제 8 장 타이머/카운터(I) - 8비트 타이머/카운터

8.1 ATmega128의 타이머/카운터

마이크로컨트롤러에서 타이머/카운터는 입력되는 펄스를 세는 장치, 즉 카운터이다. 그러나 주기가 일정한 펄스가 입력되면 펄스의 개수로 시간을 측정할 수 있으므로 타이머 역할도 한다. 내부클록은 주기가 일정하므로 일반적으로 타이머로 사용할 때는 내부클록을 사용한다.

ATmega128은 8비트 타이머/카운터 2개와 16비트 타이머 2개를 제공한다. 타이머/카운터0과 타이머/카운터2는 8비트이며 타이머/카운터1과 타이머/카운터3은 16비트 타이머/카운터이다. 각 타이머/카운터는 독립적인 프리스케일러(prescaler)를 가지고 있다. 프리스케일러는 설정 분주비로 타이머의 입력 클록의 주파수를 낮춘다. 16MHz 클록을 사용하는 보드에서 분주비를 8로 설정하면 타이머/카운터에 입력되는 펄스의 주파수는 2MHz이다.

ATmega128은 타이머/카운터에 비교유닛(Compare Unit)을 제공한다. 이를 이용하여 파형 및 PWM신호를 생성할 수 있다.

8.2 Pulse Width Modulation(PWM) 신호

PWM신호는 그림 8.1과 같이 톱니파와 비교하여 비대칭형으로 구현하거나 그림 8.2와 같이 삼각파와 비교하여 대칭형으로 구현할 수 있다.

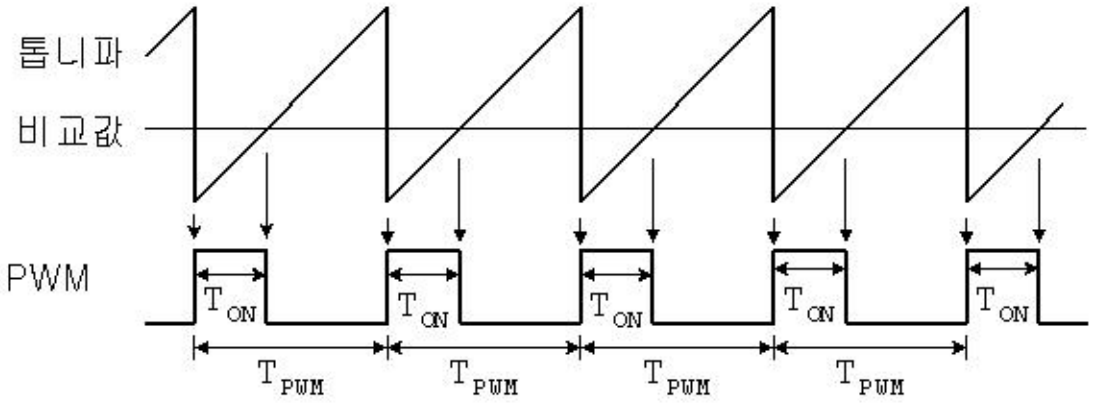

그림 8.1 비대칭형 PWM 신호

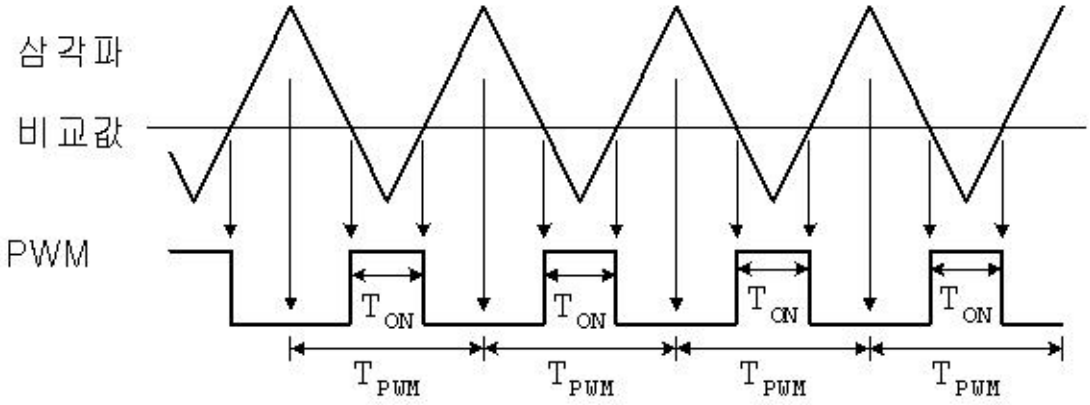

그림 8.2 대칭형 PWM 신호

☞ 두 모양 모두 톱니파(삼각파)가 비교값 보다 작을 때 HIGH를 출력하고 비교값 보다 클 때 LOW를 출력한다. 물론 출력을 반대로 할 수 있다.

☞ T_{PWM}은 일정하며 PWM신호의 반송주기(Carrier Period)라 한다. 반송주파수(Carrier frequency)는 $1/T_{PWM}$이다.

☞ 그림 8.2는 반송주기내의 파형이 대칭이므로 대칭형 PWM신호이다.

☞ T_{ON}은 펄스가 ON이 되는 시간을 말하며 비교값의 크기에 비례한다. 전체 주기 중에 ON이 되는 시간의 비를 듀티비(Duty Ratio)라하며 단위는 %를 사용한다.

$$\text{듀티비}(\%) = T_{ON}/T_{PWM} \times 100\ \% \qquad (8.1)$$

☞ PWM신호의 주기는 일정하므로 T_{ON} 또는 듀티비를 사용하여 정보를 전달한다. 즉 비교값의 크기에 대한 아날로그정보를 디지털신호인 펄스 형태로 전달할 수 있다.

☞ 듀티비 100%인 PWM신호는 DC신호가 된다. $T_{ON}=0.3T_{PWM}$이면 즉 30% 듀티비일 때는 시간당 가해지는 신호의 평균값은 DC신호의 30%가 된다. 즉 30%의 DC신호가 가해지는 것과 거의 유사한 효과를 갖게 된다. PWM신호의 장점은 ON-OFF 신호만으로 신호의 크기를 조절할 수 있고(따라서 D/A 컨버터가 필요 없음), 전력소모가 작아진다.

8.3 8비트 타이머/카운터0과 타이머/카운터2

8비트 타이머/카운터로 타이머/카운터0과 타이머/카운터2를 제공한다.

즉 타이머/카운터 값과 출력 비교 레지스터는 8비트이다. 두 타이머/카운터는 프리스케일러의 값과 외부클록을 사용하는 방법을 제외하고 같은 동작을 한다. 설명을 위해 몇 가지 용어를 정의하자.

BOTTOM	카운터 값이 0이 되면 카운터가 BOTTOM에 도달했다고 한다.
MAX	8비트 타이머/카운터에 대해 카운터 값이 0xFF(255)가 되면 MAX 값에 도달했다 말한다.
TOP	카운터 값이 최대값이 되었을 때 카운터가 TOP에 도달했다 말한다. 최대값은 0xFF이거나 비교레지스터 OCRn(n=0,2)로 설정할 수 있다.

8.3.1 8비트 타이머/카운터 0, 2의 개략도

그림 8.3은 ATmega128 매뉴얼의 8비트 타이머/카운터 개략도이다.

