/2 = 50 (클록)이 필요
  - 256-클록사이클 값을 TCNT0에 저장하고, Overflow Interrupt를 Enable 상태로 만듬
    - 위의 예에서 256-50=206 값을 TCNT0에 저장하면, 이후 Overflow Interrupt가 발생하는 시점이 원하는 지연시간 후인 100us 이후가 됨

실습 BZ-2 : 버저로 노래 연주하기

- 실습 내용
  - 버저를 이용하여 '도' 음 소리 내기
  - '산토끼' 등의 1 옥타브용 동요 음계 연주하기

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '도' 음 소리내기 (buzzer_2_1.c)
  - main() 프로그램
    - '도' 음의 주파수는 1046.6 Hz이므로, 주기는 1/1046.6 초인 956us가 되며, 478us 동안 'ON', 478us 동안 'OFF' 상태를 유지하면 됨

    - 카운터/타이머0을 이용하여 아래와 같이 세팅
      - Prescaler로 32분주를 선택하여 주기를 2us로 만듬
      - 478us/2us = 239 클록을 카운팅해야 하므로 256-239=17 값을 TCNT0에 Write
      - TIMSK의 TOIE0(Overflow 인터럽트) 설정 및 전역인터럽트 설정

실습 BZ-2 : 버저로 노래 연주하기

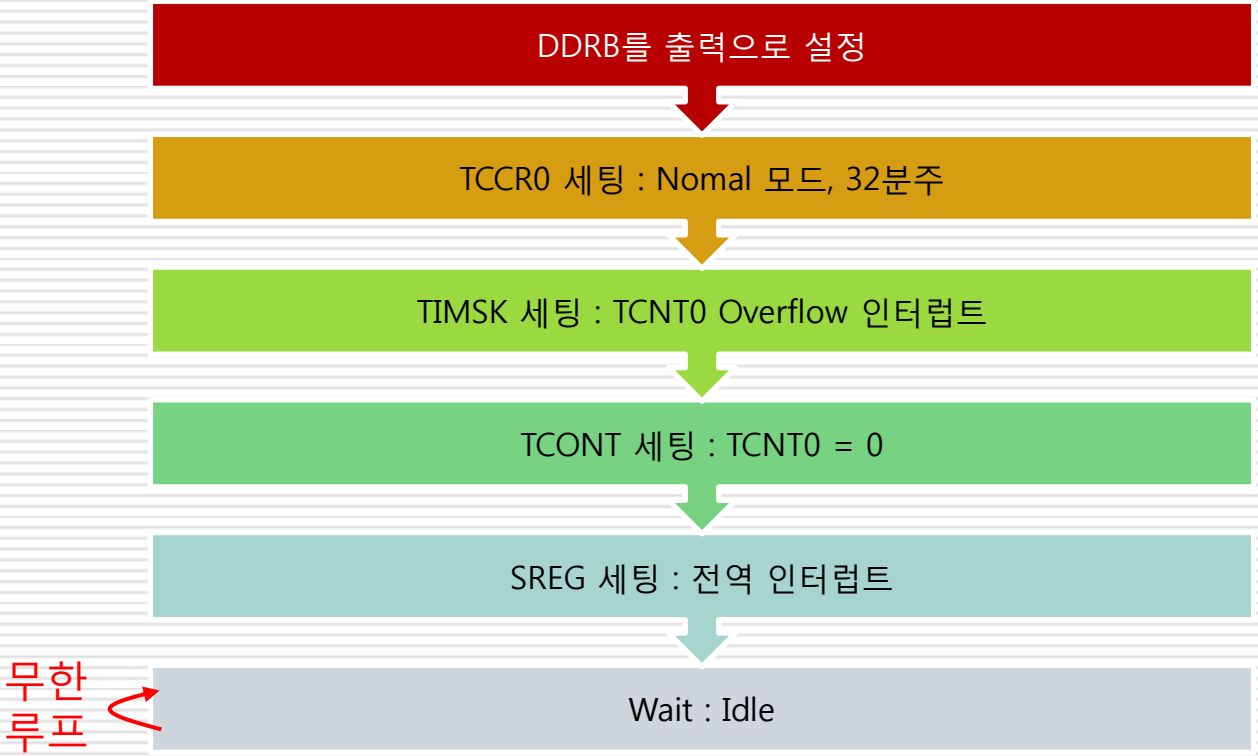
- 구동프로그램 설계 : '도' 음 소리내기 (buzzer_2_1.c)
  - 타이머/카운터0 Overflow 인터럽트 서비스 프로그램
    - state를 검사하여 'ON' 상태면 state를 'OFF' 상태로 하고 buzzer 신호를 'OFF' 시키고, 'OFF' 상태면 state를 'ON' 상태로 하고 buzzer 신호를 'ON' 시킴
    - 인터럽트가 동일한 간격으로 계속 실행될 수 있도록 TCNT0를 다시 초기값인 17로 초기화
  - 참고 : 음계에 따른 TCNT0 값 구하기

