
성 균 관 대 학 교
SIOR
로 봇 학 회

2022년 05월 15일

EMBEDDED

5 주 차

목차

EMBEDDED

타이머/카운터

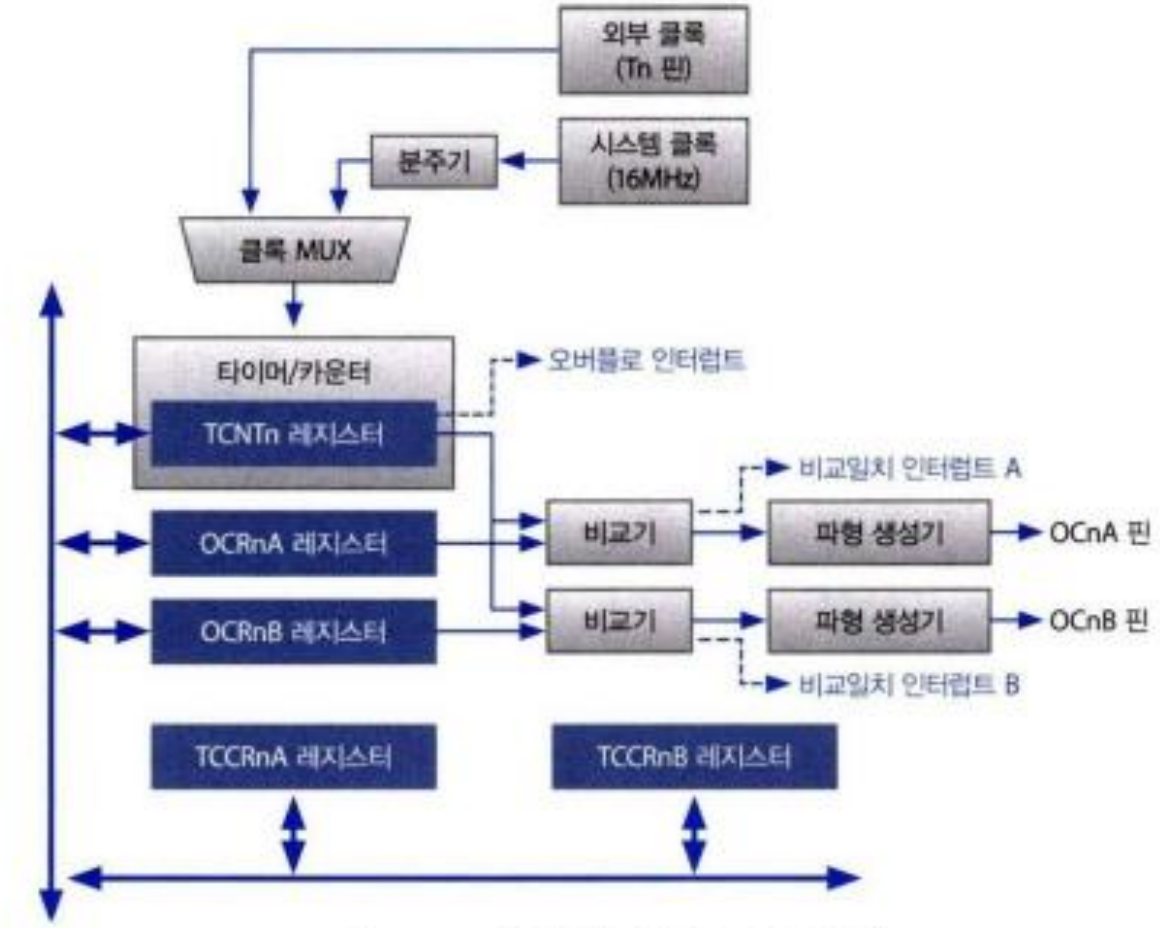

그림 14-1 타이머/카운터 블록 다이어그램

오버플로 인터럽트

코드 14-1 오버플로 인터럽트를 이용한 Blink

```
#include <avr/io.h>
#include <avr/interrupt.h>

int count = 0;                          // 오버플로가 발생한 횟수
int state = 0;                          // LED 점멸 상태

ISR(TIMER0_OVF_vect)
{
    count++;
    if(count == 32){                    // 오버플로 32회 발생 = 0.5초 경과
        count = 0;   // 카운터 초기화
        state = !state;                 // LED 상태 반전
        if(state) PORTB = 0xFF;         // LED 켜기
        else PORTB = 0x00;              // LED 끄기
    }
}

int main(void)
{
    DDRB = 0x20;                        // PB5 핀을 출력으로 설정
    PORTB = 0x00;                       // LED는 끈 상태에서 시작

    TCCR0B |= (1 << CS02) | (1 << CS00);  // 분주비를 1024로 설정

    TIMSK0 |= (1 << TOIE0);             // 오버플로 인터럽트 허용
    sei();                              // 전역적으로 인터럽트 허용

    while(1){ }
}
```

