Høgskolen i Østfold

Løsningsforslag til
2. del av
Del - EKSAMEN

Emnekode: **ITD13012**	Emne: **Datateknikk**	
Dato: **19. Mai 2014**	Eksamenstid: **kl 9:00** til kl **12:00**	
Hjelpemidler: • 4 sider (A4) (2 ark) med egne notater. • Ikke-kummuniserende kalkulator.		Faglærer: Erling Strand
Eksamensoppgaven: Oppgavesettet består av 3 sider med oppgaver og 11 sider vedlegg, totalt 14 sider. Kontroller at oppgaven er komplett før du begynner å besvare spørsmålene. *Oppgavesettet består av 3 oppgaver. Hvor mye hver oppgave teller er angitt på oppgaven. Alle spørsmål på oppgavene skal besvares.*		
Sensurdato: 11. Juni 2014 Karakterene er tilgjengelige for studenter på studentweb senest dagen etter oppgitt sensurfrist. Følg instruksjoner gitt på: http://www.hiof.no/index.php?ID=7027		

Alle C-program skal virke på Ethernut2.1 kort, med AtMega128 prosessor. (Samme som brukt i lab i våres).

Oppgave 1 (35%)

a) *Lag et hovedprogram som leser to hel tall (int) fra tastatur, tall1 og tall2, og som skriver ut det største tallet.*

```
#include <dev/board.h>
#include <stdio.h>
#include <io.h>


int main(void)
{
 unsigned long baud = 115200;
 int tall1, tall2;
```

```
    NutRegisterDevice(&DEV_DEBUG, 0, 0);

    freopen(DEV_DEBUG_NAME, "w", stdout);
    freopen(DEV_DEBUG_NAME, "r", stdin);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);

    printf("\nSkriv inn tall1:");
    scanf(”%d”, &tall1);

    printf("\nSkriv inn tall2:");
    scanf(”%d”, &tall2);

     if (tall1 > tall2)
         printf(”\nDet stoerste tallet er: %d\n”, tall1);
     else
        printf(”\nDet stoerste tallet er: %d\n”, tall2);

    for(;;);

    return 0;
}
```

b) *Skriv om programmet i a) slik at avgjørelsen av hvilket av de to tall som skal skrives ut (tall1 eller tall2) og utskriften, gjøres i en funksjon. Innlesingen av tallene gjøres i hovedprogrammet. Funksjonen skal kalles fra hovedprogrammet.*

```
#include <dev/board.h>
#include <stdio.h>
#include <io.h>

void maxtall(int x, int y);

int main(void)
{
 unsigned long baud = 115200;
 int tall1, tall2;

     NutRegisterDevice(&DEV_DEBUG, 0, 0);

    freopen(DEV_DEBUG_NAME, "w", stdout);
    freopen(DEV_DEBUG_NAME, "r", stdin);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);

    printf("\nSkriv inn tall1:");
    scanf(”%d”, &tall1);

    printf("\nSkriv inn tall2:");
    scanf(”%d”, &tall2);

    maxtall(tall1, tall2);

    for(;;);

    return 0;
}
/************/
```

```
void maxtall(int x, int y)
{
       if (x > y)
         printf(”\nDet stoerste tallet er: %d\n”, x);
       else
        printf(”\nDet stoerste tallet er: %d\n”, y);
}
```

Oppgave 2 (25 %)

a) Hvordan blir utskriften på dette programmet, hvis du skriver inn tallet 20:

```
#include <dev/board.h>
#include <stdio.h>
#include <io.h>


int main(void)
{
 unsigned long baud = 115200;
 int tall, svar;

    NutRegisterDevice(&DEV_DEBUG, 0, 0);

    freopen(DEV_DEBUG_NAME, "w", stdout);
    freopen(DEV_DEBUG_NAME, "r", stdin);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);

    printf("\nSkriv inn tall:");

    scanf(”%d”, &tall);

/**** De neste to linjene gjelder også i de neste spørsmålene ****/
    svar = (2*tall + 5) * 10;
    printf(”\nTallet er %d, og svaret er %d \n”, tall, svar);

/**** Slutt på det som også gjelder i oppgave b) ***/

    for(;;);

    return 0;
}
```

Vi regner ut svar= (2*20+5)*10= 450
Da blir utskriften:

Tallet er 20, og svaret er 450

b) *Det samme programmet som i a), men nå forandres den første linjen, av de to linjene i oppgaven til:*

```
svar = tall << 2 ;
```

Hva blir utskriften nå?

Kommentar: Binærverdien av 20 er 00010100

svar = tall << 2 betyr at bitene blir skiftet 2 plasser til venstre, og lagt inn i svar.
Da blir bitverdien i svar: 01010000. Skrevet på vanlig desimalform er det 80

Da blir utskriften:

Tallet er 20, og svaret er 80

Hvert skift til venstre er det samme som å multiplisere med 2. Da blir 2 skift til venstre 2*2=4.. 4 * 20 = 80

c) *Det samme programmet som i a), men nå forandres den første linjen, av de to linjene i oppgaven til:*

```
svar = ((2*tall + 5) * 10) & 0x03ff;
```

Hva blir utskriften nå?

Her blir resultatet av (2*tall +5) bitvis and'ed med 0x03ff. Det gir:

```
450 =     0000 0001 1100 0010
0x03ff= 0000 0011 1111 1111
Svar=    0000 0001 1100 0010
```

Som er 450

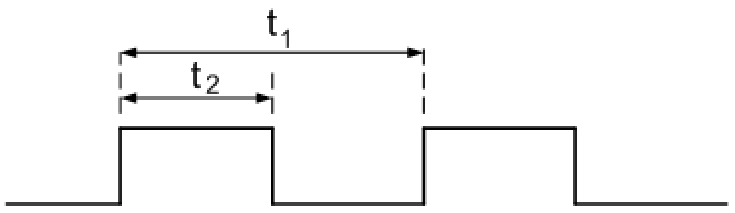
Utskriften blir da:
Tallet er 20 og svaret er 450

<u>Oppgave 3</u> (40%)

a) *Lag et program som leser et klokkesignal (01010101 osv) på inngang ICP1. Frekvensen på klokkesignalet er mellom 100 Hz og 100 KHz. Ved hver 5000 klokkepuls, skal du skrive ut frekvensen. Utskriften skal bare være frekvensen, med benevnelsen Hz bak. Lag ny linje for hver utskrift. Du skal ikke bruke interupt.*

Velger timer 1 i denne løsning. Kunne også valgt timer 3.

Inngangssignalet ser slik ut. Her er $2*t_2=t_1$



Ved 100 Hz er t_1=1/100 [s] = 0,01 [s] = 10 [ms]

Ved 100 KHz er $t_1=1/(100 \cdot 10^3)$ [s] = $0{,}01 \cdot 10^{-3}$ [s] = 10 [µs]

Tar utgangspunkt i frekvensen på prosessoren: 14,7456 MHz. Vi bruker denne som klokken til timerkretsen, Antall tellinger (diff) (økning av TCNT) ved 100 Hz og 100 KHz er.

