AVR – HW3

임베디드스쿨1기

Lv1과정

2020. 10. 09

강경수

1. REVIEW

▣ UART 더 파고들기

2020.10.08 KSS

1. ₩r ₩n의 차이?

```
UART_transmit_str("UART_INIT\r");
UART_transmit_str("UART_INIT\n");
```

← 옆과 같은 코드 실행시
결과는 똑같이 우측과 같음

```
UART_INIT
UART_INIT
UART_INIT
UART_INIT
UART_INIT
UART_INIT
```

₩r 은 현재줄의 처음 ₩n은 다음줄의 처음

2. "str"의 데이터 영역
- 문자열을 사용시 따로 데이터 버퍼에 저장하지 않고 사용하였음.
- 문자열은 data영역에 저장됨.

1. REVIEW

3. 문자열 수신 및 재송신 구현

```
int main(void)
{
    UART_init();
    uint8_t *r_data;

    while(1)
    {
        UART_receive_str();
        UART_transmit_str(r_data);
        _delay_ms(1000);
    }
    return 0;
}
```

이 부분을 인터럽트 ISR(uart_receive)로 대체해볼것..

```
unsigned char* UART_receive_str(void)
{
    uint8_t data[10] = {'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'};
    uint8_t i;

    data[0] = 1;
    i = 0;

    while(data[i] != '\0')
    {
        data[i] = UART_receive();
        i++;
    }
    return data;
}
```

정상 동작하지 않음... 관련 자료 더 찾아보고 구현해 볼것.

1. REVIEW

▣ I/O REGISTER MAPPING

```
typedef union {
    uint8_t byte;
    struct {  //익명 구조체!
        uint8_t bit0:1;
        uint8_t bit1:1;
        uint8_t bit2:1;
        uint8_t bit3:1;
        uint8_t bit4:1;
        uint8_t bit5:1;
        uint8_t bit6:1;
        uint8_t bit7:1;
    };
}ioreg8;

#define SFR_MEM8(addr) (*((volatile ioreg8*)(addr)))


#define PINB_REG SFR_MEM8(0x23)
#define DDRB_REG SFR_MEM8(0x24)
#define PORTB_REG SFR_MEM8(0x25)
#define PINC_REG SFR_MEM8(0x26)
#define DDRC_REG SFR_MEM8(0x27)
#define PORTC_REG SFR_MEM8(0x28)
#define PIND_REG SFR_MEM8(0x29)
#define DDRD_REG SFR_MEM8(0x2A)
#define PORTD_REG SFR_MEM8(0x2B)
//주소변수로 typecasting 후에 원래값 포인터.
// volatile을 쓴 이유 사실 DDRB 와 PORTB입장에서는 안써도됨
// 그러나 PIN값은 변하기 때문에 최적화로 인한 접근누락 방지를 위해 volatile 사용
```

2. ETC..

▣ TEST LCD돌려보기

```
int main(void)
{
    void MCU_init(void);
    _delay_ms(100);
    void LCD_init(void);
    _delay_ms(100);

    LCD_string(0xC0,"MYWORLD!");
    while(1)
    {
        ;
    }

    /* Replace with your application code */

}

void MCU_init(void)
{
    DDRD = 0xFF; //(TEXT LCD D0~D7)
    PORTD = 0x00;
    DDRC = 0x07; //(RS RW E OUTPUT)
    PORTC = 0x00;
}
```

2. ETC..

```
void LCD_init(void)
{
  LCD_command(0x30);
  _delay_ms(5);
  LCD_command(0x30);
  _delay_us(150);
  LCD_command(0x30);
  _delay_us(100);
  LCD_command(0x38);          // function set(8 bit, 2 line, 5x7 dot) //2i
  _delay_us(40);
  LCD_command(0x0C);          // display control(display ON, cursor OFF)
  _delay_us(40);
  LCD_command(0x06);          // entry mode set(increment, not shift)
  _delay_us(40);
  LCD_command(0x01);          // clear display
  _delay_ms(2);
}
```

