AVR - HW4

임베디드스쿨1기
Lv1과정
2020. 09. 28
손표훈

1. UART

```c
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define sbi(PORTX, BitX) (PORTX |= (1<<BitX))
#define cbi(PORTX, Bitx) (PORTX &= ~(1<<Bitx))

#define UART_BUFLEN 10
static int uartTxchar(char, FILE*);

void UART_INIT(void)
{
    sbi(UCSR0A, U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0C |= 0x06;

    sbi(UCSR0B, RXEN0);
    sbi(UCSR0B, TXEN0);
}

unsigned char UART_RX(void)
{
    while(!(UCSR0A & (1<<RXC0)));
    return UDR0;
}

unsigned char UART_TX(unsigned char data)
{
    while(!(UCSR0A & (1<<UDRE0)));
    UDR0 = data;
}

void UART_string_transmit(char *string)
{
    while(*string != '\0')
    {
        UART_TX(*string);
        string++;
    }
}
```

UART 설정 함수

UCSR0A의 U2X0를 1로 Set
Tx 속도를 Doubling 해준다.

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%

Baud Rate설정

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Data 길이 설정

Tx, Rx Enable

1. UART

UART Rx, Tx 함수

```
unsigned char UART_RX(void)
{
    while(!(UCSR0A & (1<<RXC0)));
    return UDR0;
}

unsigned char UART_TX(unsigned char data)
{
    while(!(UCSR0A & (1<<UDRE0)));
    UDR0 = data;
}

void UART_string_transmit(char *string)
{
    while(*string != '\0')
    {
        UART_TX(*string);
        string++;
    }
}
```

? UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXCn: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn flag can be used to generate a receive complete interrupt (see description of the RXCIEn bit).

수신된 데이터가 UDR레지스터에 다 채워질 때 까지 무한루프

- **Bit 6 – TXCn: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit shift register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn flag can generate a transmit complete interrupt (see description of the TXCIEn bit).

UDR레지스터에 전송할 데이터가 전송이 완료될 때 까지 무한루프

2. ltoa함수

1) ltoa 함수? long형 데이터를 문자열로 변환시켜 주는 함수

```
char *ltoa (long __val, char * __s, int __radix)
{
    if (!__builtin_constant_p (__radix)) {
    extern char *__ltoa (long, char *, int);
    return __ltoa (__val, __s, __radix);
    } else if (__radix < 2 || __radix > 36) {
    *__s = 0;
    return __s;
    } else {
    extern char *__ltoa_ncheck (long, char *, unsigned char);
    return __ltoa_ncheck (__val, __s, __radix);
    }
}
```

1) char형 포인터를 반환하는 함수

2) 변환하고자 하는 long형 데이터

3) 변환결과인 문자열을 저장하는 char형 포인터

4) 변환하고자 하는 기수를 선택
2 : 2진수, 10 : 10진수, 16 : 16진수

2. Itoa함수

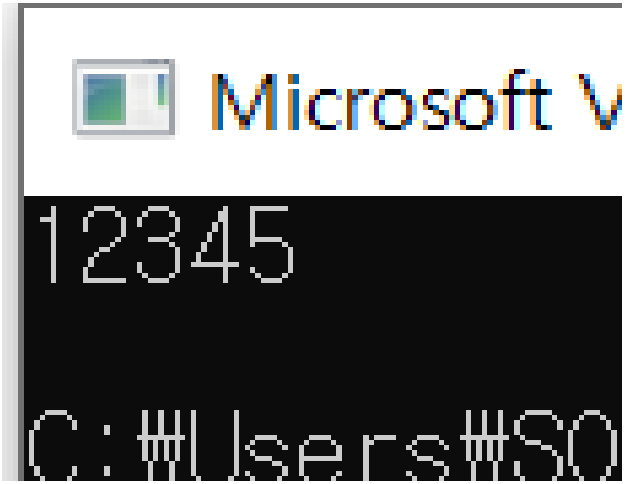
2) 사용방법

```
#define _CRT_NONSTDC_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int num = 12345;
    char str[10] = { 0, };

    _itoa_s(num, str, 10);
    printf("%s\n", str);

    return 0;
}
```



