C basic language

임베디드스쿨 2기

Lv1과정

2021. 06. 11

김효창

extern

Name Mangling

컴파일로 소스 코드에 선언된 함수 또는 변수의 이름 , 파리미터들을 변경
각 파일마다 존재할 수 있는 동일한 함수명들을 Linker 가 구분하기 위해 사용
Overloading 시 파리미터 형태에 따라 자동으로 구별하여 호출
다른 Namespace 내에 있는 같은 이름의 함수와 변수에 대해 에러 없이 구별하여 호출
모든 전역변수가 네임 맹글링 되지는 않는다.

extern : 함수명 , 전역변수명을 '심볼 ' 로 저장 , Name Mangling 방지

Register

인터럽트 루틴은 리턴 값이 없고 호출하지 않는다

처음에는 SIGNAL(vector) 형태.
나중에는 Interrput handler 에 속성을 넣어서 ISR(…) 로 변화.

```
void __vector_1(void) __attribute__((signal));
void __vector_2(void) __attribute__((signal));
void __vector_3(void) __attribute__((signal));
```

외부 인터럽트 0이 발생하면 '__vector_1'함수가 호출되고, 외부 인터럽트 1이 발생하면 '__vector_2'함수가 호출되고, 외부 인터럽트 2가 발생하면 '__vector_3'함수가 호출된다

ISR_NAKED : 레지스터 저장/복원과 reti() 를 생략한다.
영향 받는 레지스터가 없기 때문에 낭비라면 , 아래와 같이 작성.

```
ISR(INT0_vect, ISR_NAKED) {
	PORTD ^= 0x1;
	reti();
}
```

스위치 2 개로(2 개의 인터럽트 처리) 같은 LED 를 점멸하려면

```
ISR(INT0_vect){
	PORTD ^= 0x1;
}
ISR(INT1_vect, ISR_ALIASOF(INT0_vect))
```

Register

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

int main(void){
      while(1){
      }
}

ISR (ADC_vect){
      PORTB = 0x55;
}

#define ADC_vect_num 16
#define ADC_vect
#define SIG_ADC
```

iom328p.h 를 열면 인터럽트 벡터 이름을 찾을 수 있다.
계속 헤더 파일을 찾는 것이 귀찮다면 _VECTOR()
매크로 사용

```
ISR( _VECTOR(23) )
#define ANALOG_COMP_ISR _VECTOR(23)
```

```
void __vector_5 (void) __attribute__ ((signal, used, externally_visible));
```

각 벡터에 대해 __vector_N이라는 함수를 정의하지 않으면 JUMP는 __bad_interrupt의 약한 대상을 유지하지만 하나 이상의 __vector_N () 함수를 정의하면 약한 링크를 재정의하는 데 사용.

컴파일러가 인터럽트가 언제 실행될지에 대한 어떠한 가정도 할 수 없으므로, 어떤 레지스터를 저장해야 하고 어떤 레지스터를 저장하지 않아야 하는지 모른다.

Register

_SFR_IO8(0x00) S: (special) , F: (function) , R: (register) , IO8 : 8 Bit I/O
ADC 는 IO16 으로 되어 있다. (16 Bit I/O)

sfr_defs.h 열람
#define _SFR_MEM8(mem_addr) _MMIO_BYTE(mem_addr)

_MMIO_BYTE() 찾기
#define _MIMO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
memory mapped io나 peripheral 을 다룰 때 종종 쓰는 표현

mem_addr : 접근해야 하는 메모리 주소 값 ,

(* (volatile unsigned char *) (mem_addr))
mem_addr 은 unsigned char * (포인터)로 캐스팅 되어 있다.

상수는 SRAM 으로 저장하는 것이 아니라, FLASH 영역에 위치한다.
레지스터 영역에 있는 SFR 들을 접근하려면, type 부터 맞춰야
문제가 되지 않는다.

<symbol table>

type	name	address
const char	mem_addr	Code영역
unsigned char*	SFR	레지스터영역

(volatile unsigned char *) (mem_addr)

포인터가 지시하는 곳에 값을 저장. (제외하면 주소를 변경하는 명령이 된다.)

Code

(volatile unsigned char *) 0x5B
0x5B는 volatile character 포인터 타입
0x5B 주소 값이며 안에 들어가는 데이터가 8 Bit 이기 때문에 char 저장

(volatile unsigned char)0x5B → volatile char *p = 0x5B;

Code

8 Bit

일반 : TCNTn 값이 0 ~ 255 까지 증가되고 오버플로 되어 다시 0 부터 증가하며 , 이 과정을 반복

CTC : TCNTn 값이 0 부터 시작하여 OCRn 값과 일치하면 0 으로 초기화 되어 다시 증가

FAST PWM : 일반과 동일하며, PWM 신호 발생하는 기능을 추가로 사용할 수 있다.

16 Bit

일반 : 65535 까지 증가하고 다시 0 부터 시작. 0 이 되는 순간 오버플로 인터럽트 발생.

CTC : 0 부터 시작하여 ICRn 또는 OCRn 값이 되면 0으로 초기화 되어 다시 증가

FAST PWM : 최대값이 되면 다시 0 부터 시작. 최대값은 0x00FF , 0x03FF , ICRn , OCRn 중에서 선택
최대값에 도달했을 때 오버플로 인터럽트 발생

Code

퓨즈 끄는 이유 : 동작이 느리거나 업로드 상태 비 정상 된다.

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

#define F_CPU      16000000UL
#include <util/delay.h>
#include <avr/interrupt.h>

#define sbi (PORTx, BITx)        (PORTx |= (1 << BITx))
```