TCNT0 = 256 - (((1/주파수) * 1000000)/2)/2

음계	도	레	미	파	솔	라	시	도
주파수	1046.6	1174.6	1318.6	1397.0	1568.0	1760.0	1795.6	2093.2
TCNT0	17	43	66					

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '도' 음 소리내기 (buzzer_2_1.c)
  - main() 프로그램



실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '도' 음 소리내기 (buzzer_2_1.c)
  - TCNT0 Overflow 인터럽트 프로그램

If state ON → OFF, buzzer OFF, else (OFF) → ON, buzzer ON,

TCNT0 = 17 ('도' 값)

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '도' 음 소리내기 (buzzer_2_1.c)

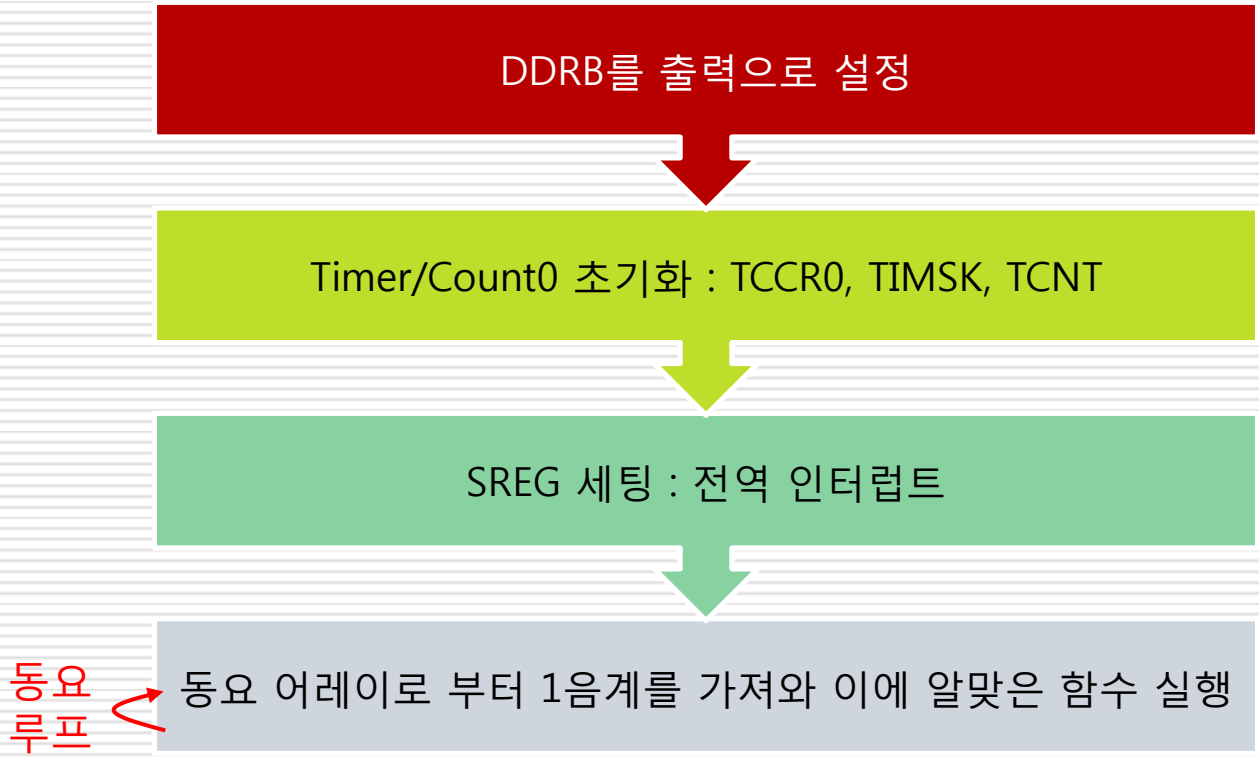
```
#include <avr/io.h>
#include <avr/interrupt.h>
#define ON 1
#define OFF 0
#define DO_data 17
volatile int state = OFF;

ISR(TIMER0_OVF_vect)
{
        if (state == ON)
        {       PORTB = 0x00;
                state = OFF;
        }
        else
        {       PORTB = 0x10;
                state = ON;     }
        TCNT0 = DO_data;
}
int main()
{
        DDRB = 0x10;
        TCCR0 = 0x03;    // 8분주
        TIMSK = 0x01;    // Overflow
        TCNT0 = DO_data;
        sei();      // 전역 인터럽트
        while(1);
}
```

