AVR - HW5

임베디드스쿨1기

Lv1과정

2020. 10. 16

김인겸

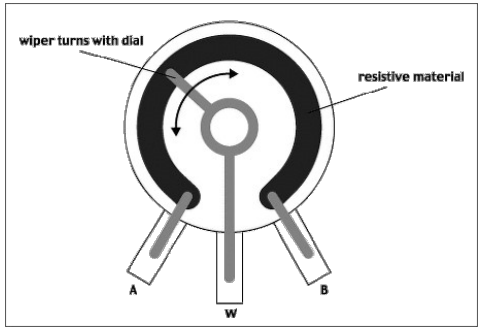
ADC복습(기준전압)

1. ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADC에서 사용하는 기준 전압 선택

REFS1	REFS2	
0	0	AREF의 입력전압 사용
0	1	AVCC의 입력전압 사용
1	0	예약됨
1	1	내부 1.1V 사용



Vcc (5V) 입력핀 GND

가변 저항은 0~5V사이의 값을 갖는데 ADC의 기준전압이 5V가 아닌 값을 넣어주면 변환값이 짤릴 수도 있음.

가변저항은 0~5V로 변하는데 기준전압이 3.3V면 문제가 발생.

변하는 값

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

레지스터 값 기준전압

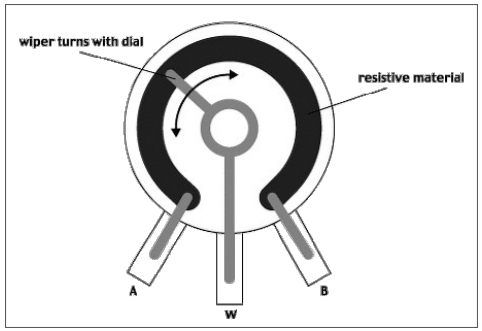
ADC복습(기준전압)

AREF : ADC가 디지털 변환을 수행할 때의 기준전압(신호 입력)
AVCC : 전원단자

ADC사용 시에 기준전압을 AVCC(전원단자)에서 사용해도 되지만 부하 회로의 영향으로 불안정할 수도 있다
→ 따라서 AREF 단자에 안정적인 신호를 인가하면 안정된 기준 전압을 보장받을 수 있다.

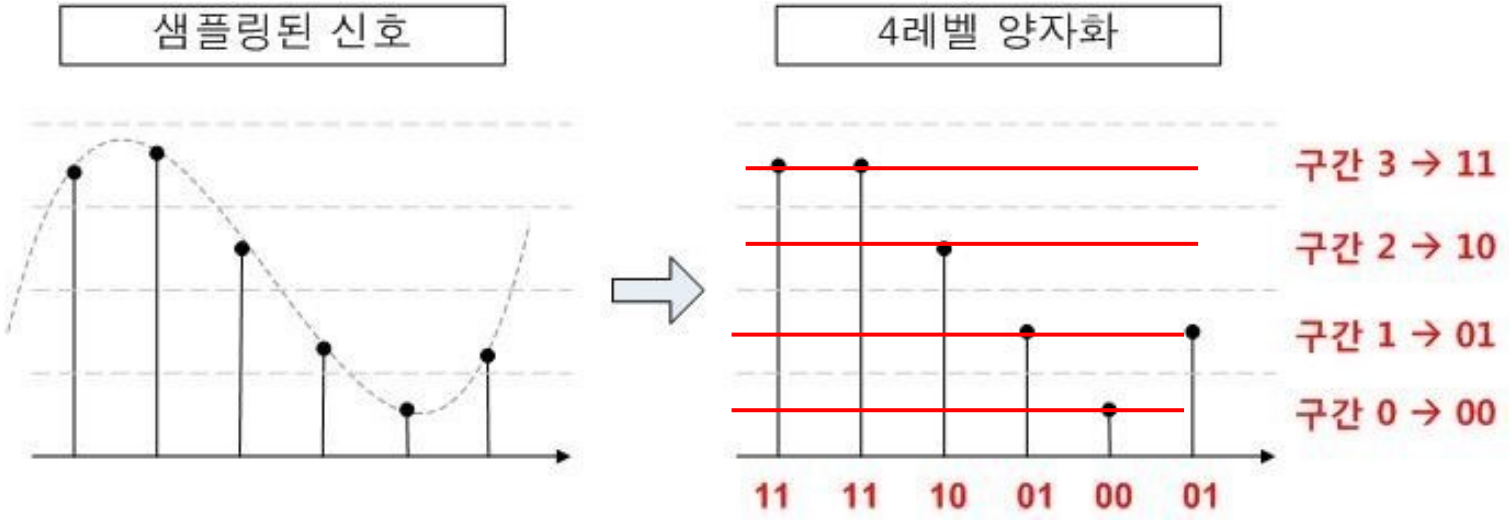
Q. ADMUX레지스터의 REFS0을 1로 설정했을 때 AVCC는 어디서 오는걸까?