비트	7	6	5	4	3	2	1	0
	TCNT0[7:0]							
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-2 TCNT0 레지스터의 구조

TCNT: 현재까지 센 펄스의 수 저장 -> 오버플로 감지

비트	7	6	5	4	3	2	1	0
	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
읽기/쓰기	W	W	R	R	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-3 TCCR0B 레지스터의 구조

TCCRnx (TCCR0B): 분주비 설정 & 카운트 시작

비트	7	6	5	4	3	2	1	0
	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
읽기/쓰기	R	R	R	R	R	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-4 TIMSK0 레지스터의 구조

TIMSKn (TIMSK0): 0번 인터럽트 활성화

EMBEDDED

오버플로 인터럽트

비트	7	6	5	4	3	2	1	0
	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
읽기/쓰기	W	W	R	R	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-3 TCCR0B 레지스터의 구조

TCCRnx (TCCR0B): 분주비 설정 & 카운트 시작

표 14-1 CS0n(n = 0, 1, 2) 비트 설정에 따른 클록 선택

CS02	CS01	CS00	설명
0	0	0	클록 소스 없음(타이머/카운터 정지)
0	0	1	분주비 1
0	1	0	분주비 8
0	1	1	분주비 64
1	0	0	분주비 256
1	0	1	분주비 1024
1	1	0	T0 핀의 외부 클록 사용. 하강 에지에서 동작한다.
1	1	1	T0 핀의 외부 클록 사용. 상승 에지에서 동작한다.

분주비가 1024가 되면 $\frac{16MHz}{1024}$ = 16KHz 클록

펄스 256개를 세는 시간은 $\frac{256}{16K} = \frac{1}{64}$초

1/64 * 32 = 0.5s

오버플로 인터럽트

비트	7	6	5	4	3	2	1	0
	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
읽기/쓰기	R	R	R	R	R	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-4 TIMSK0 레지스터의 구조

TIMSKn (TIMSK0): 0번 인터럽트 활성화

표 14-2 타이머/카운터 0번 인터럽트

벡터 번호	인터럽트	벡터 이름	인터럽트 허용 비트	
15	비교일치 인터럽트 A	TIMER0_COMPA_vect	OCIE0A	Output Compare Match A Interrupt Enable
16	비교일치 인터럽트 B	TIMER0_COMPB_vect	OCIE0B	Output Compare Match B Interrupt Enable
17	오버플로 인터럽트	TIMER0_OVF_vect	TOIE0	Overflow Interrupt Enable

오버플로 인터럽트

1KHz는 1000Hz인가, 1024Hz인가?

정확하게 이야기하면 1KHz는 1000Hz이다. 16MHz 클록은 1초에 16 × 2^{20}개의 펄스가 발생하는 것이 아니라 16 × 10^{6}개의 펄스가 발생하는 것이므로 분주비를 1024로 설정하면 클록 주파수는 16KHz가 아니라 15.625KHz가 된다. 정확하게 계산하면 분주비 1024에서 0번 타이머/카운터가 1초 동안 발생시키는 오버플로 인터럽트는 약 61.04회이며, 오버플로 인터럽트가 64회 발생하는 시간은 1초가 아니라 약 1.05초가 된다. 따라서 정밀한 시간 계산이 필요하다면 이 장의 예제들을 그대로 사용하여서는 안 된다. 이 장에서는 계산의 편의를 위해 2^{10}과 10^{3}을 흔히 동일한 값으로 취급하는 관례를 따랐다.

정밀한 시간 계산이 필요한 경우 염두에 두어야 할 또 다른 점은 클록 공급을 위해 아두이노 우노에서 사용하는 크리스털의 정밀도이다. 아두이노 우노에도 사용되고 있는 16MHz 크리스털의 정밀도는 표준편차가 0.7Hz 정도인 것으로 알려져 있다. 하지만 ATmega328의 동작 온도가 1도 상승할 때마다 크리스털의 클록 주파수는 약 0.97Hz 증가하고, 동작 전압이 1mV 증가할 때마다 크리스털의 클록 주파수는 약 0.03Hz 증가하는 등 동작 환경에 따라 클록 주파수는 가변적이다. 따라서 정밀한 시간 계산이 필요하다면 전용의 하드웨어 RTC(Real Time Clock)나 보상 회로가 추가되어 있는 클록을 사용하는 것이 바람직하다.