특정 포트의 비트 설정 매크로

```
#define cbi (PORTx, BITx)        (PORTx &= ~(1 << BITx))
```

특정 포트의 비트 클리어 매크로

```
volatile unsigned int counter = 0;
```

영문 의미 : 휘발성
레지스터 관점으로 확인해야 한다.
실제 메모리에서 값을 읽어서 데이터를 배치하고 구성.
컴파일러는 소프트웨어 , 가상 메모리를 제어 (물리 메모리는 예외) , 레지스터만 확인.
volatile 는 물리 메모리에서 Load 하여 값을 고정으로 사용.

```
uint8_t test;
```

unsigned int 8 비트 type
TI DSP 가 int 6 Byte , 12 Byte
VLIW 아키텍처(Very Long Instruction Word)
Vectorization(벡터화) 기법들의 최초 도입 사례
SSE, AVX - Intel , NEON - ARM , Nvidia GPU, AMD GPU 등등 적용됨.

Code

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

해당 인터럽트 발생 시 counter++
[0] = PCINT0_handler
[1] = PCINT1_handler
[2] = TIM_COMPA_handler
vect 가 없으면 숫자만
vect 가 있으면 배열의 인덱스에 접근하여 함수 포인터를 호출하는 개념.
SIGNAL 의 경우 GCC 확장인 attribute 를 통해 자동으로 인터럽트와 연결할 수 있음.
externally visible 이 GCC 확장으로 붙기 때문에 해당 인터럽트 발생 시
SIGNAL 에 작성한 부분의 코드가 동작하는 것을 알 수 있다.

Code

```
void counter_init(void)
{
	cbi(SREG, 7);
	SREG = AVR Status Register 로 ARM 의 CPSR 또는 Intel 의 EFLAGS 레지스터 역할을 수행.
	전역 인터럽트를 클리어하여 초기화한다.
	TCCR0A = 0;
	TCCR0B = 0;
	Timer/Counter Control Register 로 타이머 및 카운터 설정
	DDRB = 0x02;
	PORTB = 0x00;

	TCCR0A = (1 << WGM01);
	CTC 모드. OCRA pin을 사용.
	TCCR0B = (1 << CS01) | (1 << CS00); prescaler (64) 설정
	TIMSK0 = (1 << OCIE0A);
	TCNT0 = 0;
	clock / 64 → 16 MHz ÷ 64 = 250 KHz → 0.004 ms

	OCR0A = 249;
	(1 ms ÷ 0.004 ms) − 1 =249
	(249 × 0.004 ms) + 0.004 ms = 1ms

	sbi(SREG, 7);
}
```

Code

```
int main(void)
{
	counter_init();

	while(1)
	{
		if (counter == 1000) 1초
		{
			PORTB ^= 0x02;
			counter = 0;
		}
	}

	return 0;

}
```

Code

MCU 의 clock = 16 MHz 일 때 ,
1초에 16,000,000 주기 발생 , 1 주기의 시간 0.0625 us

$$\text{주기} = \frac{1}{\text{주파수}} = \frac{1}{16{,}000{,}000} = 0.0000000625\ sec = 0.0625\ us$$

1ms 시간 설정 (n = 16,000)

$$\frac{1}{1000} = \frac{1}{16{,}000{,}000} \times n$$

1주기 0.0625 us 를 16,000 번 카운터 하면 1ms 시간

8 Bit TNCT0 , TNCT2 는 256 까지 한계 → Prescaler 사용
타이머/카운터 프리스케일러 하드웨어로 계산

$$1ms = (\frac{1}{16MHz} \times prescaler) \times n$$

Code

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	

FOC : Force Output Compare
WGM Bit 가 비 PWM 모드를 지정하는 경우에만 활성화
PWM 모드에서 작동할 때 TCCR0B를 기록할 때 0으로 설정해야 한다.
논리 1을 FOC0A 비트에 쓸 때 파형 생성 장치에서 즉시 비교 일치가 강제 실행됩니다.
강제 비교의 효과를 결정하는 것은 COM0A1 : 0 비트에 있는 값입니다.

WGM : Waveform Generation Mode : 어떤 파형을 생성할 것인지 결정.

CS : Clock Select : 클록 선택 , 분주 선택

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A**1**	COM0A**0**	COM0B**1**	COM0B**0**	–	–	**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	

COM : Compare Match Output Mode :
WGM 설정에 따라 달라진다. Ocn 동작을 제어
OCn Pin 을 사용하기 위해 범용 I/O Pin 의 방향을 출력으로 설정.

Code

OCRx는 프로그래머가 작성합니다. 쓰기 후에도 PWM 모드의 경우 TOP 또는 Bottom에서만 업데이트된다는 뜻입니까?

OCRx는 이중 버퍼링되지만 일반 및 CTC 모드에서는 버퍼링이 비활성화됩니다

CTC 모드에서 TCNT는 Compare match에서 지워집니다. MAX는 항상 0xFF입니다. 그렇다면 MAX에서 TOV 플래그를 어떻게 생성 할 수 있습니까? TCNT = OCRA 일 때 TCNT가 지워지죠?