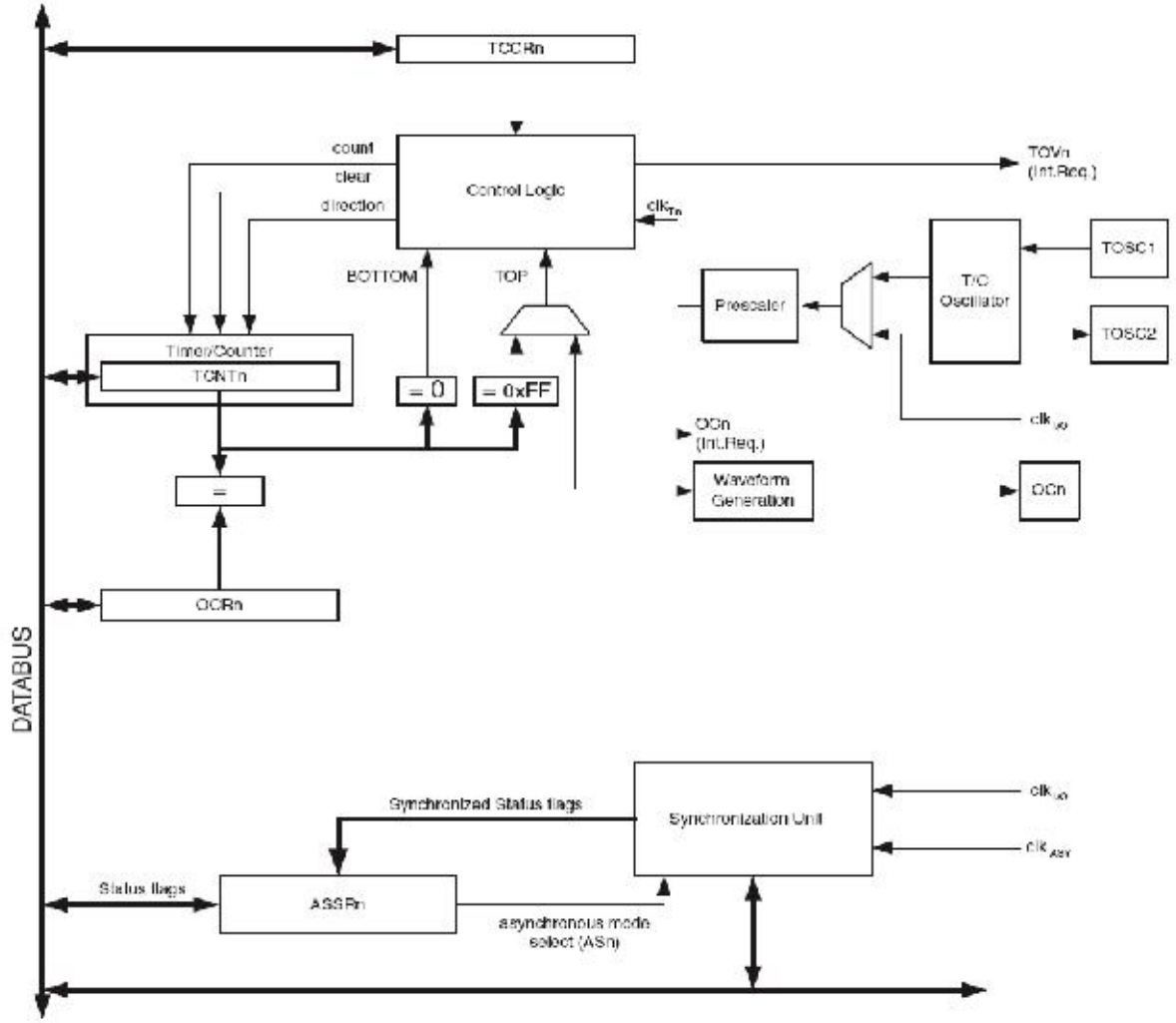

그림 8.3(a) 8비트 타이머/카운터 0 개략도(n=0)

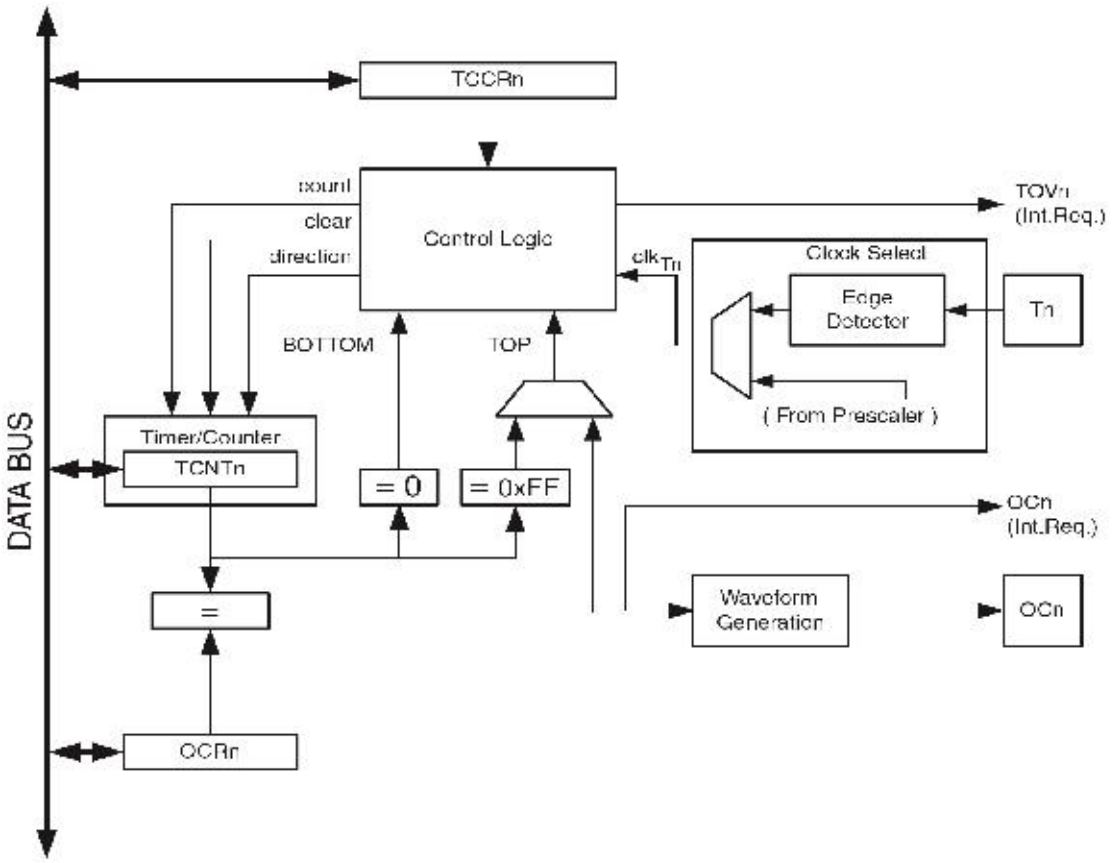

그림 8.3(b) 8비트 타이머/카운터 2 개략도(n=2)

☞ 레지스터 TCCRn(n=0,2)으로 제어로직을 설정한다.

☞ 타이머/카운터 TCNTn과 비교레지스터 OCRn을 서로 비교하여 파형을 재생하고 출력 핀 OCn에 파형이 출력된다.

☞ 타이머/카운터 0과 타이머/카운터 2는 클록의 소스 선택에서 차이를 보인다.

☞ 타이머/카운터 0은 내부클록 또는 TOSC1/TOSC2핀에 연결된 T/C Oscillator를 클록소스로 선택할 수 있다. 클록소스는 ASSR레지스터의 AS0비트로 선택한다. TOSC1/TOSC2핀은 타이머/카운터 0 내부의 T/C Oscillator와 최적화된 32.768kHz 크리스털을 연결하는 것이 좋다. TOSC1핀에 외부 클록을 직접 가하는 것은 바람직하지 않다.

☞ 타이머/카운터 2에서는 TCCR2 레지스터의 Clock Select블록에서 외부클록과 내부클록을 선택한다.

8.3.2 관련 레지스터

■ **타이머/카운터 제어레지스터(Timer/Counter Control Register):**

TCCRn(n=0,2)

비트	7	6	5	4	3	2	1	0	
	FOCn	WGMn0	COMn1	COMn0	WGMn1	CSn2	CSn1	CSn0	TCCRn
읽기/쓰기	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

• 비트 7 - FOCn: Force Output Compare

비 PWM모드로 선택되었을 때 이 비트는 활성화 된다. 이 비트에 1을 쓰면 파형생성유닛(Waveform generation unit)에 즉각적인 비교 일치(Compare Match)를 전달한다. 출력 핀 OCn의 값은 COMn1:0핀 세팅에 따라 변한다. FOCn비트의 세트는 인터럽트를 발생시키지 않는다.

• 비트 6,3 - WGMn1:0: 파형 생성 모드(Waveform generation mode)

이 비트는 카운터의 계수 순서, TOP값을 설정하는 방법 그리고 파형 생성 모드를 표 8.1과 같이 설정한다.

표 8.1 파형 생성 모드 비트 설명

모드	WGMn1	WGMn0	타이머/카운터 동작모드	TOP	OCRn 갱신시점	TOVn 플래그 세트 시점
0	0	0	Normal	0xFF	즉시	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCRn	즉시	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

• 비트 5:4 - COMn1:0: 비교 일치 출력 모드

이 비트는 출력 핀 OCn의 동작을 제어한다. 이 비트들 중 하나 이상의 비트가 세트되면 OCn은 공유하는 범용 I/O핀의 기능을 대신한다. (OC0은 PB4와 OC2은 PB7과 공유한다.) 그러나 OCn핀을 사용하기 위해서는 공유 범용 I/O핀의 방향을 출력으로 설정하여야 한다.

