AVR – HW7

임베디드스쿨1기

Lv1과정

2020. 10. 30

강경수

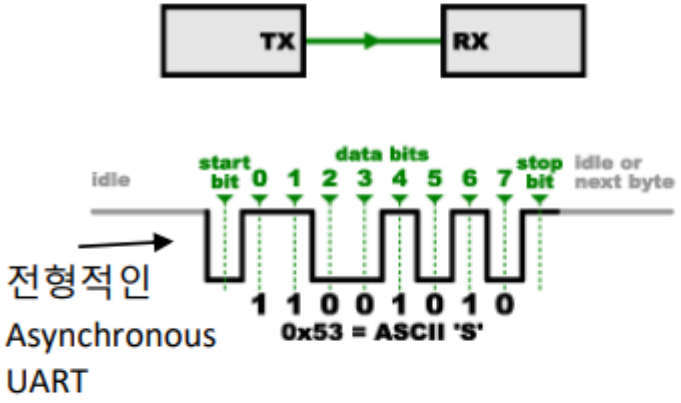
1. SPI 통신

▣ SPI통신

2020.10.27 강경수

1. SPI 통신이란?
- 일반적으로 마이크로컨트롤러가 데이터를 전송할때에 사용 됨
  (SD 카드, 센서, 시프트레지스터 등)

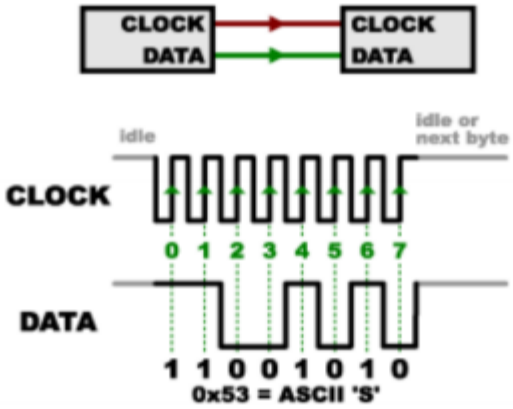
2. Asynchronous
  1) 데이터 송수신시 양쪽이 정확히 동일한 클럭으로 동작되고 있다는 보장이 없다.
  2) 이런 경우 각각의 시스템클럭이 약간의 오차가 존재할 시 제대로된 DATA 송신 및 수신이 불가능 하다
  3) 이에 대한 해결방법으로 START BIT 그리고 STOP BIT가 설정되어야 하며 또한 사전에 송신 및 수신속도가 합의되어야 한다 (19200bps 등)
  4) 추가적인 시작Bit, 정지Bit 로 인한 시간대비 DATALOSS가 발생한다



1. SPI 통신

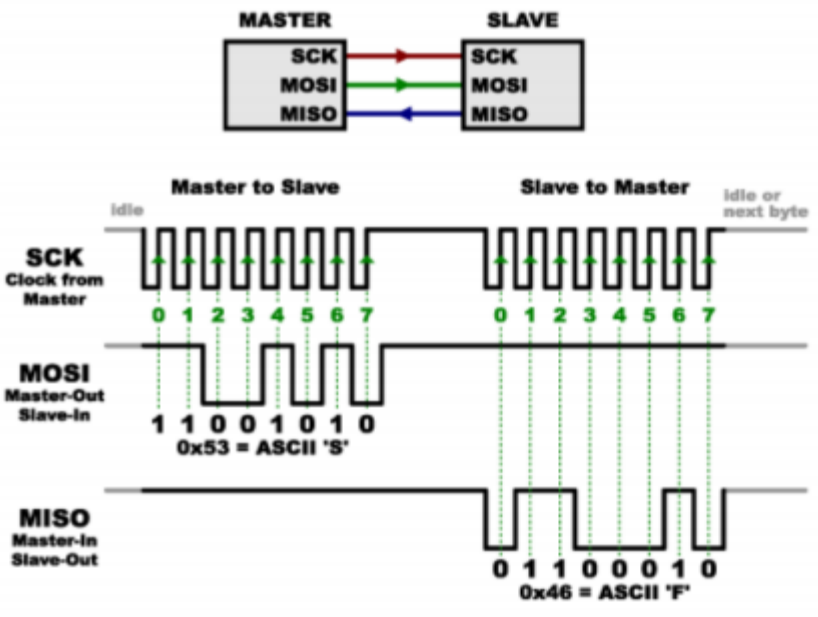
3. Synchronous

1) DATA 라인과 CLOCK LINE을 분리하여 CLOCK에 동기화 하여 DATA가 전송될 수 있게 한다.
2) CLOCK이 언제 정확히 데이터 비트를 샘플링 할지 알려준다
3) 별도의 CLOCK라인을 사용하므로 사전의 통신속도 협의가 필요하지 않다
4) CLOCK의 RISING시 혹은 FALLING시 DATA를 읽어드릴지 레지스터를 통해 제어한다.