2. ETC..

```
void LCD_command(uint8_t command)
{
    cbi(PORTC,RS);
    cbi(PORTC,EN);
    PORTD = command;
    sbi(PORTC,EN);
    asm volatile(" PUSH  R0 ");          // delay for about 250 ns
    asm volatile(" POP   R0 ");
    cbi(PORTC,EN);
    _delay_us(50);
}

void LCD_data(uint8_t data)
{
    cbi(PORTC,EN);                  // E = 0, Rs = 1
    sbi(PORTC,RS);
    PORTD = data;                   // output data
    sbi(PORTC,EN);                  // E = 1
    asm volatile(" PUSH  R0 ");          // delay for about 250 ns
    asm volatile(" POP   R0 ");
    cbi(PORTC,EN);                  // E = 0
    _delay_us(50);
}
```

2. ETC..

```
void LCD_string(uint8_t command, char *string)  /* display a string on LCD */
{
    LCD_command(command);           // start position of string
    while(*string != '\0')          // display string
    {
        LCD_data(*string);
        string++;
    }
}
```

문자 출력 실패하였음.. 더 공부해서 LCD구동해보기

2. ETC..

▣ voltaile in embeded c

2020.10.08 KSS

1. 사용하는 경우
- Memory mapped i/o
- Interrupt
- Task (RTOS)

2. 설명
- 위 사항 모두 컴파일러 입장에서는 변수가 변하지 않을 수 있는 상황임.
- 따라서 컴파일러가 단순화시켜 처리할 수 있음.(optimaztion 옵션 켜져있을시)
- 이를 방지하기위해 volatile 변수 선언
- ex) int volatile* foo; <<대부분 이렇게 datatype에 volatile 선언
  struct union에도 똑같이 적용 가능 하다.

2. ETC..

3. volatile을 적용해야 하는 예시

```
UINT1 * ptr = (UINT1 *) 0x1234;          << phireperal register mapped

// Wait for register to become non-zero.
while (*ptr == 0);
// Do something else.
```

```
void main()                              << interrupt wait
{
   ...
   while (!ext_rcvd)
   {
      // Wait
   }
   ...
}

interrupt void rx_isr(void)
{
   ...
   if (ETX == rx_char)
   {
      etx_rcvd = TRUE;
   }
   ...
}
```

```
int cntr;                                << thread

void task1(void)
{
   cntr = 0;
   while (cntr == 0)
   {
      sleep(1);
   }
   ...
}
void task2(void)
{
   ...
   cntr++;
   sleep(10);
   ...
}
```

4. 참고

- https://archive.is/D7z3E

3. TIMER/COUNTER

▣ TIMER & COUNTER

2020.10.08 ㅏ

1. TIMER와 COUNTER의 차이
- CPU에는 CLOCK이 필요료 함. (상용 PC의 경우 수G Hz의 클록)
- TIMER : MCU의 자체적인 CLOCK(크리스탈,내부OSC등)을 사용하여 동작
- COUNTER : 외부 CLOCK(NE555등)을 CLOCK핀으로 받아 동작(PRESCALE 사용 불가)

2. CLOCK이란 무엇인가?
- 클록은 MCU의 동작 단위이다. 클록을 기준으로 어셈레벨의 명령어들이 실행 된다.
- 모든 어셈명령어가 1CLOCK을 소모하는것이 아니다.
- 어셈명령어마다 소요되는 클록이 다르다.
- 아래는 이런 CLOCK을 활용하여 어셈명령어들로 만들어낸 delay함수이다.

```
void Delay_us(U08 time_us)          /* time delay for us in 16MHz */
{
  register unsigned char i;

  for(i = 0; i < time_us; i++)          // 5 cycle +
    { asm volatile(" PUSH  R0 ");       // 2 cycle +
      asm volatile(" POP   R0 ");       // 2 cycle +
      asm volatile(" NOP      ");       // 1 cycle +
      asm volatile(" NOP      ");       // 1 cycle +
      asm volatile(" NOP      ");       // 1 cycle +
      asm volatile(" PUSH  R0 ");       // 2 cycle +
      asm volatile(" POP   R0 ");       // 2 cycle = 16 cycle = 1 us for 16MHz
    }
}
```

