Chapter 16. 8비트 타이머/카운터

1. 타이머/카운터

타이머/카운터는 입력되는 펄스를 세는 장치, 즉, 카운터이다. 하지만 주기가 일정한 펄스가 입력된다면 펄스의 개수를 세서 시간을 측정할 수 있으므로 타이머의 역할도 할 수 있어 '타이머/카운터'라고 불린다. ATmega128에서는 8비트 타이머/카운터 2개(Timer/Counter 0, 2)와 16비트 타이머/카운터 2개(Timer/Counter 1, 3)를 제공하고 있다. 타이머/카운터는 다양한 용도로 활용이 가능하지만 그 구조가 복잡하여 사용하기 쉽지 않은 것도 사실이다. 먼저 8비트 타이머/카운터에 대해 살펴보자. 8비트 타이머/카운터 역시 0번과 2번 2개가 있으며 이들 역시 약간의 차이가 있다. 8비트 타이머/카운터의 구조를 비교해 보자.

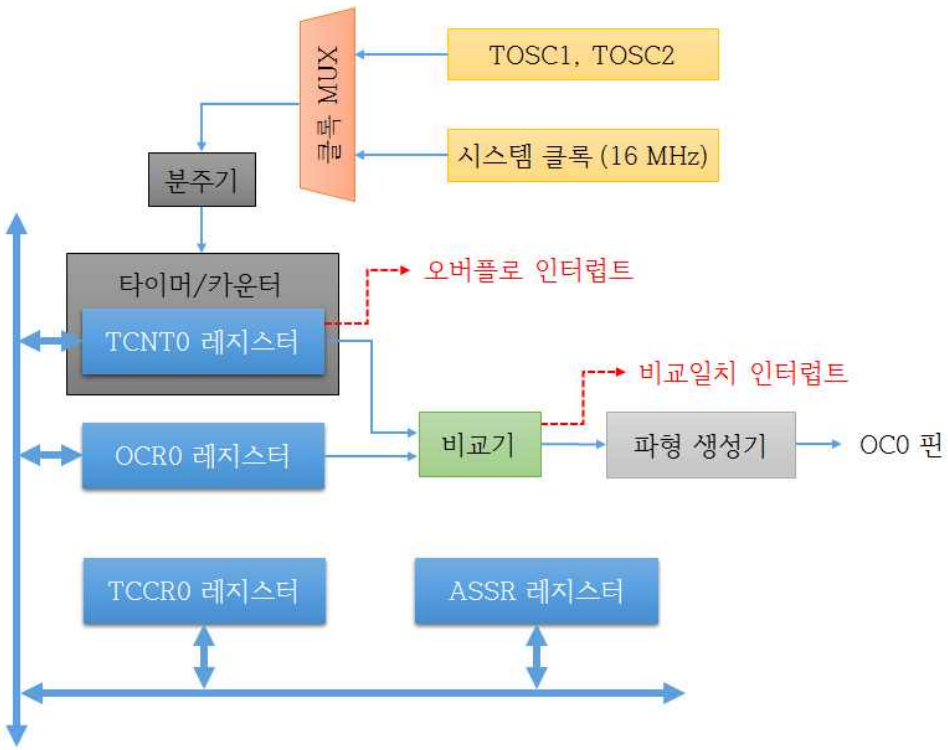

그림 1. 타이머/카운터 0번 블록 다이어그램

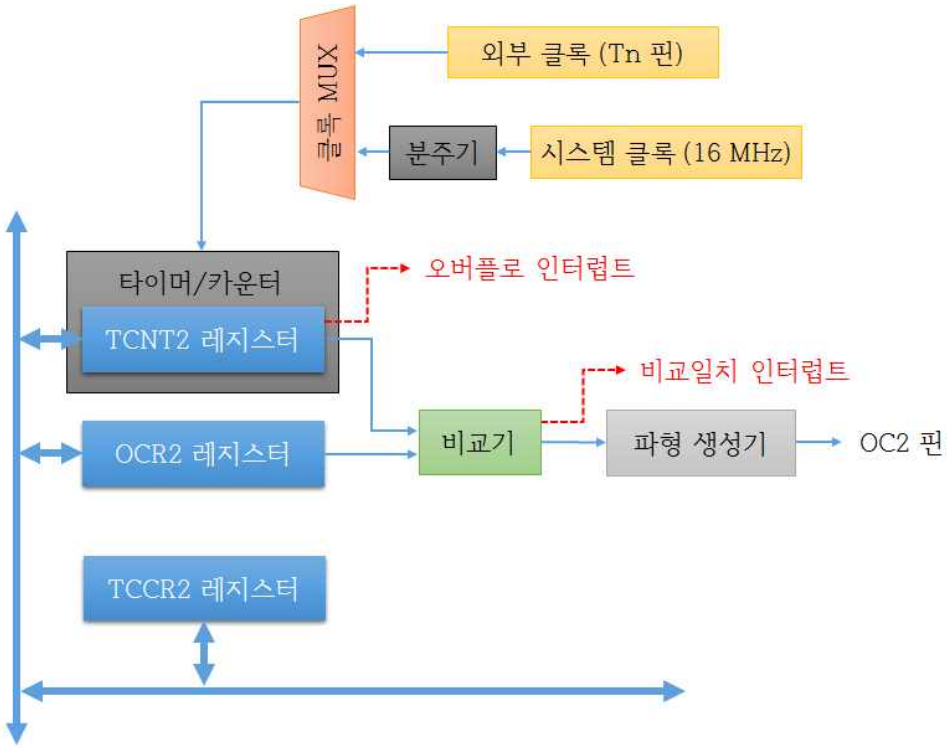

그림 2. 타이머/카운터 2번 블록 다이어그램

타이머/카운터는 기본적으로 입력 펄스를 센다. 펄스를 세기 위해서는 입력 펄스가 존재해야 하며 이는 마이크로컨트롤러의 시스템 클록이나 외부에서 주어지는 클록 중에서 선택해서 사용할 수 있다. 타이머/카운터 0번의 경우 시스템 클록이나 외부 오실레이터를 연결하여 사용할 수 있는 반면, 타이머/카운터 2번의 경우 시스템 클록이나 외부 클록을 연결하여 사용할 수 있다.