표 8.2는 비 PWM 모드(Normal 모드와 CTC모드)일 때 비트의 설정에 따른 출력 핀 OCn의 동작모드를 나타낸다.

표 8.2 비 PWM모드일 때 비교 출력 모드

COMn1	COMn0	설명
0	0	범용 I/O로 동작한다. OCn 핀은 연결되지 않음.
0	1	비교 일치가 될 때 OCn핀의 상태를 반전시킨다.
1	0	비교 일치가 될 때 OCn핀에 0을 출력한다.
1	1	비교 일치가 될 때 OCn핀에 1을 출력한다.

표 8.3은 Fast PWM모드일 때 비트의 설정에 따른 출력 핀 OCn의 동작모드를 나타낸다.

표 8.3 Fast PWM모드일 때 비교 출력 모드

COMn1	COMn0	설명
0	0	범용 I/O로 동작한다. OCn 핀은 연결되지 않음.
0	1	예약됨.
1	0	비교 일치가 될 때 OCn핀을 0으로 하고 TOP 값에서 1을 출력
1	1	비교 일치가 될 때 OCn핀을 1로 하고 TOP 값에서 0을 출력

표 8.4는 Phase Correct PWM모드일 때 비트의 설정에 따른 출력 핀 OCn의 동작모드를 나타낸다.

표 8.4 Phase Correct PWM모드일 때 비교 출력 모드

COMn1	COMn0	설명
0	0	범용 I/O로 동작한다. OCn 핀은 연결되지 않음.
0	1	예약됨.
1	0	카운터 증가모드일 때 비교 일치가 되면 OCn핀에 0을 출력한다. 카운터 감소모드일 때 비교 일치가 되면 OCn핀에 1을 출력한다.
1	1	카운터 증가모드일 때 비교 일치가 되면 OCn핀에 1을 출력한다. 카운터 감소모드일 때 비교 일치가 되면 OCn핀에 0을 출력한다.

• 비트 2:0 – CSn2:0: 클록선정

이 비트로 타이머/카운터 n이 사용하는 클록을 표 8.5와 8.6과 같이 선정한다.(타이머/카운터0과 타이머/카운터2가 서로 다르다.)

표 8.5 타이머/카운터0의 클록 선정

CS02	CS01	CS00	설명
0	0	0	타이머/카운터 정지(클록사용하지 않음)
0	0	1	clk_{TOS} (프리스케일링 없음)
0	1	0	clk_{TOS}/8(프리스케일링 입력)
0	1	1	clk_{TOS}/32(프리스케일링 입력)
1	0	0	clk_{TOS}/64(프리스케일링 입력)
1	0	1	clk_{TOS}/128(프리스케일링 입력)
1	1	0	clk_{TOS}/256(프리스케일링 입력)
1	1	1	clk_{TOS}/1024(프리스케일링 입력)

clk_{TOS}는 프리스케일러 입력 클록을 말한다. ASSR레지스터의 AS0이 0일 때는 내부클록, 1일 때는 TOSC1/TOSC2핀의 수정발진자 주파수이다.

표 8.6 타이머/카운터2의 클록 선정

CS22	CS21	CS20	설명
0	0	0	타이머/카운터 정지(클록사용하지 않음)
0	0	1	$clk_{I/O}$ (프리스케일링 없음)
0	1	0	$clk_{I/O}$/8(프리스케일링 입력)
0	1	1	$clk_{I/O}$/64(프리스케일링 입력)
1	0	0	$clk_{I/O}$/256(프리스케일링 입력)
1	0	1	$clk_{I/O}$/1024(프리스케일링 입력)
1	1	0	T2핀의 외부 클록사용. 하강모서리에서 클록을 함
1	1	1	T2핀의 외부 클록사용. 상승모서리에서 클록을 함

1. $clk_{I/O}$는 내부 클록을 말한다.
2. 외부 클록 사용모드를 사용하면 T2핀이 출력방향으로 설정되더라도 클록으로 동작을 한다.

■ **타이머/카운터 레지스터(Timer Counter Register):TCNTn(n=0,2)**

비트	7	6	5	4	3	2	1	0	
	TCNTn[7:0]								TCNTn
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

타이머/카운터 레지스터로서 읽기/쓰기를 직접 할 수 있다.

■ **출력 비교 레지스터(Output Compare Register): OCRn(n=0,2)**

비트	7	6	5	4	3	2	1	0	
	OCRn[7:0]								OCRn
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

OCRn은 타이머/카운터 레지스터 TCNTn과 지속적으로 비교된다. 비교 일치는 출력비교일치 인터럽트나 OCn핀에 파형을 출력할 때 사용된다.

■ **타이머/카운터 인터럽트 플래그 레지스터(Timer/Counter Interrupt Flag Register): TIFR**

비트	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

- 비트 1,7 - OCFn(n=0,2) : 출력 비교 플래그 (Output Compare Flag) 타이머 값 TCNTn과 비교레지스터 값 OCRn이 같을 때 플래그 OCFn이 세트된다. 이 플래그는 해당 인터럽트 서비스루틴이 수행되면서 자동으로 지워진다. 이 비트를 지우는 다른 방법은 OCFn비트에 1을 쓰면 플래그가 지워진다.
- 비트 0,6 - TOVn(n=0,2) : 타이머/카운터 n 오버플로 플래그 (Timer/Counter Overflow Flag)

타이머 값 TCNTn에 오버플로가 생길 때 플래그 TOVn이 세트된다. 이 플래그는 해당 인터럽트 서비스루틴이 수행되면서 자동으로 지워진다. 이 비트를 지우는 다른 방법은 TOVn비트에 1을 쓰면 플래그가 지워진다. PWM모드에서는 TCNTn이 0x00될 때 플래그가 세트된다.

■ 타이머/카운터 인터럽트 마스크 레지스터(Timer/Counter Interrupt Mask Register): TIMSK

비트	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

• 비트 1,7 - OCIEn(n=0,2) : 출력 비교 일치 인터럽트 허용
(Output Compare Match Interrupt Enable)

OCIEn비트가 1이고 SREG레지스터의 I비트가 1이면 출력비교 일치 인터럽트를 허용한다. 출력 비교 일치 인터럽트는 타이머 값 TCNTn과 비교레지스터 값 OCRn이 같을 때 즉 출력 비교 플래그 OCFn가 세트될 때 발생한다.

• 비트 0,6 - TOIEn(n=0,2) : 타이머 오버플로 인터럽트 허용
(Timer/Counter Overflow Interrupt Enable)

TOIEn비트가 1이고 SREG레지스터의 I비트가 1이면 타이머 오버플로 인터럽트를 인터럽트를 허용한다. 타이머 오버플로 인터럽트는 타이머 값 TCNTn가 오버플로가 생길 때 즉 타이머/카운터 오버플로 플래그 TOVn이 세트될 때 발생한다.

■ 비동기 상태 레지스터(Asynchronous Status Register): ASSR (타이머/카운터 0에만 해당)

비트	7	6	5	4	3	2	1	0	
	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB	ASSR
읽기/쓰기	R	R	R	R	R/W	R	R	R	
초기 값	0	0	0	0	0	0	0	0	

• 비트 3 - AS0: 비동기 타이머/카운터0
비트가 0으로 설정되면 타이머/카운터 0의 입력은 내부 클록이다. 비트가 1로 설정되면 타이머/카운터 0은 비동기모드로 설정된다. 이 때 타이머/카운터 0의 입력은 TOSC1핀에 연결된 수정발진자의 펄스이다.
• 비트 2 - TCN0UB: Timer/Counter0 Update Busy
• 비트 1 - OCR0UB: Output Compare Register0 Update Busy
• 비트 0 - TCR0UB: Timer/Counter Control Register0 Update Busy

■ 특수기능 IO 레지스터(Special Function IO Register): SFIOR

비트	7	6	5	4	3	2	1	0	
	TSM	-	-	-	ACME	PUD	PSR0	PSR321	SFIOR
읽기/쓰기	R/W	R	R	R	R/W	R/W	R/W	R/W	
초기 값	0	0	0	0	0	0	0	0	

• 비트 7 - TSM : 타이머/카운터 동기 모드
(Timer/Counter Synchronization Mode)
ATmega128에는 두 개의 프리스케일러가 있다. 타이머/카운터 0가 프리스케일러 하나를 사용하고 타이머/카운터1,2,3은 다른 하나의 프리스케일러를 공유한다. TSM비트를 1로 세트하면 타이머/카운터 동기 모드가 된다. 이 때 PSR0와 PSR321에 써진 값을 보존하므로 PSR0와 PSR321 리셋 신호가 그대로 유지된다. 따라서 이 모드를 사용하면 해당 타이머/카운터를 정지시키고 같은 타이머/카운터 값을 설정할 수 있다. TSM비트를 0으로 지우면 PSR0와 PSR321이 하드웨어에 의해 0으로 지워지므로 타이머/카운터를 동시에 시작시킬 수 있다.
• 비트 1 - PSR0 : 프리스케일러 리셋 타이머/카운터 0
(Prescaler Reset Timer/Counter0)
이 비트가 1로 세트되면 타이머/카운터0의 프리스케일러가 리셋된다. 표준적으로 이 비트는 하드웨어에 의해 즉시 0으로 지워진다. 타이머/카운터0이 비동기모드로 동작 중에 이 비트에 1이 써지면 이 비트는 프리스케일러가 리셋될 때까지 1이 유지된다. TSM비트가 1로 세트되어 있을 때 이 비트는 하드웨어에 의해 0으로 지워지지 않는다.