Bruker formelen: diff=krystallfrekvens * (1/frekvens på inngangssignalet)

Neddeler	Gir frekvens fc	Økning av TCNT 100 Hz diff	Økning av TCNT 100 KHz diff
1	14,7456 MHz	147456 (maks er 65535)	147
8	1,8432 MHz	18432	18
64	230,4 KHz	2304	2
256	57,6 KHz	576	0,5 (umulig)
1024	14,4 KHz	144	0,1 (umulig)

For å dekke hele området fra måling fra 100 Hz til 100 KHz må vi velge en neddeler på 8

Da blir programkoden:

```
#include <dev/board.h>
#include <stdio.h>

int main(void)
{
   unsigned long baud = 115200;  /* Hastighet paa com-porten */
   int tall1=0, tall2, diff=0, i=0,
   float frekvens;

/****** Initier skriving til skjerm, via com-porten ************/
    NutRegisterDevice(&DEV_DEBUG, 0, 0);

    freopen(DEV_DEBUG_NAME, "w", stdout);
    freopen(DEV_DEBUG_NAME, "r", stdin);
    ioctl(_fileno(stdout), UART_SETSPEED, &baud);

/************** Initier timer *******************/
 TCCR1B = (1<<ICNC1) | (1 << CS11);   /** dele krystallfrekvens paa 8: CS11=1 **/
                        /** har også valgt noise canceler: ICNC1. Ikke nødvendig her */

/**** Skriv ut info om at initiering er ferdig ***********/
printf("\nInitierig klar\n");

      while(1) {
/***** Vent paa en negative flanke paa ICP1 inngangen (=PD4) *****************/
               while (!(TIFR & (1<<ICF1)))
               ;

/********* Nullstill flagg i timer ******************/
               TIFR = (1<< ICF1);

/***** Les timerverdi og finn diff  ***************/

               tall2 = ICR1;
               diff = tall2 - tall1;
               tall1 = tall2;

/***** Skriv ut for hver 5000'ende lesing, ****/
               if (! ((i++) % 5000) ) {
                       frekvens = 1843200/ diff;
                       printf("\n%.0f [Hz]", frekvens);
                        }
       }

 return 0;
}
/*********Slutt paa hovedprogram ********/
```

Denne løsningen vil gi en liten feil ved høyere frekvenser. Ved den største frekvensen vil den vise 102400 Hz, da det skulle ha vært 100000 Hz. Det er fordi differansen i avlesningstallene da er så liten som 18.

b) *Skriv om programmet i a) slik at du bruker interupt på å lese klokkesignalet. Utskriften for hver 5000 klokkepuls skal nå ligge i hovedprogrammet.*

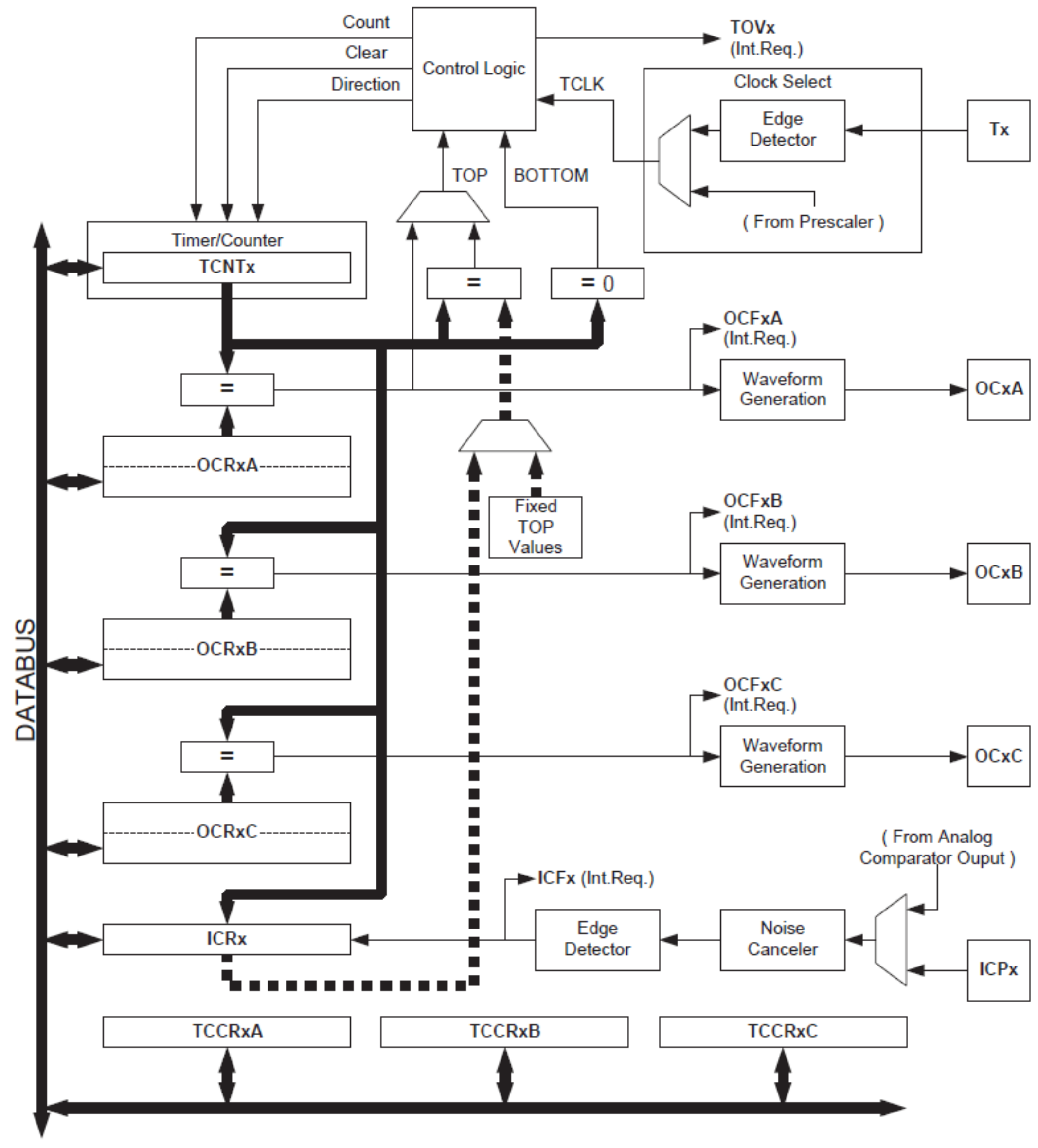
```
#include <dev/board.h>
#include <stdio.h>
#include <avr/interrupt.h>

/*** Globale variable *****/
int tall1=0, tall2, diff=0, set = 0;

int main(void)
{
    unsigned long baud = 115200;  /* Hastighet paa com-porten */
    int  i=0;
   float frekvens;

/****** Initier skriving til skjerm, via com-porten ************/
    NutRegisterDevice(&DEV_DEBUG, 0, 0);

    freopen(DEV_DEBUG_NAME, "w", stdout);
    freopen(DEV_DEBUG_NAME, "r", stdin);
    ioctl(_fileno(stdout), UART_SETSPEED, &baud);

/************** Initier timer *******************/
    TCCR1B = (1<<ICNC1) | (1 << CS11);   /** dele krystallfrekvens paa 8: CS11=1 **/
                      /** har også valgt noise canceler: ICNC1. Ikke nødvendig her */

/**** Slaa paa interuptmask *********/
   TIMSK = (1 << TICIE1);

/**** Skriv ut info om at initiering er ferdig ***********/
   printf("\nInitierig klar\n");

/**** Slaa paa interupt hovedbryter *********/
   sei();

  while(1) {
          if (set = = 1) {

/***** Skriv ut for hver 5000'ende lesing, ****/
                 if (! ((i++) % 5000) ) {
                       frekvens = 1843200/ diff;
                       printf("\n%.0f [Hz]", frekvens);
                        }
                set = 0;
```