OCRA가 MAX와 같지 않으면 TCNT는 MAX에 도달 할 수 없으므로 TOV가 발생할 수 없습니다.

WGM02	WGM01	WGM00	Timer/Counter Mode operation	TOP	Upadate of OCRx	TOV Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
0	1	0	CTC	OCRA	Immediate	MAX
0	1	1	Fast PWM	0xFF	BOTTOM	MAX
1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Code

이중 버퍼링
타이머가 작동 중에 OCR 을 변경하거나 TOP 을 변경할 때 잘못하면 오작동 하기 때문에
비교일치 시점에 업데이트 하는 방식 , 글리치 현상 방지하는 것.

CTC 모드에는 이중 버퍼링 기능이 없기 때문이다.

OCR0에 작성된 새 값이 TCNT0의 현재 값보다 낮으면 카운터는 비교 일치를 놓치게 된다.

그런 다음 카운터는 최대값(0xFF)까지 계산하고 비교 일치가 발생하기 전에 0x00에서 시작하여 암호화 해야 한다..

OCR0을 변경하면 출력 파형에서 글리치가 발생할 수 있습니다. 원치 않는 상황을 방지 할 수 있는 하드웨어 솔루션 ?

아니면 업데이트하기 전에 OCR0을 확인하는 것

"PWM 생성 중에도 TCNTn의 값은 계속 증가하며, 이 값보다 작은 값이 OCRn에 쓰여지면, 그 주기에서는 타이머/카운터n가 출력비교에 실패하여 TCNTn값이 0xFFFF(16bit에서)까지 증가하였다가 0x0000으로 되면서 저런 문제가 발생한다.“

CTC 모드에서 OCRn에 이중버퍼링 기능이 없어서 생기는 문제 → glitch

OCR1A 를 줄이면 TCNT 값을 확인하여 TCNT 값이 OCR1A 를 초과하지 않도록 한다.
OCR1A 값에 도달하기 전에 TCNT 값을 0 으로 Clear ????????

Code

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	

TCNT : 카운터 값을 읽기/쓰기 하는 레지스터
n = MCU clock ÷ 원하는 시간 ÷ 프리스케일러

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	

TIMSK 의 OCIE0A 비트가 1 로 기록되고
SREG 의 I- 비트가 설정되면 Timer / Counter0 비교 일치 A 인터럽트가 활성화.

```
TIMSK0 = (1 << OCIE0A);
TCNT0 = 0;
OCR0A = 249;
sbi(SREG, 7);
```

OCIE0A : TCNT 와 OCR0A 값이 동일하면 인터럽트가 발생하도록 제어하는 Bit

15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 compare match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

Code

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	

OCR0 : Output Compare Register
TCNT0 값과 비교하여 인터럽트 또는 OC0 pin으로 출력 신호 발생.

OCR1A 및 OCR1B는 타이머가 PWM을 생성하는 핀.
모드에 따라 업데이트는 전환 또는 설정 또는 해제된다.

Code

인터럽트를 사용하기 위해서 #include 〈avr/interrupt.h〉 선언
ISR : 인터럽트가 발생했을 때 이를 받아서 처리하는 함수
vector : 발생한 인터럽트가 어느 것이냐에 따라 미리 정해진 식별자.

```
ISR(vector){
      …
}
```

TIMER0_COMPA_vect : 타이머0 CTC 인터럽트 함수
1ms 마다 count 값 증가

Timer/Counter0 Compare match A 가 발생하면 ISR(TIMER0_COMPA_vect) 의 루틴을 수행하라는 의미

인터럽트가 발생하면 해당하는 플래그 비트가 SET,
플래그 비트에 의해 인터럽트가 요청, 전체 인터럽트 허가 비트 I 와 해당 인터럽트 허가 비트가 모두 1 로 설정되어 있으면, 인터럽트가 요청되어 인터럽트 벡터의 주소를 찾아서 ISR 을 수행.
ISR 수행되고 있을 때, 자동으로 SREG 의 I 비트를 클리어

인터럽트 모드에 돌입하게 되면 다른 인터럽트 발생이 억제 되는 것
ISR안에서 'I' 비트를 set하게 되면 다시 인터럽트가 발생할 수 있다. 인터럽트 안에서 다시 인터럽트가 발생하는 것을 nested interrupt

Code

sbi (DDRC, PB2); 출력 , cbi (DDRC, PB2); 입력

sbi(PORTx, 2); Bit 2 는 High , cbi(PORTx, 2); Bit 2 는 Low

sbi(EIMSK, 0); EIMSK 의 Bit 0 을 SET

sei() : set global interrupt flag enable (SREG |= 0x80;)
cli() : clear global interrupt flag disable (SREG &= 0x7F;)

```
#define cbi(reg, bit)      reg &= ~_BV(bit)
#define sbi(reg, bit)      reg  |=   _BV(bit)
#define BV(bit)     ( 1 << BIT )
```

volatile

인터럽트 서비스 루틴 내에 사용된 전역 변수와 하드웨어 주소 등에 사용되는 전역 변수들은
레지스터가 아닌 메모리, 어떤 상황으로부터 독립할 수 있는 별도의 메모리 영역에 할당 될 수 있도록
변수 앞에 volatile 를 붙인다.

```
a = 1;
a = 2;
```

최적화하면 a = 1을 삭제하고 a = 2만 실행
a라는 것이 device의 address를 나타내고 0x10000번지에 0x01이란 값을 Write해서 device에 명령을 주는 것이라고 가정하면.. 그 명령은 순서대로 수행해야 한다.