8비트 타이머/카운터의 경우 0에서 255까지만 셀 수 있다. ATmega128의 내부 클록은 16MHz 속도로 동작하므로 256번 카운트하는 시간은 256/16M = 0.016ms로 긴 시간 간격을 가지는 타이머를 만들 수 없다. 따라서 분주기(prescalor)를 사용하여 사용된 클록에 비해 주기가 긴 클록을 생성하여 긴 시간을 측정할 수 있도록 하고 있다. 타이머/카운터 0번과 2번에서 분주기의 위치가 서로 다르다는 점도 눈여겨 볼 필요가 있다.

현재까지 센 펄스의 수는 TCNTn (n = 0, 2) 레지스터(Timer/Counter Register)에 기록된다. 펄스의 개수가 셀 수 있는 최댓값을 넘어서면 TCNTn 레지스터는 0으로 초기화되며 이때 오버플로 인터럽트(Overflow Interrupt)가 발생한다. 오버플로가 발생하는 시간은 시스템 클록과 분주비에 의해 결정되며 분주비는 몇 가지 값 중 하나만을 사용할 수 있어 다양한 시간 간격을 측정하기는 어렵다. 따라서 보다 다양한 시간 간격 측정을 위해서 사용할 수 있도

록 비교일치 인터럽트(Compare Match Interrupt)가 준비되어 있다. 비교일치 인터럽트는 TCNTn 레지스터에 저장된 현재까지의 펄스 개수가 미리 설정된 값과 동일할 때 발생한다. 인터럽트가 발생하는 시점을 나타내는 '미리 설정된 값'은 OCRn (n = 0, 2) (Output Compare Register) 레지스터에 저장된다. 비교일치 인터럽트 발생과는 별도로 비교일치가 발생하면 OCn (n = 0, 2) 핀을 통해 지정된 파형을 출력하도록 할 수 있다. 타이머/카운터의 세부 동작들은 TCCRn (n = 0, 1, 2; x = A, B) 레지스터(Timer/Counter Control Register)를 통해 제어된다.

2. 오버플로 인터럽트

먼저 오버플로 인터럽트를 사용해 보자. 코드 1은 타이머/카운터 0번을 사용하여 0.5초 간격으로 PB0 핀에 연결된 LED를 점멸하는 코드이다.