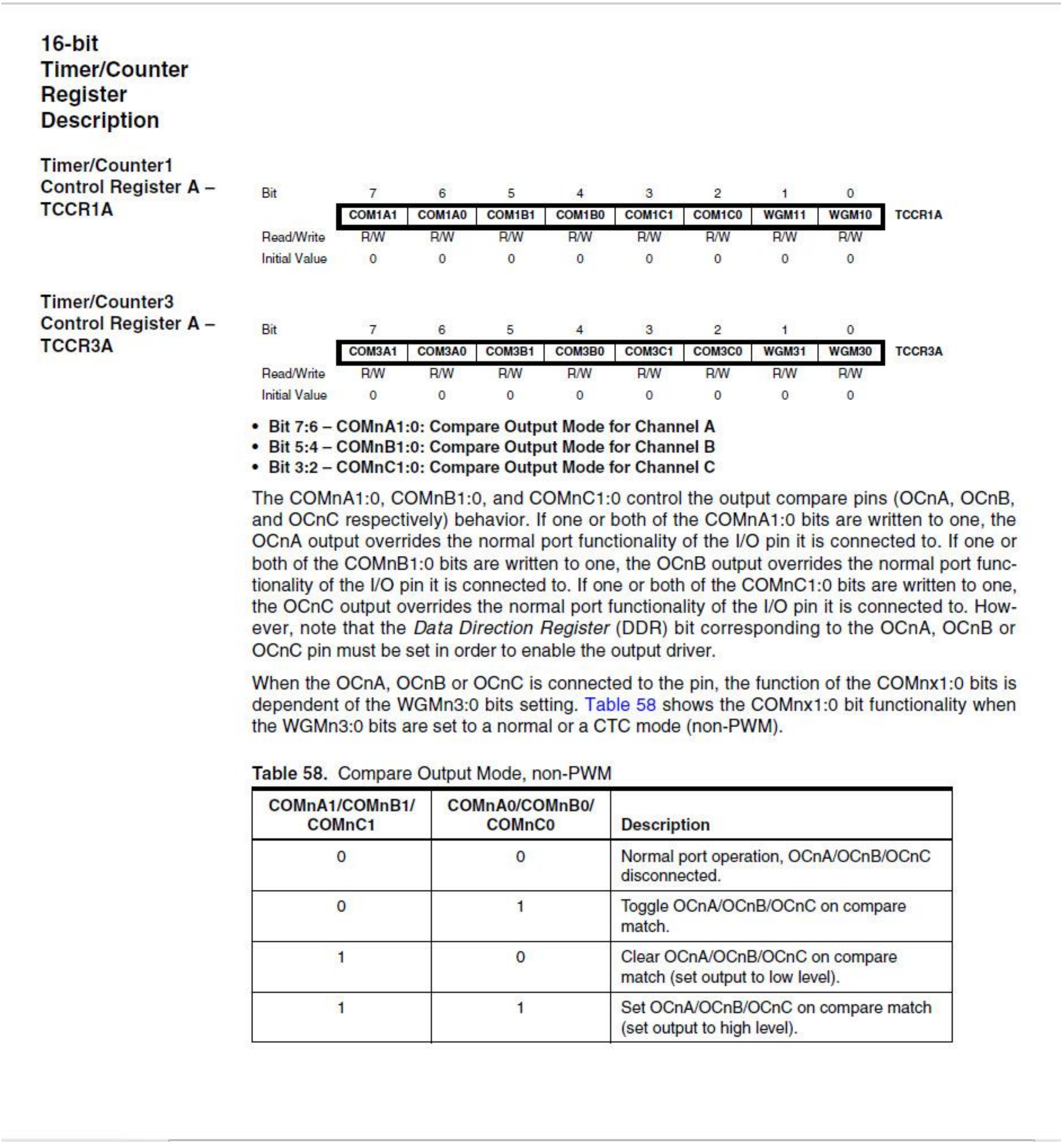
```
            }    /**** end   if set ****/
}  /**** end while(1) ***/
 return 0;
}

/*********Slutt paa hovedprogram ********/

/*********Interuptrutine ************/
SIGNAL(SIG_INPUT_CAPTURE1)
{
 tall2 = ICR1;
 diff = tall2 - tall1;
 tall1 = tall2;
 set = 1;

 }
```

VEDLEGG

Frekvensen på prosessoren er 14,7456 MHz

Figure 46. 16-bit Timer/Counter Block Diagram





16-bit Timer/Counter Register Description

Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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	

Timer/Counter3 Control Register A – TCCR3A

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
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	

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

The COMnA1:0, COMnB1:0, and COMnC1:0 control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bits are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register* (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver.

When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnx1:0 bits is dependent of the WGMn3:0 bits setting. Table 58 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to a normal or a CTC mode (non-PWM).

Table 58. Compare Output Mode, non-PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

