CHAPTER 15

INPUT CAPTURE AND WAVE GENERATION IN AVR

OBJECTIVES

Upon completion of this chapter, you will be able to:

- >> **Understand the compare and capture features of the AVR**
- >> **Generate pulses with different frequencies**
- >> **Explain how the wave generators of timers work**
- >> **Explain the different operation modes of Timer0 and Timer1**
- >> **Explain how the capture feature of Timer1 works**
- >> **Code programs for the capture feature in Assembly and C**

In Chapter 9, you learned how to use AVR timers to generate delay and count external events. AVR timers have other features as well. They can be used for generating different square waves or capturing events and measuring the frequency and duty cycle of waves. These usages are discussed in this chapter and Chapter 16. In Sections 15.1 and 15.2 you learn to generate waves using 8-bit and 16-bit timers, respectively. In Section 15.3 you learn to capture events and measure the frequency and duty cycle of waves. You can find the C versions of the programs in Section 15.4.

SECTION 15.1: WAVE GENERATION USING 8-BIT TIMERS

Examine Figure 15-1. As mentioned in Chapter 9, for each timer there is, at least, an OCRn register (like OCR0 for Timer0). The value of this register is constantly compared with the TCNTn register, and when a match occurs, the OCFn flag will be set to high.

As shown in Figures 15-1 and 15-2, in each AVR timer there is a waveform generator. The waveform generator can generate waves on the OCn pin. The WGMn and COMn bits of the TCCR register determine how the waveform generator works. When the TCNTn register reaches Top or Bottom or compare match