8.3.3 동작모드

8비트 타이머의 동작모드는 TCCRn레지스터의 WGMn1:0비트와 COMn1:0로 설정한다.

(가) 표준 모드

☞ TCCRn레지스터의 WGMn1:0=0로 설정하는 가장 간단한 동작모드이다.
☞ 타이머/카운터 값 TCNTn은 항상 증가를 하며 값이 8비트의 최대 값(TOP=0xFF)을 지나면 다시 바닥(BOTTOM=0x00)부터 다시 카운트를 시작한다.
☞ TCNTn이 0xFF에서 0x00으로 변할 때 0x00이 되는 시점에서 오버플로가 발생하고 오버플로 플래그 TOVn이 세트된다.
☞ 출력 비교 레지스터를 사용하여 주어진 시점에 출력 비교 값 일치 인터럽트를 발생시킬 수 있다. 그러나 출력 비교 레지스터를 사용하여 파형을 생성하는 것은 과도한 CPU시간을 소비하기 때문에 권고되지 않는다.

(나) CTC(Clear Timer on Compare Match) 모드

☞ TCCRn레지스터의 WGMn1:0=2로 설정하는 동작모드이다.
☞ 타이머/카운터 값 TCNTn는 출력비교 레지스터 OCRn값과 일치될 때까지 증가하고 이 후 TCNTn는 0으로 지워지고, 출력 비교 인터럽트 플래그 OCFn비트가 세트된다.
☞ 출력 핀 OCn의 동작은 COMn1:0의 값으로 제어한다.(표8.2) OCn핀에 출력을 하려면 공유 범용 I/O핀의 방향을 출력으로 설정해야 한다.

그림 8.4는 CTC모드에서 COMn1:0=1로 설정되었을 때 출력 핀, 비교인터럽트의 동작을 보여준다. 그림에서 조그만 수평바는 TCNTn값이 OCRn과 일치할 때를 나타낸다. TCNTn값이 OCRn과 일치하면 TCNTn이 0으로 지워지고 비교 일치 인터럽트 플래그 OCFn이 세트된다. 동시에 COMn1:0 세팅에 따라 출력 핀 OCn의 값은 반전(Toggle)된다.

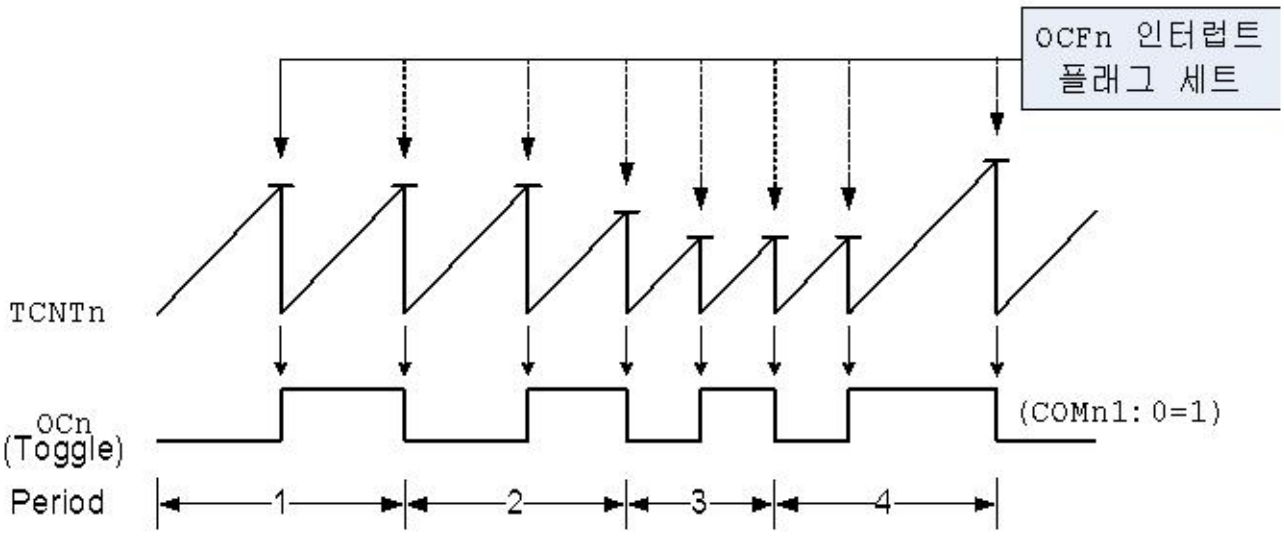

그림 8.4 CTC모드의 동작

☞ 생성되는 파형의 주파수는 비교 레지스터 OCRn으로 설정할 수 있다.

- 프리스케일러 값이 N이면 내부 클록 N개마다 TCNTn값이 1씩 증가한다. 따라서 TCNTn이 1 증가하는 데 필요한 소요시간 t_t는

$$t_t = N / f_{clk\_I/O} \qquad (8.2)$$

여기서 $f_{clk_I/O}$는 내부클록 주파수를 말한다.

- TCNTn은 0부터 OCRn값 까지 증가하였다가 0으로 리셋되므로 0부터 다음 0까지 소요 카운트 값은 OCRn+1이다.
- TCNTn이 0으로 리셋 될 때마다 출력 OCn핀의 상태가 반전되므로 한 파형을 만드는 데 0으로 리셋이 두 번 일어나야 하므로 소요 카운트 값은 2·(OCRn+1)이다.
- 따라서 파형의 주기 T_{OCn}과 주파수 f_{OCn}은

$$T_{OCn} = 2\cdot(OCRn+1)\cdot t_t = 2\cdot N\cdot(OCRn+1)/f_{clk\_I/O} \text{ sec} \qquad (8.3)$$

$$f_{OCn} = 1/T_{OCn} = f_{clk\_I/O}/(2\cdot N\cdot(OCRn+1)) \text{ Hz} \qquad (8.4)$$

☞ 각 타이머의 프리스케일러 값은 표 8.5와 표8.6을 참고하여 설정한다.

예) 내부 클록 16MHz를 사용, 프리스케일러 값이 8, OCRn=15로 설정되면 파형의 주파수 및 주기는 다음과 같다.

$$t_t = 8 / 16\times10^6 = 0.5 \times10^{-6} \text{ sec} = 0.5\ \mu\text{sec}$$

$$T_{OCn} = 2\cdot8\cdot(15+1)/16\times10^6 \text{ sec} = 16\ \mu\text{sec}$$

$$f_{OCn} = 16\times10^{6}/(2\cdot8\cdot(15+1))Hz = 0.0625MHz$$
$$= 62.5KHz$$

(다) Fast PWM 모드

이 모드는 그림 8.1과 같은 비대칭형 PWM신호를 만들어 낸다.

☞ TCCRn레지스터의 WGMn1:0=3으로 설정하는 동작모드이다.
☞ 타이머/카운터 값 TCNTn은 항상 증가를 하며 값이 8비트의 최대값(MAX=0xFF)을 지나면 다시 바닥(BOTTOM=0x00)부터 다시 카운트를 시작한다. 이 때 오버플로 플래그 TOVn이 세트되고 OCRn값이 업데이트된다. 타이머/카운터값이 PWM생성을 위한 톱니파를 형성한다.
☞ 타이머/카운터 값 TCNTn는 출력비교 레지스터 OCRn값과 일치될 때와 TCNTn이 0이 될 때 COMn1:0값의 설정(표 8.3)에 따라 핀 OCn에 출력을 한다. 그리고 비교 일치 인터럽트 플래그 OCFn이 세트된다.