컴파일러는 단순하게 생각하고, a = 1 수행하고 a=2를 수행하면 시간만 까먹으니.. a=2만 수행하도록 한다.

flag값을 여러 번 읽는 경우

주변장치용 레지스터에 많이 사용되지만, OS 를 사용하는 경우에 TASK 간 (또는 ISR) 공유되는
변수에도 사용

컴파일러는 값이 자주 바뀌는 것을 고려해서 컴파일 한다.

코딩에 언급된 그대로 실행하라.
외부와 연결되는 I/O 부분과 연관 있는 변수에 사용
volatile 을 사용하지 않으면 외부 칩의 제어 순서가 바뀔 수 있다.

Code

1.
unsigned int counter = 0; 의 전역 변수 (※ volatile 제외)

컴파일러에 의하여 count 값을 레지스터에 저장

레지스터 저장된 값과 == 1000 비교

SIGNAL (TIMER0_COMPA_vect) 발생으로 count 증가

다시 == 1000 비교 (※ count 증가된 값을 비교하지 않고 0을 다시 비교한다.)

SIGNAL 함수 갱신된 값이 반영되지 않음.

2.
volatile unsigned int counter = 0;

레지스터에 저장하지 않고 실제 메모리에서 값을 읽어서 == 1000 비교

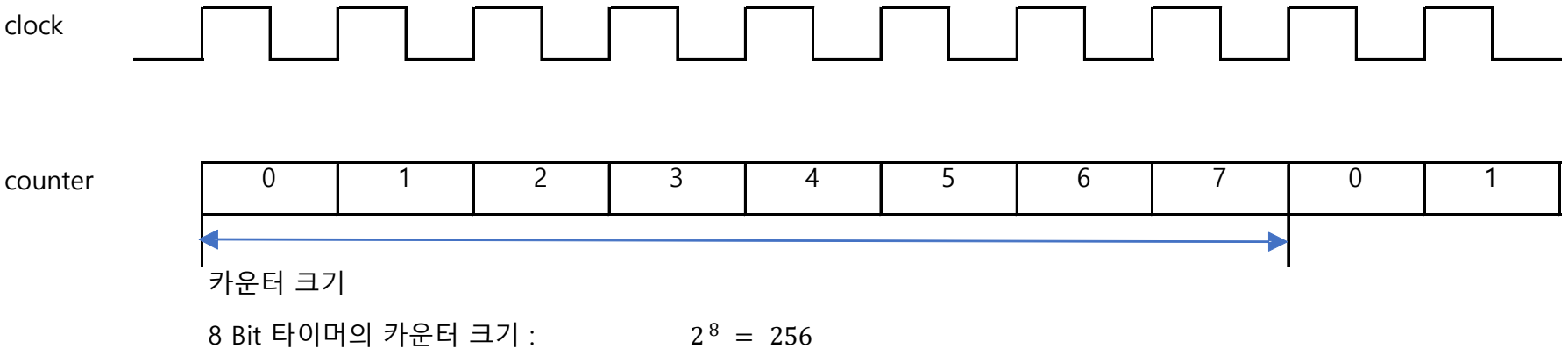
SIGNAL (TIMER0_COMPA_vect) 발생으로 count 증가