4. MASTER AND SLAVE

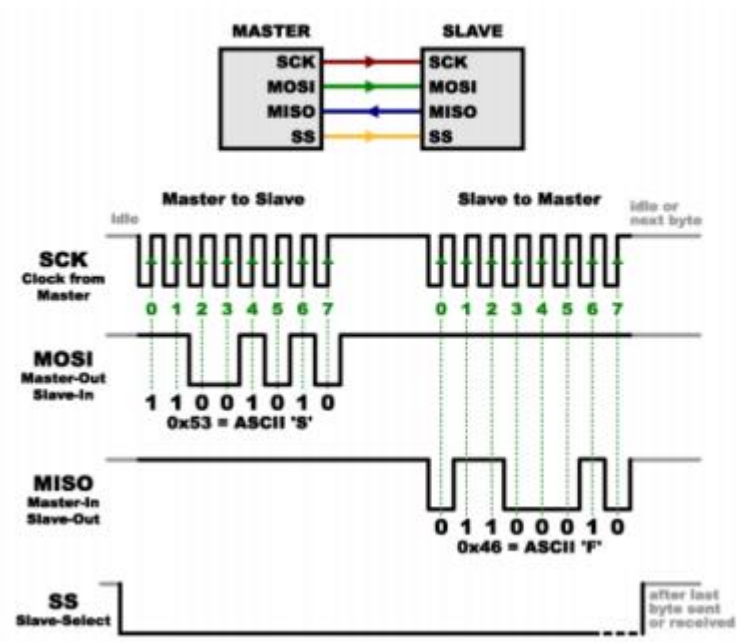
1) MATER란 CLOCK을 만들어내는 시스템
2) MOSI(MASTER OUT SLAVE IN) MASTER가 SLAVE에게 DATA를 전송
3) MISO를 통해 SLAVE는 MASTER의 DATA에 응답함.
4) MASTER가 CLOCK을 생성하기 때문에 MASTER는 SLAVE의 응답이 **언제 얼마나** DATA를 반환하는지 알고 있어야 한다.
5) **따라서 SLAVE의 DATA가 필요할 경우 반드시 MASTER가 먼저 DATA를 요청하는 명령을 보내야 한다**



1. SPI 통신

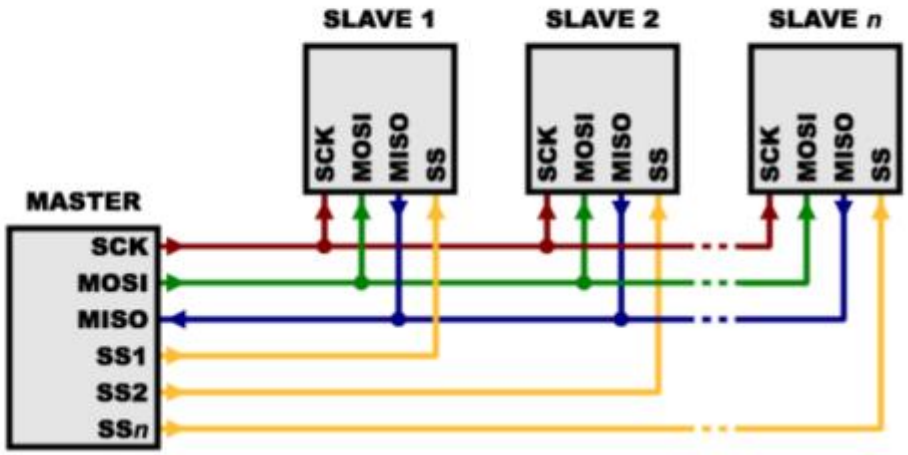
5. S/S 란 무엇인가?