그림 8.5는 Fast PWM 모드에서 출력 핀의 동작을 보여준다.

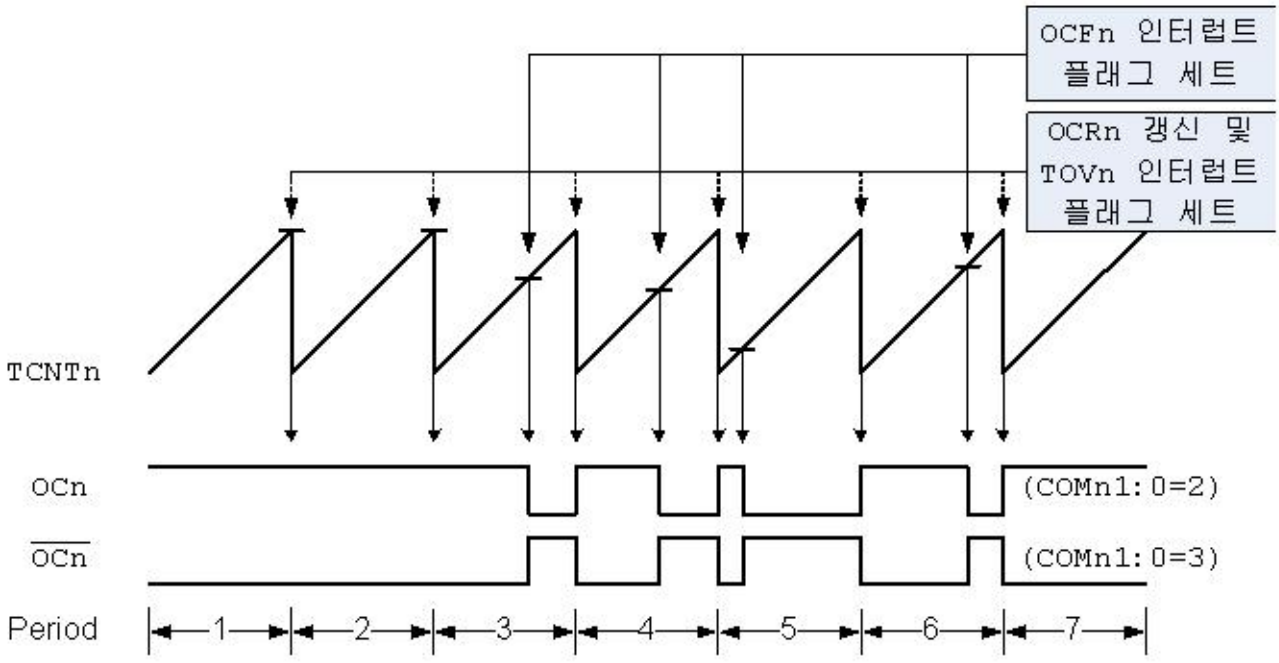

그림 8.5 Fast PWM 모드의 동작

☞ TCNTn의 값이 0으로 리셋되는 시점이 PWM 신호의 시작시점이므로 PWM신호의 반송주기는 카운터의 값이 0부터 다음 0까지이므로 256 카운트에 해당한다. 따라서 반송주기와 주파수는

$$T_{OCnPWM} = 256 \cdot t_t = 256 \cdot N / f_{clk\_I/O} \quad sec \qquad (8.5)$$

$$f_{OCnPWM} = 1 / T_{OCnPWM} = f_{clk\_I/O} / (256 \cdot N) \quad Hz \qquad (8.6)$$

☞ PWM 신호의 듀티비는 출력 비교레지스터 값으로 설정할 수 있다.

$$\text{듀티비}(\%) = OCRn / 256 \times 100\ (\%)$$

☞ FAST PWM모드의 해상도는 8bit 즉 256등분이다.

(라) Phase Correct PWM 모드

이 모드는 그림 8.2와 같은 대칭형 PWM신호를 만들어 낸다. 신호의 대칭특성으로 모터의 구동에 적합하다.

☞ TCCRn레지스터의 WGMn1:0=1로 설정하는 동작모드이다.

☞ 타이머/카운터 값 TCNTn은 최대값(MAX=0xFF)값까지 증가하였다가 BOTTOM값 감소하며 삼각파를 형성한다. 타이머/카운터값이 0이될 때 오버플로 플래그 TOVn이 세트된다. OCRn값은 TCNTn이 최대값이 될 때 업데이트가 된다.

☞ 타이머/카운터 값 TCNTn는 출력비교 레지스터 OCRn값과 일치될 때 증가모드와 감소모드에서 COMn1:0값의 설정(표8.4)에 따라 핀 OCn에 출력을 한다.

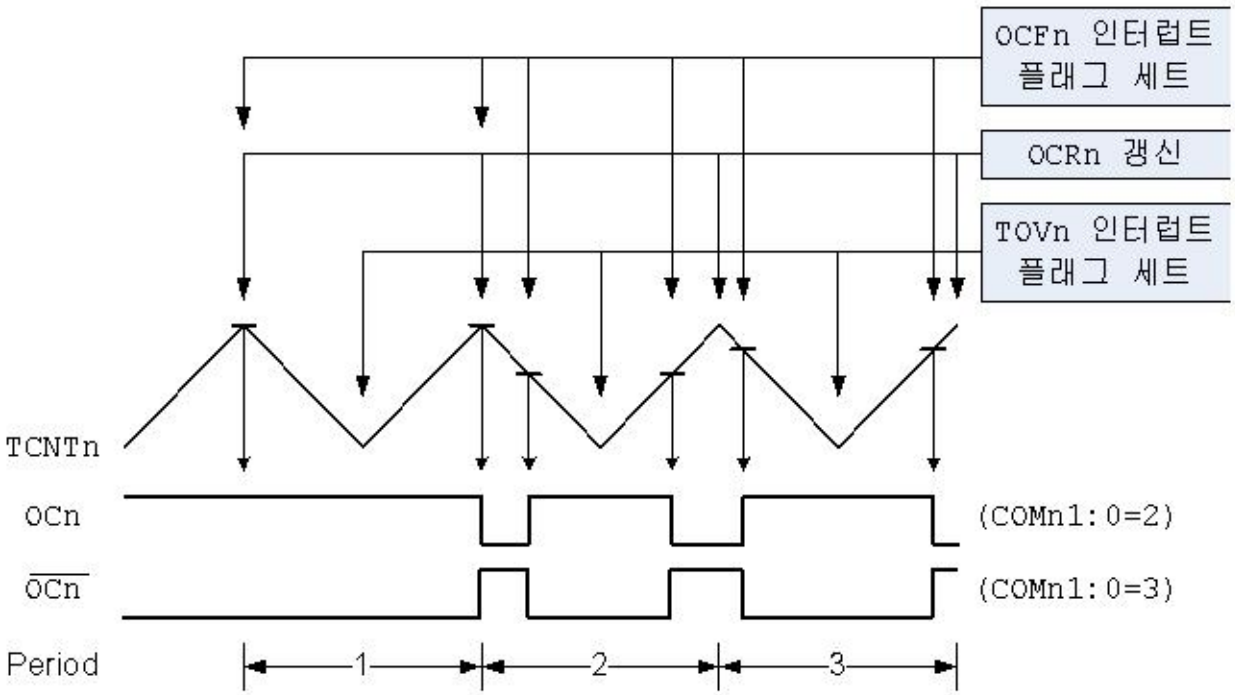

그림 8.6 Phase Correct PWM 모드의 동작

그림 8.6은 Phase correct PWM 모드에서 출력 핀의 동작을 보여준다.

☞ TCNTn의 값이 MAX값이 되는 시점이 PWM 신호의 시작시점이므로 PWM신호의 반송주기동안 카운터의 값은 0xFF(255)->0x00->0xFF으로 변한다. 카운트로는 510에 해당한다. 따라서 반송주기와 주파수는

$$T_{OCnPWM} = 510 \cdot t_t = 510 \cdot N / f_{clk\_I/O} \quad sec \qquad (8.7)$$

$$f_{OCnPWM} = 1 / T_{OCnPWM} = f_{clk\_I/O} / (510 \cdot N) \quad Hz \qquad (8.8)$$

☞ PWM 신호의 듀티비는 출력 비교레지스터 값으로 설정할 수 있다.

$$\text{듀티비}(\%) = OCRn / 256 \times 100\ (\%)$$

☞ Phase Correct PWM모드의 해상도는 8bit 즉 256등분이다.

☞ Phase Correct PWM모드의 PWM신호 반송주기는 Fast PWM모드의 약 두 배이다.