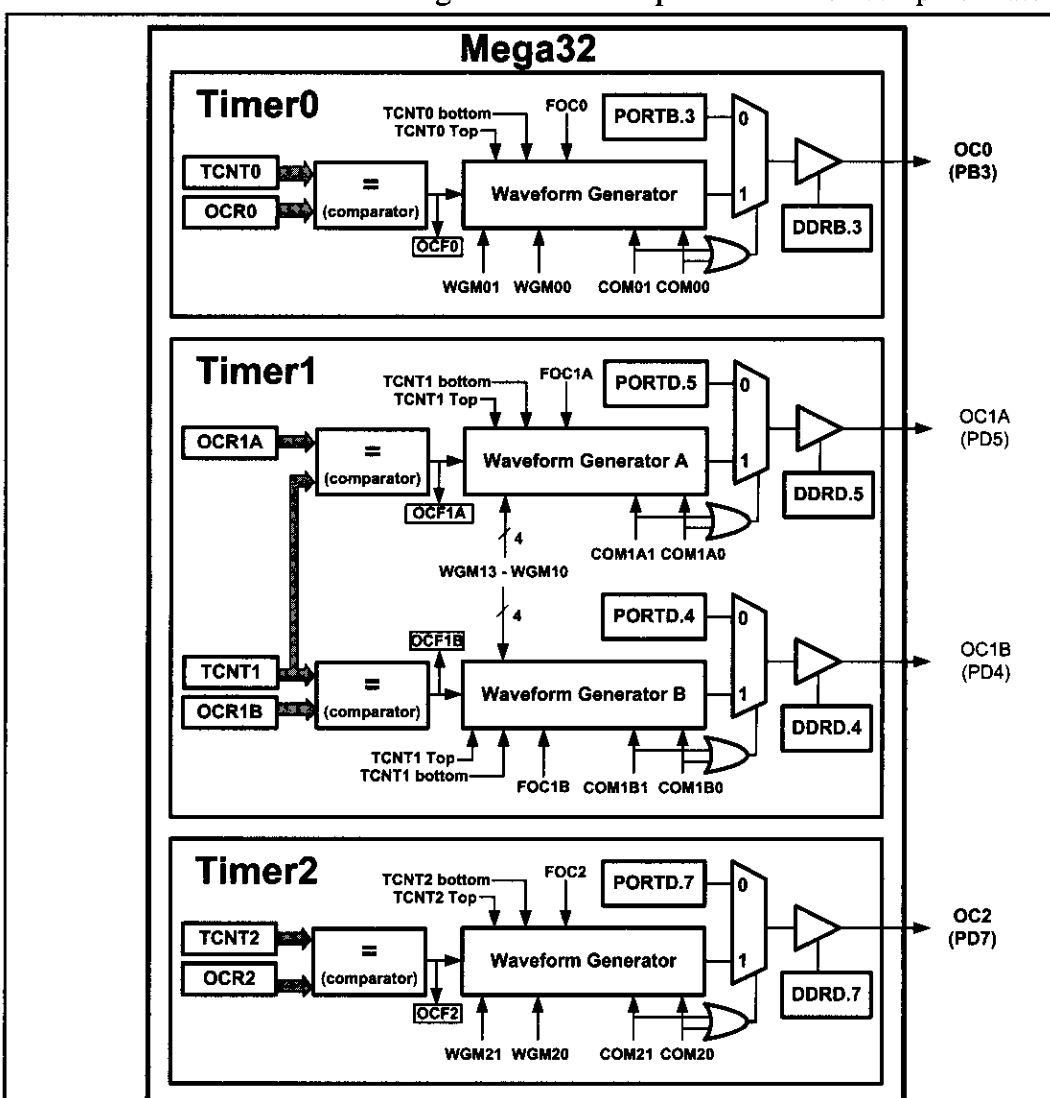


Figure 15-1. Waveform Generators in ATmega32

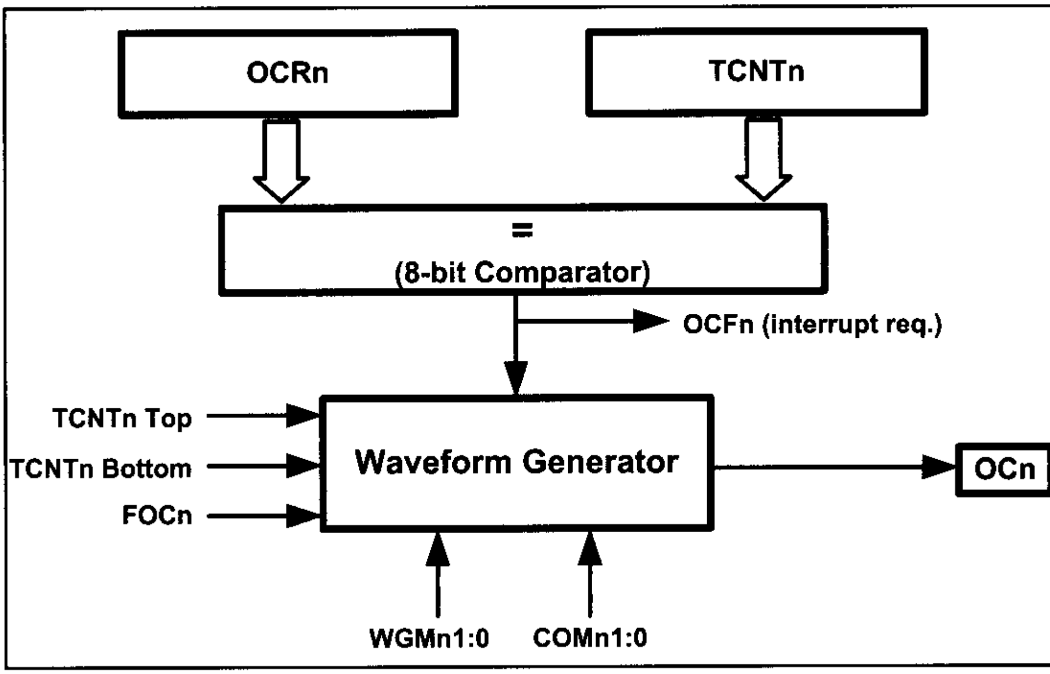


Figure 15-2. Waveform Generator

occurs, the waveform generator is informed. Then the waveform generator changes the state of the OC0 pin according to the mode of the timer (WGM01:00 bits of the TCCR0 register) and the COM01 (Compare Output Mode) and COM00 bits. See Figure 15-4.

In ATmega32/ATmega16, OC0 is the alternative function of PB3. In other words, the PB3 functions as an I/O port when both COM01 and COM00 are zero. Otherwise, the pin acts as a wave generator pin controlled by a waveform generator. See Figures 15-1 and 15-3. Notice that, since the DDR register represents the direction of the I/O pin, we should set the OC0 pin as an output pin when we want to use it for generating waves.

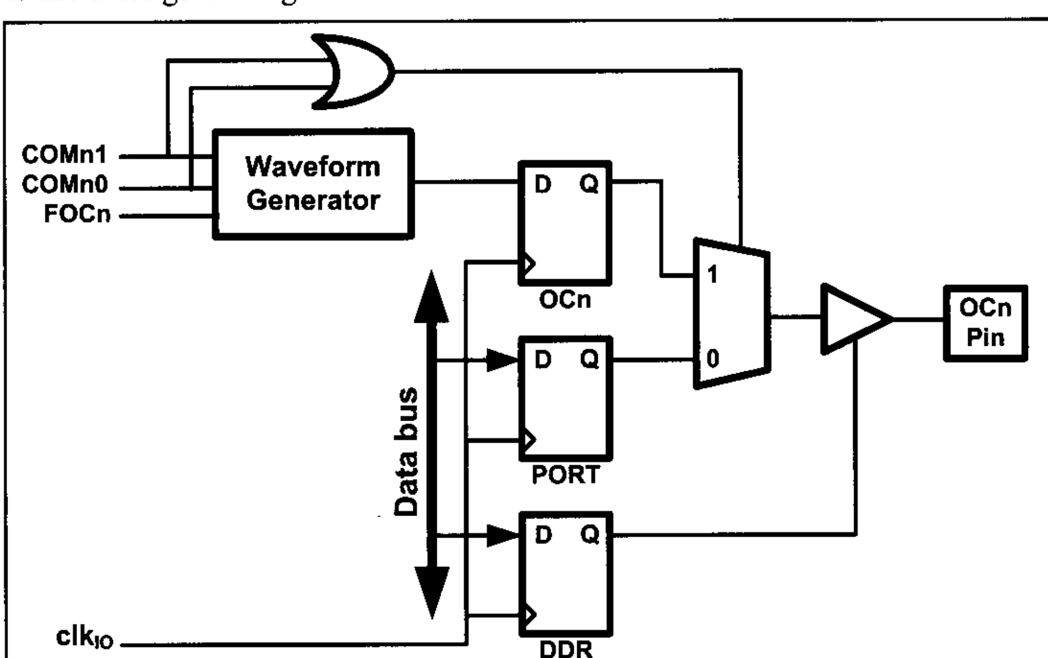


Figure 15-3. DDR Register and Waveform Generator

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force Output compare: Writing 1 to it forces the wave generator to act as if a compare match has occurred.

WGM01:00	D6	D3	Timer0 mode selector bits
	0	0	Normal
	0	1	CTC (Clear Timer on Compare match)
	1	0	PWM, phase correct
	1	1	Fast PWM

COM01:00 D5 D4 Compare Output Mode; The table shows what the wave generator does on compare match when the timer is in Normal or CTC mode:

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

CS02:00	D2D1D0	Timer0 clock selector
	0 0 0	No clock source (Timer/Counter stopped)
	0 0 1	clk (no prescaling)
	0 1 0	clk / 8
	0 1 1	clk / 64
	1 0 0	clk / 256
	1 0 1	clk / 1024