count 계속 증가된 값을 다시 읽어서 == 1000 비교

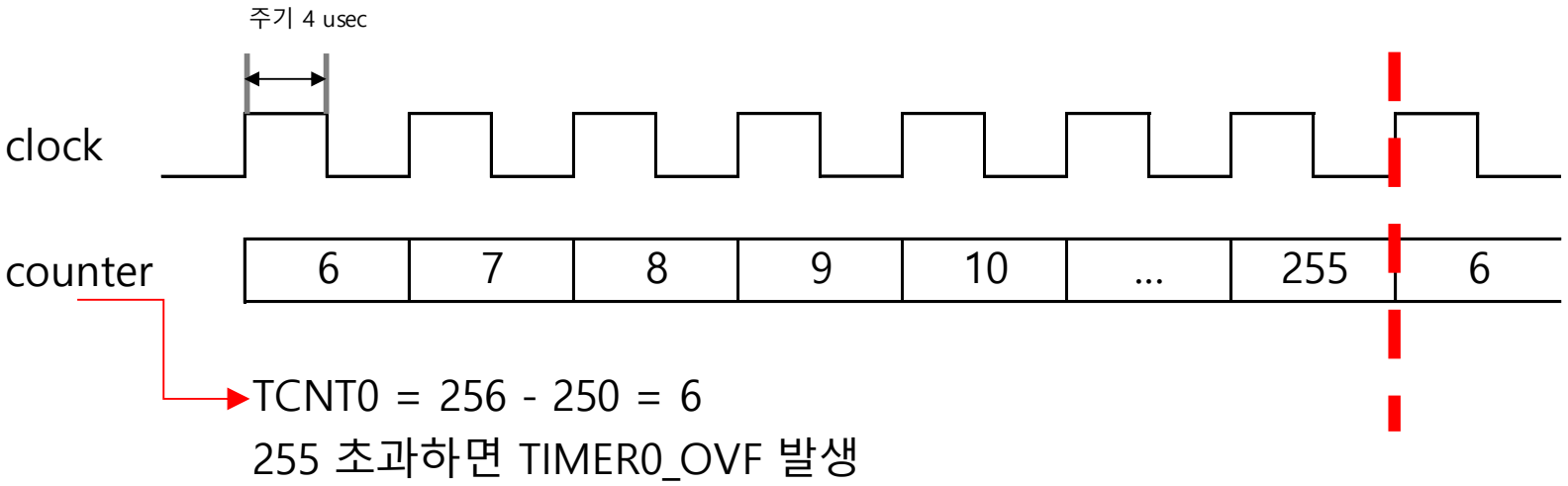
Code

타이머/카운터 : OS 의 Task 스케줄링 구현

Prescaler : 타이머에 공급하는 입력 클럭의 속도를 조절하는 분주기.
고주파 전기 신호를 정수 나눗셈에 의해 낮은 주파수로 줄이는 데 사용되는 전자 계수 회로.
기본 타이머 클럭 주파수(CPU 클럭 주파수일 수도 있고 다소 높거나 낮은 주파수일 수도 있음)를 가져 와서 프리스케일러 레지스터가 구성된 방식에 따라 타이머에 공급하기 전에 일부 값으로 나눈다.
목적은 사용자가 원하는 속도로 타이머를 기록하도록 하는 것.
분해능과 최대 주기 사이의 비율을 조정할 수 있다.



Code



16 MHz / 64 = 0.25 MHz → 4 usec 마다 1 clock
250번 clock 발생하면 1 msec
No prescaling 하면 1/16 MHz = 0.0625 usec 마다 1 clock

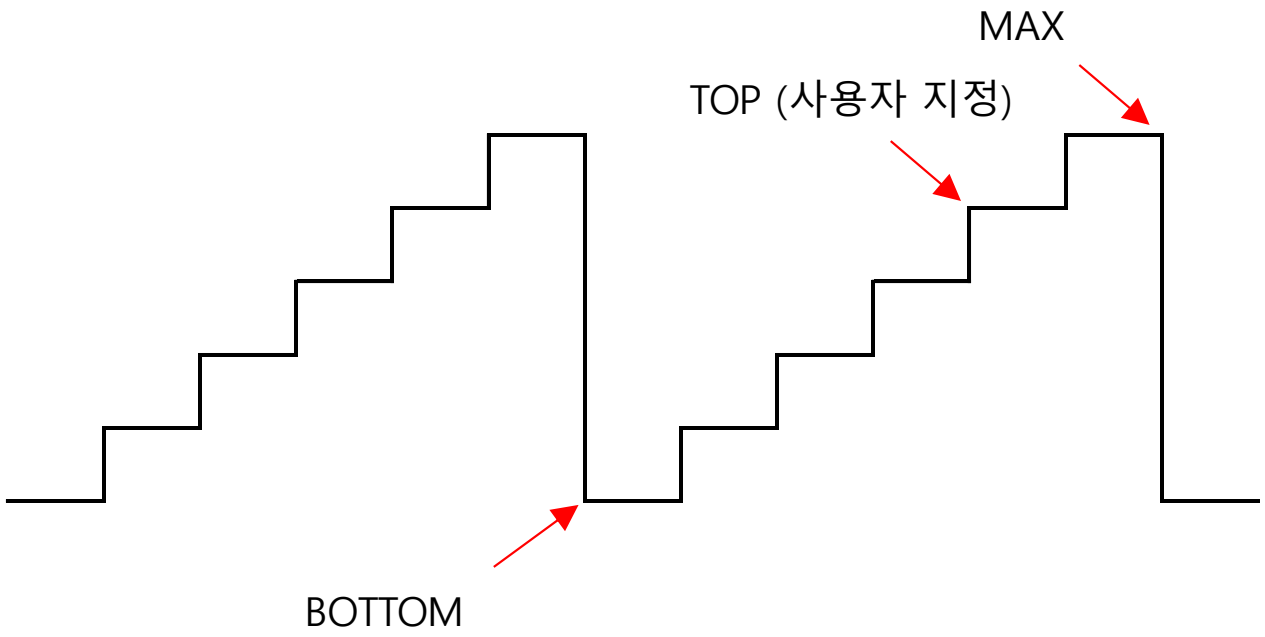
8 Bit 타이머의 최대값 256
TCNT0 = 256 − 250 = 6;
4 usec × 250 = 1 msec

```
ISR(TIMER0_OVF_vect)
{
    ...
}
```

Code

TCNT 값이 증가하여, OCR 값과 일치하면 출력 비교 인터럽트 발생.
TCNT 는 계속 증가하여 65535 에서 초과되는 순간, 오버플로우 인터럽트 발생.

BOTTOM : 타이머/카운터가 동작할 수 있는 최소값.
MAX : 타이머/카운터가 동작할 수 있는 최대값.
TOP : 비교일치 OCR 설정 값 , 최대값 내에서 사용자가 상한선 지정.