옆의 그림의 아날로그 입력 핀이 5V라서 AVCC가 5V인 건가?
vcc가 5V라서 AVCC가 5V인 건가?..



Vcc 입력핀 GND
(5V)

ADC복습



분해능이 1024 → 위 그림의 빨간색 줄이 1024개(0~1023) 까지 있다는 의미
ADC가 양자화 및 부호화를 거쳐 그 값이 ADC레지스터에 값이 저장됨.

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

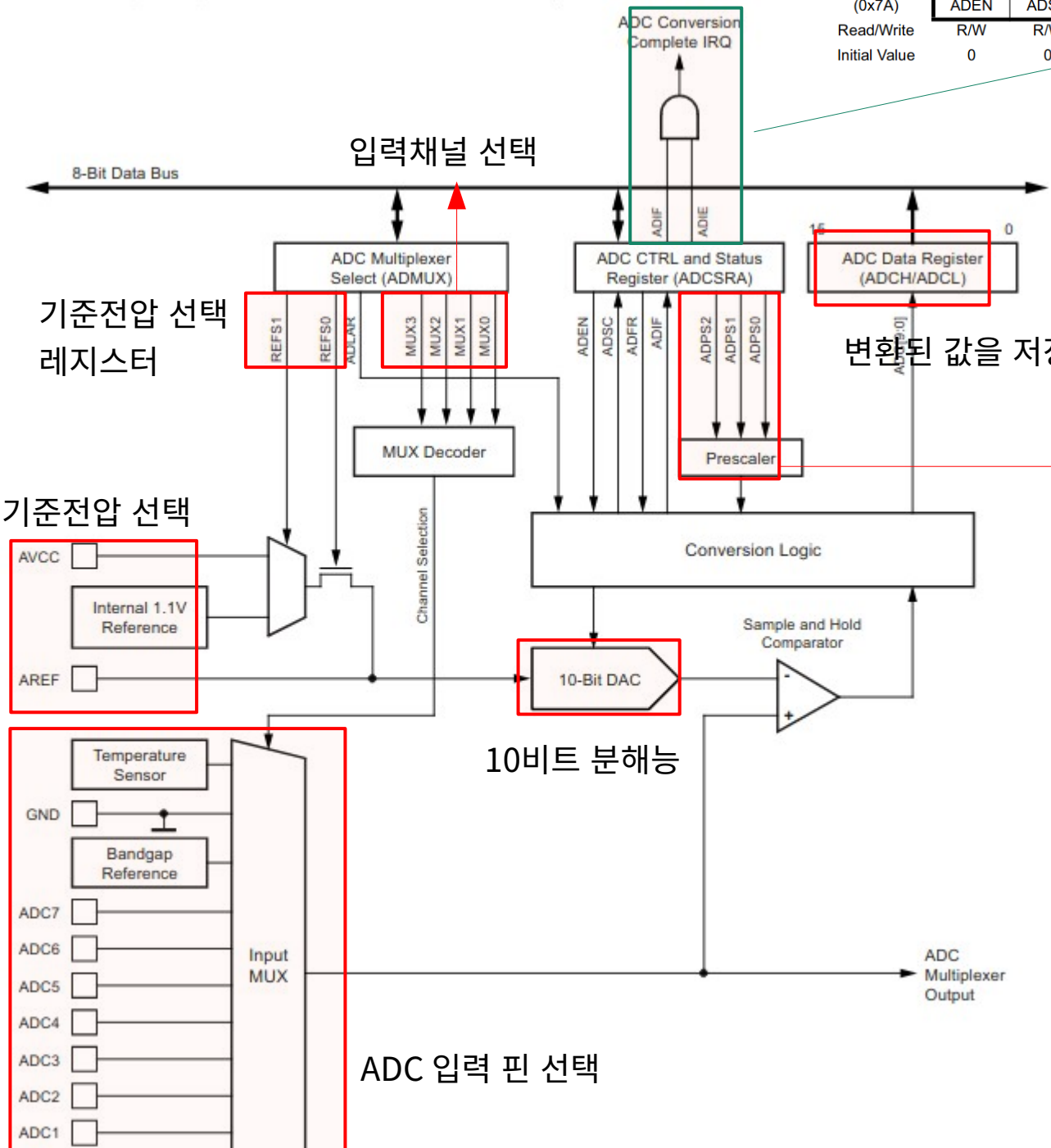