3. CLOCK을 만들어 내는 방법

High-speed on-chip oscillator

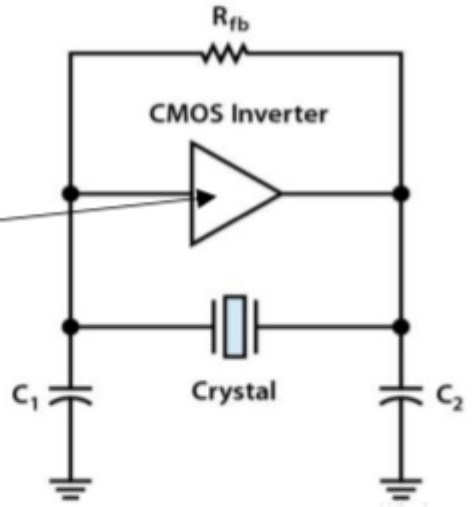
- Select from 32 MHz, 24 MHz, 16 MHz, 12 MHz, 8 MHz, 6 MHz, 4 MHz, 3 MHz, 2 MHz, and 1 MHz
- High accuracy: +/- 1.0 % (V_{DD} = 1.8 to 5.5 V, T_A = -20 to +85°C)

RL78 DATASHEET중 일부 발췌.

위와 같이 MCU내부에서 자체적으로 CLOCK을 만들어 낼 수 있고 (외부 크리스탈에비해 오차가 크다)

또는 CRYSTAL과 CAP을 달아주어 CLOCK 을 만들어낸다.

또 다른 방법으로는 일체형 OSCILATOR (전압인가하면 발진함) 를 사용하여 발생하는 CLOCK을 인가하는 방법이 있다.

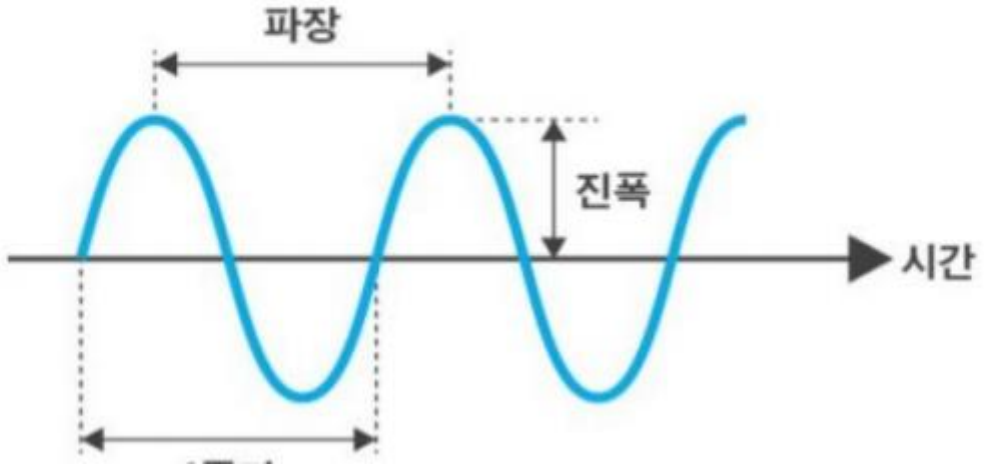


3. TIMER/COUNTER

4. 8bit? 16bit? 어떤 의미인가?
- Atemga328p의경우 8bit 16bit timer/counter가 존재함.
- 8bit의 경우 최대갑이 255, 16bit의 경우 65535이므로 딱 봐도 16bit가 더 정밀하게 TIMER COUNTER를 사용 할 수 있음을 알 수 있다.

→ 더 자세한 내용은 뒤에서 데이터 시트를 분석하며 알 수 있다.

5. Prescaler란 무엇인가?
- 내부 혹은 외부의 CLOCK을 USER가 마음대로 자르고 붙이고 할 수 있다.
- 즉 16Mhz의 경우 6.25us(1/16Mhz)주기를 갖는다.
- 이때 6.25us마다 어떤 이벤트 혹은 인터럽트가 발생한다하면 사용자는 이를 활용 할 수 있다.
- 하지만 위에서 봤듯이 8bit의 경우 최대값이 255 이다.
- 이때 255*6.25us = 1593us 즉 1.53ms에 불과하다.
- 이는 짧아서 활용하기 힘들다. 이 6.25us를 더 길게 해줄 수 있는게 분주비 즉, **PRESCALER**이다
- MCU에는 이런 PRESALCING할 수 있는 회로가 있고 이를 사용자는 레지스터를 통해 제어 할 수 있는것이다.