비트	7	6	5	4	3	2	1	0
	OCR0[7:0]							
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-5 OCR0x(x = A, B) 레지스터의 구조

EMBEDDED

비교일치 인터럽트

코드 14-2 비교일치 인터럽트를 이용한 Blink 1

```
#include <avr/io.h>
#include <avr/interrupt.h>

int count = 0;                          // 비교일치가 발생한 횟수
int state = 0;                          // LED 점멸 상태

ISR(TIMER0_COMPA_vect)
{
    count++;
    TCNT0 = 0;                          // 자동으로 0으로 변하지 않는다.
    if(count == 64){                    // 비교일치 64회 발생 = 0.5초 경과
        count = 0;                      // 카운터 초기화
        state = !state;                 // LED 상태 반전
        if(state) PORTB = 0xFF;         // LED 켜기
        else PORTB = 0x00;              // LED 끄기
    }
}

int main(void)
 {
     DDRB = 0x20;                       // PB5 핀을 출력으로 설정
     PORTB = 0x00;                      // LED는 끈 상태에서 시작

     TCCR0B |= (1 << CS02) | (1 << CS00);  // 분주비를 1024로 설정

     OCR0A = 128;                       // 비교일치 기준값

     TIMSK0 |= (1 << OCIE0A);           // 비교일치 인터럽트 허용
     sei();                             // 전역적으로 인터럽트 허용

     while(1){ }
 }
```

코드 14-3 비교일치 인터럽트를 이용한 Blink 2

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile int count = 0;                 // 비교일치가 발생한 횟수
int state = 0;                          // LED 점멸 상태

ISR(TIMER0_COMPA_vect)
{
    count++;
    TCNT0 = 0;                          // 자동으로 0으로 변하지 않는다.
}

int main(void)
{
    DDRB = 0x20;                        // PB5 핀을 출력으로 설정
    PORTB = 0x00;                       // LED는 끈 상태에서 시작

    TCCR0B |= (1 << CS02) | (1 << CS00);  // 분주비를 1024로 설정

    OCR0A = 128;                        // 비교일치 기준값

    TIMSK0 |= (1 << OCIE0A);            // 비교일치 인터럽트 허용
    sei();                              // 전역적으로 인터럽트 허용

    while(1){
        if(count == 64){                // 비교일치 64회 발생 = 0.5초 경과
            count = 0;                  // 카운터 초기화
            state = !state;             // LED 상태 반전
            if(state) PORTB = 0xFF;     // LED 켜기
            else PORTB = 0x00;          // LED 끄기
        }
    }
}
```

인터럽트를 짧게 만들어 줌!

EMBEDDED

비교일치 인터럽트

코드 14-4 비교일치 인터럽트를 이용한 Blink 3

```
#include <avr/io.h>
#include <avr/interrupt.h>

int state = 0;                          // LED 점멸 상태

ISR(TIMER1_COMPA_vect)
{
    TCNT1 = 0;                          // 자동으로 0으로 변하지 않는다.

    state = !state;                     // LED 상태 반전
    if(state) PORTB = 0xFF;             // LED 켜기
    else PORTB = 0x00;                  // LED 끄기
}

int main(void)
{
    DDRB = 0x20;                        // PB5 핀을 출력으로 설정
    PORTB = 0x00;                       // LED는 끈 상태에서 시작

    TCCR1B |= (1 << CS12) | (1 << CS10); // 분주비를 1024로 설정

    OCR1A = 0x2000;                     // 비교일치 기준값

    TIMSK1 |= (1 << OCIE1A);            // 비교일치 인터럽트 허용
    sei();                              // 전역적으로 인터럽트 허용