	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

Figure 15-4. TCCR0 (Timer/Counter Control Register) Register

Wave generation Normal and CTC modes

When Timer0 is in CTC (WGM01:0 = 10) or Normal (WGM01:0 = 00) mode after a compare match occurs, the OC0 pin can perform one of the following actions, depending on the value of the COM01:0 bits:

(a) Remain unaffected
(b) Toggle the OC0 pin
(c) Clear (Drive low) the OC0 pin
(d) Set (Drive high) the OC0 pin

We use the COM01 and COM00 bits to select one of the above actions; as shown in Figure 15-4. See Example 15-1.

Notice that in the CTC mode, when the compare match occurs, the timer value will be set to zero, while in the Normal mode the timer counts up until it reaches the top value.

Setting (driving high) the OC0 pin

There are many applications for the compare feature. One application can be to count the number of people going through a door and closing the door when a certain number is reached. See Example 15-2.

Example 15-1

Using Figure 15-4, find the TCCR0 register value to:
(a) Set high the OC0 pin upon match. Use external clock, falling edge, and Normal mode.
(b) Toggle the OC0 pin upon match. Use external clock, falling edge, and CTC mode.

Solution:

(a) TCCR0 =

0	0	1	1	0	1	1	0
FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**

(b) TCCR0 =

0	0	0	1	1	1	1	0
FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**

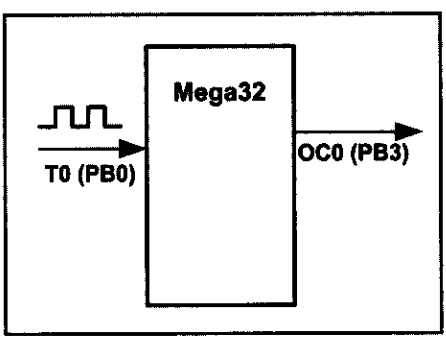
Example 15-2

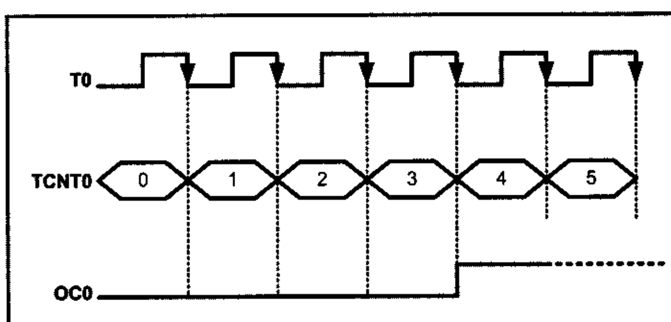
Write a program that (a) after 4 external clocks turns on an LED connected to the OC0 pin, (b) toggles the OC0 pin every 4 pulses.