ADC복습

ADCSRA – ADC Control and Status Register A

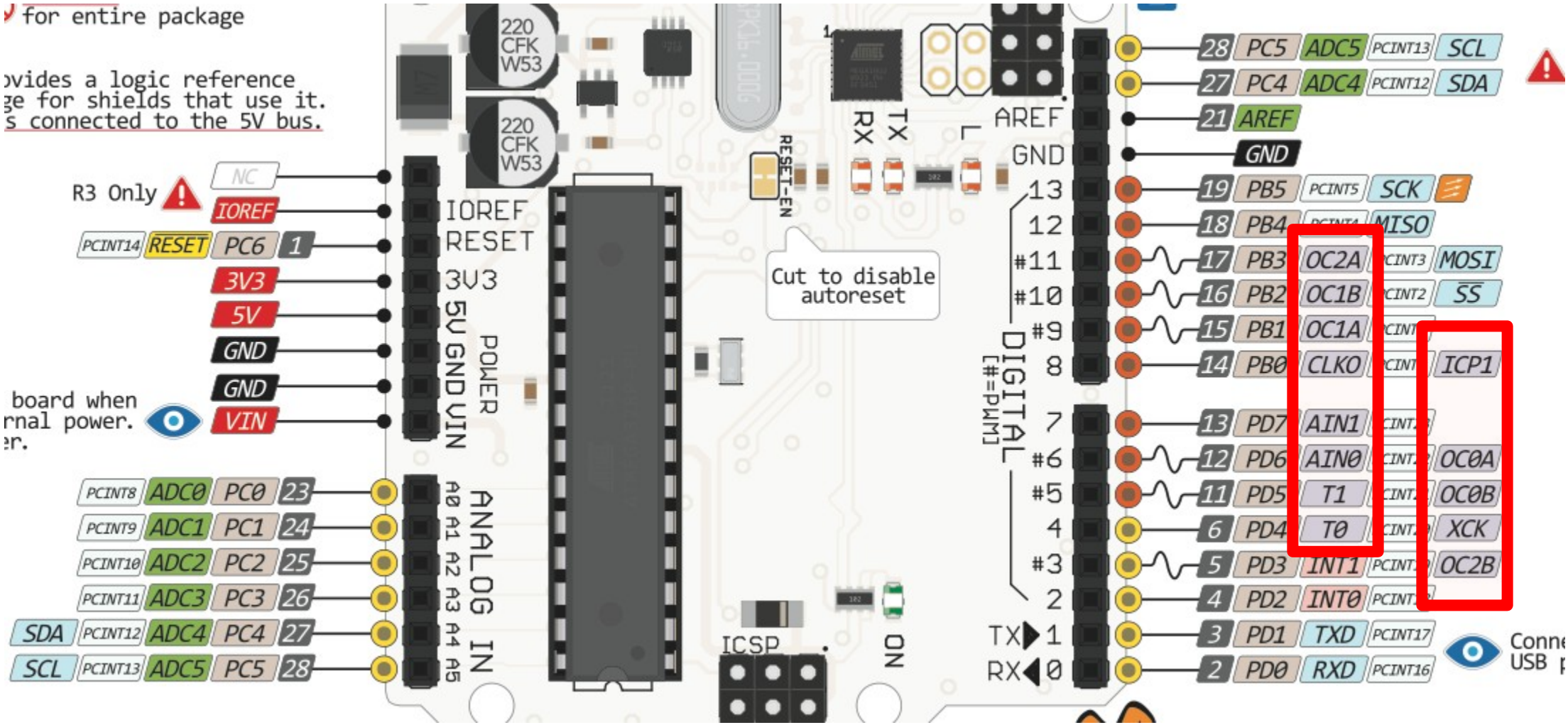
Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADIF : 변환 완료되면 1로 설정
ADIE : 인터럽트 허용 비트(init할 때 설정함)
위 두 개 비트가 AND게이트로 연결되어 있음
1 1 → 1 이 되면서 ADC변환완료 인터럽트실행