    while(1){ }
}
```

타이머/카운터 1번 (16bit) 사용

비트	15	14	13	12	11	10	9	8
TCNT1H	TCNT1 [15:8]							
TCNT1L	TCNT1 [7:0]							
비트	7	6	5	4	3	2	1	0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-6 TCNT1 레지스터의 구조

비트	15	14	13	12	11	10	9	8
OCR1xH	OCR1x [15:8]							
OCR1x1L	OCR1x [7:0]							
비트	7	6	5	4	3	2	1	0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-7 OCR1x(x = A, B) 레지스터의 구조

1clk = 1024/16M [s]
0.5 [s] = 8K * clk [s]

EMBEDDED

파형 출력

표 14-3 비교일치 인터럽트 시 파형 출력 핀

타이머/카운터	파형 출력 핀		아두이노 핀 번호
0	OC0A	PD6	6
	OC0B	PD5	5
1	OC1A	PB1	9
	OC1B	PB2	10
2	OC2A	PB3	11
	OC2B	PD3	3

비트	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10
읽기/쓰기	R/W	R/W	R/W	R/W	R	R	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14-8 TCCR1A 레지스터의 구조

표 14-4 OC1A 핀의 출력

COM1A1	COM1A0	설명
0	0	OC1A 핀으로 데이터가 출력되지 않으며 OC1A 핀은 일반적인 범용 입출력 핀으로 동작한다.
0	1	비교일치가 발생하면 OC1A 핀의 출력은 반전된다.
1	0	비교일치가 발생하면 OC1A 핀의 출력은 LOW 값으로 바뀐다.
1	1	비교일치가 발생하면 OC1A 핀의 출력은 HIGH 값으로 바뀐다.

코드 14-5 파형 생성 1

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(TIMER1_COMPA_vect)
{
    TCNT1 = 0;                                  // 자동으로 0으로 변하지 않는다.
}

int main(void)
{
    TCCR1B |= (1 << CS12) | (1 << CS10);        // 분주비를 1024로 설정

    OCR1A = 0x2000;                             // 비교일치 기준값

    // 비교일치 인터럽트 발생 시 OC1A 핀의 출력을 반전
    TCCR1A |= (1 << COM1A0);

    DDRB |= (1 << PB1);                         // OC1A 핀(PB1 핀)을 출력으로 설정

    TIMSK1 |= (1 << OCIE1A);                    // 비교일치 인터럽트 허용
    sei();                                      // 전역적으로 인터럽트 허용

    while(1){ }
}
```

표 14-5 PWM이 아닌 파형 생성 모드

모드	TCCR1B		TCCR1A		설명
	WGM13	WGM12	WGM11	WGM10	
0	0	0	0	0	정상 모드(디폴트 모드)
4	0	1	0	0	CTC 모드
12	1	1	0	0	CTC 모드
13	1	1	0	1	-

파형 출력

코드 14-6 파형 생성 2

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    TCCR1B |= (1 << CS12) | (1 << CS10);  // 분주비를 1024로 설정

    OCR1A = 0x2000;                       // 비교일치 기준값

    // 비교일치 인터럽트 발생 시 OC1A 핀의 출력을 반전
    TCCR1A |= (1 << COM1A0);
    TCCR1B |= (1 << WGM12);               // CTC 모드 선택