코드 1. 오버플로 인터럽트를 이용한 LED 점멸

```
#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 = 0x01;             // LED 켜기
    else PORTB = 0x00;                  // LED 끄기
  }
}

int main(void)
{
```

```
    DDRB = 0x01;                                // PB0 핀을 출력으로 설정
    PORTB = 0x00;                               // LED는 끈 상태에서 시작

    // 분주비를 1024로 설정
    TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);

    TIMSK |= (1 << TOIE0);                      // 오버플로 인터럽트 허용

    sei();                                      // 전역적으로 인터럽트 허용

    while(1){ }
}
```

LED를 일정한 시간 간격으로 점멸하는 코드는 그다지 신기할 것이 없다. 하지만 코드 1은 타이머/카운터 0번의 오버플로 인터럽트를 사용하여 구현하였다는 데 의미가 있다. 현재까지 센 펄스의 개수는 TCNT0에 기록된다. TCNT0 레지스터의 구조는 그림 3과 같으며 8비트의 카운트 값을 유지한다.

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 3. TCNT0 레지스터 구조

타이머/카운터는 디폴트로 비활성 상태에 있으므로 타이머/카운터를 사용하기 위해서는 활성 상태로 변경하여야 한다. 코드 1에서 타이머/카운트를 활성 상태로 변경하는 부분은 TCCR0 (Timer/Counter Control Register) 레지스터에 분주비를 설정하는 부분이다. TCCR0 레지스터의 구조는 그림 4와 같다.

비트	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
읽기/쓰기	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 4. TCCR0 레지스터 구조

7번 비트 FOC0 (Force Output Compare) 비트, 6번에서 3번까지의 COM0n (n = 0, 1) 비트, 1번에서 0번까지의 COM0n (n = 0, 1) 비트는 비교일치 인터럽트와 관련된 비트로 뒷부분에서 다시 다룬다.

2번에서 0번까지의 CS0n (n = 0, 1, 2) (Clock Select) 비트는 분주비를 설정하기 위해 사용하는 비트로 코드 1에서는 1,024의 분주비를 설정하고 있다. CS0n 비트 설정에 따른 분주비는 표 1과 같다.

CS02	CS01	CS00	설명
0	0	0	클록 소스 없음 (타이머/카운터 정지)
0	0	1	분주비 1 (분주 없음)
0	1	0	분주비 8
0	1	1	분주비 32
1	0	0	분주비 64
1	0	1	분주비 128
1	1	0	분주비 256
1	1	1	분주비 1024

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

CS0n의 디폴트값은 000_2으로 타이머/카운터는 동작하지 않는다. 코드 1에서는 CS0n에 111_2 값을 설정함으로써 분주비가 1024가 되도록 하였다. ATmega128의 경우 16MHz 클록을 사용하므로 분주비가 1024가 되면 $\frac{16MHz}{1024} \approx 16KHz$ 클록이 타이머/카운터에 공급된다. 타이머/카운터 0번이 255까지 세는데 걸리는 시간은 얼마나 될까? 1초에 16K 개의 펄스가 타이머/카운터 0번에 주어지므로 256개 펄스를 세는 시간은 $\frac{256}{16K} = \frac{1}{64}$초가 된다. 따라서 코드 1에서는 256개 펄스를 세었을 때 발생하는 오버플로 인터럽트를 32번 체크하여 0.5초 경과를

계산하고 LED 상태를 반전시킨다.

인터럽트 역시 디폴트로 발생하지 않도록 설정되어 있으므로 타이머/카운터 0번의 오버플로 인터럽트를 활성화시켜 주어야 한다. 인터럽트 활성화 비트는 TIMSK (Timer/Counter Interrupt Mask Register) 레지스터에 존재한다. 그림 5는 TIMSK 레지스터의 구조를 나타낸다.