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '산토끼' 연주하기 (buzzer_2_2.c)
  - '도' 음 소리낸 것과 동일하게 다른 음계도 계산된 값을 TCNT0에 세팅하는 방법을 적용하여 소리낼 수 있음
  - 동요의 음계를 어레이로 저장하여 이를 순서대로 실행
  - main() 프로그램
    - 카운터/타이머0 초기화 (TCCR0, TIMSK, TCNT0)
    - 동요 어레이로 부터 한 음씩 읽어서 음계를 소리내는 함수 실행
    - 단, 음계를 소리내는 함수는 TCNT0 Overflow 인터럽트를 이용하여 정확한 음계를 소리냄
  - TCNT0 Overflow 인터럽트 서비스 루틴
    - state를 검사하여 'ON', 'OFF' 상태에 따라 state를 반대로 변경하고 buzzer도 반대로 ON, OFF 시킴
    - 인터럽트가 동일한 간격으로 계속 실행될 수 있도록 TCNT0를 음계에 알맞은 값으로 초기화

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : '산토끼" 연주하기 (buzzer_2_2.c)
  - main() 프로그램



실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : "산토끼" 연주하기 (buzzer_2_2.c)
  - TCNT0 Overflow 인터럽트 프로그램

If state ON → OFF, buzzer OFF, else (OFF) → ON, buzzer ON,

TCNT0 = 음계값

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : "산토끼" 연주하기 (buzzer_2_2.c)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#define DO 0
#define RE 1
#define MI 2
#define FA 3
#define SOL 4
#define RA 5
#define SI 6
#define DDO 7
#define EOS -1
```

```
#define ON 0
#define OFF 1
volatile int state, tone;
char f_table[8] = {17, 43, 66, 77, 97,
114, 117, 137};
int song[] = {SOL, MI, MI, SOL, MI,
DO, RE, MI, RE, DO, MI, SOL, DDO,
SOL, DDO, SOL, DDO, SOL, MI, SOL,
RE, FA, MI, RE, DO, EOS};
```

실습 BZ-2 : 버저로 노래 연주하기

- 구동프로그램 설계 : "산토끼" 연주하기 (buzzer_2_2.c)

```
ISR(TIMER0_OVF_vect)
{
        if (state == ON)
        {
                PORTB = 0x00;
                state = OFF;
        }
        else
        {
                PORTB = 0x10;
                state = ON;
        }
        TCNT0 = f_table[tone];
}
```

```
int main()
{
        int i=0;
        DDRB = 0x10;
        TCCR0 = 0x03;// 8분주
        TIMSK = 0x01;// Overflow
        TCNT0 = f_table[song[i]];
        sei();
        do {
                tone = song[i++];
                _delay_ms(500);
        }while(tone != EOS);
}
```

숙제

- 내용 : “산토끼” 연주하기 프로그램 작성
  - <buzzer_2_2.c> 동요연주하기 프로그램은 아래 2가지 면에서 문제점이 있는데, 이를 해결하여 음계과 길이까지 잘 맞는 프로그램으로 수정 작성
    - 한 음의 연주 길이가 모두 동일함 (힌트 : 연주 시간의 문제)
    - 동일한 음이 연이어서 나오면 한 개의 음으로 합쳐져서 들림 (힌트 : 잠시 끊어지는 묵음을 사이에 넣으면 됨)
- 제출 기한 : 다음 수업시간 시작 전까지
- 제출 방법 : eclass에 “학번-이름-BUZZ.zip” 파일로 제출

묻고 답하기

Q & A