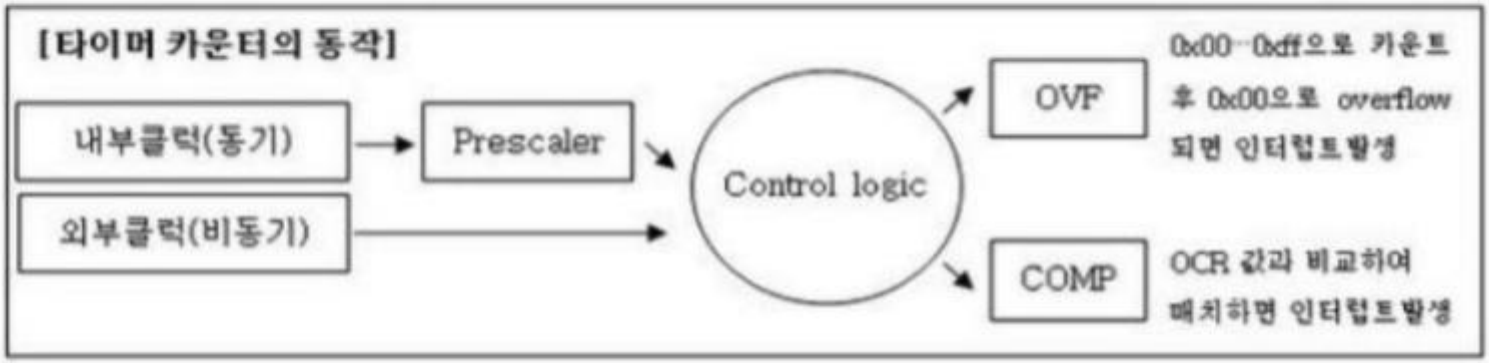
3. TIMER/COUNTER

6. 인터럽트 발생 조건

- 위에서 클록을 사용하여 어떤 이벤트 혹은 인터럽트 발생 조건을 만든다고 하였다.
  두 가지 인터럽트 발생 조건에 대해서 알아보자.