Figure 23-1. Analog to Digital Converter Block Schematic Operation



Timer/Counter 관련 아두이노 우노 핀맵



Timer/Counter0, 2

Timer : 입력되는 파형이 시스템 클럭에서 들어옴
Counter : 입력되는 파형이 외부 클럭에서 들어옴

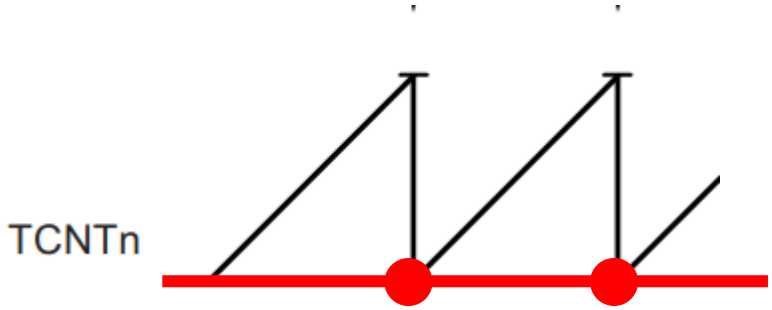
인터럽트
1. 타이머 오버플로우(Timer Overflow) 인터럽트
2. 출력 비교 일치(Output Compare Match) 인터럽트

Timer/Counter종류(Atmega128) (*Atemga328p는 0,1,2 이렇게 세 개 있다.)
- Timer/Counter0, Timer/Counter2 (8비트)
  0x00~0xFF까지 카운트
  Timer/Counter0은 외부 클럭 입력 가능, Timer/Counter2는 불가능.

- Timer/Counter1, Timer/Counter3 (16비트)
  0x00~0xFFFF까지 카운트

Timer/Counter0, 2 동작모드

1. 일반 모드(Normal)
: TCNT 값이 MAX → BOTTOM될 때 오버플로우 인터럽트 발생.