8.4 8비트 타이머/카운터 실습

8.4.1 주기적으로 LED 패턴 이동하기

여기서는 6.4절에서 수행하였던 LED 패턴이동 실습을 타이머/카운터를 사용하여 수행하여 보자. 6.4절에서는 시간지연을 msec_delay()함수 또는 _delay_ms()함수를 사용하였으나 이 절에서는 정확한 시간지연을 위해 타이머/카운터를 사용한다. LED회로는 그림 6.2를 사용한다.

다음 방법을 통해 LED 패턴을 주기적으로 이동하도록 하자.

☞ 주기적으로 인터럽트를 발생시키고, 인터럽트서비스루틴에서 LED 패턴을 이동시킨다.
☞ 주기적으로 인터럽트를 발생시키는 방법은 타이머 오버플로 인터럽트와 출력 비교 일치 인터럽트를 사용할 수 있다.
☞ 표준 모드에서는 타이머 오버플로 인터럽트와 출력 비교 일치 인터럽트 중 하나를 사용하면 되고, CTC모드에서는 비교 일치 인터럽트를 사용한다.

우선 각 프리스케일러의 분주비에 따라 인터럽트의 최대 발생주기를 계산하여 보자.

프리스케일러 출력 클록 당 소요시간은 식(8.2)로서 주어진다.

$$t_t = N / f_{clk\_I/O}$$

여기서 N은 프리스케일러의 분주비이고 $f_{clk_I/O}$는 내부클록 주파수이다. 8비트 레지스터인 TCNTn레지스터 또는 OCRn은 0~255까지 셀 수 있으므로 최대 주기는

$$T_{max} = 256 \cdot t_t = 256 \cdot N / f_{clk\_I/O} \qquad (8.9)$$

이다. 표8.7은 16MHz클록을 사용할 때 프리스케일러 분주비에 따른 인터럽트 최대 발생주기를 나타낸다.

표 8.7 분주비에 따른 인터럽트 최대 발생주기

분주비(N)	1	8	32†	64	128†	256	1024
최대주기 (msec)	0.016	0.128	0.512	1.024	2.048	4.096	16.384

†타이머/카운터 2는 제공하지 않는 분주비임

가장 큰 분주비를 선택하여도 최대주기가 약 16msec이다. 이 주기로 패턴을 이동하면 너무 빨라 모두 켜져 있는 것처럼 보인다. 느린 주기로 패턴을 이동하여 패턴이동을 보기 위해 다음 방법을 고려하자.

☞ 10msec 주기로 인터럽트를 건다.
☞ 인터럽트 서비스루틴에서 인터럽트가 걸린 회수를 계산하여 긴 시간을 측정하고 시간이 되면 패턴을 이동시킨다. (예: 200msec주기로 패턴을 이동하려면 인터럽트가 20번 걸렸을 때 패턴을 이동하면 된다.)

우선 주어진 시간간격으로 인터럽트를 거는 방법을 생각해보자.

(가) 표준모드에서 타이머/카운터 오버플로 인터럽트를 이용하는 방법

타이머에 적절한 초기 값을 쓰면 타이머 시작부터 타이머 오버플로가 발생할 때까지의 시간을 조절할 수 있다.

(예) 타이머에 0xF0를 쓰고 타이머를 시작하면 0xFF에서 0x00으로 넘어가는 시점에서 오버플로 인터럽트가 발생하므로

0xFF - 0xF0 + 1 = 0x10 = 16

클록 후에 타이머 오버플로 인터럽트가 발생한다.

(예) 16클록 후에 타이머 오버플로 인터럽트를 걸려면 타이머에

0xFF - 16 + 1 = 0xF0

를 설정하고 타이머를 동작시키면 된다.

T_s msec후에 인터럽트를 발생시키려면 T_s msec에 해당하는 클록 수를 계산하여야 한다. 이를 변수 icount라 하면

$$\begin{aligned} \text{icount} &= T_s \times 10^{-3} / t_t \quad \text{<== 식 (8.2)} \\ &= T_s \times 10^{-3} / (N / f_{clk\_I/O}) \\ &= T_s \times 10^{-3} \times 16 \times 10^{6} / N \\ &= T_s \times 16 \times 1000 / N \quad (8.10) \end{aligned}$$

여기서 N은 분주비이고 $f_{clk\_I/O}$=16MHz이다. 따라서 타이머에

$$\text{TCNTn} = 0\text{xFF} - \text{icount} + 1 = 256 - \text{icount} \quad (8.11)$$

을 설정하고 타이머를 동작시키면 T_s msec 후에 인터럽트가 발생한다.

프로그램 8.1은 10msec마다 타이머/카운터 0 오버플로 인터럽트를 걸고 인터럽트 횟수가 20일 때 즉 200msec가 되었을 때 LED 패턴을 이동시키는 프로그램이다.

```
#include <avr/io.h>
#include <avr/interrupt.h>
static unsigned char pattern[8]   // LED 패턴 테이블
     = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};

ISR(TIMER0_OVF_vect) // 타이머0 오버플로 인터럽트 서비스루틴
{
     static int  index=0;     // 패턴 인덱스
     static char n_enter=0;   // 인터럽트 횟수

     TCNT0 = 100;             // 10msec 후에 인터럽트발생
     n_enter++;               // 인터럽트 횟수 증가

     if(n_enter == 20)        // 200msec
     {
          n_enter=0;
          PORTA = pattern[index++];    // 패턴이동
          if(index==8) index=0;
     }
}
int main()
{
     DDRA  = 0xFF;        // A 포트를 출력으로 설정
     PORTA = 0xFF;        // LED를 끈다.

     // 타이머/카운터 0 설정
     TCCR0 = 0x00;        // 표준모드, 타이머 정지
     TCNT0 = 100;         // 타이머 초기 값 설정

     // 인터럽트 설정
     TIMSK = (1<<TOIE0); // 타이머0 오버플로 인터럽트 허용
     sei();               // 전역 인터럽트 허용

     TCCR0 |= (7<<CS00); // 분주비 1024로 타이머 시작

     while(1);
}
```

프로그램 8.1 오버플로 인터럽트를 사용한 주기적 LED 패턴이동

☞ 타이머/카운터 0 오버플로 인터럽트의 매크로는 표7.2로부터 TIMER0_OVF_vect이다.

☞ 분주비 1024를 선택하였을 때 10msec후에 인터럽트를 걸려면 타이머 0에 설정하여야 할 값은 식(8.10)과 (8.11)로부터 다음과 같다.

식(8.10) ==> icount = 10 × 16 × 1000 / 1024 = 156

식(8.11) ==> TCNT0 = 256 - 156 = 100

☞ 타이머 오버플로 인터럽트가 20번 발생할 때 LED 패턴을 이동시키면 200msec마다 패턴이 이동하게 된다.

☞ 타이머/카운터0을 표준모드, OC0핀을 사용하지 않을 때 타이머 제어 레지스터 TCCR0의 각 비트는

WGM01 =0, WGM00 =0, COM01:0 =0, CS02:0 =0(타이머 정지)

==> TCCR0 = 0x00;

☞ 타이머0 오버플로 인터럽트를 허용한다. ==> TIMSK = (1<<TOIE0)

☞ 분주비 1024로 타이머를 동작시키려면 CS02:0 = 7로 설정하면 된다. TCCR0의 CS02:0핀들만 0x07로 변경하여야 하므로

==> TCCR0 |= (7<<CS00); // TCCR0 = (7<<CS00)이 아님

☞ 그림 8.7과 같이 인터럽트 서비스루틴에서 다시 10msec후에 인터럽트를 발생시키므로 주기적으로 인터럽트가 걸린다.