Code

```
# include <avr/io.h>
#define F_CPU      16000000UL
#include <util/delay.h>
#include <avr/interrupt.h>

#define sbi (PORTx, BITx)        (PORTx |= (1 << BITx))
#define cbi (PORTx, BITx)        (PORTx &= ~(1 << BITx))

void fast_pwm_servo(void)
{
      cbi(SREG, 7);
      sbi(TCCR1A, COM1A1);
      sbi(TCCR1A, COM1B1);
      non-inverted PWM
      sbi(TCCR1A, WGM11);
      sbi(TCCR1B, WGM13);
      sbi(TCCR1B, WGM12);
      MODE 14 (FAST PWM)
      sbi(TCCR1B, CS11);
      PRESCALER = 8 , 16 MHz ÷ 8 = 2 MHz

      ICR1 = 39999; 20 ms → 50 Hz
      DDRB = 0x02;

      sbi(SREG, 7);
}
```

Code

```
int main(void)
{
	fast_pwm_servo();

	while(1)
	{
		OCR1A = 2000;   90도 , OCR1A = 1000;
		_delay_ms(1000);

		OCR1A = 3000;   0도
		_delay_ms(1000);

		OCR1A = 4000;   -90도  , OCR1A = 5000;
		_delay_ms(1000);

		OCR1A = 3000;   0도
		_delay_ms(1000);
	}

	return 0;
}
```

Code

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
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	

Table 15-2. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match (set output to low level).
1	1	Set OC1A/OC1B on compare match (set output to high level).

COM1A1 , COM1A0 , COM1B1 , COM1B0 , COM1C1 , COM1C0 는 OC1A , OC1B , OC1C 출력 pin 동작을 제어

타이머/카운터 1 , 3은 PWM 지원
(타이머/카운터 0 , 2 는 지원하지 않음)

OCR1A 값과 일치하는 순간 OC1A (PB1) 의 출력이 결정된다.

COM11	COM10	non – inverted PWM
0	0	Normal Port. GPIO로 동작
0	1	비교 일치가 될 때 OC pin 의 상태 반전
1	0	비교 일치가 될 때 OC pin 에 0 출력
1	1	비교 일치가 될 때 OC pin 에 1 출력

Code

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
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	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

파형 발생 모드 = TCCR1A (WGM11 , WGM10) + TCCR1B (WGM13, WGM12)
프리스케일러 값 설정 = CS12 ~ CS10

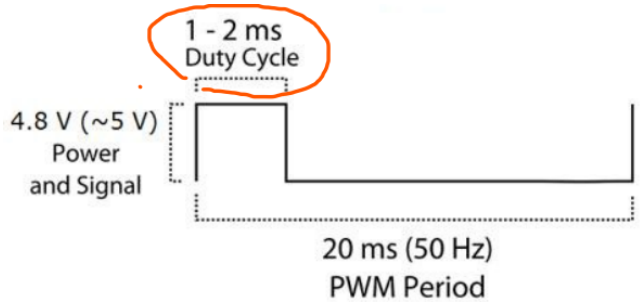
Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (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 T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Code

ICR 을 TOP 으로 사용하는 14 번 모드 Fast PWM 사용

$$50\ Hz\ 의\ PWM = \frac{F\_CPU}{Prescaler \times (TOP + 1)} = \frac{16000000}{8 \times (ICR + 1)}$$



Duty 변경 → OCR1A/OCR1B 값을 설정
카운터 값이 일치하면 OC1A/OC1B 핀이 High 상태에서 Low 상태로 변경된다.

20 ms 중 2 ms 일 때 90° : (40000 × 2) / 20 = 4000
20 ms 중 1.5 ms 일 때 0° : (40000 × 1.5) / 20 = 3000
20 ms 중 1 ms 일 때 −90° : (40000 × 1) / 20 = 2000

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \times (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

SG90 서보 모터

0 ~ 180 ° 범위
Duty cycle 로 원하는 각도 설정.

PWM Period 를 20 ms (주기 : 50Hz) 로 설정

빨강 = Vcc (4.8V ~ 5V,)
노란색 = 접지
주황색 = PWM 입력

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.

Code

ICR , OCR 은 High , Low 로 분할된 16 Bit 이다.
A , B 채널은 ICR1 = 40000; OCR1A = 4000; 입력해도 되지만, C 는 그런 기능이 없다.
타이머 3의 경우도 직접 따로 입력해야 한다.

초기 설정 → 주기 50Hz , 20ms
duty cycle 조절 → 1ms ~ 2ms 사이에서 각도 조절

TCNT 로 증가되는 속도 제어
prescaler : 들어가는 클럭 개수만큼 속도를 줄이는 것
1 클럭이 들어갈 때 증가되는 속도 16 / 8 = 2 MHz = 0.5 us