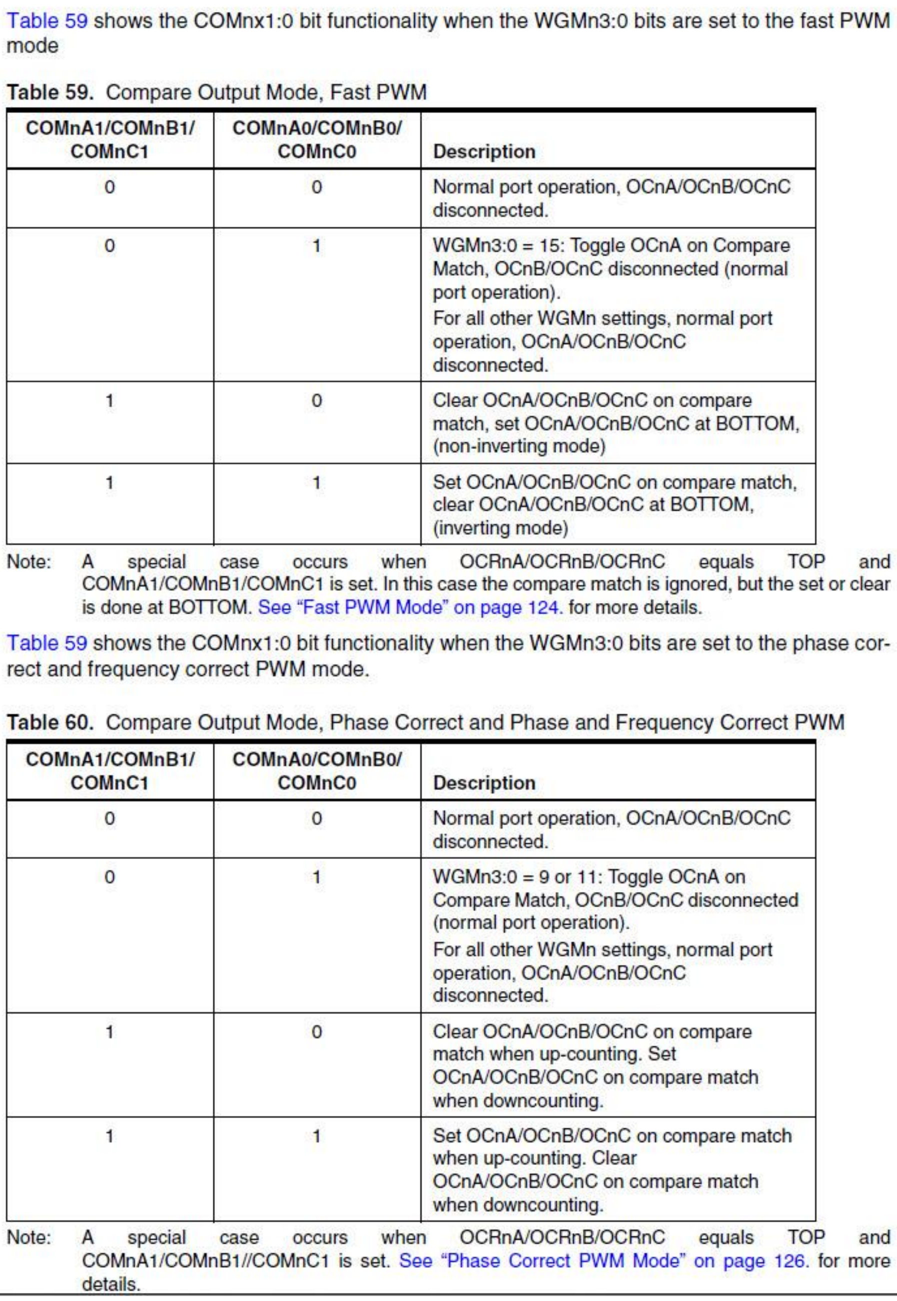


Table 59 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode

Table 59. Compare Output Mode, Fast PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGMn3:0 = 15: Toggle OCnA on Compare Match, OCnB/OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM, (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM, (inverting mode)

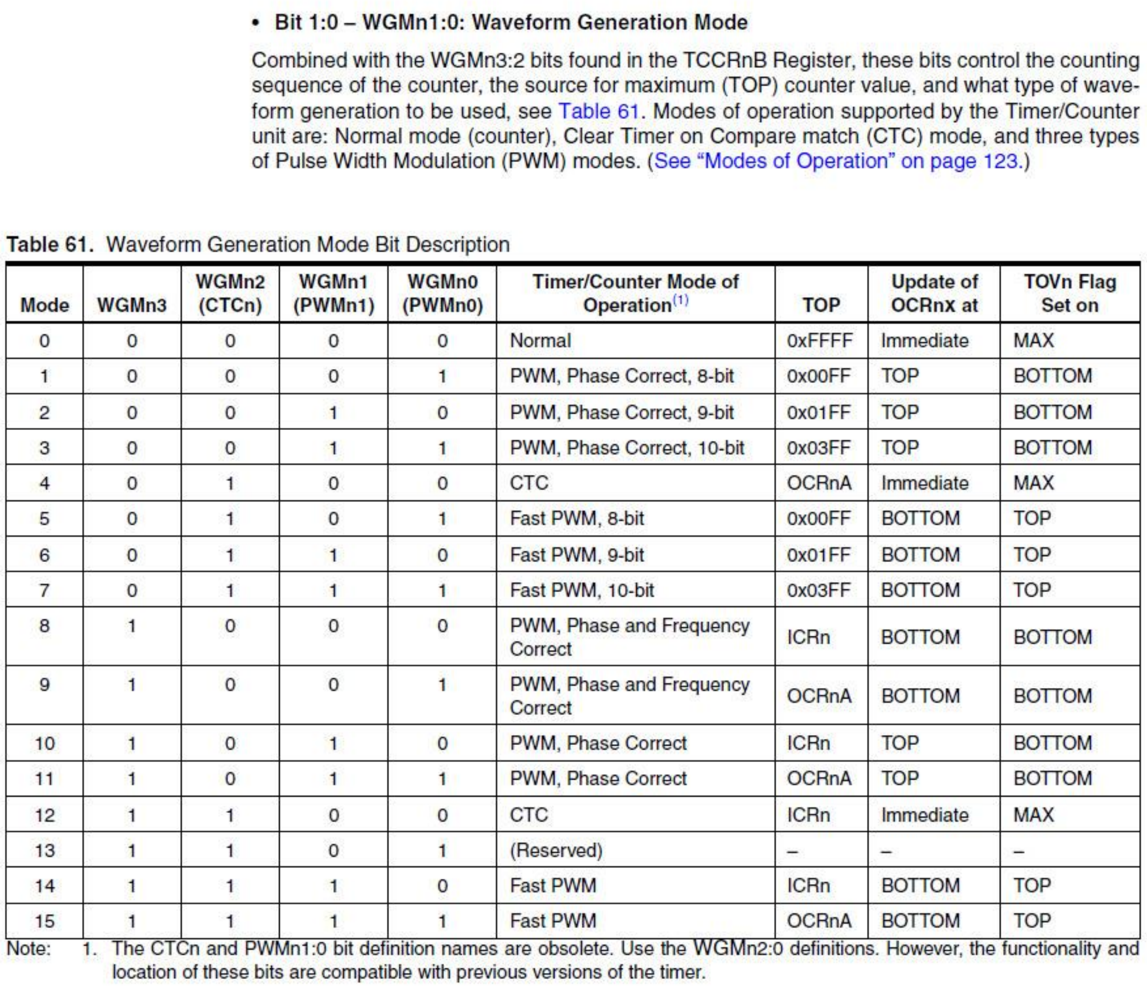
Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See "Fast PWM Mode" on page 124. for more details.

Table 59 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct and frequency correct PWM mode.

Table 60. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGMn3:0 = 9 or 11: Toggle OCnA on Compare Match, OCnB/OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when downcounting.
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when downcounting.

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1//COMnC1 is set. See "Phase Correct PWM Mode" on page 126. for more details.