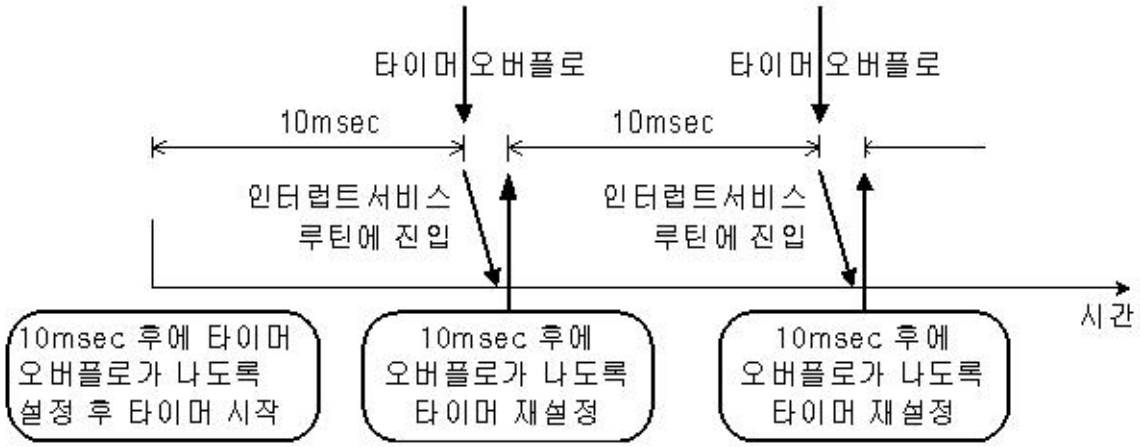

그림 8.7 오버플로를 이용하여 10msec 주기 인터럽트 걸기

☞ 그림 8.7에서 보여주듯 인터럽트가 걸린 후 인터럽트서비스 루틴에 진입하는 데 시간이 걸리므로 오버플로 발생 후 약간 시간이 지난 후 타이머 재설정이 이루어진다. 따라서 타이머 오버플로 사이의 시간 간격은 10msec보다 약간 길다.

(나) 표준모드에서 출력 비교 일치 인터럽트를 이용하는 방법

비교일치 인터럽트는 비교레지스터 OCR0과 타이머레지스터 TCNT0이 일치 할 때 발생한다. 그림 8.8은 비교일치 인터럽트를 사용하여 주기적 인터럽트를 발생시키는 방법을 보여주고 있다.

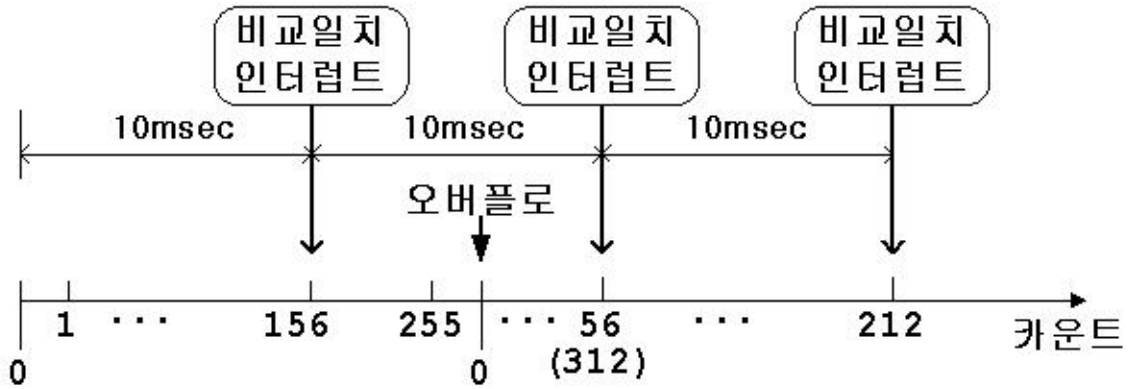

그림 8.8 오버플로를 이용하여 10msec 주기 인터럽트 걸기

☞ 타이머를 동작시키면 그림 8.8과 같이 8비트 타이머는 0->255까지 세고 다음에는 오버플로가 발생하면서 다시 0부터 값을 센다.

☞ 분주비 1024일 때 10msec에 해당하는 카운트는 156이므로

첫 인터럽트가 발생하는 TCNT0의 값 :

```
TCNT0 = 0 + 156(0x9C)
```

둘째 인터럽트가 발생하는 TCNT0의 값 :

```
TCNT0 = 156 + 156 = 312(0x138)
```

- 타이머0는 8비트 타이머이므로 255(0xFF)에서 오버플로가 발생하여 이 값은 312-256 = 56(0x38)이 된다.

셋째 인터럽트가 발생하는 TCNT0의 값 :

```
TCNT0 = 56 + 156 = 212(0xD4)
```

☞ 따라서 OCR0을 0x9C, 0x38, 0xD4 ... 순으로 설정하면 10msec마다 비교일치 인터럽트가 발생한다. 이는 다음과 같이 계산하면 된다.

```
OCR0 = OCR0 + 156;
```

TCNT0과 같이 두 번째 인터럽트에 대해서는 OCR0 = 0x138이 되나 OCR0은 8비트 레지스터이므로 하위바이트 0x38이 OCR0에 저장된다.

```
#include <avr/io.h>
#include <avr/interrupt.h>
static unsigned char pattern[8]   // LED 패턴 테이블
     = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
ISR(TIMER0_COMP_vect)          // 비교일치 인터럽트 서비스루틴
{
     static int  index=0;      // 패턴 인덱스
     static char n_enter=0;    // 인터럽트 횟수
     OCR0 += 156;              // 10msec 후에 인터럽트발생
     n_enter++;                // 인터럽트 회수 증가
     if(n_enter == 20)         // 200msec
     {
          n_enter=0;
          PORTA = pattern[index++];    // 패턴이동
          if(index==8) index=0;
     }
}
int main()
{
     DDRA  = 0xFF;         // A 포트를 출력으로 설정
     PORTA = 0xFF;         // LED를 끈다.
     // 타이머/카운터 0 설정
     TCCR0 = 0x00;         // 표준모드, 타이머 정지
     OCR0  = 156;          // 10msec후에 비교일치 발생
     // 인터럽트 설정
     TIMSK = (1<<OCIE0);// 타이머0 비교일치 인터럽트 허용
     sei();                // 전역 인터럽트 허용
     TCCR0 |= (7<<CS00); // 분주비 1024로 타이머 시작
     while(1);
}
```

프로그램 8.2 비교일치 인터럽트를 사용한 주기적 LED 패턴이동

프로그램 8.2는 비교일치인터럽트를 사용하여 LED패턴을 이동시킨다.

☞ 타이머0 비교일치 인터럽트의 매크로는 TIM0_COMP_vect이다.

☞ 타이머0 비교일치 인터럽트를 허용한다. ==> TIMSK = (1<<OCIE0)

☞ 프로그램 8.2는 그림 8.8에서 보여주듯 정확히 10msec마다 비교일치 인터럽트가 발생한다.

(다) CTC모드를 이용하는 방법

그림 8.4의 CTC모드를 설정하면 카운터값 TCNTn이 비교값 OCRn까지 증가하였다가 리셋되고 비교일치 인터럽트를 발생한다. 한번 비교값 OCRn을 설정하면 계속적으로 인터럽트가 발생한다.(방법 (나)에서는 인터럽트 서비스루틴에서 계속적으로 OCRn값을 재설정하여야 한다.)

여기서는 10msec주기로 비교 일치 인터럽트를 발생시키고 인터럽트 회수를 세어 20번(200msec)마다 LED패턴을 이동하도록 한다. 아울러 타이머0의 출력핀인 OC0(PB4와 공유)에 비교 일치 인터럽트가 발생할 때마다 상태가 반전되는 펄스를 출력하여 보자.

☞ 1024 분주비를 사용하였을 때 10msec는 156 카운트에 해당한다.

```
        ==> OCR0 = 156
```

☞ CTC 모드의 설정 : WGM01=1, WGM00=0
출력핀 OC0를 반전모드로 설정: COM01:0 = 0x01
타이머 정지: CS02:0 = 0x00

```
   ==> TCCR0 = (1<<WGM01)|(1<<COM00); // (=0b00011000)
```

☞ OC0(PB4)핀을 출력으로 설정: PB4를 제외한 PORTB의 핀은 방향을 보존한다.

```
        ==> DDRB |= (1<<DDB4);  // (=0b00010000)
```

☞ 타이머0 비교 일치 인터럽트 허용

```
        ==> TIMSK = (1<<OCIE0); // (=0b00000010)
```

☞ 타이머를 1024분주비로 시작: CS02:1 = 0x07
TCCR0의 CS02:0핀들만 0x07로 변경하여야 하므로

```
        ==> TCCR0 |= 0x07; // TCCR0 = 0x07이 아님
```

프로그램 8.3은 CTC모드로 LED 패턴을 200msec 주기로 이동시키는 프로그램이다. 출력 핀 OC0를 CTC모드와 연계시켰으므로 OC0핀에 인터럽트 주기의 두 배인 20msec의 주기를 갖는 펄스가 생성된다.

실습 : 프로그램 8.1, 8.2, 8.3을 수행한다. 단 프로그램 8.3을 수행할 때는 OC0(PB4)핀의 출력을 오실로스코프로 관찰한다.