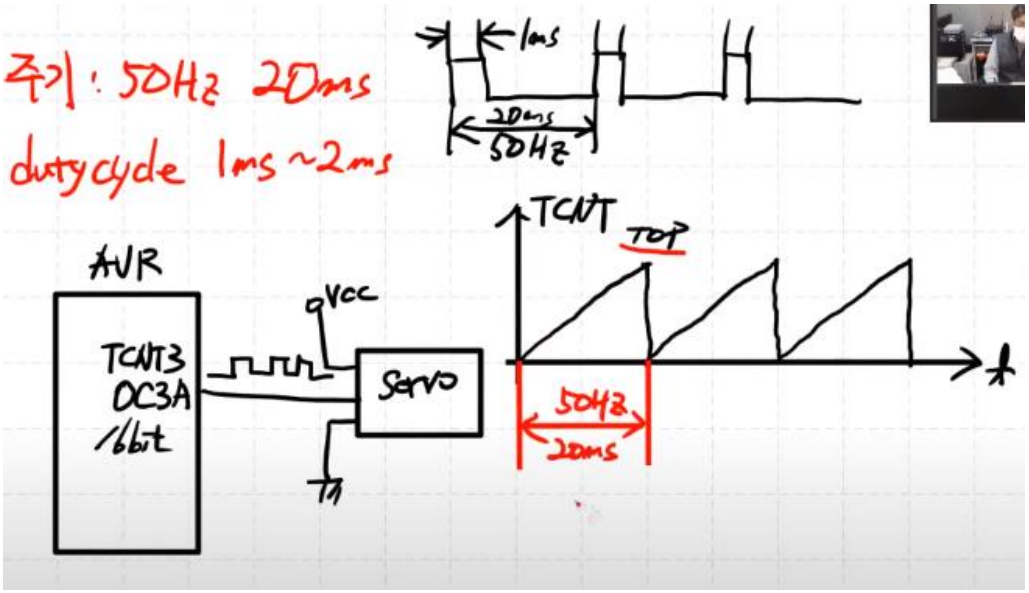
TCNT 값의 1 증가되는 속도는 0.5 us.

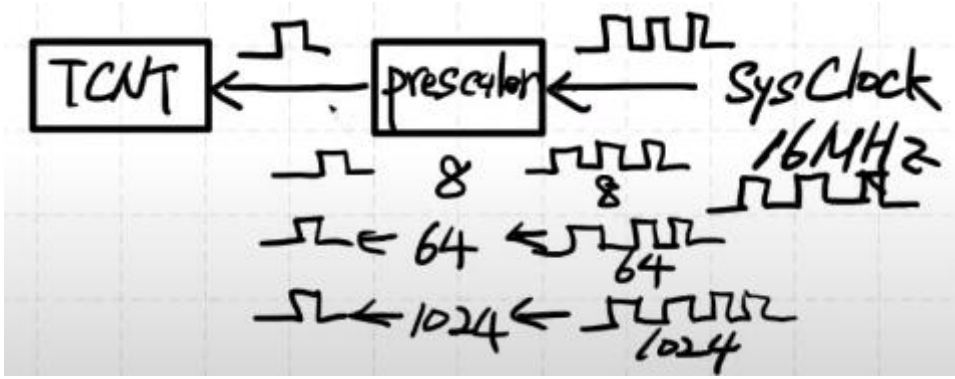
TOP 값까지 올라가는 시간 : 20 ms (이거는 모터 스펙으로 인해 정해진 주기)

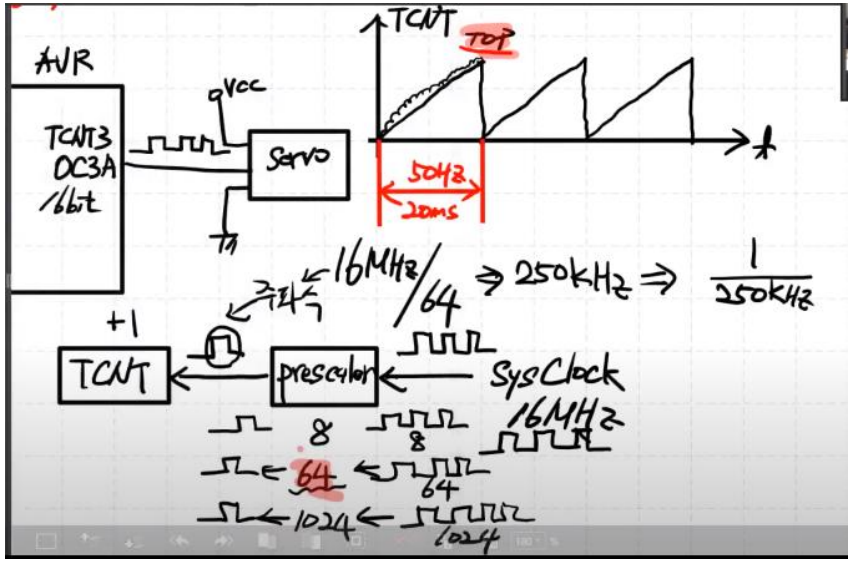
히스테리시스 ???? : 입력이 감소 또는 증가에 따라 그 값이 일치하지 않고 오차를 보이는 현상

서보모터 SG90 은 소형이므로 전류 소비량도 적은 편.
서보모터에 전원을 공급하는 전류 조건을 충족해야 한다. (회전 오동작)

Code







TCNT 1증가 속도 4μs.