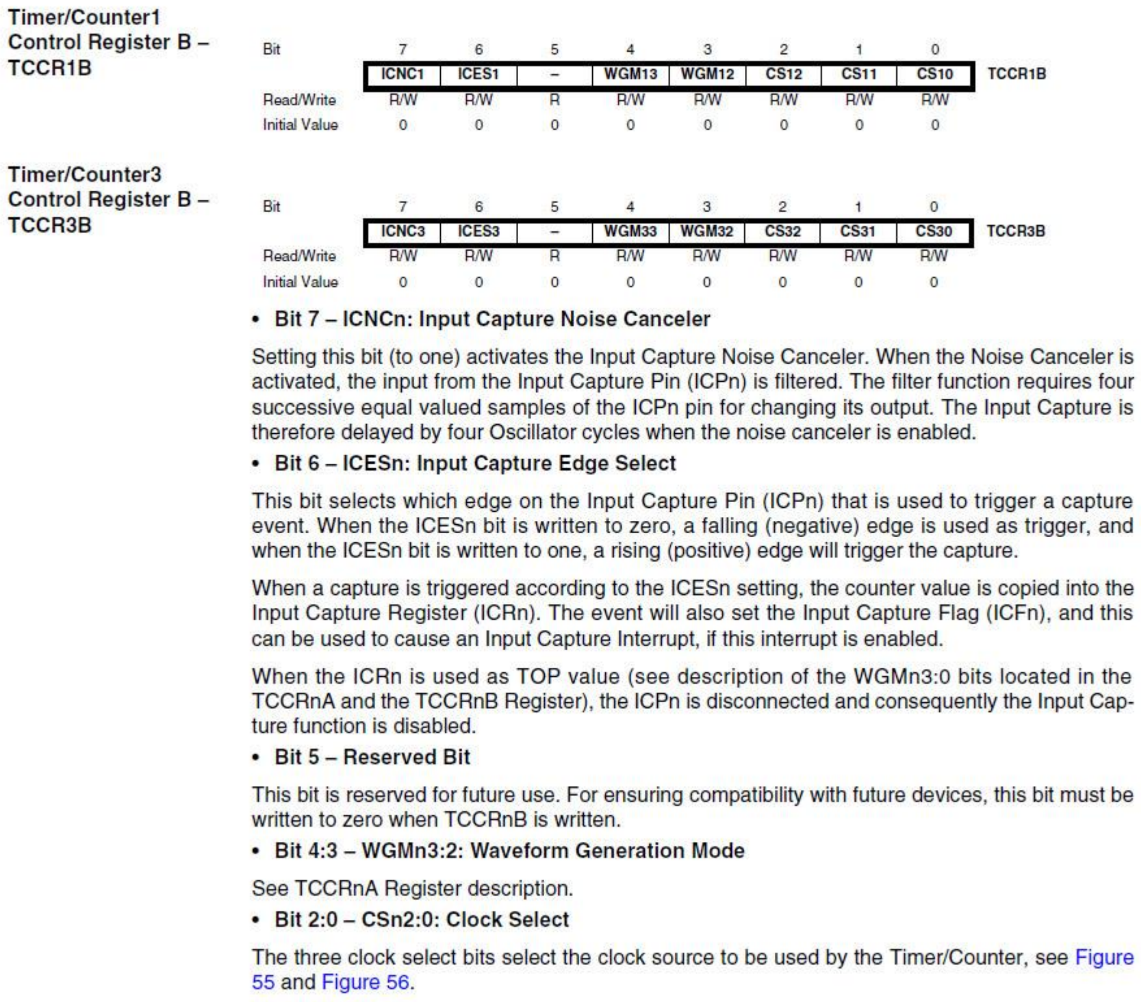
- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 61. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See “Modes of Operation” on page 123.)

Table 61. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation[1]	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.



Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	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	

Timer/Counter3 Control Register B – TCCR3B

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
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	

- **Bit 7 – ICNCn: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICESn: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICPn) that is used to trigger a capture event. When the ICESn bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the Input Capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**

See TCCRnA Register description.

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 55 and Figure 56.

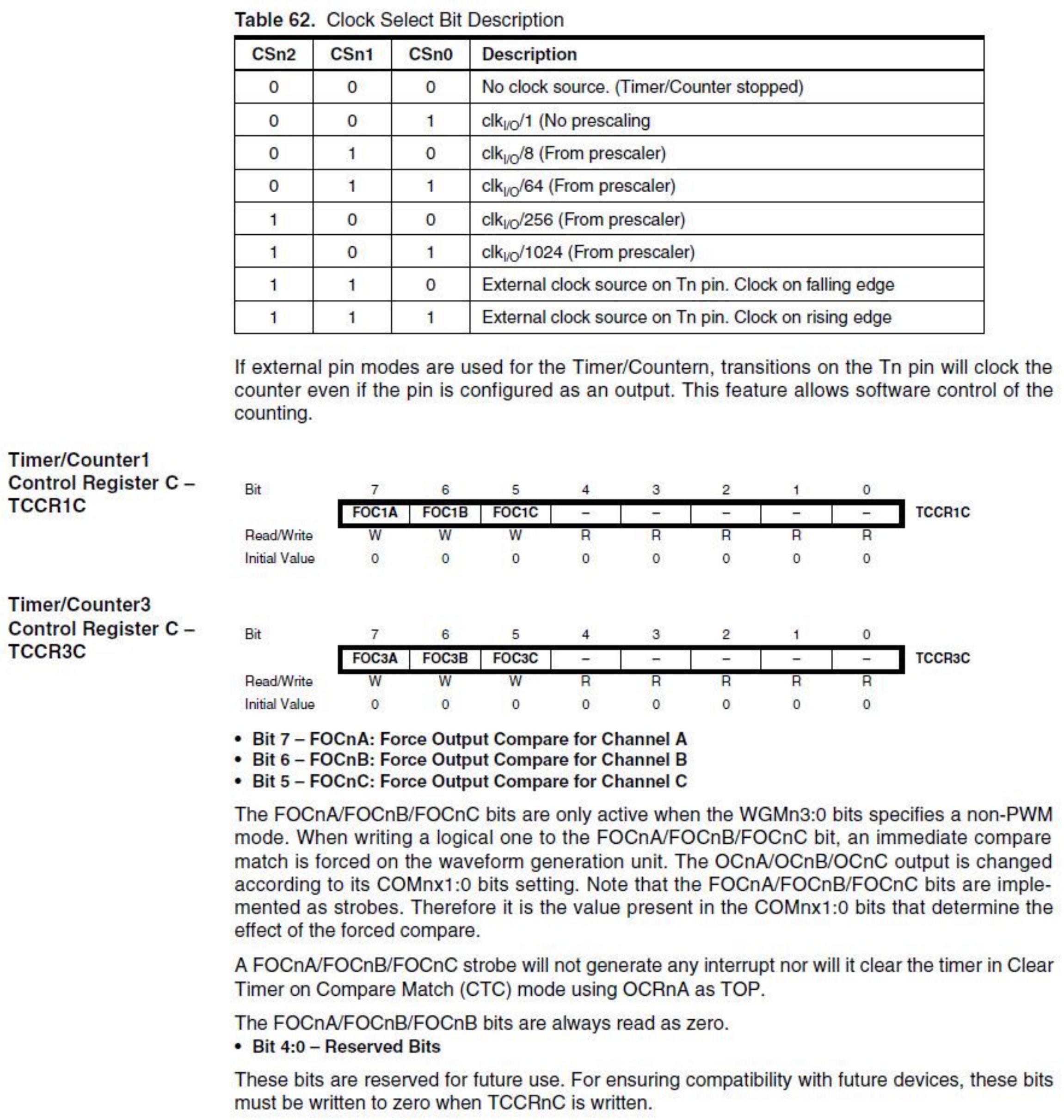


Table 62. Clock Select Bit Description

CSn2	CSn1	CSn0	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 Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Timer/Counter1 Control Register C – TCCR1C

Bit	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter3 Control Register C – TCCR3C