과제 : prac6-7.c는 main()에서 주어진 시간간격으로 도트매트릭스의 행을 바꾸어가면서 주어진 LED패턴을 켠다. 이를 5msec마다 타이머 인터럽트 서비스루틴에서 행을 바꾸며 LED패턴 변경하도록 변경하라.

```
#include <avr/io.h>
#include <avr/interrupt.h>
static unsigned char pattern[8]   // LED 패턴 테이블
     = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
ISR(TIMER0_COMP_vect)          // 비교일치 인터럽트 서비스루틴
{
     static int  index=0;      // 패턴 인덱스
     static char n_enter=0;    // 인터럽트 횟수

     n_enter++;                // 인터럽트 횟수 증가
     if(n_enter == 20)         // 200msec
     {
          n_enter=0;
          PORTA = pattern[index++];    // 패턴이동
          if(index==8) index=0;
     }
}
int main()
{
     DDRA  = 0xFF;         // A 포트를 출력으로 설정
     DDRB |= (1<<DDB4);    // PB4(OC0)를 출력으로 설정
     PORTA = 0xFF;         // LED를 끈다.
     // 타이머/카운터 0 설정 : CTC 모드, 타이머 정지, OC0핀 사용
     TCCR0 = (1<<WGM01)|(1<<COM00);
     OCR0  = 156;          // 10msec마다 비교일치 발생
     // 인터럽트 설정
     TIMSK = (1<<OCIE0);// 타이머0 비교일치 인터럽트 허용
     sei();                // 전역 인터럽트 허용

     TCCR0 |= (7<<CS00); // 분주비 1024로 타이머 시작
     while(1);
}
```

프로그램 8.3 CTC모드를 사용한 주기적 LED 패턴이동

8.4.2 PWM신호로 LED 밝기 조절하기

8.2절에서 언급한 바와 같이 PWM신호의 듀티비로 조절하면 DC신호의 평균값을 조절할 수 있으므로 LED를 PWM신호로 구동을 하면 밝기를 조절할 수 있다. 실습을 위하여 그림 8.9와 같이 OC0핀에 LED를 연결한다.

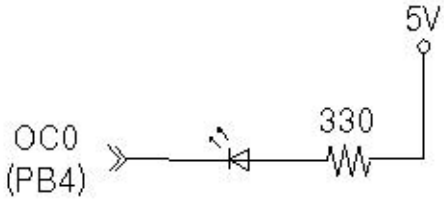

그림 8.9 PWM 출력에 LED연결

표 8.8은 분주비에 따른 각 PWM 모드의 주파수 및 주기를 보여준다.

표 8.8 분주비 따른 PWM신호의 주파수 및 주기

분주비(N)		1	8	32†	64	128†	256	1024
Fast PWM	주파수(KHz)	62.5	7.8125	1.9531	0.9766	0.4883	0.2441	0.0610
	주기(msec)	0.016	0.128	0.512	1.024	2.048	4.096	16.384
P.C PWM	주파수(KHz)	31.348	3.9216	0.9804	0.4902	0.2451	0.1225	0.0306
	주기(msec)	0.0319	0.255	1.020	2.040	4.080	8.160	32.64

†타이머/카운터 2는 제공하지 않는 분주비임

프로그램 prac8-4.c는 분주비 8을 사용하여 FAST PWM모드로 OC0(PB4)에 연결된 LED의 밝기를 조절하는 프로그램이다. PWM신호의 듀티비는 외부인터럽트 INT0과 INT1을 사용하여 증가/감소시킨다.

☞ FAST PWM 모드의 설정 : WGM01=1, WGM00=1
출력핀 OC0의 모드 설정 : COM01:0 = 0x02
타이머 정지 : CS02:0 = 0x00
==> TCCR0=(1<<WGM01)|(1<<WGM00)|(2<<COM00);
// (=0b01101000)

☞ OC0(PB4)핀을 출력으로 설정: PB4를 제외한 PORTB의 핀은 방향을 보존한다.

==> DDRB |= (1<<DDB4); // (=0b00010000)

☞ 타이머를 분주비 8로 시작: CS02:0 = 0x02
TCCR0의 CS02:0핀들만 0x02로 변경하여야 하므로
==> TCCR0 |= (2<<CS00); // TCCR0 = (2<<CS00)이 아님

☞ 듀티비는 변수 duty로 표시하며 외부 인터럽트서비스루틴에서 변경한다. 이 값을 OCR0에 설정하면 듀티비가 변한다.

과제 :

1. prac8-4.c를 수행하면서 스위치 입력에 따라 변화되는 OC0핀 출력의 PWM 듀티비를 오실로스코프로 관찰하고 아울러 연결된 LED의 밝기를 관찰하라.
2. 출력핀 OC0의 모드 설정을 COM01:0 = 0x03이도록 prac8-4.c를 변경하고 스위치 입력에 따른 PWM신호와 LED의 밝기를 관찰하라.
3. Phase Correct PWM모드로 프로그램 prac8-4.c를 수정하고 과제 1을 수행하라.
4. 그림 8.8에서 LED 대신 소형 DC모터를 그림 8.10과 같이 연결하고 prac8-4.c를 수행하면서 스위치 입력에 따른 DC모터의 속도를 관찰하라.
   - NPN 트랜지스터가 스위치 역할을 한다.
   - NPN 트랜지스터 스위치는 입력 신호가 5V일 때 ON이 되고 입력신호가 0V일 때 OFF가된다.
   - 모터의 전원은 외부전원을 인가하고, 외부전원의 접지와 보드의 접지를 연결한다.

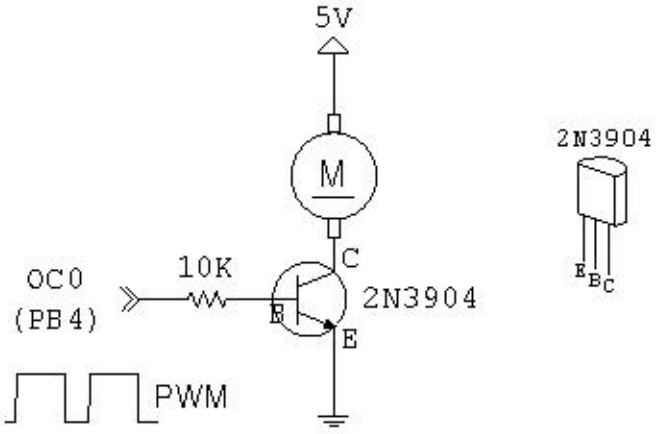

그림 8.10 소형 DC모터 연결

[프로그램 prac8-4.c]

```
//=====================================================
// 실습 8-4 : PWM 신호로 LED 밝기 조절
//=====================================================

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define DEBOUNCING_DELAY     20   // msec
void msec_delay(int n);           // 시간지연함수
static volatile unsigned char duty = 125;   // 초기 듀티비

ISR(INT0_vect)          // 외부인터럽트 0 서비스루틴
{
     duty += 5;                   // 듀티비 5 증가
     if(duty >= 250) duty = 250;

     msec_delay(DEBOUNCING_DELAY);
     while(~PIND & 0x01);     // 스위치 해제를 기다림
     msec_delay(DEBOUNCING_DELAY);
     EIFR = (1<<INTF0);       // INT0 인터럽트 플래그 리셋
}

ISR(INT1_vect)          // 외부인터럽트 1 서비스루틴
{
     duty -= 5;                   // 듀티비 5 감소
     if(duty <= 5) duty = 5;

     msec_delay(DEBOUNCING_DELAY);
     while(~PIND & 0x02);     // 스위치 해제를 기다림
     msec_delay(DEBOUNCING_DELAY);
     EIFR = (1<<INTF1);       // INT1 인터럽트 플래그 리셋
}

int main()
{
     DDRB |= (1<<DDB4);       // PB4(OC0)를 출력으로 설정
```

```
    // 타이머/카운터 0 설정, FAST PWM 모드, 타이머 정지
    TCCR0 = (1<<WGM01)|(1<<WGM00)|(2<<COM00);

    // 인터럽트 설정,ISC11:0=2, ISC01:0=2(하강모서리)
    EICRA = (2<<ISC10)|(2<<ISC00);
    EIMSK = (1<<INT0) |(1<<INT1);// 외부인터럽트0,1 허용

    sei();                          // 전역 인터럽트 허용

    TCCR0 |= (0x02<<CS00);          // 분주비 8로 타이머 시작

    while(1)
    {
        OCR0 = duty;   // PWM 신호의 듀티비를 바꾼다.
    }
}

// 시간지연함수
//===========================================

void msec_delay(int n)
{
    for(; n >0; n--)          // 1msec 시간지연을 n회 반복
        _delay_ms(1);         // 1msec 시간지연
}
```