비트	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
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 5. TIMSK 레지스터 구조

TIMSK 레지스터의 비트 중 타이머/카운터 0번과 관련된 비트는 1번 OCIE0 비트와 0번 TOIE0 비트이다. OCIE0 (Timer/Counter0 Output Compare Match Interrupt Enable) 비트는 비교일치 인터럽트 활성화를 위해 사용되고, TOIE0 (Timer/Counter0 Overflow Interrupt Enable) 비트는 오버플로 인터럽트 활성화를 위해 사용된다. 코드 1에서는 오버플로 인터럽트 활성화를 위해 TOIE0 비트를 세트하고 있다.

오버플로나 비교일치와 같이 인터럽트에 해당하는 조건을 만족하면 TIFR (Timer/Counter Interrupt Flag Register) 레지스터의 해당 비트가 먼저 세트된다. TIFR 레지스터의 해당 비트가 세트되었을 때 TIMSK 레지스터의 해당 비트가 세트된 상태라면 실제로 인터럽트가 발생하게 된다. TIFR 레지스터의 구조는 그림 6과 같다.

비트	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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 6. TIFR 레지스터의 구조

1번 OCF0 (Output Compare Flag) 비트는 비교일치가 발생한 경우 세트되는 비트이며, 0번 TOV0 (Timer/Counter0 Overflow Flag) 비트는 오버플로가 발생한 경우 세트되는 비트이다.

타이머/카운터 0번과 관련된 인터럽트에는 카운터가 카운트할 수 있는 범위를 넘어서는 경우 발생하는 오버플로 인터럽트와, 미리 설정된 값과 동일한 경우 발생하는 비교일치 인터럽트의 2종류가 있으며, 타이머/카운터 2번 역시 마찬가지다. 표 2는 0번과 2번 타이머/카운터와 관련된 인터럽트와 각 인터럽트를 허용하는 비트를 요약한 것이다.

벡터 번호	인터럽트	벡터 이름	인터럽트 허용 비트	
10	비교일치 인터럽트	TIMER2_COMP_vect	OCIE2	Timer/Counter 2 Output Compare Match Interrupt Enable
11	오버플로 인터럽트	TIMER2_OVF_vect	TOIE2	Timer/Counter 2 Overflow Interrupt Enable
16	비교일치 인터럽트	TIMER0_COMP_vect	OCIE0	Timer/Counter 0 Output Compare Match Interrupt Enable
17	오버플로 인터럽트	TIMER0_OVF_vect	TOIE0	Timer/Counter 0 Overflow Interrupt Enable

표 2. 타이머/카운터 0번 및 2번과 관련된 인터럽트

* 1KHz는 1000Hz인가, 1024Hz인가?

정확하게 이야기 하자면 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. 분주비 1024에서 0번 타이머/카운터가 1초 동안 발생시키는 오버플로 인터럽트는 64회가 아니라 약 61.04회이며, 오버플로 인터럽트가 64회 발생하는 시간은 1초가 아니라 약 1.05초가 된다. 따라서 정밀한 시간 계산이 필요하다면 코드 1에서와 같이 1초에 64회 인터럽트가 발생하는 것으로 가정해서는 안된다. 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 ATmega128에 클리스탈의 정밀도이다. ATmega128에 사용되고 있는 16MHz 크리스탈의 정밀도는 표준편차가 0.7Hz 정도인 것으로 알려져 있다. 하지만 ATmega128에 동작 온도가 1도 상승할 때마다 크리스탈의 클록 주파수는 약 0.97Hz 증가하고, 동작 전

압이 1mV 증가할 때마다 크리스탈의 클록 주파수는 약 0.03Hz 증가하는 등 동작 환경에 따라 클록 주파수는 별할 수 있다. 따라서 정밀한 시간 계산이 필요하다면 전용의 하드웨어 RTC(Real Time Clock)나 보상 회로가 추가되어 있는 클록을 사용하여야 한다.

3. 비교일치 인터럽트

코드 1과 동일한 동작을 하는 코드를 비교일치 인터럽트를 사용해서 작성한 예가 코드 2이다.