이진법	팔진법	십진법	십육진법	모양	85진법 (아스키 85)	이진법	팔진법	십진법	십육진법	모양	85진법 (아스키 85)	이진법	팔진법	십진법	십육진법	모양	85진법 (아스키85)
0100000	040	32	20	␣		1000000	100	64	40	@	31	1100000	140	96	60	`	63
0100001	041	33	21	!	0	1000001	101	65	41	A	32	1100001	141	97	61	a	64
0100010	042	34	22	"	1	1000010	102	66	42	B	33	1100010	142	98	62	b	65
0100011	043	35	23	#	2	1000011	103	67	43	C	34	1100011	143	99	63	c	66
0100100	044	36	24	$	3	1000100	104	68	44	D	35	1100100	144	100	64	d	67
0100101	045	37	25	%	4	1000101	105	69	45	E	36	1100101	145	101	65	e	68
0100110	046	38	26	&	5	1000110	106	70	46	F	37	1100110	146	102	66	f	69
0100111	047	39	27	'	6	1000111	107	71	47	G	38	1100111	147	103	67	g	70
0101000	050	40	28	(	7	1001000	110	72	48	H	39	1101000	150	104	68	h	71
0101001	051	41	29	)	8	1001001	111	73	49	I	40	1101001	151	105	69	i	72
0101010	052	42	2A	*	9	1001010	112	74	4A	J	41	1101010	152	106	6A	j	73
0101011	053	43	2B	+	10	1001011	113	75	4B	K	42	1101011	153	107	6B	k	74
0101100	054	44	2C	,	11	1001100	114	76	4C	L	43	1101100	154	108	6C	l	75
0101101	055	45	2D	-	12	1001101	115	77	4D	M	44	1101101	155	109	6D	m	76
0101110	056	46	2E	.	13	1001110	116	78	4E	N	45	1101110	156	110	6E	n	77
0101111	057	47	2F	/	14	1001111	117	79	4F	O	46	1101111	157	111	6F	o	78
0110000	060	48	30	0	15	1010000	120	80	50	P	47	1110000	160	112	70	p	79
0110001	061	49	31	1	16	1010001	121	81	51	Q	48	1110001	161	113	71	q	80
0110010	062	50	32	2	17	1010010	122	82	52	R	49	1110010	162	114	72	r	81
0110011	063	51	33	3	18	1010011	123	83	53	S	50	1110011	163	115	73	s	82
0110100	064	52	34	4	19	1010100	124	84	54	T	51	1110100	164	116	74	t	83
0110101	065	53	35	5	20	1010101	125	85	55	U	52	1110101	165	117	75	u	84
0110110	066	54	36	6	21	1010110	126	86	56	V	53	1110110	166	118	76	v	
0110111	067	55	37	7	22	1010111	127	87	57	W	54	1110111	167	119	77	w	
0111000	070	56	38	8	23	1011000	130	88	58	X	55	1111000	170	120	78	x	
0111001	071	57	39	9	24	1011001	131	89	59	Y	56	1111001	171	121	79	y	
0111010	072	58	3A	:	25	1011010	132	90	5A	Z	57	1111010	172	122	7A	z	
0111011	073	59	3B	;	26	1011011	133	91	5B	[	58	1111011	173	123	7B	{	
0111100	074	60	3C	<	27	1011100	134	92	5C	\	59	1111100	174	124	7C	|	
0111101	075	61	3D	=	28	1011101	135	93	5D	]	60	1111101	175	125	7D	}	
0111110	076	62	3E	>	29	1011110	136	94	5E	^	61	1111110	176	126	7E	~	
0111111	077	63	3F	?	30	1011111	137	95	5F	_	62						