Bit	7	6	5	4	3	2	1	0	
	FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

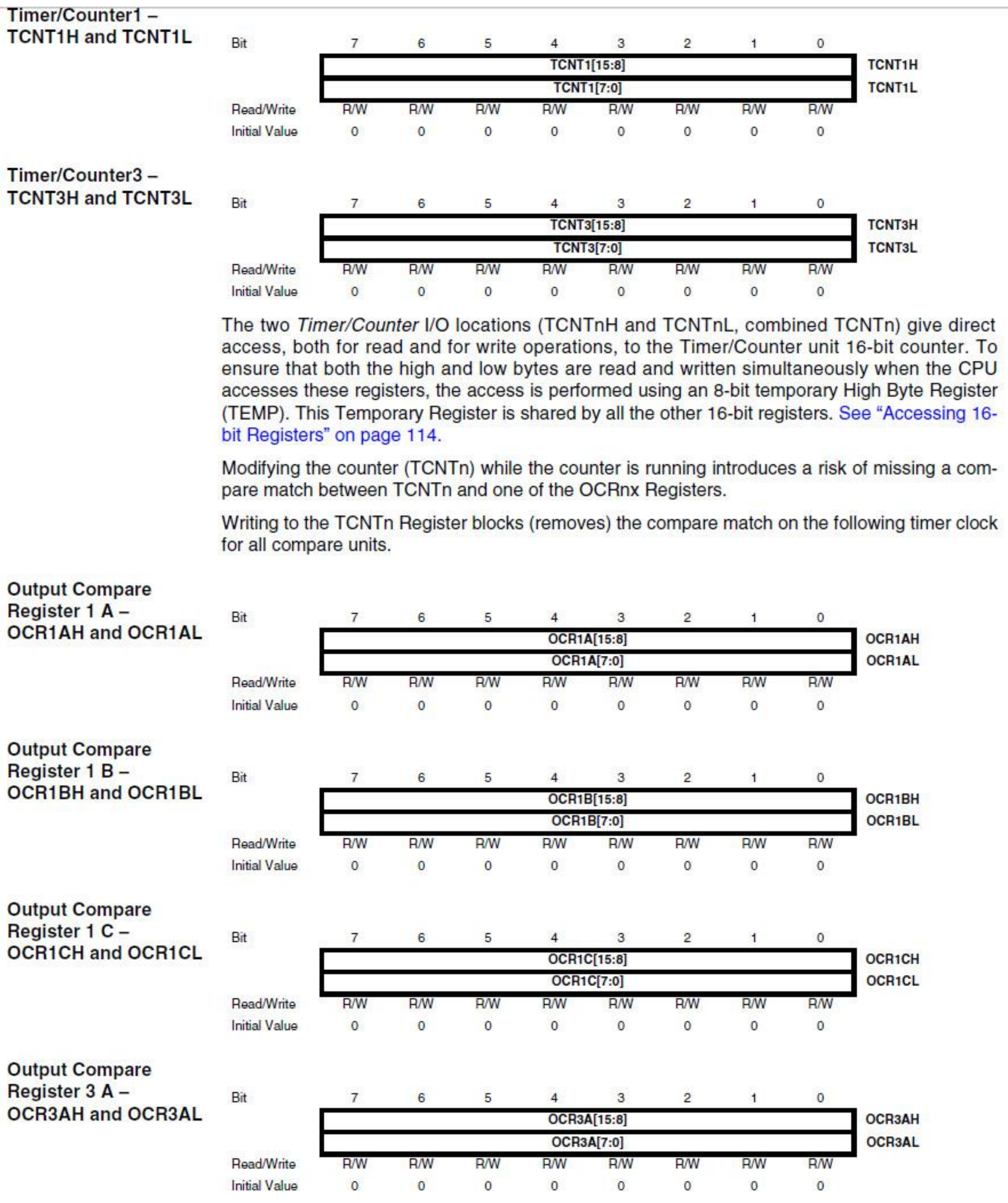
The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnx1:0 bits setting. Note that the FOCnA/FOCnB/FOCnC bits are implemented as strobes. Therefore it is the value present in the COMnx1:0 bits that determine the effect of the forced compare.

A FOCnA/FOCnB/FOCnC strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCRnA as TOP.

The FOCnA/FOCnB/FOCnB bits are always read as zero.

- **Bit 4:0 – Reserved Bits**

These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be written to zero when TCCRnC is written.



Timer/Counter1 – TCNT1H and TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

Timer/Counter3 – TCNT3H and TCNT3L

Bit	7	6	5	4	3	2	1	0	
	TCNT3[15:8]								TCNT3H
	TCNT3[7:0]								TCNT3L
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	

The two *Timer/Counter* I/O locations (TCNTnH and TCNTnL, combined TCNTn) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This Temporary Register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 114.

Modifying the counter (TCNTn) while the counter is running introduces a risk of missing a compare match between TCNTn and one of the OCRnx Registers.

Writing to the TCNTn Register blocks (removes) the compare match on the following timer clock for all compare units.

Output Compare Register 1 A – OCR1AH and OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
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	

Output Compare Register 1 B – OCR1BH and OCR1BL

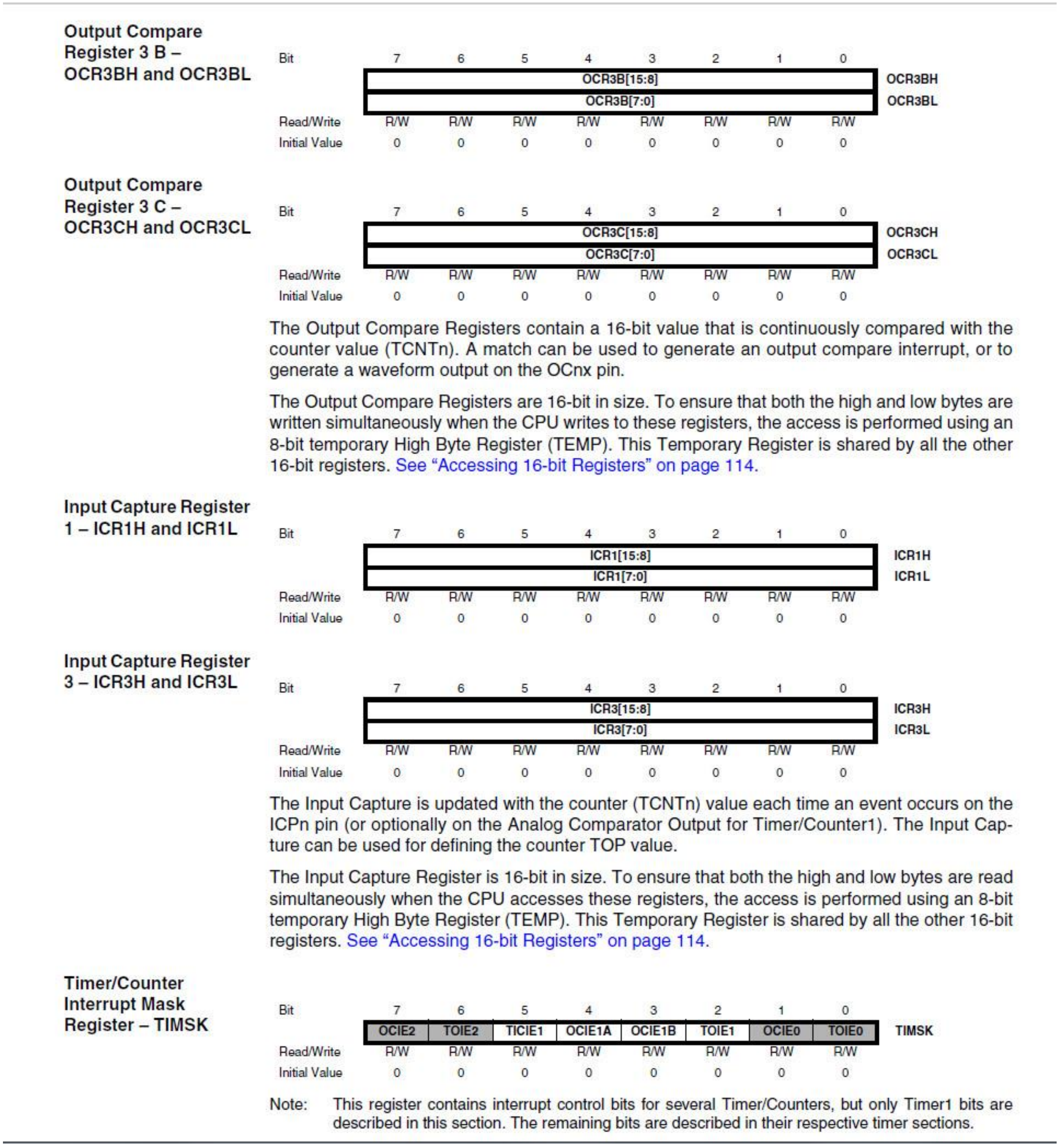
Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Output Compare Register 1 C – OCR1CH and OCR1CL