1)SLAVE SELECT의 약자로 ACTIVE LOW 로 동작하게 된다. MASTER에서 SLAVE로 DATA를 전송하기 전에 S/S를 LOW로 하여 SLAVE를 활성화 하며 SLAVE모두 사용 후 S/S는 다시 HIGH가 된다

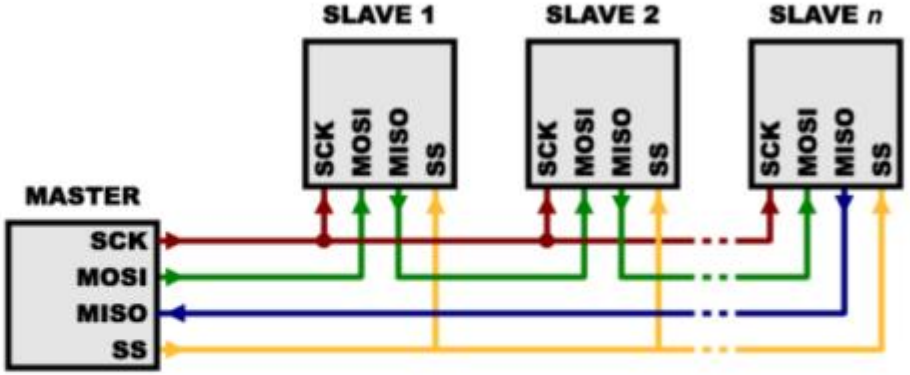


1. SPI 통신

6. Multiple slaves 예시
 - SS핀을 여러 개 두어 각각의 SLAVE들을 개별 제어 한다.



7. Daisy Chain 방식 예시
 - Daisy chain 예시 다음장 참고

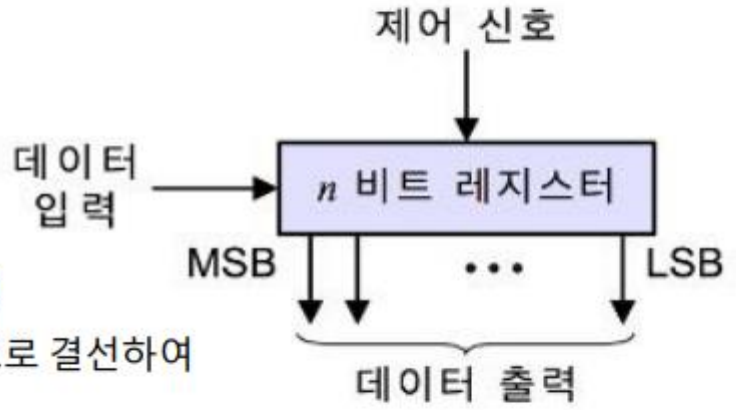


1. SPI 통신

▣ Shift Register

1. SHIFT REGISTER란 무엇을 이야기 하는가?

-. 아래와 같은 0,1을 저장할 수 있는 플립플롭을 직렬로 연결하여, 여러 비트로 구성한 2진수를 저장할 수 있게 한 것.