코드 2. 비교일치 인터럽트를 이용한 LED 점멸 1

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

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

ISR(TIMER0_COMP_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 = 0x01;             // LED 켜기
    else PORTB = 0x00;                  // LED 끄기
  }
}

int main(void)
{
  DDRB = 0x01;                          // PB5 핀을 출력으로 설정
  PORTB = 0x00;                         // LED는 끈 상태에서 시작
```

```
    // 분주비를 1024로 설정
    TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);

    OCR0 = 128;                          // 비교일치 기준값

    TIMSK |= (1 << OCIE0);               // 비교일치 인터럽트 허용

    sei();                               // 전역적으로 인터럽트 허용

    while(1){ }
}
```

비교일치 인터럽트를 사용하기 위해서는 먼저 OCR0 (Output Compare Register) 레지스터에 비교값을 설정해주어야 한다. OCR0 레지스터의 구조는 그림 7과 같으며, TCNT0 레지스터의 값과 비교하기 위한 8비트 값을 저장하고 있다.

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 7. OCR0 레지스터 구조

코드 2에서는 OCR0 레지스터의 값을 128로 설정하여 오버플로 인터럽트보다 2배 빨리 인터럽트가 발생하도록 하였다. 따라서 코드 1에서는 오버플로 인터럽트가 32회 발생한 경우 0.5초가 경과한 것으로 판단하는 반면, 코드 2에서는 비교일치 인터럽트가 64회 발생한 경우 0.5초가 경과한 것으로 판단하고 있다. 한 가지 주의할 점은 비교일치가 발생한 경우 TCNT0 레지스터에 저장된 현재까지의 카운트 값이 자동으로 0으로 초기화되지 않는다는 점이다. 따라서 인터럽트 서비스 루틴에서 비교일치 인터럽트가 발생할 때마다 TCNT0 레지스터의 값을 0으로 설정해주고 있다. 만약 'TCNT0 = 0;' 문장을 삭제한다면 비교일치 인터럽트는 (128에서 오버플로를 거쳐 다음 128이 될 때까지) 256 클록마다 발생하여 오버플로 인터럽트와 동일한 주기를 가지게 된다. 인터럽트의 횟수에 따라 시간 경과를 계산하고 있으므로 'TCNT0 = 0;' 문장을 삭제하면 LED는 0.5초가 아니라 1초 간격으로 깜빡거리게 된다.

코드 2에서는 LED를 점멸하는 코드가 인터럽트 서비스 루틴에 포함되어 있다. 하지만 인터럽트 서비스 루틴은 가능한 짧게 작성하는 것이 좋다는 점을 기억하는가? 인터럽트 서비스 루틴이 실행되는 동안에는 인터럽트가 발생하지 못하며 인터럽트 서비스 루틴 실행에 시간이 걸리므로 정확한 타이밍 계산이 힘들 수 있다. 코드 2에서 LED를 점멸하는 코드를 main 함수의 이벤트 루프로 옮겨온 코드가 코드 3이다. 단, 이 경우 count 변수는 volatile로 선언되어야 한다는 점을 잊지 말아야 한다.

코드 3. 비교일치 인터럽트를 이용한 LED 점멸 2

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

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

ISR(TIMER0_COMP_vect)
{
  count++;
  TCNT0 = 0;                               // 자동으로 0으로 변하지 않는다
}

int main(void)
{
  DDRB = 0x01;                             // PB0 핀을 출력으로 설정
  PORTB = 0x00;                            // LED는 끈 상태에서 시작

  // 분주비를 1024로 설정
  TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);

  OCR0 = 128;                              // 비교일치 기준값

  TIMSK |= (1 << OCIE0);                   // 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 = 0x01;           // LED 켜기
        else PORTB = 0x00;                // LED 끄기
      }
    }
}
```

4. 파형 출력

비교일치가 발생하는 경우 인터럽트 이외에도 지정된 핀을 통해 신호를 출력하는 것이 가능하며, 그림 1과 그림 2에서 파형 생성기를 통해 OCn (n = 0, 2 핀에 연결된 부분이 여기에 해당한다. 표 3에서 알 수 있듯이 8비트 타이머/카운터는 1개의 비교일치 인터럽트만 가능하지만 16비트 타이머/카운터는 3개의 비교일치 인터럽트가 가능하다. 표 3은 각 카운터/타이머를 통해 파형을 출력할 수 있는 핀을 요약한 것이다.