Solution:

(a)

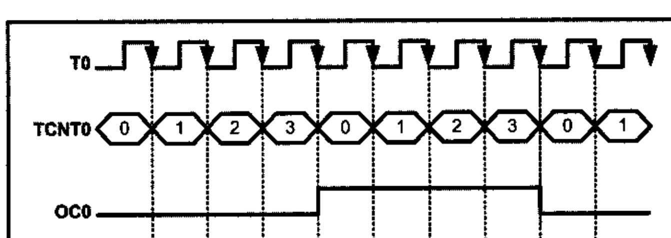
```
.INCLUDE "M32DEF.INC"
      CBI   DDRB,0            ;PB0(T0) pin as input
      SBI   DDRB,3            ;PB3(OC0) pin as output
      LDI   R20, 3
      OUT   OCR0,R20          ;OCR0 = 3  the final count
      LDI   R20, 0
      OUT   TCNT0,R20         ;TCNT0 = 0
      LDI   R20,0x36          ;external clk, Normal mode, set OC0
      OUT   TCCR0,R20         ;load TCCR0 and start counting
HERE: RJMP  HERE
```





(b)

```
.INCLUDE "M32DEF.INC"
      CBI   DDRB,0
      SBI   DDRB,3
      LDI   R20, 3
      OUT   OCR0,R20
      LDI   R20, 0
      OUT   TCNT0,R20
      LDI   R20,0x1E          ;external clk, CTC mode, toggle OC0
      OUT   TCCR0,R20         ;load TCCR0 and start counting
HERE: RJMP  HERE
```



Notice that there is no need to monitor the OCF0 flag, which means the AVR can do other tasks.

Generating square waves

To generate square waves we can set the timer to Normal mode or CTC mode and set the COM bits to the toggle mode (COM01:00 = 01). The OC0 pin will be toggled on each compare match and a square wave will be generated. See Figure 15-5. See Examples 15-3 and 15-4.

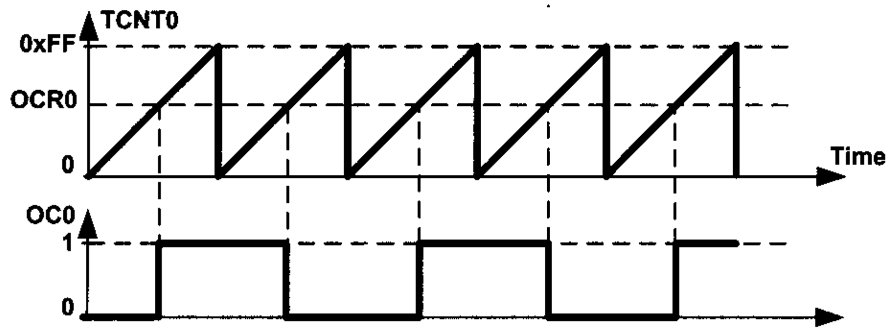


Figure 15-5. Generating Square Wave Using Normal

Example 15-3

Find the value for TCCR0 if we want to program Timer0 as a Normal mode square wave generator and no prescaler.

Solution:

TCCR0 =

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
0	0	0	1	0	0	0	1

Example 15-4

Assuming XTAL = 8 MHz, calculate the frequency of the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,3     ;PB3 as output
      LDI   R22,100
      OUT   OCR0,R22   ;set the match value
      LDI   R22,0x11   ;COM01:00 = Toggle, Mode = Normal, no prescaler
      OUT   TCCR0,R22 ;load TCCR0 and start counting
HERE: RJMP  HERE
```

Solution:

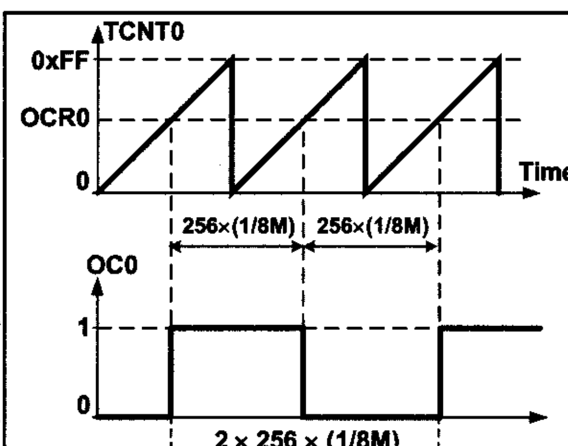
There are 256 clocks between two consecutive matches. Therefore

$T_{timer\ clock} = 1/8\ \text{MHz} = 0.125\ \mu s$

$T_{wave} = 2 \times 256 \times 0.125\ \mu s = 64\ \mu s$

$F_{wave} = 1/64\ \mu s = 15{,}625\ \text{Hz} = 15.625\ \text{kHz}$

Note: In Normal mode, when match occurs, the OC0 pin toggles and the timer continues to count up until it reaches the top value.



Generating square waves using CTC

The CTC mode is better than Normal mode for generating square waves, since the frequency of the wave can be easily adjusted using the OCR0 register. See Figure 15-6. In CTC mode, when OCR0 has a lower value, compare match occurs earlier and the period of the generated wave is smaller (higher frequency). When the OCR0 has a higher value, compare match occurs later and the period of the wave is longer (lower frequency).

Notice that in the CTC mode, when the compare match occurs, the timer value will be set to zero, while in the Normal mode the timer counts up until it reaches the top value. See Examples 15-5 through 15-7.

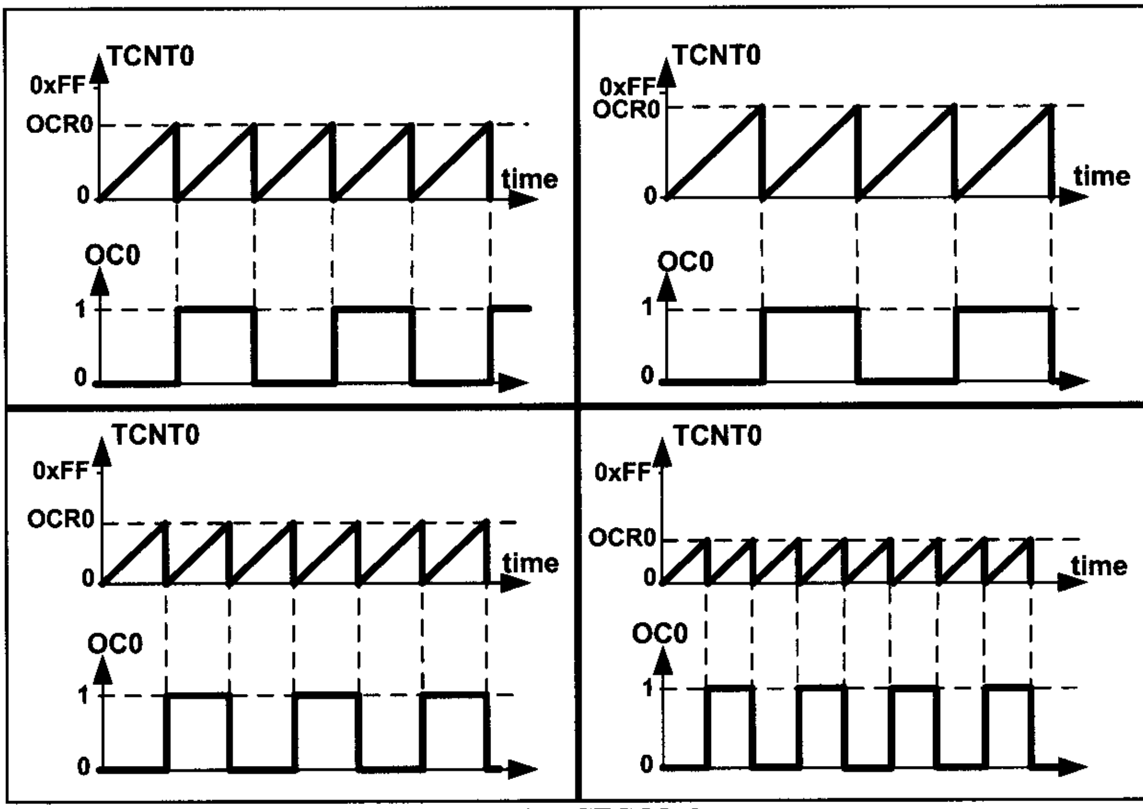


Figure 15-6. Generating Square Wave Using CTC Mode

Example 15-5

Find the value for TCCR0 if we want to program Timer0 as a CTC mode square wave generator and no prescaler.

Solution:

WGM01:00 = 10 = CTC mode
COM01:00 = 01 = Toggle
CS02:00 = 001 = No prescaler
FOC0 = 0

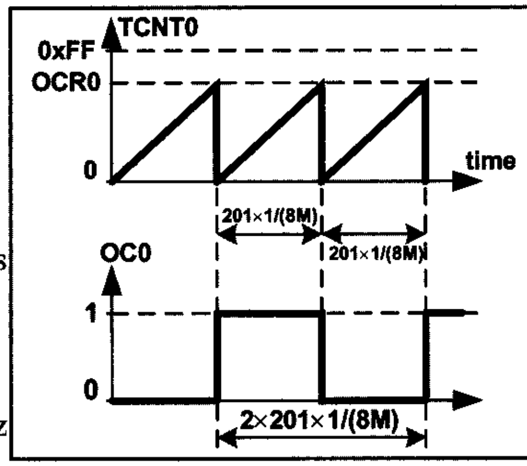
TCCR0 =

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
0	0	0	1	1	0	0	1

Example 15-6

Assuming XTAL = 8 MHz, calculate the frequency of the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,3
      LDI   R20,0x19   ;COM01:00 = Toggle, Mode = CTC, no prescaler
      OUT   TCCR0,R20
      LDI   R22,200
      OUT   OCR0,R22
HERE: RJMP  HERE
```



Solution:

Between two consecutive matches it takes 200 + 1 = 201 clocks and

$T_{timer\ clock}$ = 1/8 MHz = 0.125 μs

T_{wave} = 2 × 201 × 0.125 μs = 50.25 μs

F_{wave} = 1/50.25 μs = 19,900 Hz = 19.900 kHz

Example 15-7

In Example 15-6, calculate the frequency of the wave generated in each of the following cases:

(a) OCR0 is loaded with 50

(b) XTAL = 4 MHz and OCR0 is loaded with 95

(c) prescaler is 8, XTAL = 1 MHz, OCR0 = 150

(d) prescaler is N, XTAL = F_{OSC}, OCR0 = X

Solution:

(a) 50 + 1 = 51 clocks and $T_{timer\ clock}$ = 0.125 μs ➔ T_{wave} = 2 × 51 × 0.125 μs = 12.75 μs

F_{wave}= 1 / 50.25 μs = 19,900 Hz = 19.900 kHz

(b) 95 + 1 = 96 clocks and $T_{timer\ clock}$ = 1 / 4 MHz = 0.25 μs

➔ T_{wave} = 2 × 96 × 0.25 μs = 48 μs ➔ F_{wave} = 1 / 48 μs = 20,833 Hz = 20.833 kHz

(c) 150 + 1 = 151 clocks and $T_{timer\ clock}$ = 8 × 1 / 1 MHz = 8 μs

➔ T_{wave} = 2 ×151 × 8 μs = 2416 μs ➔ F_{wave} = 1 / 2416 μs = 413.9 Hz

(d) X + 1 clocks and $T_{timer\ clock}$= N × 1/F_{OSC} = N/F_{OSC}

➔ T_{wave} = 2 × (X + 1) × N / F_{OSC} ➔ F_{wave} = 1 / T_{wave} = F_{OSC} / [2N(X + 1)]

Generating pulses using CTC mode

When a timer is in the CTC mode and COM is in the toggle mode, the value of the OCRn represents how many clocks it counts before it toggles the pin. This way, we can generate different pulses by loading different values into the OCRn register. See Figure 15-7 and Example 15-8.