Bit	7	6	5	4	3	2	1	0	
	OCR1C[15:8]								OCR1CH
	OCR1C[7:0]								OCR1CL
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	

Output Compare Register 3 A – OCR3AH and OCR3AL

Bit	7	6	5	4	3	2	1	0	
	OCR3A[15:8]								OCR3AH
	OCR3A[7:0]								OCR3AL
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	



Output Compare Register 3 B – OCR3BH and OCR3BL

Bit	7	6	5	4	3	2	1	0	
	OCR3B[15:8]								OCR3BH
	OCR3B[7:0]								OCR3BL
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	

Output Compare Register 3 C – OCR3CH and OCR3CL

Bit	7	6	5	4	3	2	1	0	
	OCR3C[15:8]								OCR3CH
	OCR3C[7:0]								OCR3CL
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	

The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNTn). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OCnx pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This Temporary Register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 114.

Input Capture Register 1 – ICR1H and ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
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	

Input Capture Register 3 – ICR3H and ICR3L

Bit	7	6	5	4	3	2	1	0	
	ICR3[15:8]								ICR3H
	ICR3[7:0]								ICR3L
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	

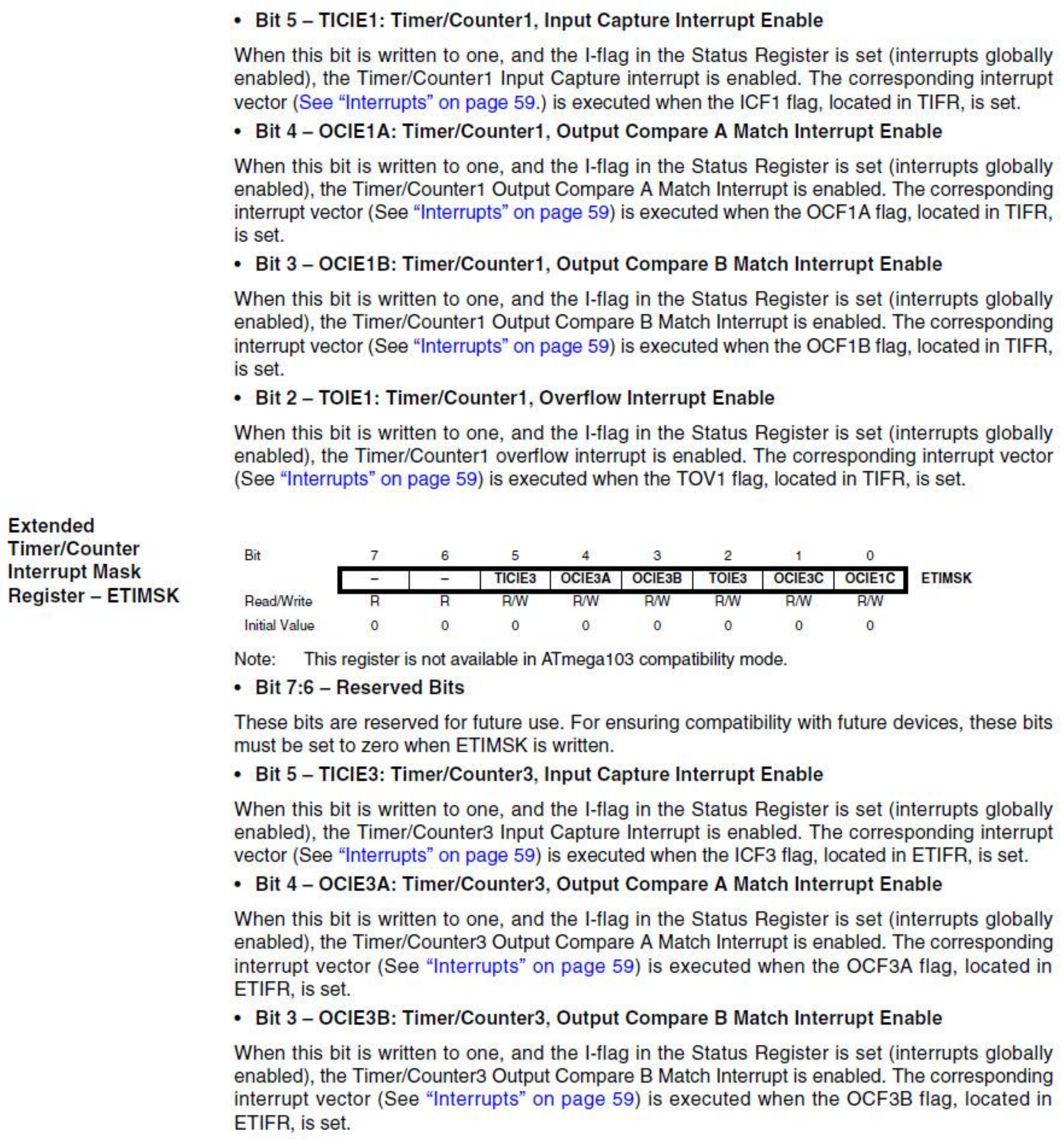
The Input Capture is updated with the counter (TCNTn) value each time an event occurs on the ICPn pin (or optionally on the Analog Comparator Output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This Temporary Register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 114.

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

Note: This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.



- **Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59.) is executed when the ICF1 flag, located in TIFR, is set.

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF1A flag, located in TIFR, is set.

- **Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF1B flag, located in TIFR, is set.

- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the TOV1 flag, located in TIFR, is set.

Extended Timer/Counter Interrupt Mask Register – ETIMSK

Bit	7	6	5	4	3	2	1	0	
	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	ETIMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: This register is not available in ATmega103 compatibility mode.

- **Bit 7:6 – Reserved Bits**

These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be set to zero when ETIMSK is written.