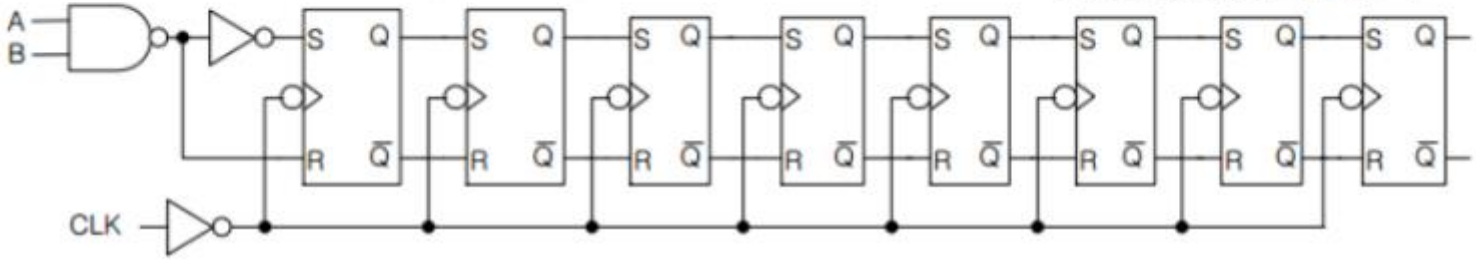
SPI통신시 위와같이

DAISY CHAIN 방식으로 결선하여

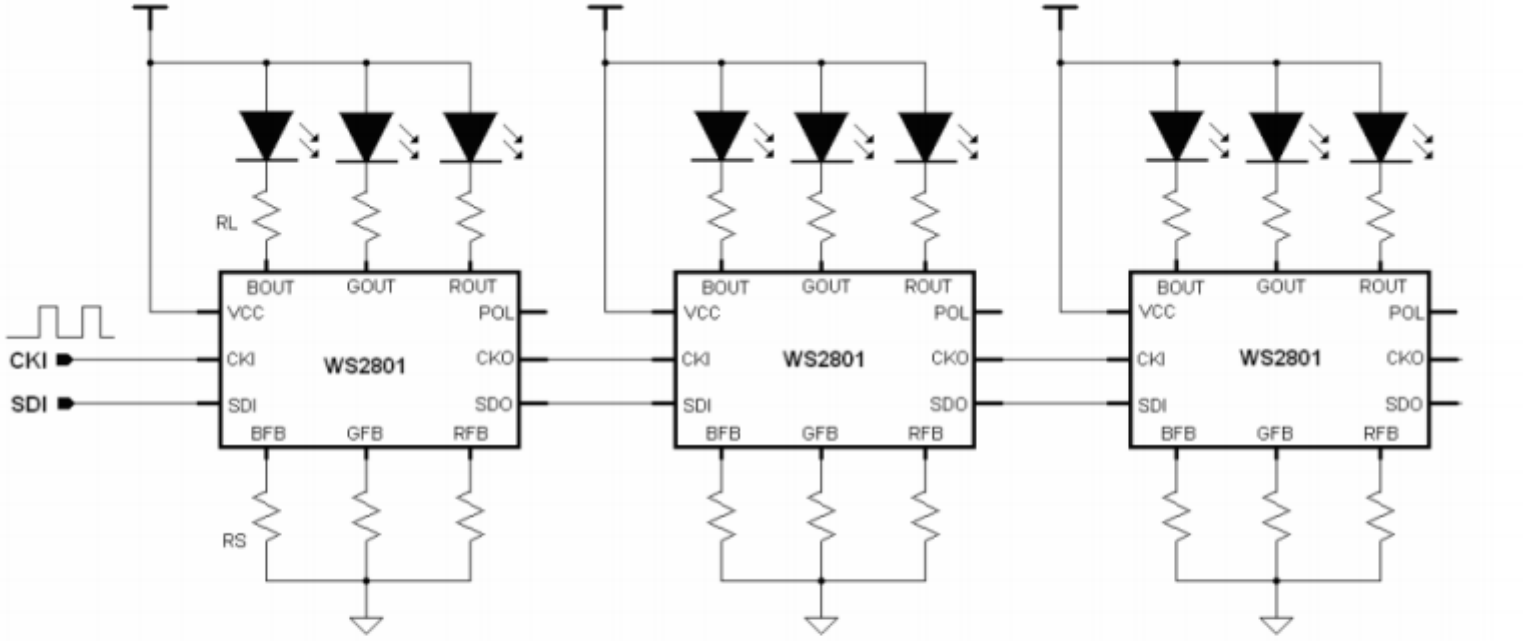
사용 할 수 있다.

대표적인 사용 IC) 74HC95 , WS2801 (LED Drvier),WS2812(LED Driver and LED)

DAISY CHAIN 결선 방식



1. SPI 통신



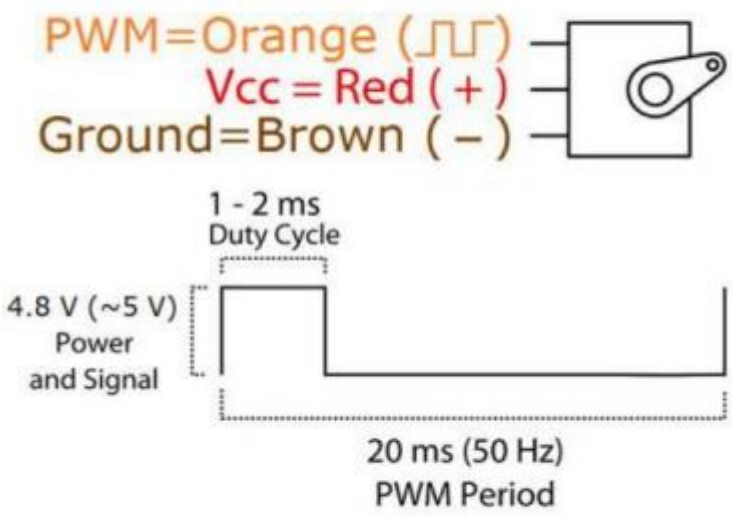
위와같이 한개의 DATA LINE을 가지고 많은
CHIP을 직렬로 컨트롤 할 수 있다.