2. ltoa함수

3) 소스코드 분석(10진수만 구현해봄... 16, 2진수는 다시 검토필요..)

```
40  char *ltoa(long num, char *str, int radix)
41  {
42    bool isNeg = false;
43    char *ptrTmp = str;
44    long tmp;
45
46    switch(num)
47    {
48      case 0:
49        *str++ = '0';
50        str = '\0';
51        return str;
52      default:
53        break;
54    }
55
56    if(num < 0 && radix == 10)
57    {
58      isNeg = true;
59      num = -1 * num;
60    }
61
62    while(num != 0)
63    {
64      tmp = num % radix;
65      *str++ = (tmp > 9) ? (tmp-10) + 'a' : tmp + '0';
66      num = num/radix;
67    }
68
69    switch(isNeg)
70    {
71      case true:
72        *str++ = '-';
73        break;
74      default:
75        break;
76    }
77
78    strrev(ptrTmp);
79
80    *str = '\0';
81  }
```

입력된 long형 변수의 숫자가 0일 때
문자열로 0을 반환

입력된 숫자가 음수일 때(10진수만 해당되는 코드)
숫자를 음수로 변환

#64 : 입력된 숫자를 radix(기수)로 나누고 나머지를
임시 변수에 저장한다.

#65 : ASCII코드 변환을 위해 나머지가 9보다
큰 경우 16진수의 경우 10='a'이므로
나머지-10에 + 'a'를 한다
예)나머지 = 15라면, 15-10 = 5 -> 5 + 'a' = 'f'
10진수의 경우 ASCII코드 0에 나머지를 더하면 됨.
*str++은 파라미터로 받은 배열의 첫번째 주소에
결과를 더한 후 주소 값 1씩 증가

#66 : 진수변환을 위해 기수로 나눈 몫을 다시 num에
대입한다.