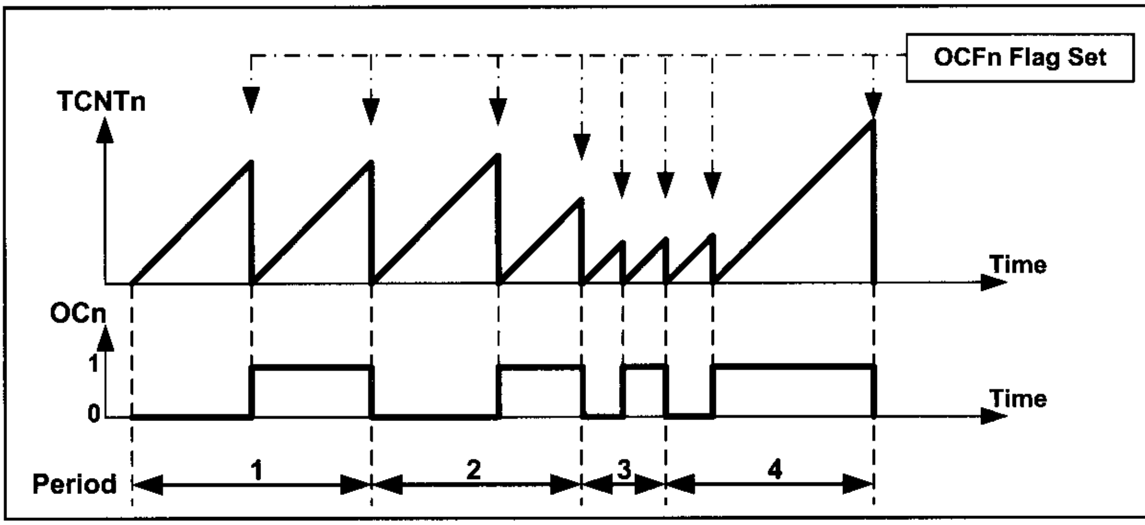


Figure 15-7. Generating Different Pulses Using CTC and Toggle Modes

Example 15-8

Assuming XTAL = 1 MHz, draw the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,3
BEGIN:LDI   R20,69
      OUT   OCR0,R20   ;OCR0 = 69
      LDI   R20,0x19
      OUT   TCCR0,R20  ;CTC, no prescaler, set on match
L1:   IN    R20,TIFR
      SBRS  R20,OCF0   ;skip next instruction if OCF0 = 1
      RJMP  L1
      LDI   R16,1<<OCF0
      OUT   TIFR,R16   ;clear OCF0
      LDI   R20,99
      OUT   OCR0,R20   ;OCR0 = 99
      LDI   R20,0x29
      OUT   TCCR0,R20  ;CTC, no prescaler, clear on match
L2:   IN    R20,TIFR
      SBRS  R20,OCF0   ;skip next instruction if OCF0 = 1
      RJMP  L2
      LDI   R16,1<<OCF0 ;clear OCF0
      OUT   TIFR,R16
      RJMP  BEGIN
```

Solution:

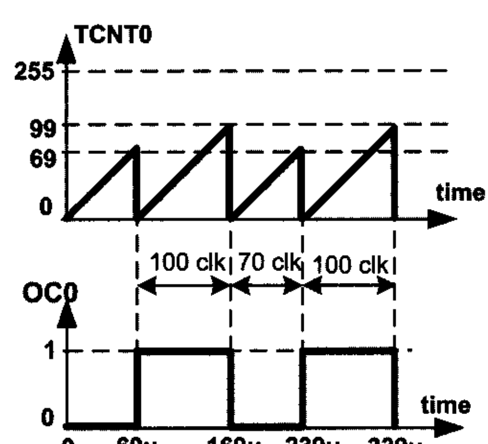
$T_{timer\ clock} = 1 / 1\ \text{MHz} = 1\ \mu s$

$T_0 = 70 \times 1\ \mu s = 70\ \mu s$

$T_1 = 100 \times 1\ \mu s = 100\ \mu s$

$T_{wave} = 70\ \mu s + 100\ \mu s = 170\ \mu s$

$F_{wave} = 1 / 170\ \mu s = 5882\ \text{Hz}$



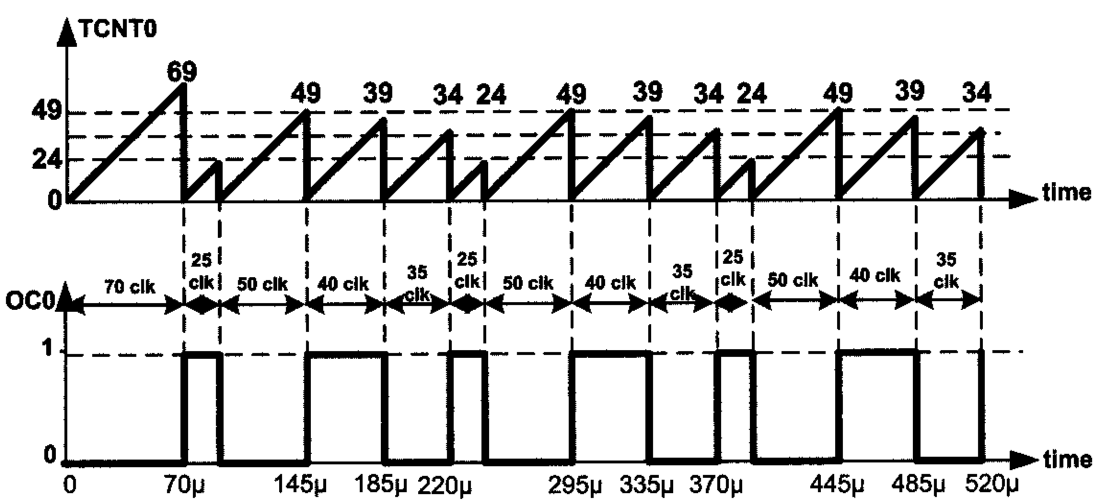
To load values to the OCRn we can use the compare match interrupt as well. Upon a compare match, the pin will be toggled and an interrupt will be invoked. Using the interrupt we can define the duration that OCn will be in the current state by loading a proper value into the OCR0 register. See Figure 15-7 and Example 15-9.

Example 15-9

Assuming XTAL = 1 MHz, draw the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
.ORG  0x0
      RJMP  MAIN
.ORG  0x14                          ;compare match interrupt vector
      DEC   R29                        ;R29 = R29 - 1
      BRPL  L1                         ;if (R29 >= 0) go to L1
      LDI   R30,WAVE_TABLE<<1          ;Z points to WAVE_TABLE
      LDI   R29,3                      ;R29 = 3
L1:   LPM   R28,Z+                     ;R28 = [Z], Z = Z + 1
      OUT OCR0,R28                     ;OCR0 = 99
      RETI                             ;return from interrupt
WAVE_TABLE:      .DB   24,49,39,34
MAIN: LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)
      OUT   SPL,R20          ;initialize stack
      SBI   DDRB,3           ;PB3 as output
      LDI   R20,69
      OUT   OCR0,R20         ;OCR0 = 69
BEGIN:LDI   R20,0x19
      OUT   TCCR0,R20        ;CTC, no prescaler, toggle on match
      LDI   R20,1<<OCIE0
      OUT   TIMSK,R20        ;activate compare match interrupt
      SEI
HERE: RJMP  HERE
```

Solution:



FOC0 (Force Output Compare) flag

Sometimes you might need to force the waveform generator to act as if a compare match has occurred. This can be done by setting the FOC0 bit of the TCCR0 register. See Example 15-10.