1) OVERFLOW : 특정 레지스터(여기선 TCNT)가 꽉 차면 (8bit 기준 255) 인터럽트 발생

2) Output compare match : 특정 레지스터(여기선 TCNT) 와 사용자 설정값이 같으면 인터럽트 발



7. 세 가지 TIMER/COUNTER 사용 방법

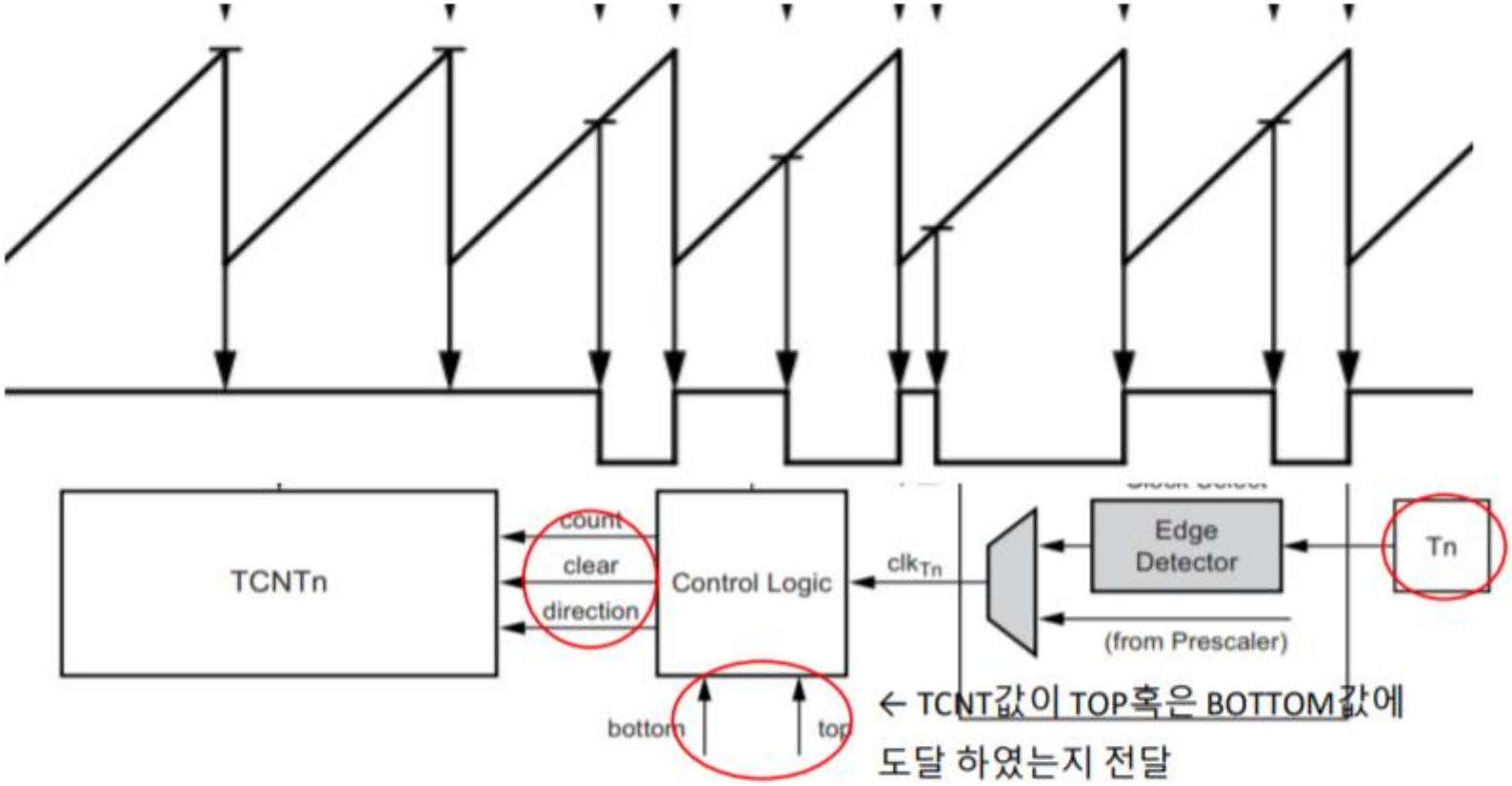
1) NORMARL : 일반적인 OVERFLOW로 발생하는 인터럽트를 사용하는 모드

2) CTC : Output Compare match, OVERFLOW가 아닌 특정값에 인터럽트 발생하는 모드

3) FAST PWM :

4) PC PWM :