2. SERVO

▣ SERVO MOTOR 제어

2020.10.27 강경수

1. SERVO MOTOR SPEC



Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

- 모델명 : SG90
- 주파수 : 50Hz (모든 RC 주파수)
- 전압 : 5V
- DUTY : 1~2ms
- 특징 : 서보모터는 지속적으로 펄스 파형을 가하여야 함.
- 서보모터 위치

위치	DUTY
FULL LEFT	1ms
MIDDLE	1.5ms
FULL RIGHT	2ms

2. SERVO

2. PWM 파형 만들기

```
void 16bit_timer_init(void)
{
        sbi(TCCR1A,WGM11);
        sbi(TCCR1B,WGM12); //16BIT FAST PWM TOP VAL : ICR1
        sbi(TCCR1B,WGM13);
        sbi(TCCR1B,CS11); //PRESCALE : 8 2MHZ

        ICR1 = 40000; //20ms주기 0.5us * 40,000 = 0.2ms

        TCCR1A |= (1<<COM1A1); //비반전 모드

        DDRB |= (1<<PORTB1); // OC1A OUTPUT MODE

        OCR1A = 1000; //
}
```

14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP

Table 15-3. Compare Output Mode, Fast PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See Section 15.9.3 "Fast PWM Mode" on page 101 for more details.

0	1	0	$clk_{I/O}$/8 (from prescaler)

2. SERVO

3. RX 인터럽트 루틴을 사용하여 서보모터 제어하기

```
void UART_INIT(void){
    sbi(UCSR0A, U2X0); //U2X0 = 1 --> Baudrate 9600 = 207

    UBRR0H = 0x00;
    UBRR0L = 207; //Bauddrate 9600

    UCSR0C |= 0x06; //1stop bit, 8bit data

    sbi(UCSR0B, RXEN0);  //enable receiver and transmitter
    sbi(UCSR0B, TXEN0);
}
ISR(USART_RX_vect)
{
    rx_buf[rx_cnt] = UDR0;

    if((rx_buf[rx_cnt]=='\r')||(rx_buf[rx_cnt]=='\n'))
    {
        rx_buf[rx_cnt+1] = '\0'; //atoi를 사용하기 위함.
        rx_flag = 1;
        rx_cnt = 0;
    }
    rx_cnt++;
}
```

2. SERVO

```
void rx_str_check(void)
{
    if(rx_flag==1)
    {
        Rx_to_Servo();
    }
}
void Rx_to_Servo(void)
{
    int temp;

    temp = atoi(rx_buf); //'\0'까지 확인후 ascii to intiger

    if((PULSE_MAX>=temp)&&(temp>=PULSE_MIN))
    {
        OCR1A = temp;
    }
}
```

원래 정석대로 라면

OCR1AH, OCR1AL 8bit씩

나누어 값을 할당하지만

avr/io.h에서 이와같이 한번에

word로 쓸 수 있게 지원하는것 같다.