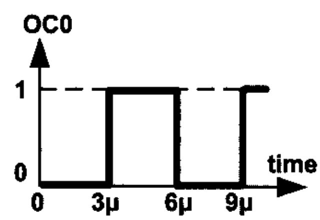
Example 15-10

Assuming XTAL = 1 MHz, draw the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,3
      LDI   R20,0x98
BEGIN:OUT   TCCR0,R20   ;CTC,timer stopped,toggle on match,FOC0=1
      RJMP  BEGIN
```

Solution:

The wave generator is in toggle mode. So, it toggles on compare match. Setting the FOC0 bit causes the wave generator to act as if a real compare match has occurred. The execution of instructions "`OUT TCCR0,R20`" and "`RJMP BEGIN`" takes 1 and 2 clocks, respectively. So, toggle occurs after 1 + 2 = 3 clocks.



Generating waves using Timer2

We can generate waves using Timer2 or any other 8-bit timer the same way as we did using Timer0. We should simply use the proper registers and monitor the associated flag.

As the prescaler values are different in Timer2 we should be careful to load TCCR2 with proper value. For example, if we load 0x14 into TCCR0 the prescaler is 256, whereas loading TCCR2 with 0x14 means a prescaler of 64. See Examples 15-11 and 15-12.

Example 15-11

Rewrite the program of Example 15-4 using Timer2.

Solution:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRD,7      ;OC2 (PD7) as output
      LDI   R22,100
      OUT   OCR2,R22   ;set the match value
      LDI   R22,0x11   ;COM21:20=Toggle, Mode=Normal, no prescaler
      OUT   TCCR2,R22 ;load TCCR2 and start counting
HERE: RJMP  HERE
```

Example 15-12

Rewrite the program of Example 15-6 using Timer2.

Solution:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRD,7        ;OC2 (PD7) as output
      LDI   R22,0x19
      OUT   TCCR2,R22     ;COM21:20 = Toggle, Mode = CTC, no prescaler
      LDI   R22,200
      OUT   OCR2,R22      ;OCR2 = 200
HERE: RJMP HERE
```

Review Questions

1. True or false. In ATmega32, Timer0 has a wave generator.
2. True or false. CTC mode can be used to generate square waves.
3. True or false. To generate waves the OC0 pin must be configured as an input pin.
4. Give the pin number used by the wave generator of Timer0 in ATmega32.

SECTION 15.2: WAVE GENERATION USING TIMER1

In Chapter 9, we discussed Timer1. In this section we first discuss the different modes of Timer1 in more detail and then show how to generate waves using Timer1.

The different modes of Timer1

The WGM13, WGM12, WGM11, and WGM10 bits define the mode of Timer1, as shown in Figure 15-8. Timer1 has 16 different modes. Of these 16 modes, mode 13 is reserved (not implemented). These modes can be categorized into five groups: Normal, CTC, Fast PWM, Phase Correct PWM, and Phase and Frequency Correct PWM. We learned about the operation of the first two categories in Chapter 9; the operation of the other categories will be discussed in this part. Before discussing the operation of the different modes we should define the meaning of Top.

Top in Timer1

Top is the highest value that the TCNT register reaches while counting. In 8-bit timers (e.g., Timer0) the top value is 0xFF except for the CTC mode, whose top can be defined by OCRn. See Figure 15-8. In 16-bit timers such as Timer1 the top values are as follows:

- In Normal mode (mode 0) the top value is 0xFFFF.
- In some modes the top value is fixed and is other than the maximum; the top value can be 0xFF, 0x1FF, or 0x3FF.
- In some other modes the top can be defined by either the OCR1A register or the ICR1 register. See Figure 15-8.

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	

ICNC1 D7 Input Capture Noise Canceller
0 = Input Capture Noise Canceller is disabled
1 = Input Capture Noise Canceller is enabled

ICES1 D6 Input Capture Edge Select
0 = Capture on the falling (negative) edge
1 = Capture on the rising (positive) edge

D5 Not used

WGM13:WGM12 D4 D3 Timer1 mode

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 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	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

CS12:CS10 D2D1D0 Timer1 clock selector

D2 D1 D0	
0 0 0	No clock source (Timer/Counter stopped)
0 0 1	clk (no prescaling)
0 1 0	clk / 8
0 1 1	clk / 64
1 0 0	clk / 256
1 0 1	clk / 1024
1 1 0	External clock source on the T1 pin. Clock on falling edge
1 1 1	External clock source on the T1 pin. Clock on rising edge

Figure 15-8. TCCR1B (Timer 1 Control) Register

CTC mode

As shown in Figure 15-8, modes 4 and 12 operate in the CTC mode. They are almost the same. The only difference between them is that in mode 4, the top value is defined by OCR1A, whereas in mode 12, ICR specifies the top.

As mentioned in Chapter 9, in mode 4, the timer counts up until it reaches OCR1A; then the timer will be cleared and the OCF1A flag will be set as a result of compare match. See Figure 15-9.

In mode 12, the timer counts up until it reaches ICR; then the timer will be cleared and the ICF1 flag will be set, as shown in Figure 15-10. So, in mode 12, the timer works almost the same way as mode 4. See Example 15-13 and compare it with Example 9-22.

In other words, in Normal, CTC, and Fast PWM, the timer counts up until it reaches the top and then rolls over to zero. But the top value is different in the different modes and as a result, different flags are set when the timer rolls over. When the top value is a fixed value, the TOV1 flag is set; when the OCR1A register defines the top, the OCF1 flag will be set; and when the top is defined by the ICR1 register, the ICF1 flag will be set. See Figures 15-9 through 15-11.

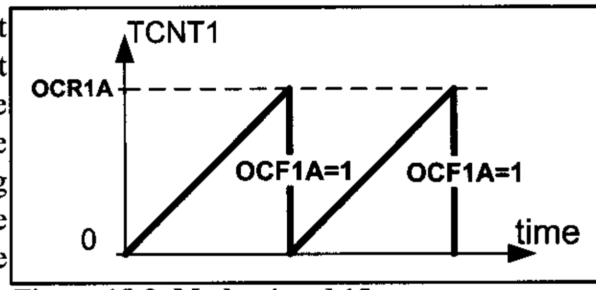


Figure 15-9. Modes 4 and 15

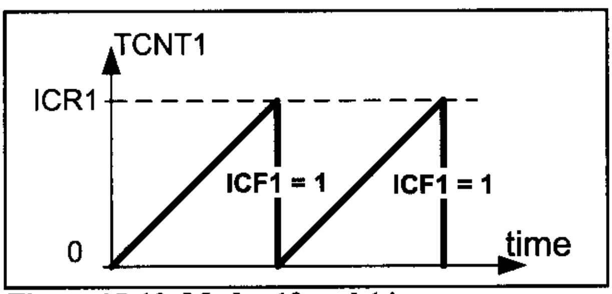


Figure 15-10. Modes 12 and 14

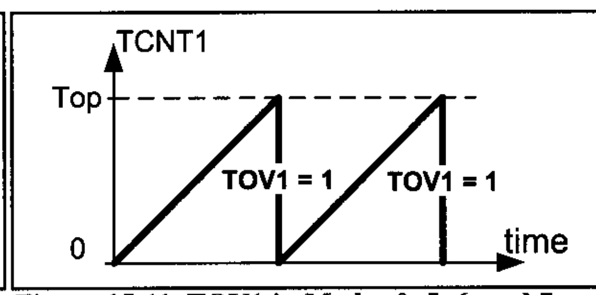


Figure 15-11. TOV1 in Modes 0, 5, 6, and 7

You might find the contents of these two pages confusing. There is no need to memorize the details. All you need to know is how the timer counts in each of the five categories of operations (Normal, CTC, etc.) and how to use the information mentioned in Figure 15-8. The following is a summary:

Counting:

In Normal, CTC, and Fast PWM modes the timer counts up until it reaches the top value. Then the timer rolls over to zero and a flag is set:

- If the top is a fixed value, TOV1 will be set.
- If the OCR1A register represents the top, the OCF1A will be set.
- If the ICR1 register defines the top, the ICF1 will be set.

Highlights of Figure 15-8:

- Column 6 (Timer/Counter Mode of Operation): mentions which of the five operation modes (Normal, CTC, Fast PWM, etc.) it belongs to.
- Column 7 (Top): represents the highest value that the timer reaches while counting; in some modes the top is a fixed value such as 0xFF, 0x1FF, 0x3FFF, and 0xFFFF, while in the others the top value can be determined by the OCR1A or ICR1 register.
- Column 8 is discussed in Chapter 16.

Example 15-13

Rewrite Example 9-27 using the ICR1 flag.

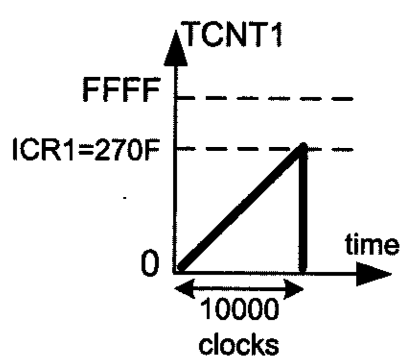
Solution:

To wait 10,000 clocks we should load the ICR1 flag with 10,000 – 1 = 9999 = 0x270F and use mode 14.