    DDRB |= (1 << PB1);                   // OC1A 핀(PB1 핀)을 출력으로 설정

    while(1){ }
}
```

CTC 모드: 자동으로 카운터 초기화
-> 인터럽트 루틴 제거

EMBEDDED

아두이노의 타이머

MsTimer2 라이브러리를 이용하여 할 수 있다고 함.

그냥 delay() 쓰는게 편하지 않을까요?ㅋㅎ

EMBEDDED

PWM 이란?

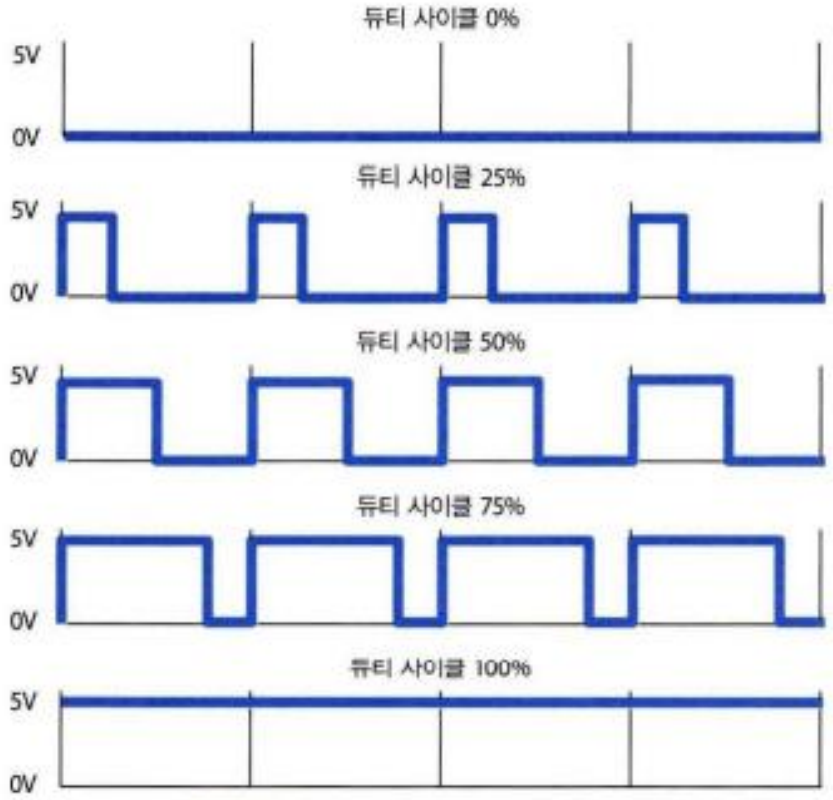

그림 15-6 PWM 신호의 듀티 사이클

PWM 모드

표 15-2 타이머/카운터 동작을 위한 용어 정의

단어	설명
BOTTOM	카운터의 값이 0x00일 때를 가리킨다.
MAX	카운터의 값이 0xFF일 때를 가리킨다.
TOP	카운터가 가질 수 있는 최댓값을 가리킨다. 오버플로 인터럽트의 경우 TOP은 0xFF이지만, 비교일치 인터럽트의 경우 사용자가 설정한 값이 TOP이 된다.

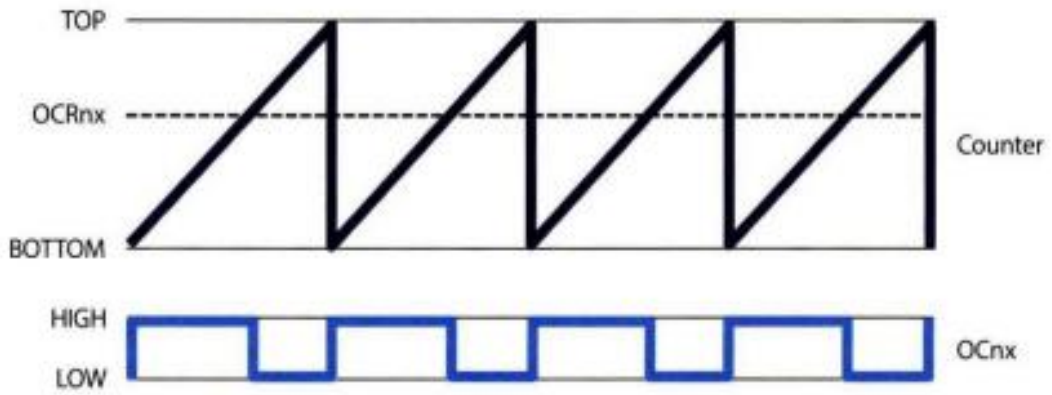

그림 15-8 비반전 고속 PWM 모드에서의 PWM 파형 생성

PWM 모드

고속 PWM 모드

$$f_{fast\,PWM} \approx \frac{f_{osc}}{N \cdot \mathrm{TOP}}$$