타이머/카운터	파형 출력 핀	ATmega128 핀 번호
0	OC0	PB4
1	OC1A	PB5
	OC1B	PB6
	OC1C	PB7
2	OC2	PB7
3	OC3A	PE3
	OC3B	PE4
	OC3C	PE5

표 3. 비교일치 인터럽트 시 파형 출력 핀

코드 4는 0번 타이머/카운터에서 비교일치 인터럽트가 발생하는 경우 파형 생성 기능을 이용하여 OC0 핀(PB4)에 연결된 LED를 점멸하는 코드이다.

코드 4. 파형생성 1

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

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

ISR(TIMER0_COMP_vect)
{
  count++;
  TCNT0 = 0;                               // 자동으로 0으로 변하지 않는다
}

int main(void)
{
  // 파형 출력 핀인 OC0(PB4) 핀을 출력으로 설정
  DDRB = 0x10;
  PORTB = 0x00;                            // LED는 끈 상태에서 시작

  // 분주비를 1024로 설정
  TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);

  OCR0 = 255;                              // 비교일치 기준값

  // 비교일치 인터럽트 발생 시 OC0 핀의 출력을 반전
  TCCR0 |= (1 << COM00);

  TIMSK |= (1 << OCIE0);                   // 비교일치 인터럽트 허용

  sei();                                   // 전역적으로 인터럽트 허용

  while(1){ }
}
```

코드 4를 업로드 하고 LED를 살펴보자. LED가 아주 빠른 속도로 깜빡거리고 있음을 발견할

수 있다. 코드 4의 어디에도 LED를 점멸하는 코드는 찾아볼 수 없다. 하지만 LED는 깜빡거리고 있다. 비밀은 바로 그림 1과 그림 2의 파형 생성기에 있다. 파형 생성기는 비교일치 인터럽트가 발생할 때마다 OC0 핀의 출력을 반전시키도록 함으로써 LED가 깜빡거리도록 해준다. 코드 4가 코드 2, 코드 3과 비교했을 때 달라진 점은 TCCR0 레지스터의 COM00 비트를 세트하여 파형 생성기의 동작을 설정하고 있는 점이다.

TCCR0 레지스터에 대해 설명하기 전에 한 가지 언급하고 지나갈 사실이 있다. 코드 4는 LED를 아주 빠른 속도로 깜빡거리게 하고 있다. LED의 깜빡거리는 속도를 좀 더 느리게 하고 싶겠지만 현재로서는 LED의 깜빡거리는 속도를 더 느리게 할 수 없다. 분주비를 1024로 설정한 경우 오버플로 인터럽트는 1초에 약 32회 발생한다. 코드 3에서는 카운터 값이 255에 도달할 때마다 비교일치 인터럽트가 발생하므로 역시 1초에 약 32회 인터럽트가 발생하고 있다. 인터럽트가 발생할 때마다 파형 생성기는 현재 OC0의 출력을 반전시키고 있으므로 1초에 LED는 약 32회 출력이 반전된다. 코드 2와 코드 3에서 인터럽트가 발생한 횟수를 기준으로 시간을 측정한 것을 기억할 것이다. 하지만 파형 생성기는 비교일치 인터럽트가 발생할 때마다 자동으로 출력을 반전시키므로 코드 2나 코드 3과 같은 방법으로는 시간을 조절할 수는 없다. 파형 생성기를 통해 더 긴 시간 간격으로 LED를 깜빡거리도록 하기 위해서는 8비트 타이머/카운터가 아니라 16비트 타이머/카운터를 사용하면 된다. 이 장에서는 8비트 타이머/카운터를 사용하고 있으므로 LED를 제어하는 코드 없이 ATmega128에서 제공하는 하드웨어인 파형 생성기를 통해 자동으로 LED를 깜빡거릴 수 있다는 점에 만족하도록 하자.

그림 8은 파형 생성기 동작 설정에 사용된 TCCR0 레지스터의 구조를 나타낸다.

비트	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
읽기/쓰기	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 8. TCCR0 레지스터 구조

2번에서 0번까지의 CS0n (n = 0, ... , 2) 비트는 분주비 설정을 위해 사용되었다.