3. 16BIT TIMER

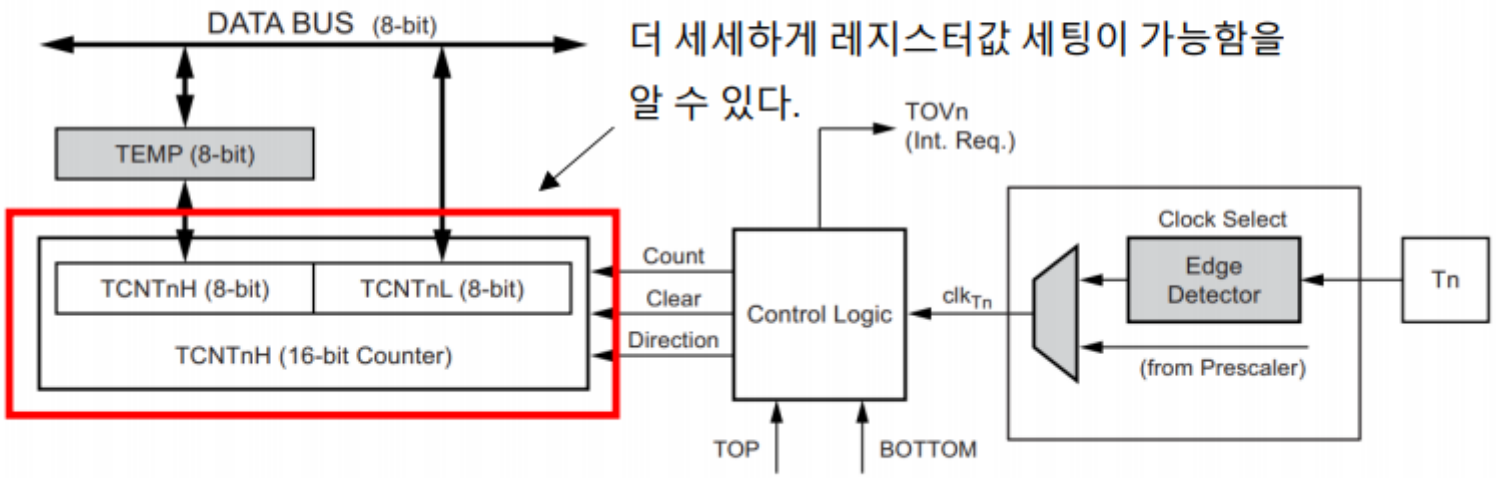
▣ TIMER/COUNTER

2020.10.27 강경수

1. 16BIT TIMER/COUNTER

Figure 15-2. Counter Unit Block Diagram

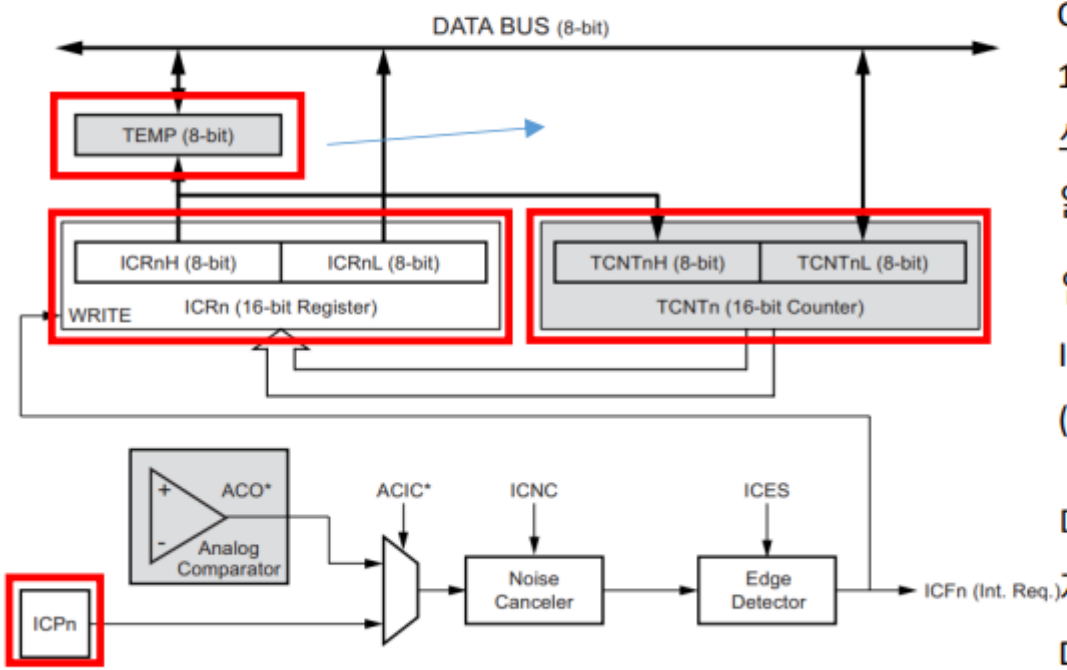
8BIT TIMER/COUNTER에 비하여
더 세세하게 레지스터값 세팅이 가능함을
알 수 있다.