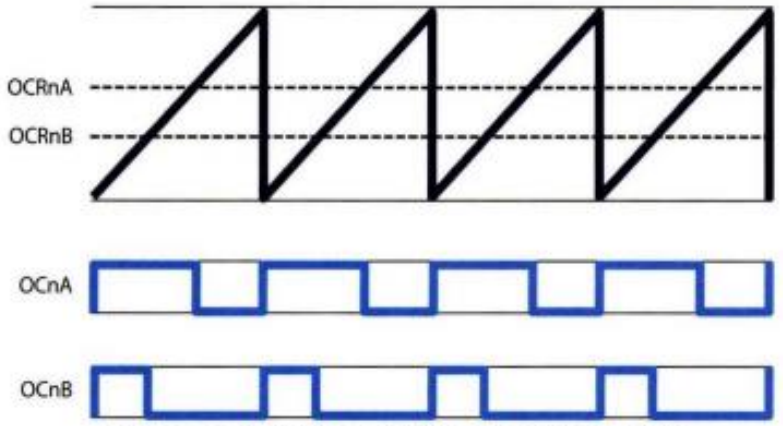

그림 15-10 고속 PWM 모드에서 비교일치 값에 따른 파형

위상 교정 PWM 모드

$$f_{PC\,PWM} \approx \frac{f_{osc}}{N \cdot 2 \cdot \mathrm{TOP}}$$

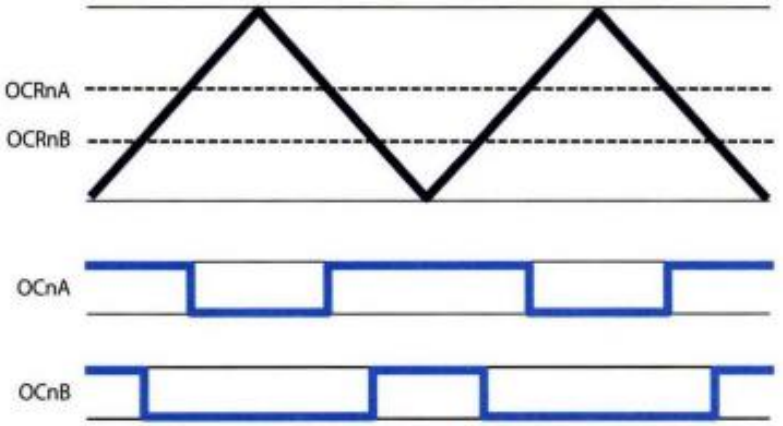

그림 15-11 위상 교정 PWM 모드에서 비교일치 값에 따른 파형

PWM 모드

위상 및 주파수 교정 PWM 모드

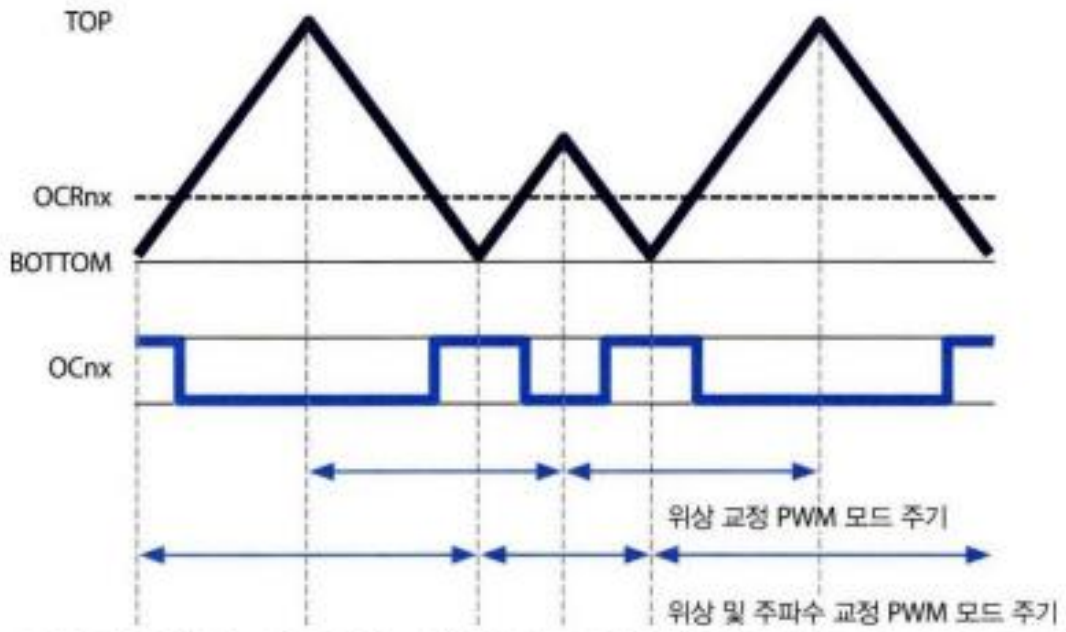

그림 15-13 TOP 값이 바뀌는 경우 위상 교정 PWM 모드와 위상 및 주파수 교정 PWM 모드에서의 주기

PWM 모드

표 15-8 WGM 비트 설정에 따른 1번 타이머/카운터의 파형 생성 모드

모드	TCCR1B		TCCR1A		설명	TOP
	WGM13	WGM12	WGM11	WGM10		
0	0	0	0	0	정상 모드	0xFFFF
1	0	0	0	1	8비트 위상 교정 PWM 모드	0x00FF
2	0	0	1	0	9비트 위상 교정 PWM 모드	0x01FF
3	0	0	1	1	10비트 위상 교정 PWM 모드	0x03FF