3. TIMER/COUNTER



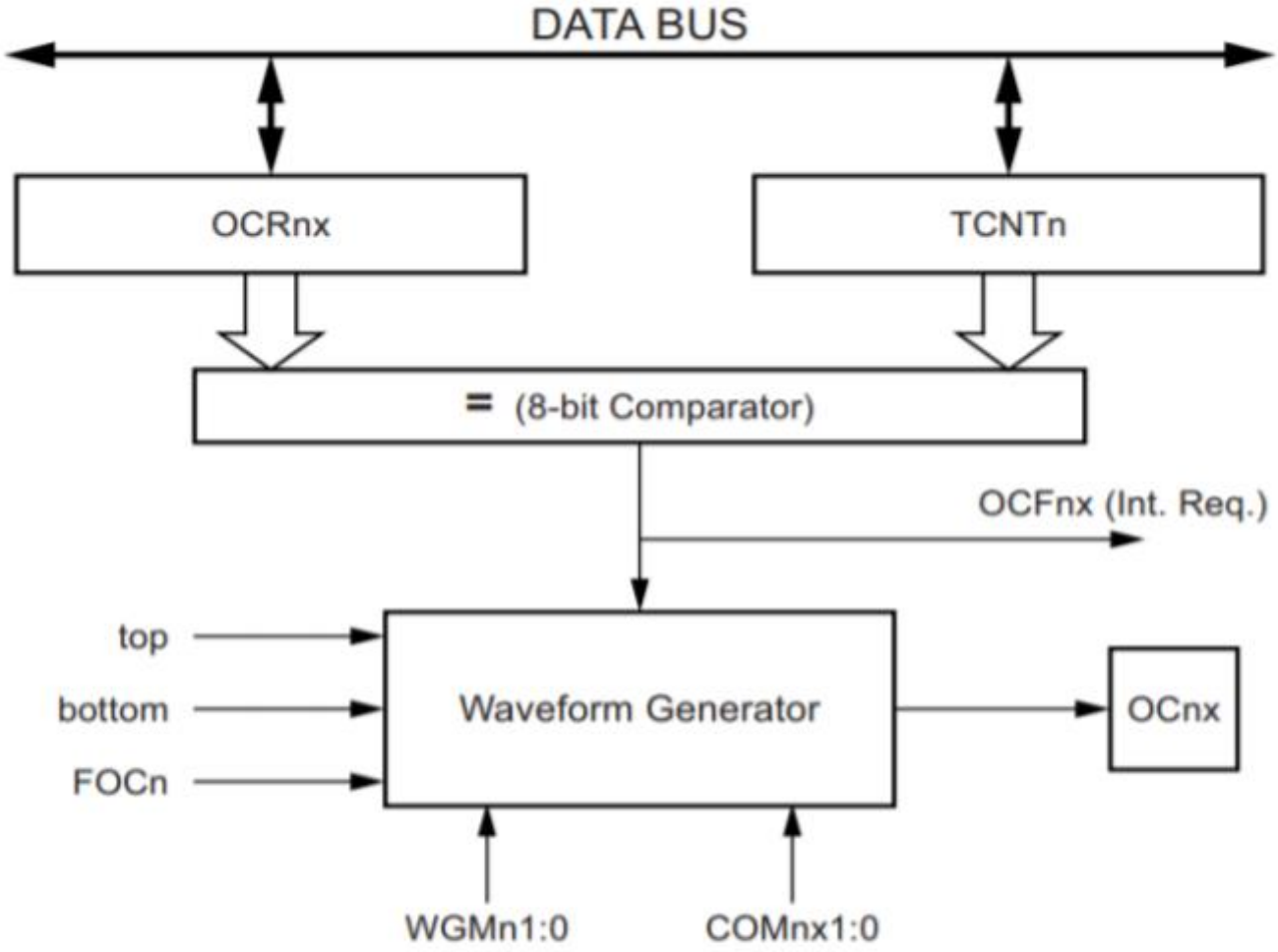
① 외부클럭(Tn) 혹은 Prescaler를 거친 내부 clock중 하나만 선택(MUX를 거치기 때문) 되어 Control Logic에 전달되고