- **Bit 5 – TICIE3: Timer/Counter3, Input Capture Interrupt Enable**

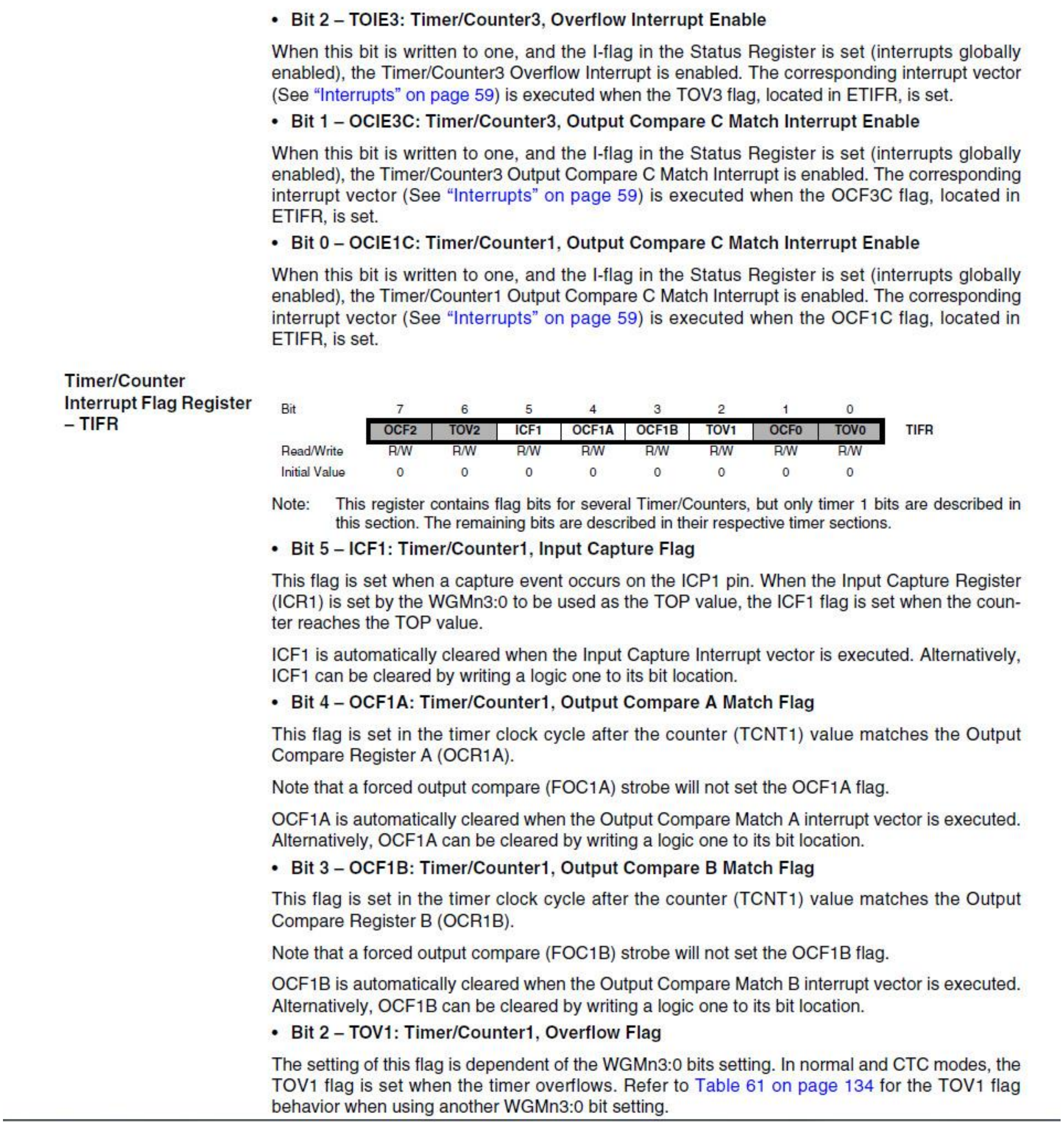
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter3 Input Capture Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the ICF3 flag, located in ETIFR, is set.

- **Bit 4 – OCIE3A: Timer/Counter3, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter3 Output Compare A Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF3A flag, located in ETIFR, is set.

- **Bit 3 – OCIE3B: Timer/Counter3, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter3 Output Compare B Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF3B flag, located in ETIFR, is set.



- **Bit 2 – TOIE3: Timer/Counter3, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter3 Overflow Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the TOV3 flag, located in ETIFR, is set.

- **Bit 1 – OCIE3C: Timer/Counter3, Output Compare C Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter3 Output Compare C Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF3C flag, located in ETIFR, is set.

- **Bit 0 – OCIE1C: Timer/Counter1, Output Compare C Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare C Match Interrupt is enabled. The corresponding interrupt vector (See "Interrupts" on page 59) is executed when the OCF1C flag, located in ETIFR, is set.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

Note: This register contains flag bits for several Timer/Counters, but only timer 1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGMn3:0 to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a forced output compare (FOC1A) strobe will not set the OCF1A flag.

OCF1A is automatically cleared when the Output Compare Match A interrupt vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a forced output compare (FOC1B) strobe will not set the OCF1B flag.

OCF1B is automatically cleared when the Output Compare Match B interrupt vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 2 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In normal and CTC modes, the TOV1 flag is set when the timer overflows. Refer to Table 61 on page 134 for the TOV1 flag behavior when using another WGMn3:0 bit setting.

Definisjoner fra <avr/signal.h>:

SIG_INTERRUPT0	– eksternt avbrudd 0
SIG_INTERRUPT1	– eksternt avbrudd 1
SIG_INTERRUPT2	– eksternt avbrudd 2
SIG_INTERRUPT3	- eksternt avbrudd 3
SIG_INTERRUPT4	– eksternt avbrudd 4
SIG_INTERRUPT5	– eksternt avbrudd 5
SIG_INTERRUPT6	- eksternt avbrudd 6
SIG_INTERRUPT7	- eksternt avbrudd 7
SIG_OUTPUT_COMPARE2	– output compare timer 2
SIG_OVERFLOW2	– overflow timer 2
SIG_INPUT_CAPTURE1	– input capture timer 1
SIG_OUTPUT_COMPARE1A	– output compare A timer 1
SIG_OUTPUT_COMPARE1B	– output compare B timer 1
SIG_OVERFLOW1	– timer 1 overflow
SIG_OUTPUT_COMPARE0	– output compare timer 0
SIG_OVERFLOW0	- overflow timer 0
SIG_SPI	- SPI avbrudd
SIG_UART_RECV	- UART datamottak klart
SIG_UART1_RECV	- UART1 datamottak klart
SIG_UART_DATA	- UART dataregister klart for nye data
SIG_UART1_DATA	- UART1 dataregister klart for nye data
SIG_UART_TRANS	- UART datautsendelse ferdig
SIG_UART1_TRANS	- UART1 datautsendelse ferdig
SIG_ADC	- AD-omforming ferdig
SIG_EEPROM	- EEPROM klar
SIG_COMPARATOR	- Analog komparator klar

Uavhengig av hvilken avbruddstype man benytter er det svært viktig at avbruddsfunksjonene er så korte som mulig. Løkker for å vente på statussignaler eller andre former for venting er forbudt. Det er svært viktig at avbrudds-funksjonen raskt overlater styringen til hovedprogrammet igjen, slik at andre avbruddskilder eventuelt kan betjenes hurtig.

2.2.1.1 SIGNAL

```
SIGNAL (SIG_NAME)   // SIG_NAME hentes fra <avr/signal.h>
{
   // Her kommer instruksjonene
   // som skal utføres
   // i avbruddsfunksjonen.
}
```

Denne avbruddsfunksjonen utføres altså med avbruddsystemet avslått, og vil i de fleste studentprosjekter være den foretrukne typen.

Typisk plasseres avbruddsfunksjonen(e) foran *main*() i programmet. Angivelsen av avbruddstypen (SIG_NAME) hentes fra listen av definisjoner fra <avr/signal.h>.

2.2.1.2 INTERRUPT

```
INTERRUPT (SIG_NAME) //SIG_NAME hentes fra <avr/signal.h>
{
   // Her kommer instruksjonene
   // som skal utføres
   // i avbruddsfunksjonen.
}
```

Denne avbruddsfunksjonen utføres altså med avbruddsystemet aktivt. Dette gir mer fleksible men også mer uoversiktlige løsninger. Det anbefales at man forsøker å benytte avbruddstypen SIGNAL så langt dette er mulig.