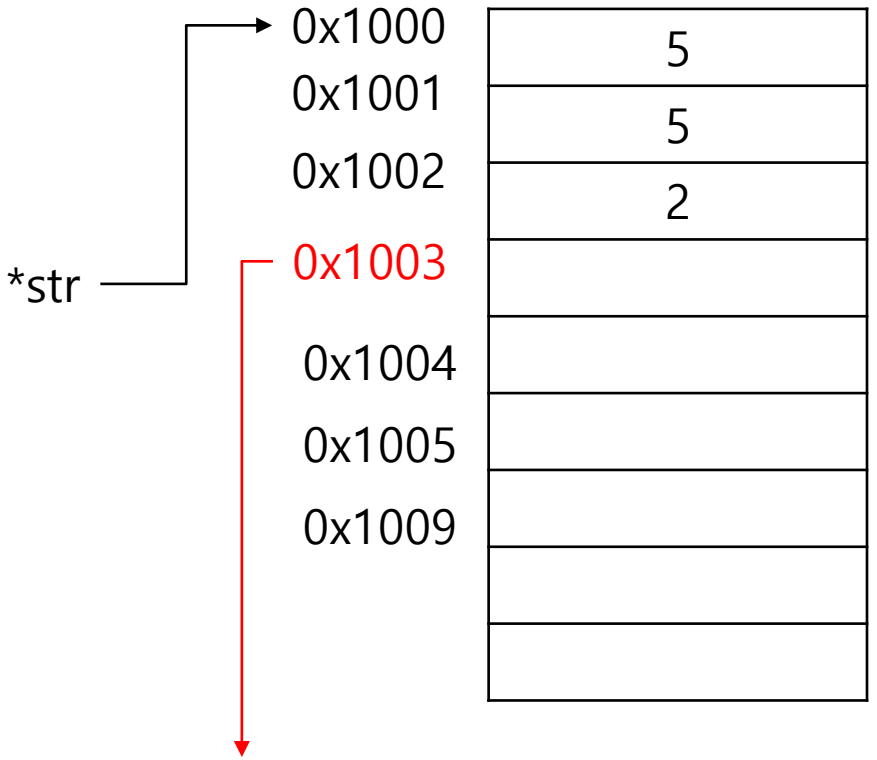
2. ltoa함수

3) 소스코드 분석(10진수만 구현해봄... 16, 2진수는 다시 검토필요..)

```
40  char *ltoa(long num, char *str, int radix)
41  {
42    bool isNeg = false;
43    char *ptrTmp = str;
44    long tmp;
45
46    switch(num)
47    {
48      case 0:
49        *str++ = '0';
50        str = '\0';
51        return str;
52      default:
53        break;
54    }
55
56    if(num < 0 && radix == 10)
57    {
58      isNeg = true;
59      num = -1 * num;
60    }
61
62    while(num != 0)
63    {
64      tmp = num % radix;
65      *str++ = (tmp > 9) ? (tmp-10) + 'a' : tmp + '0';
66      num = num/radix;
67    }
68
69    switch(isNeg)
70    {
71      case true:
72        *str++ = '-';
73        break;
74      default:
75        break;
76    }
77
78    strrev(ptrTmp);
79
80    *str = '\0';
81  }
```

#69 : 음수의 경우 62번행의 결과 종료 후 *str이 가리키는 주소에 '-'문자를 대입

예) num이 -255의 경우 str에 다음과 같이 데이터가 메모리에 할당이 된다.



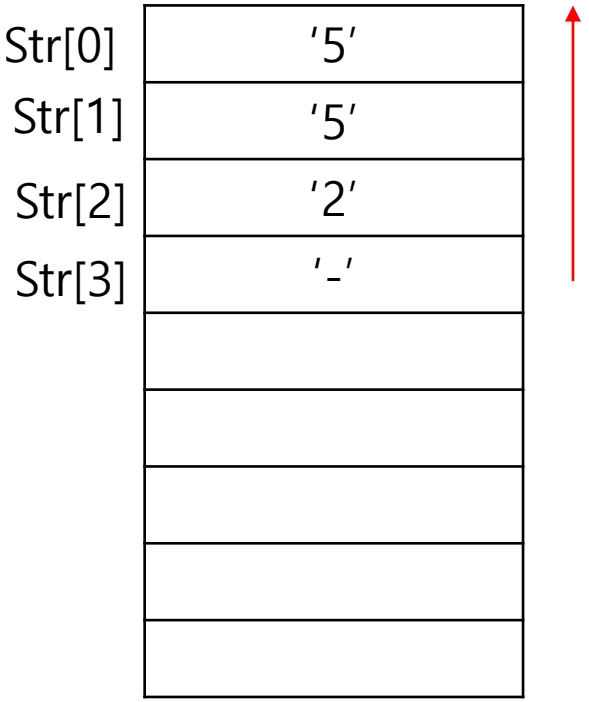
69번째 행일 때 *str은 str[3]과 같다

2. ltoa함수

3) 소스코드 분석(10진수만 구현해봄... 16, 2진수는 다시 검토필요..)

```
40  char *ltoa(long num, char *str, int radix)
41  {
42    bool isNeg = false;
43    char *ptrTmp = str;
44    long tmp;
45
46    switch(num)
47    {
48      case 0:
49        *str++ = '0';
50        str = '\0';
51        return str;
52      default:
53        break;
54    }
55
56    if(num < 0 && radix == 10)
57    {
58      isNeg = true;
59      num = -1 * num;
60    }
61
62    while(num != 0)
63    {
64      tmp = num % radix;
65      *str++ = (tmp > 9) ? (tmp-10) + 'a' : tmp + '0';
66      num = num/radix;
67    }
68
69    switch(isNeg)
70    {
71      case true:
72        *str++ = '-';
73        break;
74      default:
75        break;
76    }
77
78    strrev(ptrTmp);
79
80    *str = '\0';
81  }
```