4	0	1	0	0	CTC 모드	OCR1A
5	0	1	0	1	8비트 고속 PWM 모드	0x00FF
6	0	1	1	0	9비트 고속 PWM 모드	0x01FF
7	0	1	1	1	10비트 고속 PWM 모드	0x03FF
8	1	0	0	0	위상 및 주파수 교정 PWM 모드	ICR1
9	1	0	0	1	위상 및 주파수 교정 PWM 모드	OCR1A
10	1	0	1	0	위상 교정 PWM 모드	ICR1
11	1	0	1	1	위상 교정 PWM 모드	OCR1A
12	1	1	0	0	CTC 모드	ICR1
13	1	1	0	1	-	-
14	1	1	1	0	고속 PWM 모드	ICR1
15	1	1	1	1	고속 PWM 모드	OCR1A

PWM 모드

표 15-3 WGM 비트 설정에 따른 2번 타이머/카운터의 파형 생성 모드[39]

모드	TCCR2B	TCCR2A		설명	TOP
	WGM22	WGM21	WGM20		
0	0	0	0	정상 모드	0xFF
1	0	0	1	위상 교정 PWM 모드	0xFF
2	0	1	0	CTC 모드	OCR2A
3	0	1	1	고속 PWM 모드	0xFF
4	1	0	0	-	-
5	1	0	1	위상 교정 PWM 모드	OCR2A
6	1	1	0	-	-
7	1	1	1	고속 PWM 모드	OCR2A

SPI