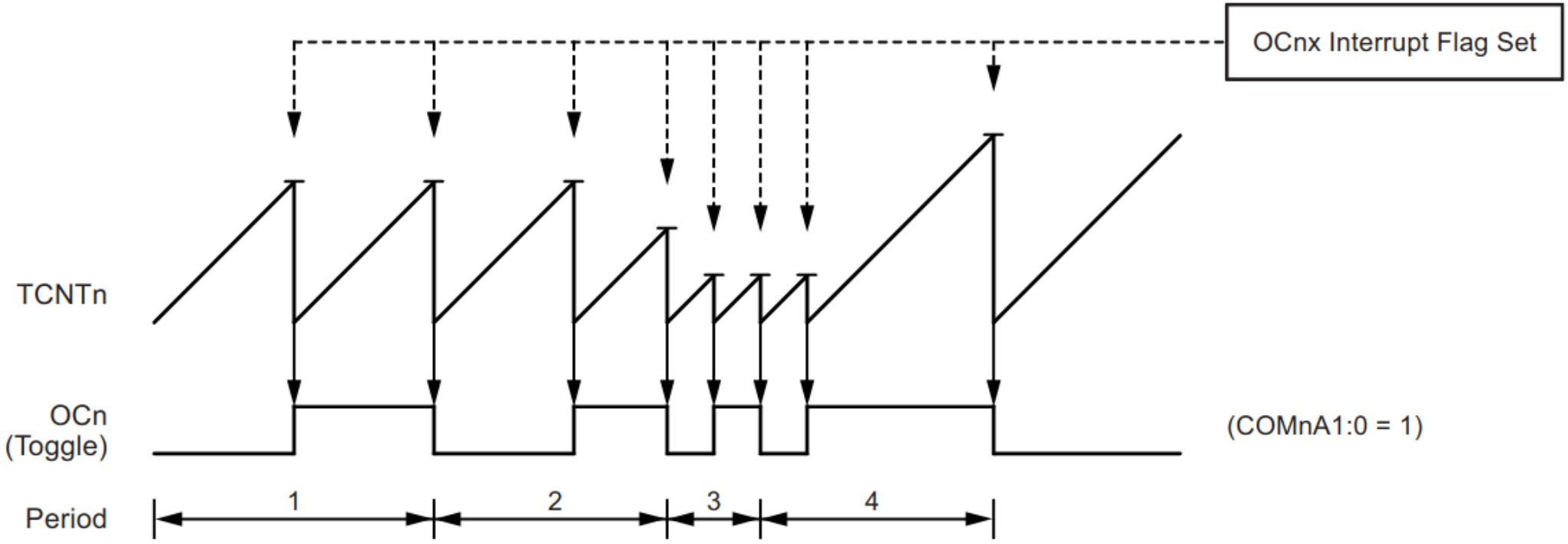


BOTTOM으로 떨어지고 +1클럭
후에 인터럽트 발생.