#78 : 아래와 같이 배열에 할당된 문자를 역순으로 배치



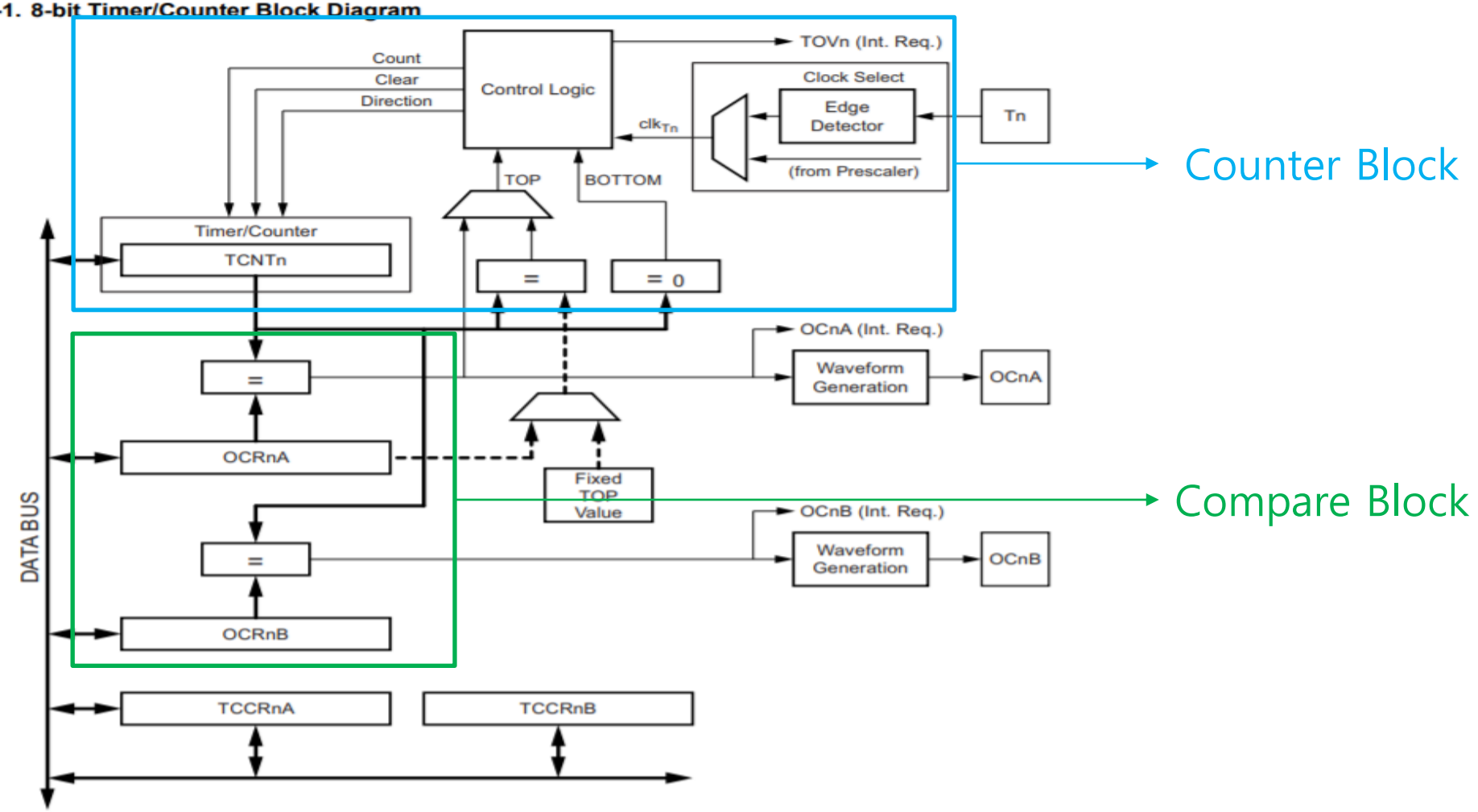
#80 : 마지막 str[4]에 '₩0' null문자를 대입

```
23  void strrev(char *str)
24  {
25    int n = 0;
26    char tmp;
27
28    while(str[n] != '\0')
29      n++;
30
31    for(int i = 0; i < n; i++)
32    {
33      tmp = str[i];
34      str[i] = str[n-1];
35      str[n-1] = tmp;
36      n--;
37    }
38  }
```

2. Timer/Counter

1) Timer/Counter란? 내/외부의 클럭 소스를 사용하여 사용자가 필요로 하는 일정한 주기로 CPU에 신호를 보내는 주변장치이다.