Top 값까지 가는 시간 20ms

$4\mu s \times x = 20ms$

$x = \frac{20 \times 10^{-3}}{4 \times 10^{-6}} = 5 \times 10^{3} = 5000$

질의응답

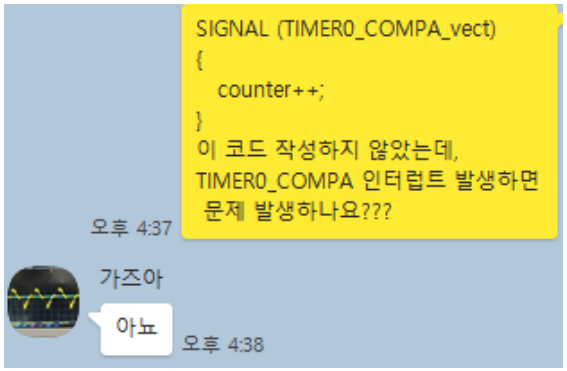
Q : __attribute__ ((externally_visible)) 의미
A : 외부에 노출한다는것은 인터럽트 벡터 테이블을 외부에서 읽어와서 동작시켜야 하기 때문

Q : avr 에서만 사용할 수 있는 지시문?
A : 어디서든 사용 가능

Q : 내용을 이해했는데... 무언가 잘 모르겠습니다 ... 심볼 테이블 ? 리로케이션 테이블? 사전지식이 있어야???
A : 그 부분은 운영체제에 대한 깊은 이해를 요구합니다. 당장은 파악하기 어려우니 우선 레벨 2까진 조금 참아. 실제로 해당 부분을 코딩 해줘야 운영체제 개발이 가능

Q : "외부 클럭을 적용할 때는 mcu 의 안정적인 작동을 보장하기 위해 적용된 클럭 주파수의 급격한 변화를 피해야 한다. 한 클럭 사이클에서 다음 클럭 사이클로 2% 이상의 주파수 변화는 예측할 수 없는 동작으로 이어질 수 있다. “
A : PLL과 관련된 내용 , 당연히 nMHz를 만든다고 하면 PLL 구동시 2% 이상의 오차를 만들면 안되겠죠

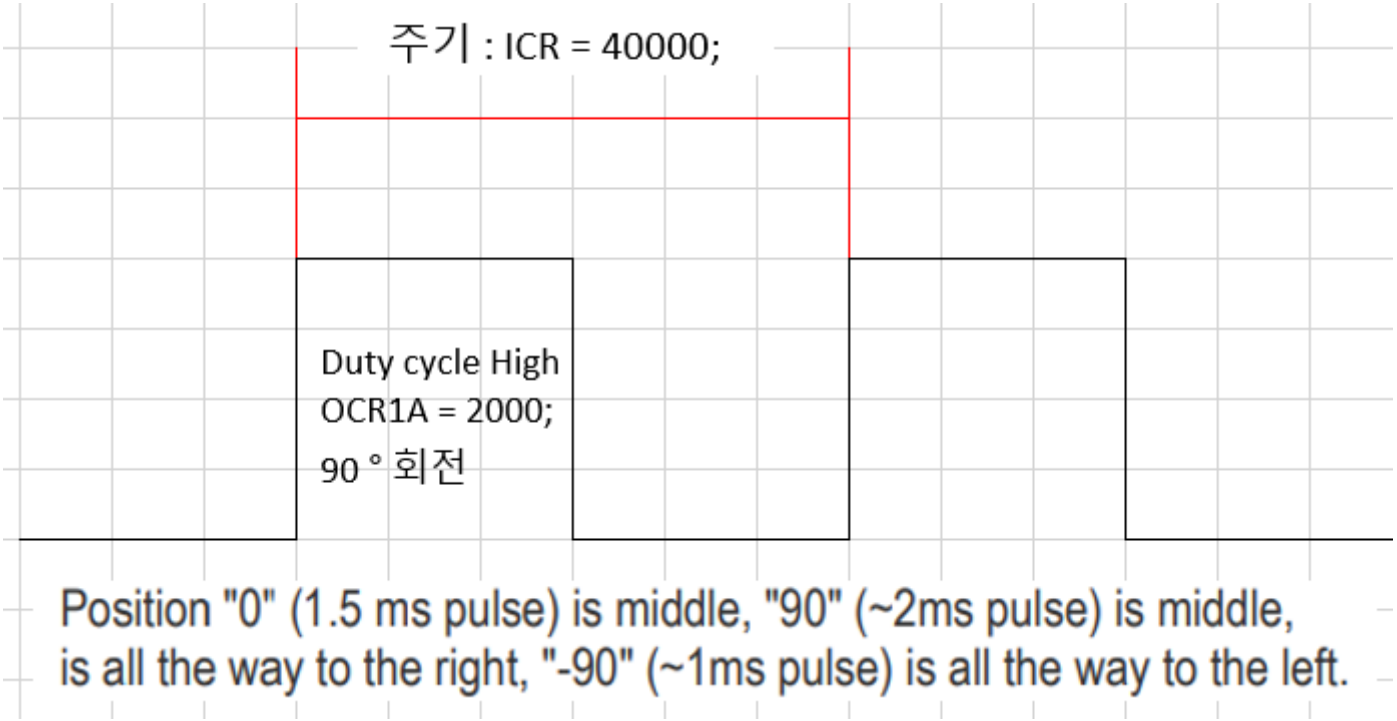
Q : 모든 PORT 가 플로팅 상태여서 처음부터 " cbi(SREG,7) : 모든 인터럽트 비활성화 " 배치하신 건가요???
A : 넵



질의응답

Q :일반, CTC 는 OCR 업데이트 시점 : 설정 즉시이기 때문에 이중 버퍼링이 없다. -> 이중 버퍼링이 없어서 glitch 현상 발생 (타이머가 작동 중에 OCR 을 변경하거나 TOP 을 변경할 때 잘못하면 오작동한다.
A : 데이터 무결성이 깨지면서 문제가 발생할 수 있다는 의미

Q : ICR = 40000 = 20ms = 펄스 개수
A : 네



End of Document