Timer/Counter0, 2 동작모드

2. CTC모드 : OCR값을 설정하고 TCNT값이 OCR값 + 1에 도달하면 인터럽트 발생.

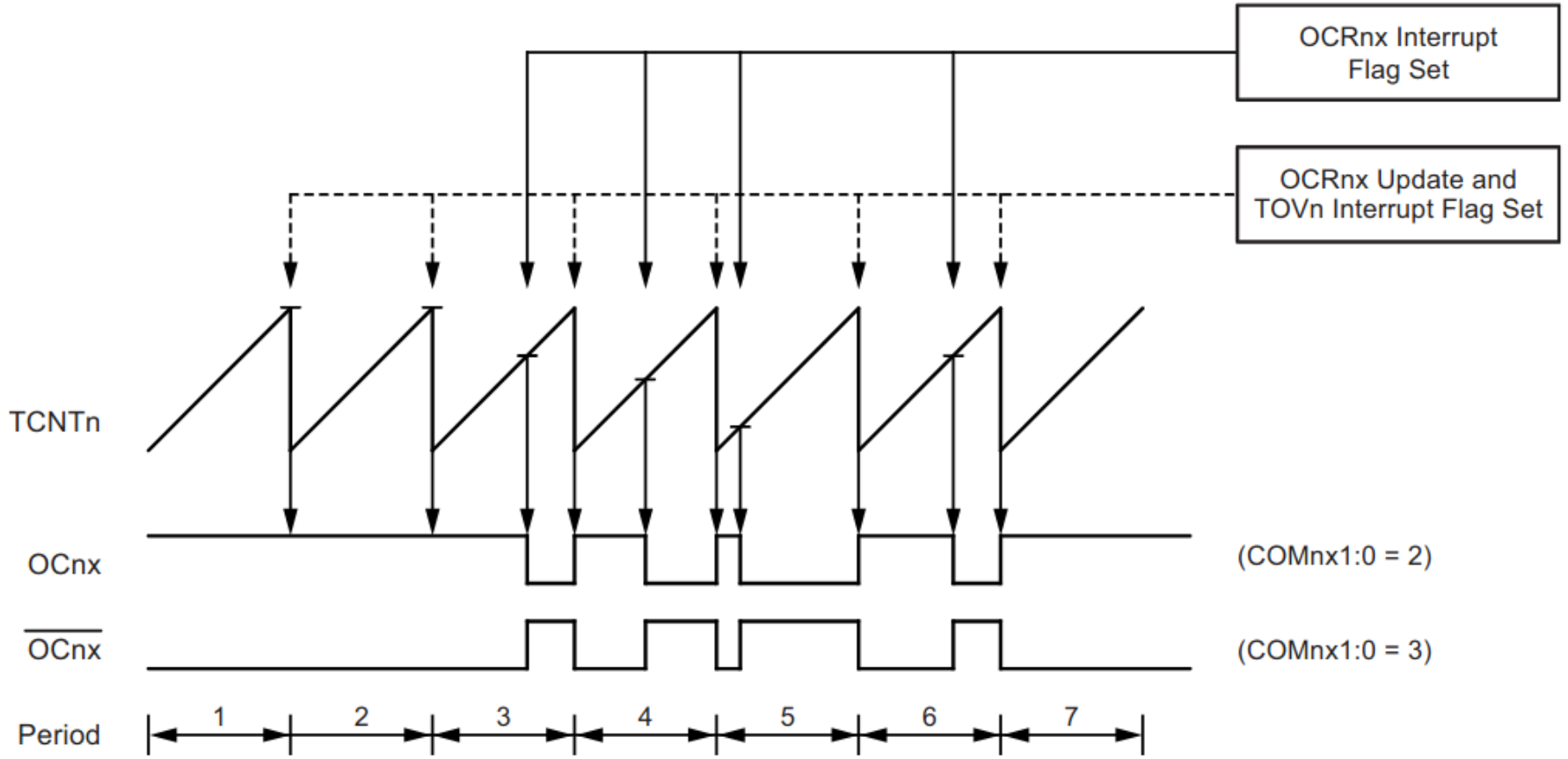
Figure 14-5. CTC Mode, Timing Diagram



Timer/Counter0, 2 동작모드

3. 고속PWM모드 : TCNT값이 OCR값과 같아지면 비교일치 인터럽트 발생 → OC는 0또는 1로
TCNT값이 0이 되면 인터럽트 발생 → OC는 1또는 0으로
(TCCR의 COM비트 설정이 필요함.)

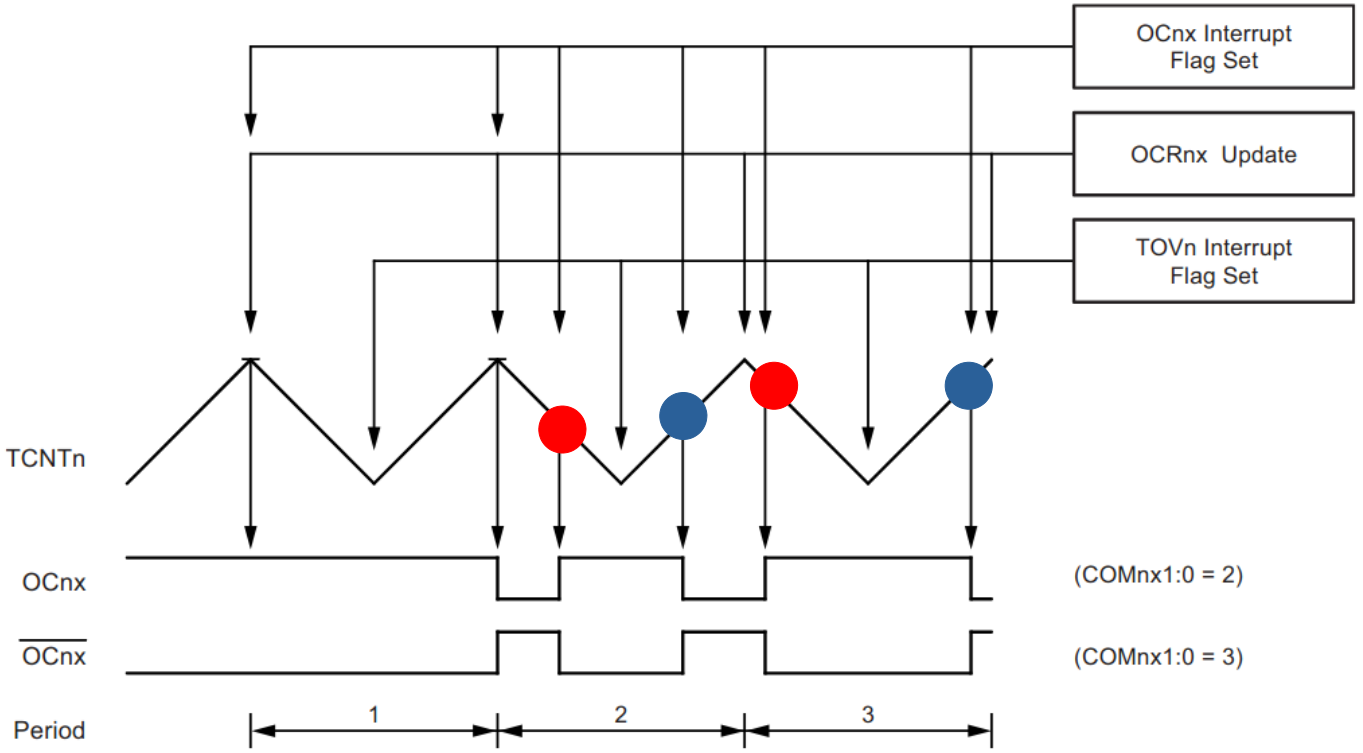
Figure 14-6. Fast PWM Mode, Timing Diagram