6번 WGM00 비트와 3번 WGM01 비트는 파형 생성 모드를 설정하는 비트로 코드 4에서는 디폴트값인 00_2, 정상 모드(Normal Mode)가 사용되었다. WGM0n 비트 설정에 따른 파형 생성 모드는 표 4와 같다.

모드 번호	WGM01 (CTC)	WGM00 (PWM)	타이머/카운터 모드	TOP
0	0	0	정상	0xFF
1	0	1	위상 교정 PWM	0xFF
2	1	0	CTC	OCR0
3	1	1	고속 PWM	0xFF

표 4. WGM0n (n = 0, 1) 비트 설정에 따른 파형 생성 모드

4개의 모드 중 PWM과 관련된 모드는 별도의 장에서 다루며, 이 장에서는 0번과 2번 모드에 대해서만 다룬다. 정상 모드와 CTC(Clear Timer on Compare match) 모드의 차이는 글자 그대로 비교일치가 발생한 경우 TCNT0 레지스터의 값을 0으로 설정하는지의 여부에 있다. 이전 코드에서는 정상 모드를 사용하였으며 TCNT0 값이 자동으로 0으로 설정되지 않아 인터럽트 서비스 루틴에서 TCNT0 값을 0으로 설정하였다. 하지만 CTC 모드를 사용하면 자동으로 TCNT0 값이 0으로 설정되므로 편리하다. 이러한 차이는 표 4에서 TOP 값에서도 찾아볼 수 있다. TOP 값은 카운터가 가질 수 있는 최댓값으로 TOP 값에 이르면 자동으로 0으로 리셋된다.

5번 COM01 비트와 4번 COM00 비트는 비교일치 출력 모드(Compare Match Output Mode)를 설정하는 비트로 파형 생성 모드에 따라 달라지지만 정상 모드와 CTC 모드의 경우에는 동일하다. 정상 모드와 CTC 모드에서 COM0n (n = 0, 1) 비트 설정에 따른 비교일치 출력 모드는 표 5와 같다.

COM01	COM00	설명
0	0	OC0 핀으로 데이터가 출력되지 않으며 OC0 핀은 일반적인 범용 입출력 핀으로 동작한다.
0	1	비교일치가 발생하면 OC0 핀의 출력은 반전된다.
1	0	비교일치가 발생하면 OC0 핀의 출력은 LOW 값으로 바뀐다. (clear)
1	1	비교일치가 발생하면 OC0 핀의 출력은 HIGH 값으로 바뀐다. (set)

표 5. PWM 이외의 모드에서 COM0n (n = 0, 1) 비트 설정에 따른 비교일치 출력 모드

코드 4에서는 COM0n 값으로 01_2를 사용하여 비교일치 인터럽트가 발생할 때마다 출력을 반

전시키고 있다.

TCCR0 레지스터의 7번 비트 FOC0 (Force Output Compare) 비트는 WGM 비트가 PWM 이외의 모드로 설정된 경우에만 세트될 수 있다. FOC0 비트가 세트되면 비교일치가 발생한 것과 동일한 효과가 파형 출력 핀 OC0 핀으로 이루어진다. FOC0 비트 세트는 일회성으로 필요할 때마다 1을 써주어야 한다. FOC0 비트는 쓰기 전용으로 읽기가 가능하기는 하지만 항상 영(0)의 값을 반환한다.

코드 4에서는 정상모드를 사용하여 인터럽트 처리 루틴에서 TCNT0 값을 강제로 0으로 리셋하였다. 이를 CTC 모드를 사용하도록 수정한 예가 코드 5로, 인터럽트 처리 루틴에서 TCNT0 값을 리셋할 필요가 없다는 점을 유의해서 살펴보자.

코드 5. 파형생성 2

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

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

ISR(TIMER0_COMP_vect)
{
  count++;
}

int main(void)
{
  // 파형 출력 핀인 OC0(PB4) 핀을 출력으로 설정
  DDRB = 0x10;
  PORTB = 0x00;                            // LED는 끈 상태에서 시작

  // 분주비를 1024로 설정
  TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);

  TCCR0 |= (1 << WGM01);                   // CTC 모드

  OCR0 = 255;                              // 비교일치 기준값
```

```
    // 비교일치 인터럽트 발생 시 OC0 핀의 출력을 반전
    TCCR0 |= (1 << COM00);

    TIMSK |= (1 << OCIE0);                  // 비교일치 인터럽트 허용

    sei();                                  // 전역적으로 인터럽트 허용

    while(1){ }
}
```