Figure 14-1. 8-bit Timer/Counter Block Diagram



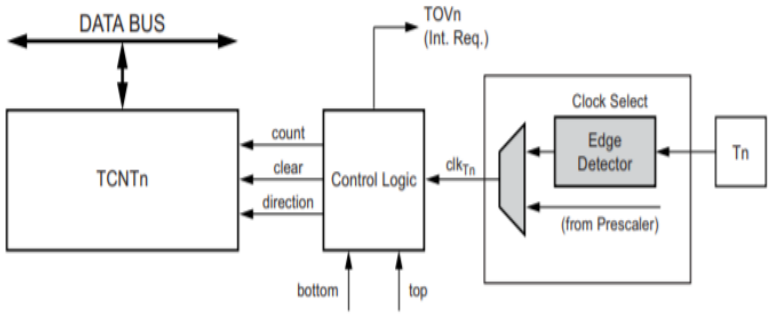
2. Timer/Counter

2) Timer/Counter원리

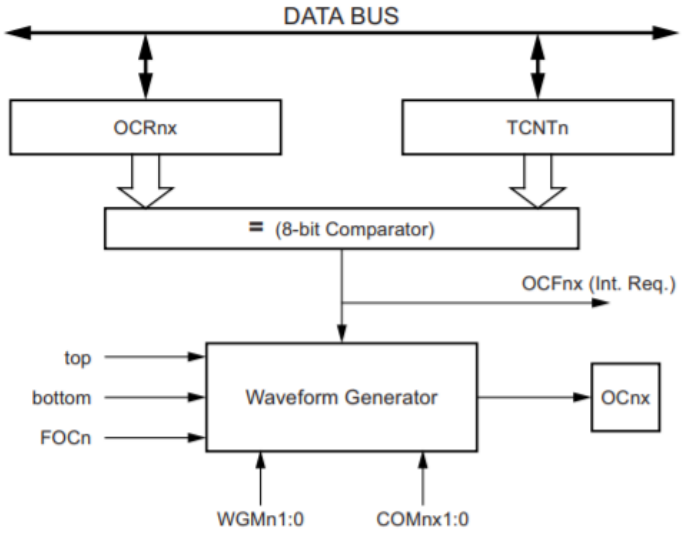
Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 14-2 shows a block diagram of the counter and its surroundings.

Figure 14-2. Counter Unit Block Diagram



Output Compare Unit, Block Diagram



- Counter Block : 시스템 클럭을 입력 받아 펄스의 개수를 센다.
  펄스개수 값은 TCNTn에 저장됨.
- Timer/Counter는 보통 분해능을 가지며 ATmega328의 Timer/Counter0는 8bit의 분해능을 가진다.
- 시스템 클럭을 0~255(8bit)로 분주 시킬 수 있다.
- Output Compare Unit은 Counter블럭의 펄스개수와 사용자가 설정한 OCR값을 비교하여 일치하면 CPU에 일치신호를 보낸다

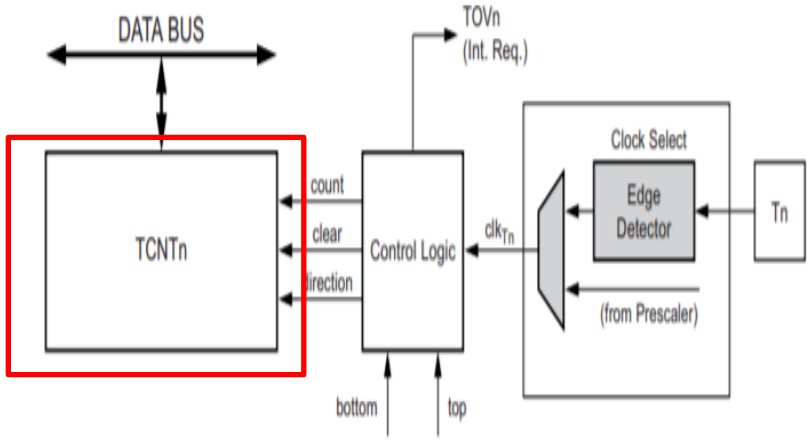
2. Timer/Counter

2) Timer/Counter의 동작모드
-Normal Mode

Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 14-2 shows a block diagram of the counter and its surroundings.