② CONTROLLOGIC은 설정에 따라서(DIRECTION) count 전달

③ TCNT는 DATABUS를 통해 접근할 수 있으며 Overflow발생시 TOVn플래그 SET

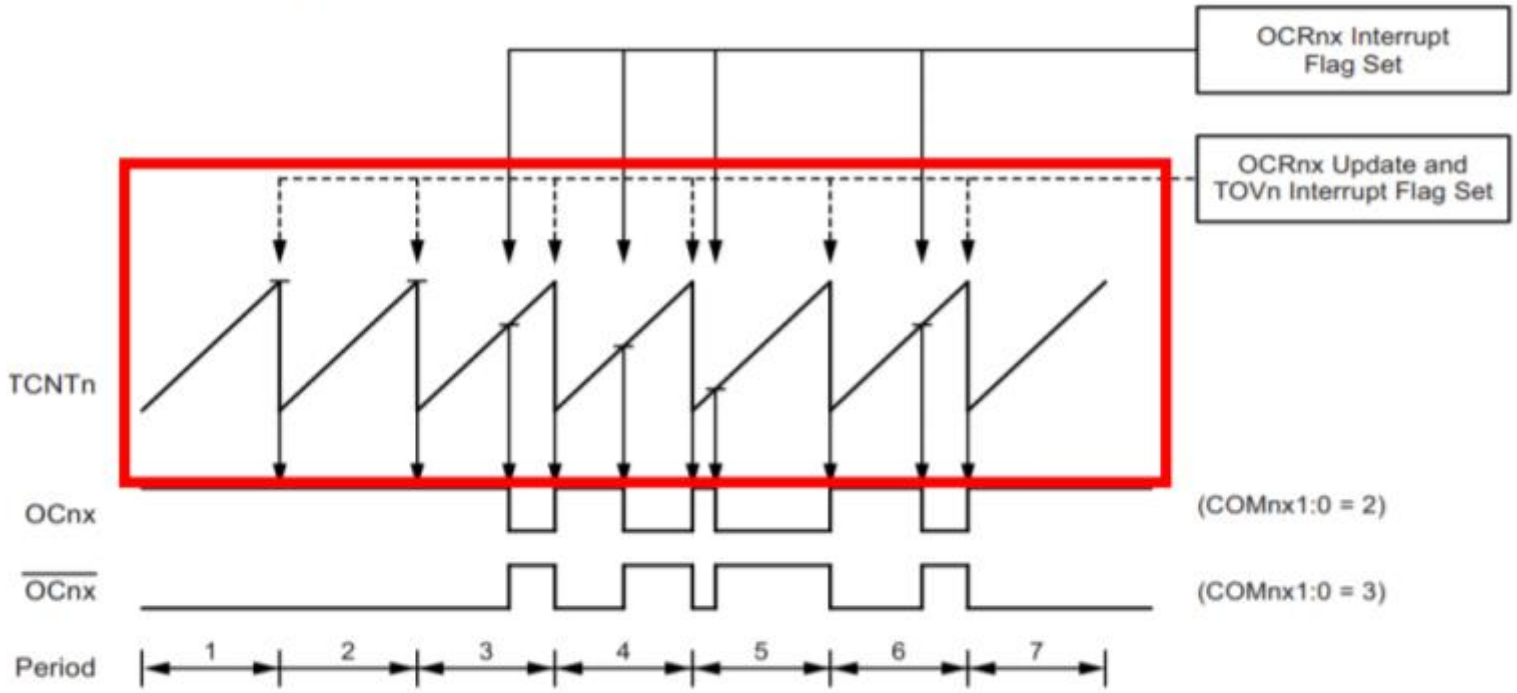
3. TIMER/COUNTER

10. CTC모드 블록 분석



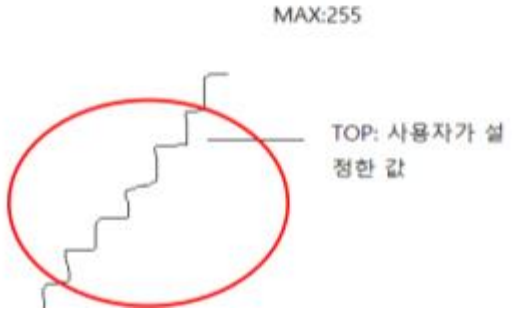
① OCRnx레지스터와 TCNTn레지스터 값이 같으면 OCFnx 레지스터 Flag set → 인터럽트 발생
② Ocnx레지스터가 추후 PWM파형 값을 변경하는데 쓰임

11. FAST PWM 파형 발생 원리



① OCRnx레지스터는 설정값에 따라 CTC interrupt 발생시 토글되는데 이때 이 원리를 이용하여 PWM파형을 발생시키는 것을 볼 수 있다.