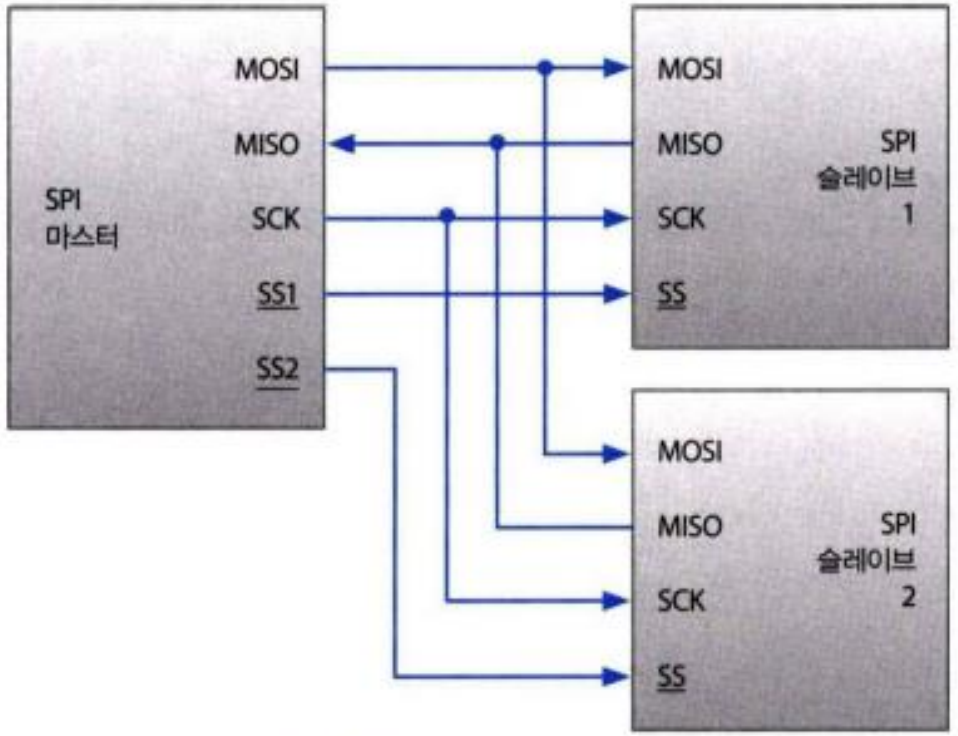

그림 16-2 일대다 SPI 연결

SPI

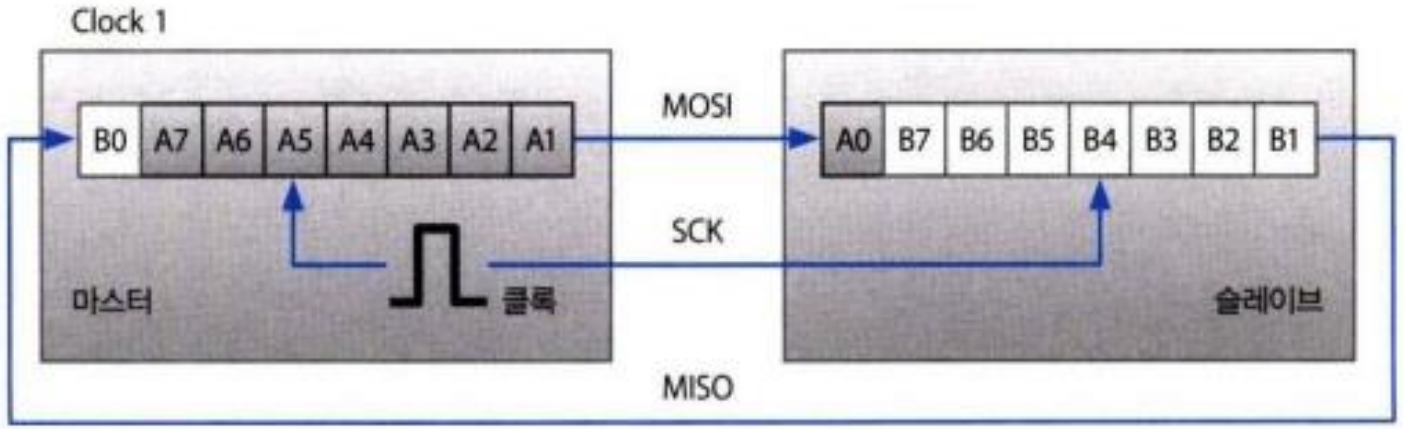

그림 16-4 1비트 데이터 전송

SPI

- **CPOL**: SPI 버스가 유휴 상태일 때의 클록 값을 결정한다. CPOL = 0이면 비활성 상태일 때 SCK는 LOW 값을 가지며, CPOL = 1이면 비활성 상태일 때 SCK는 HIGH 값을 가진다.
- **CPHA**: 데이터를 샘플링하는 시점을 결정한다. CPHA = 0이면 데이터는 비활성 상태에서 활성 상태로 바뀌는 에지에서 샘플링되고, CPHA = 1이면 데이터는 활성 상태에서 비활성 상태로 바뀌는 에지에서 샘플링된다.

SPI

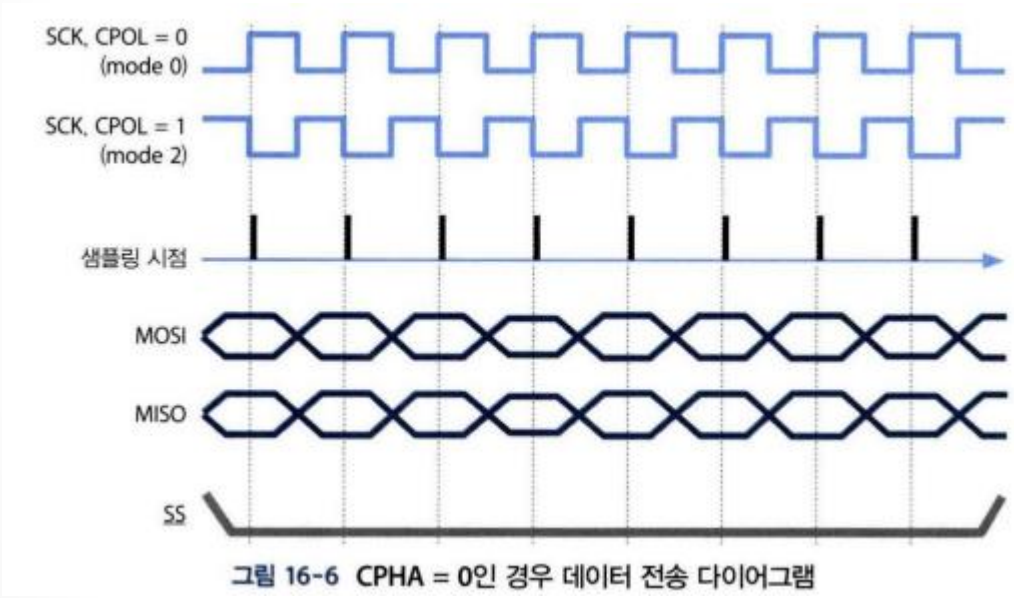

그림 16-6 CPHA = 0인 경우 데이터 전송 다이어그램

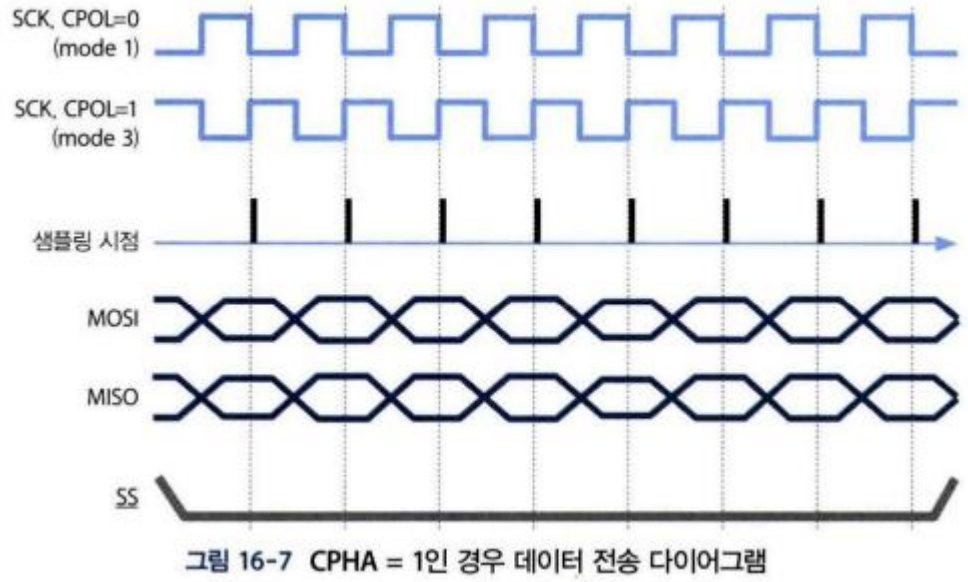

그림 16-7 CPHA = 1인 경우 데이터 전송 다이어그램

CPOL: CLK의 비활성 상태 결정(HIGH, LOW)
CPHA: Rising / Falling Edge 결정

성 균 관 대 학 교

Thank You

로 봇 동 아 리