Figure 14-2. Counter Unit Block Diagram



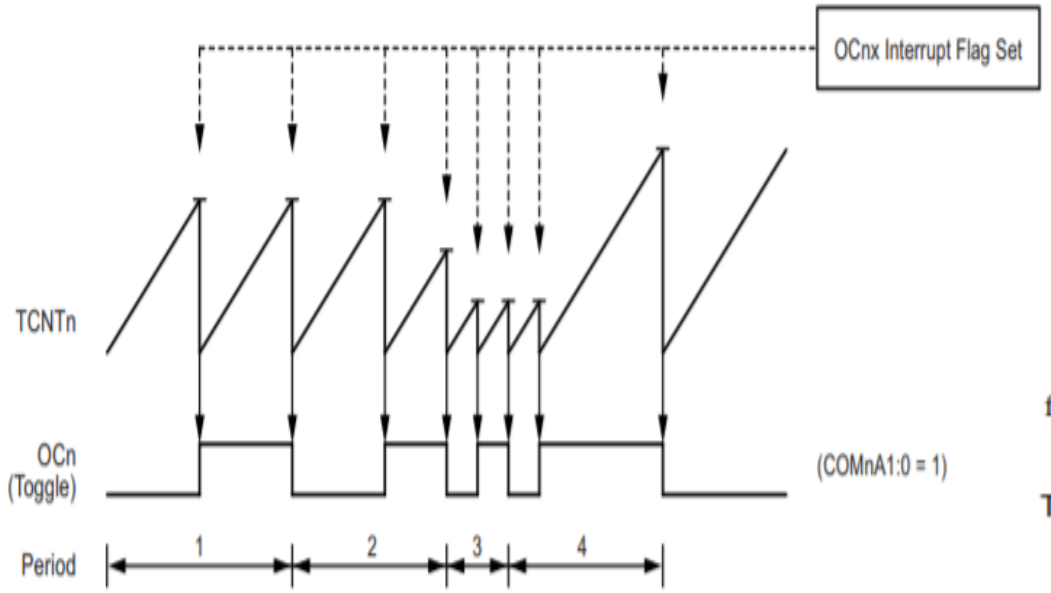
- TCNTn레지스터의 설정 값에 따라 설정된 TCNTn ~ 255까지 카운트를 하며, MAX값인 255가되면 다시 설정된 TCNTn으로 초기화된다.

- 위 과정을 반복하면서 CPU에 일정한 주기로 인터럽트 신호를 보냄

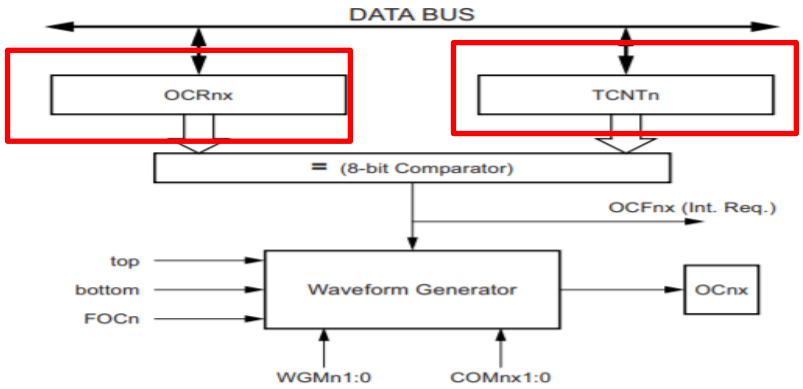
- 시스템 클럭이 100us이고, TCNT = 56이라면, 인터럽트 신호는 100us X (256-56) 20ms마다 발생한다.