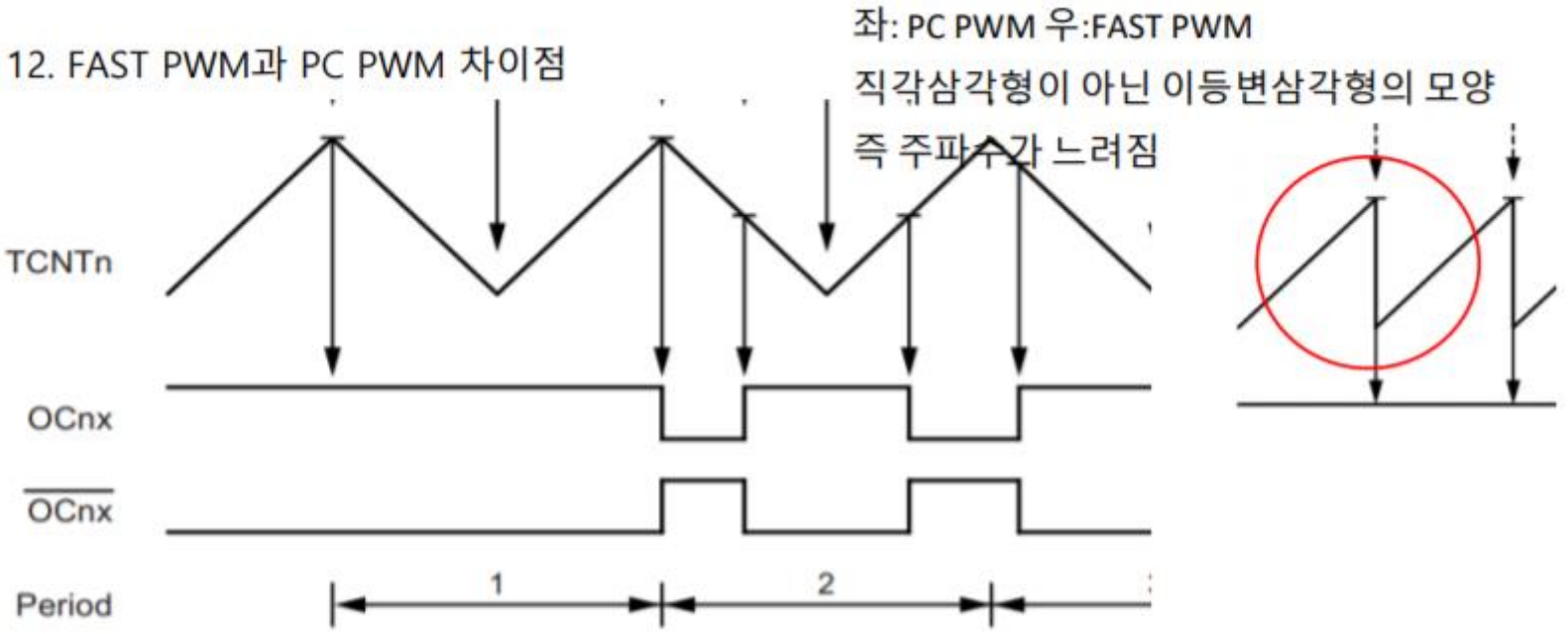
① 또한 이 삼각파의 경우도 자세하게 확대하여 볼 경우



← 옆과 같은 계단식 모양으로 TCNT값(일정 CLOCK 마다 증가하는 값)이 쌓여나가는 과정이라 볼 수 있다.

3. TIMER/COUNTER

12. FAST PWM과 PC PWM 차이점



① FAST PWM에 비하여 파형 모양봤을때 주파수가 1/2배가 됨을 알 수 있다.
① 하지만 DUTY비의 분해능은 2배이기 때문에 오히려 더 정밀한 PWM제어가 가능하다.

3. TIMER/COUNTER

13. Glithch? 이중버퍼?
- TOP 값을 결정하는 레지스터를 TIMER/COUNT 실행 중 변경하면 어떤일이 발생할까?

CASE01) TOP : 100 → 150
변경되는 시점의 카운트 레지스터가 0이던 10이던 50이던 150까지 도달후 0으로 초기화 되기때문에 문제없음.

CASE02) TOP : 100 → 50
변경되는 시점에 카운트 레지스터 0일 경우 → 문제없음
변경되는 시점에 카운트 레지스터 70일 경우 → **문제발생**
이미 50을 지나친 상태이기 때문에 MAX값까지 도달하는 과도기 이후 정상작동하게 됨.

- 위와 같은 현상을 TIMER/COUNTER에서 Glitch라 한다.
- 해결방법 : 버퍼를 사용해 잠시 버퍼에 변경사항을 저장해 두었다가
  카운트 레지스터 BOTTOM값에 도달시 변경
- ATMEGA328P의 경우 PWM모드에는 버퍼 적용 돼 있으나 NORMAL 및 CTC의 경우 적용 돼 있지 않으므로 사용시 주의해야함.