Timer/Counter0, 2 동작모드

4. Phase Correct PWM모드
TCNT값이 증가하다가 OCR과 같아지면 비교일치 인터럽트발생 → OC는 0또는 1
TCNT값이 감소하다가 OCR과 같아지면 비교일치 인터럽트발생 → OC는 1또는 0
(TCCR의 COM비트 설정이 필요함.)

Figure 14-7. Phase Correct PWM Mode, Timing Diagram



상승엣지

하강엣지

Timer/Counter0, 2 레지스터

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

비교일치 인터럽트 발생 시 set할지 clear할지 결정.

모드 선택

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF

Timer/Counter0, 2 레지스터

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter 클럭 분주비 선택

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

ex)
CPU클럭 : 16MHz
분주비 : 1024 일때

Timer 클럭 주파수 : 16M/1024 = 15.625KHz
Timer 클럭 주기 = 64us

0~255까지 셀 수 있는 8비트 타이머는
1클럭 당 64us의 주기를 가지므로
16.384ms까지의 시간을 측정할 수 있다.

Timer/Counter0, 2 레지스터

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable

  Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable
- Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable

Timer/Counter0, 2 레지스터

TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

타이머 카운터의 값이 들어감

0~255

OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

비교 일치 값을 설정

OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

비교 일치 값을 설정

volatile

선언방법

```
volatile int vi = 1;
volatile int* pvi = &vi;

int i1 = vi;

// ...

int i2 = vi;
```

volatile

1. 최적화에서 제외하기 위해서 사용.

최적화란? 컴파일러가 시스템의 효율을 위해서 코드를 간단하게 조정하는 과정.

ex1)

```
static int foo;

void bar(void)
{
    foo = 0;

    while (foo != 255);
}
```

```
void bar_optimized(void)
{
    foo = 0;

    while (true);
}
```

ex2)

```
int i = 0;

while (i < 10)
    i++;

printf("%d\n", i);
```

```
int i = 10;     // 반복문을 없애버리고 10을 할당

printf("%d\n", i);     // 10
```

volatile

예시 1을 봐보자

만약 foo변수가 하드웨어에 의해서 값이 달라지는 레지스터 값이라면 다른 동작에 의해 값이 변경될 수 있다.

그런데 1번처럼 최적화를 하면 foo변수의 변화를 코드에 적용시킬 수 없는 문제점이 발생한다.

따라서 인터럽트 및 주변 I/O장치와 밀접하게 관련있는 임베디드 시스템을 설계할 때 volatile을 많이 쓴다.

volatile

2. 캐쉬 메모리 사용 금지하고 항상 메모리에서 접근

프로그램에서 자주 쓰이는 변수는 캐쉬 메모리에 저장돼서 속도를 빠르게 한다.

두 개 이상의 프로세서가 동일한 변수에 접근할 때 그 변수가 메모리에도 있고 캐쉬에도 있으면 원하지 않는 값으로 뒤바뀌는 현상이 발생할 수 있다.

따라서 volatile을 사용함으로써 변수를 캐쉬에 저장하는 작업을 금지하고 오직 메모리에서만 접근하도록 만든다
이때 뮤텍스, 세마포어 같은 방식을 추가로 이용해서 값이 변화하는 것을 막자.