3. 16BIT TIMER

2. Input Captuer Unit

- TIMER/COUNTER는 크게 세가지 역할을 한다고 볼 수 있음
  - (1) 일정 시간 간격으로 주기적인 이벤트를 발생하는 것이다. **(Periodic Timer)**
  - (2) 일정 시간 간격, 혹은 시간 경과 후에 외부로 신호를 발생할 수 있다. **(Output Compare)**
  - (3) 외부 신호의 발생 시각을 알 수 있는 용도로 활용할 수도 있다. **(Input Capture)**



OCR1A,OCR1B를 제외한
16BIT REGISTER의 경우
쓰기 : HIGH -> LOW
읽기 : LOW -> HIGH

입력된 순간의 TCNTn값이
ICRn레지스터에 저장된다
(이때 인터럽트 발생 가능)

따라서, ICRn값을 임시변수에
저장해서 비교하면 쉽게
Duty 및 Pulse 계산 가능할듯?

3. 16BIT TIMER

3. Capture 를 통한 duty 및 pusle계산

```
void input_capture_timer_init(void)
{
    sbi(TCCR1A,WGM11);
    sbi(TCCR1B,WGM12); //16BIT FAST PWM TOP VAL : ICR1
    sbi(TCCR1B,WGM13);
    sbi(TCCR1B,CS11); //PRESCALE : 8 2MHZ

    sbi(TCCR1B,ICNC1) //NOISE CANCEL
    sbi(TCCR1B,ICES1); //rising 때 인터럽트 걸리게 setting

    sbi(TIFR1,ICF1); //input caputer interrupt flag, 인터럽트 실행시 자동으로 0.
    sbi(TIMSK1,ICIE1); //input capture interrupt enable

    ICR1 = 40000; //20ms주기 0.5us * 40,000 = 0.2ms
    TCCR1A |= (1<<COM1A1); //비반전 모드
    DDRB |= (1<<PORTB1); // OC1A OUTPUT MODE
    OCR1A = 1000; //
}
```

3. 16BIT TIMER

```
ISR(TIMER1_CAPT_vect)
{
    if(capture_cnt==0)
    {
        duty_start = ICR1;
        cbi(TCCR1B,ICES1); //falling 때 인터럽트 걸리게 setting
        capture_cnt++;
    }
    else if(capture_cnt==1)
    {
        duty_last = ICR1;
        capture_cnt++;
    }
    else(capture_cnt==2)
    {
        pulse_last = ICR1;
        capture_cnt = 0
        capture_end = 1;
    }
}
void cal_pwm(void)
{
    int duty;
    int pulse;

    duty = duty_start-duty_last;
    pulse = pulse_last;
}
```

TIMER OVERFLOW 발생할 경우
DUTY, PULSE 계산 함수 재작성 필요.

4. 8BIT TIMER COUNTER REVIEW

▣ TIMER/COUNTER REIVEW

2020.10.27 강경수

1. CTC TIMER 복습

```
SIGNAL(TIMER0_COMPA_vect)
{
    counter++;
}

void OCM_timer_Init(void);

int main(void)
{
    /* Replace with your application code */
    OCM_timer_Init();

    while (1)
    {
        if(counter==1000)
        {
            PORTB = (~PORTB);
            counter = 0;
        }
    }
```

1ms마다 인터럽트 발생
즉 1*1000 1초마다 LED
점멸하는것을 볼 수 있다.

4. 8BIT TIMER COUNTER REVIEW

```
void OCM_timer_Init(void)
{
    cbi(SREG ,7); // I 레지스터 전역인터럽트 허용
    TCCR0A = 0;  //사용 하지 않음.
    TCCR0B = 0;
    PORTB = 0x00;
    DDRB  = 0xff;

    TCCR0A = (1<<WGM01); //PWM 출력 안함

    TCCR0B = (1<<CS01)|(1<<CS00); //clk/64 16Mhz / 64 => 0.004ms
    TIMSK0 = (1<<OCIE0A); //OCIE2A 레지스터 활성화 OCR2A==TCNT시 인터럽트
    TCNT0 = 0; //초기값 0
    OCR0A = 249; //0.004 X 250 = 1 즉 1ms초마다 인터럽트 루틴 발생
    sbi(SREG,7);
};
```

4. 8BIT TIMER COUNTER REVIEW

```
void OCM_timer_Init(void)
{
cbi(SREG ,7); // I레지스터 허용

TCCR0A = 0;  //사용 하지 않음.
TCCR0B = 0;

PORTB = 0x00;
DDRB  = 0xff;

TCCR0A = (1<<WGM01)|(1<<COM0A0); //OC0A TOGGLE

TCCR0B = (1<<CS02)|(1<<CS00); //clk/1024 16Mhz / 64 => 0.000064ms

TIMSK0 = (1<<OCIE0A); // OCR0A와 TCNT0값이 같을시 인터럽트 발생

TCNT0 = 0;
DDRD = 0xff;
OCR0A = 249;
sbi(SREG,7);
};
```