2) Timer/Counter의 동작모드 -CTC모드

Figure 14-5. CTC Mode, Timing Diagram



Output Compare Unit, Block Diagram



- TCNT값과 설정한 OCR레지스터의 값과 비교 후 일치하면 CPU에 인터럽트 신호를 발생하는 모드

- 주파수 계산은 아래와 같다

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \times N \times (1 + OCRnx)}$$

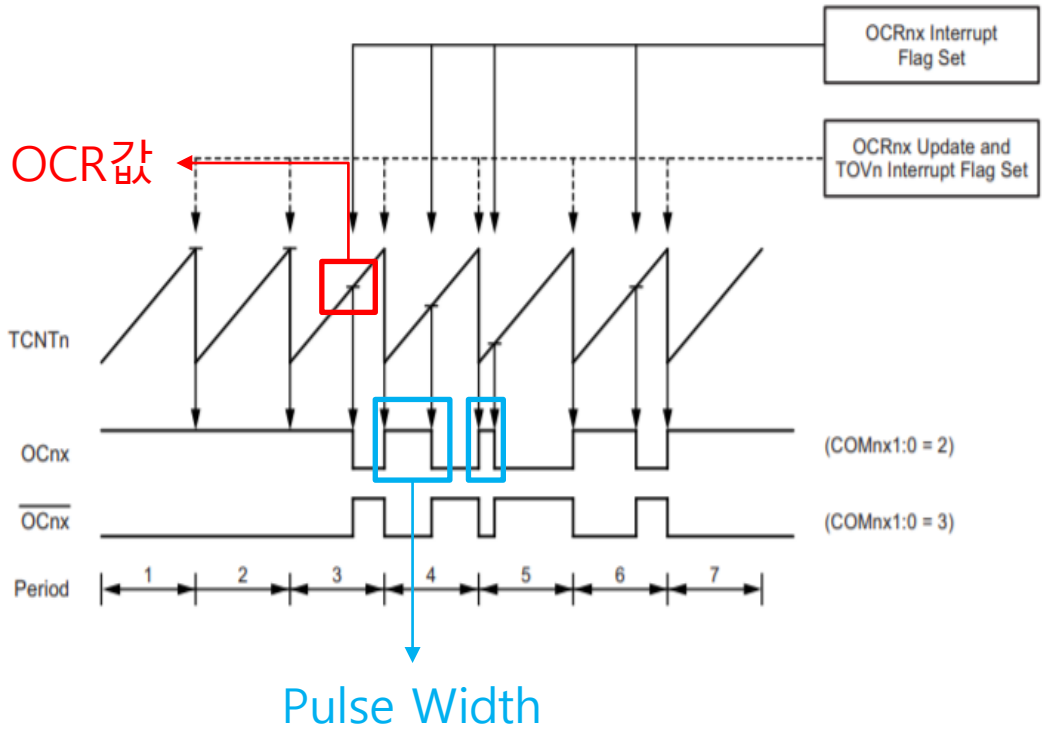
The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

- 프리스케일러 값이란? 시스템클럭을 타이머/카운터 모듈에 입력하기전 분주하기 위한 값이다.

2. Timer/Counter

2) Timer/Counter의 동작모드
- FAST PWM

Figure 14-6. Fast PWM Mode, Timing Diagram



$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \times 256}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

주파수 설정!!

- PWM = Pulse Width Modulation 으로 펄스폭 변조를 말한다.
- Duty ratio = 펄스 주기에서 High구간의 비율을 말한다..
  예) 1초의 주기에서 0.5초동안 High이면 Duty Ratio = 50%
- TCNT와 동작은 비슷하지만 OCR값과 일치해도 TCNT는 255까지 계속 값을 카운트한다..
- 설정한 OCR값이 TCNT값보다 작으면, OCn핀에서 Low를 출력하며
- OCR값 설정을 통해 Duty Ratio를 설정할 수 있다.