4. 8BIT TIMER COUNTER REVIEW

2. CTC PWM OUTPUT

```
volatile unsigned int counter = 0;

SIGNAL(TIMER0_COMPA_vect)
{
}

void OCM_timer_Init(void);

int main(void)
{
/* Replace with your application code */
OCM_timer_Init();

    while (1)
    {
    }
}
```

4. 8BIT TIMER COUNTER REVIEW

```
void OCM_timer_Init(void)
{
cbi(SREG ,7); // I레지스터 허용

TCCR0A = 0;  //사용 하지 않음.
TCCR0B = 0;

PORTB = 0x00;
DDRB  = 0xff;

TCCR0A = (1<<WGM01)|(1<<COM0A0); //OC0A TOGGLE PWM출력, CTC MODE

TCCR0B = (1<<CS02)|(1<<CS00); //clk/1024 16Mhz / 1024 => 0.000064ms

TIMSK0 = (1<<OCIE0A); // OCR0A와 TCNT0값이 같을시 인터럽트 발생

TCNT0 = 0;
DDRD = 0xff;
OCR0A = 249; //0.016초마다 인터럽트 발생, 파형 TOGGLE
sbi(SREG,7);
};
```

CTC MODE DUTY 는 무조건 50%

4. 8BIT TIMER COUNTER REVIEW

3. FAST PWM OUTPUT

```
int main(void)
{
    /* Replace with your application
    Fast_PWM_Mode();
    while (1)
    {
    }
}
SIGNAL(TIMER0_COMPA_vect)
 {
    if(LED_FLAG == 0)
    {
        OCR0A += 1;
    }

    if(OCR0A==255)
    {
        LED_FLAG =1;
    }
```

LED가 밝아졌다가
어두워졌다가 반

```
void Fast_PWM_Mode(void)
{
    cbi(SREG,7);
    sbi(TCCR0A,WGM01); //FAST PWM
    sbi(TCCR0A,WGM00); //FAST PWM
    sbi(TCCR0A,COM0A1); // CLEAR ON COMPARE MATCH
    sbi(TCCR0B,CS02); //PRESCALE 1024
    sbi(TCCR0B,CS00); //PRESCALE 1024
    sbi(TIMSK0,OCIE0A); //OCIE0A INTERRUPT활성화
    DDRD = 0xff; //DDRD 만 출력으로 돼 있으면 된다.
    PORTD = 0x00; //PORTD는 출력이던 뭐던 상관 없다.
    OCR0A = 0;
    sbi(SREG,7);
}
```

```
    if(LED_FLAG == 1)
    {
        OCR0A -= 1;
    }

    if(OCR0A==0)
    {
        LED_FLAG =0;
    }
```

4. 8BIT TIMER COUNTER REVIEW

4. PHASE CORRECT PWM OUTPUT

```
void Phase_Correct_PWM_Init(void);

uint8_t LED_FLAG = 0;

SIGNAL(TIMER0_COMPA_vect)
{
    OCR0A += 1;

    if(OCR0A == 255)
    {
        OCR0A = 0;
    }
}

int main(void)
{
    /* Replace with your application code */
    Phase_Correct_PWM_Init();
    while (1)
    {
    }
}
```

4. 8BIT TIMER COUNTER REVIEW

```
void Phase_Correct_PWM_Init(void)
{
    cbi(SREG,7);
    sbi(TCCR0A,WGM02);
    sbi(TCCR0A,WGM00); //PWM PHASE CORRECT OCRA TOP VALUE
    sbi(TCCR0A,COM0A1);// UP COUNT COMPARE시 CLEAR DOWN COUNT COPMAR시 S
    sbi(TCCR0B,CS02);
    sbi(TCCR0B,CS00); //PRESCLAE 1024
    sbi(TIMSK0,OCIE0A); //TIMER COUNTER INTERRUPT ENABLE
    DDRD = 0xff; //DDRD 만 출력으로 돼 있으면 된다.
    PORTD = 0x00; //PORTD는 출력이던 뭐던 상관 없다.
    OCR0A = 0;
    sbi(SREG,7);
}
```