```
.INCLUDE "M32DEF.INC"

      LDI   R16,HIGH(RAMEND)  ;initialize stack pointer
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16
      SBI   DDRB,5            ;PB5 as an output
BEGIN:SBI   PORTB,5           ;PB5 = 1
      RCALL DELAY_1ms
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY_1ms
      RJMP  BEGIN

DELAY_1ms:
      LDI   R20,HIGH(9999)
      OUT   ICR1H,R20         ;TEMP = 0x27
      LDI   R20,LOW(9999)
      OUT   ICR1L,R20         ;ICR1L = 0x0F, ICR1H = TEMP
      LDI   R20,0
      OUT   TCNT1H,R20        ;TEMP = 0x0
      OUT   TCNT1L,R20        ;TCNT1L = 0x0, TCNT1H = TEMP
      LDI   R20,0x02
      OUT   TCCR1A,R20        ;WGM11:10 = 10
      LDI   R20,0x19
      OUT   TCCR1B,R20        ;WGM13:12 = 11, CS = CLK, mode = 14
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,ICF1          ;if ICF1 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,1<<ICF1
      OUT   TIFR,R20          ;clear ICF1 flag
      LDI   R19,0
      OUT   TCCR1B,R19        ;stop timer
      OUT   TCCR1A,R19
      RET
```



Waveform generators in Timer1

In examining Figures 15-12 and 15-13 we see that Timer1 has two independent waveform generators: Waveform Generator A and Waveform Generator B.

The compare match between OCR1A and TCNT1 affects Waveform Generator A, and the wave generated by Waveform Generator A shows up on the OC1A pin.

The compare match between OCR1B and TCNT1 affects Waveform Generator B, and the wave generated by Waveform Generator B shows up on the OC1B pin.

The COM1A1 and COM1A0 bits have control over Waveform Generator A; whereas COM1B1 and COM1B0 control Waveform Generator B. All of the COM bits are in the TCCR1A register, as shown in Figure 15-14.

The operation mode of Timer1 (WGM13, WGM12, WGM11, and WGM10 bits of TCCR1A and TCCR1B) affect both generators, as shown in Figures 15-12 and 15-13.

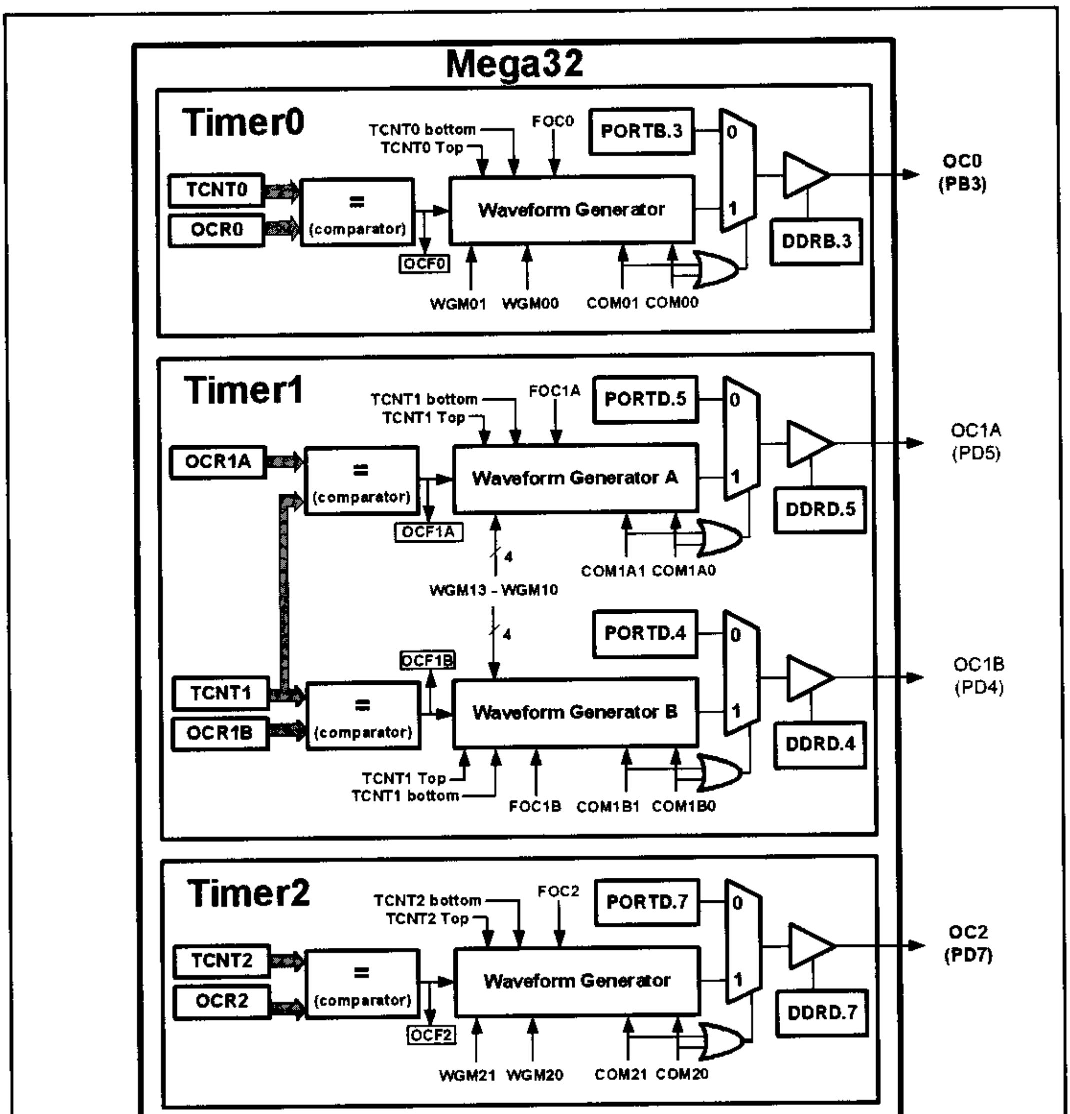


Figure 15-12. Waveform Generators in ATmega32

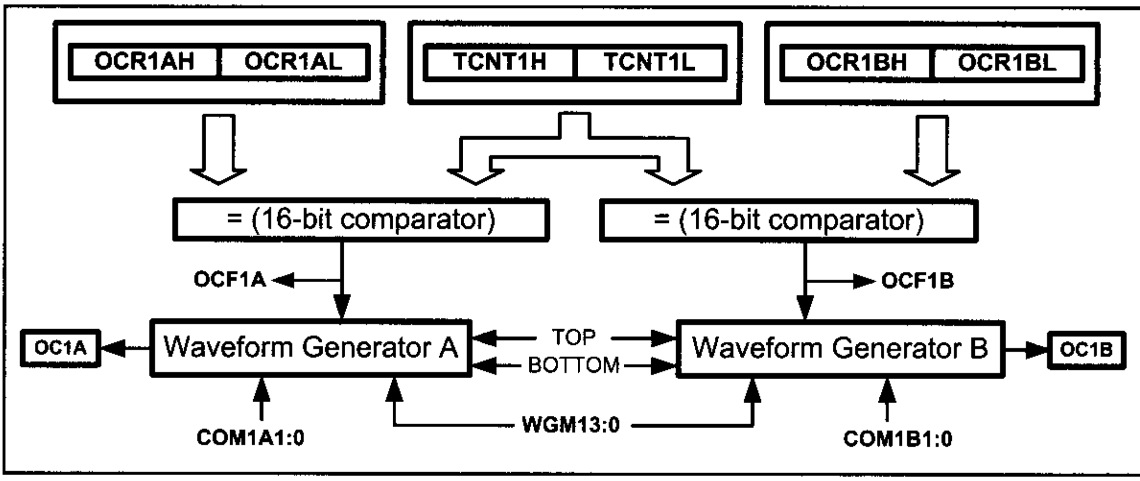


Figure 15-13. Simplified Waveform Generator Block Diagram

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
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

COM1A1:COM1A0 D7 D6 Compare Output Mode for Channel A

COM1A1	COM1A0	Description
0	0	Normal port operation, OC1A disconnected
0	1	Toggle OC1A on compare match
1	0	Clear OC1A on compare match
1	1	Set OC1A on compare match

COM1B1:COM1B0 D5 D4 Compare Output Mode for Channel B

COM1B1	COM1B0	Description
0	0	Normal port operation, OC1B disconnected
0	1	Toggle OC1B on compare match
1	0	Clear OC1B on compare match
1	1	Set OC1B on compare match

FOC1A D3 Force Output Compare for Channel A

FOC1B D2 Force Output Compare for Channel B

WGM11:10 D1 D0 Timer1 mode (discussed in Figure 15-8)

Figure 15-14. TCCR1A (Timer 1 Control) Register

In ATmega32/ATmega16, OC1A and OC1B are the alternative functions of PD5 and PD4, respectively. In other words, the PD5 pin functions as an I/O port when both COM1A1 and COM1A0 are zero. Otherwise, the pin acts as a wave generator pin controlled by Waveform Generator A. PD4 functions as an I/O port when both COM1B1 and COM1B0 are zero. Otherwise, the pin acts as a wave generator pin controlled by Waveform Generator B, as shown in Figure 15-12.

Notice that the DDR register represents the direction of the OC1A and OC1B pins all the time. Thus, we should be careful in setting the OC1A and OC1B pins as output pins when we want to use them for generating waves.

The waveform generators of Timer1 work almost the same as those of Timer0. In the following pages we will see the operation of Timer1 in the different modes.

Wave generation in Normal and CTC modes

When Timer1 is in CTC (WGM13:0 = 0100 or WGM13:0 = 1100) or Normal (WGM13:0 = 0000) mode after a compare match occurs, the waveform generators can perform one of the following actions, depending on the values of COM1A1:0 and COM1B1:0 bits, respectively:

(a) Remain unaffected
(b) Toggle the OC1x pin (OC1A or OC1B)
(c) Clear (drive low) the OC1x pin
(d) Set (drive high) the OC1x pin

The COM1A1 and COM1A0 bits select the operation of OC1A, while COM1B1 and COM1B0 select the operation of OC1B, as shown in Figure 15-14. See Example 15-14.

Example 15-14

Using Figures 15-8 and 15-14, find the values of the TCCR1A and TCCR1B registers if we want to clear the OC1A pin upon match, with no prescaler, internal clock, and Normal mode.

Solution:

WGM13:10 = 0000 = Normal mode
COM1A1:0 = 10 = Clear
CS12:10 = 001 = No prescaler

TCCR1A =

1	0	0	0	0	0	0	0
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B =

0	0	0	0	0	0	0	1
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Generating square waves

To generate square waves we can set the timer to Normal or CTC mode and set the COM1x1 and COM1x0 bits of one of the Waveform Generators to the toggle mode (COM1A1:0 = 01 to generate waves with Waveform Generator A or COM1B1:0 = 01 for generating waves using Waveform Generator B).

The OC1x pin will be toggled on each compare match and a square wave will be generated, as shown in Figure 15-15. See Examples 15-15 and 15-16.

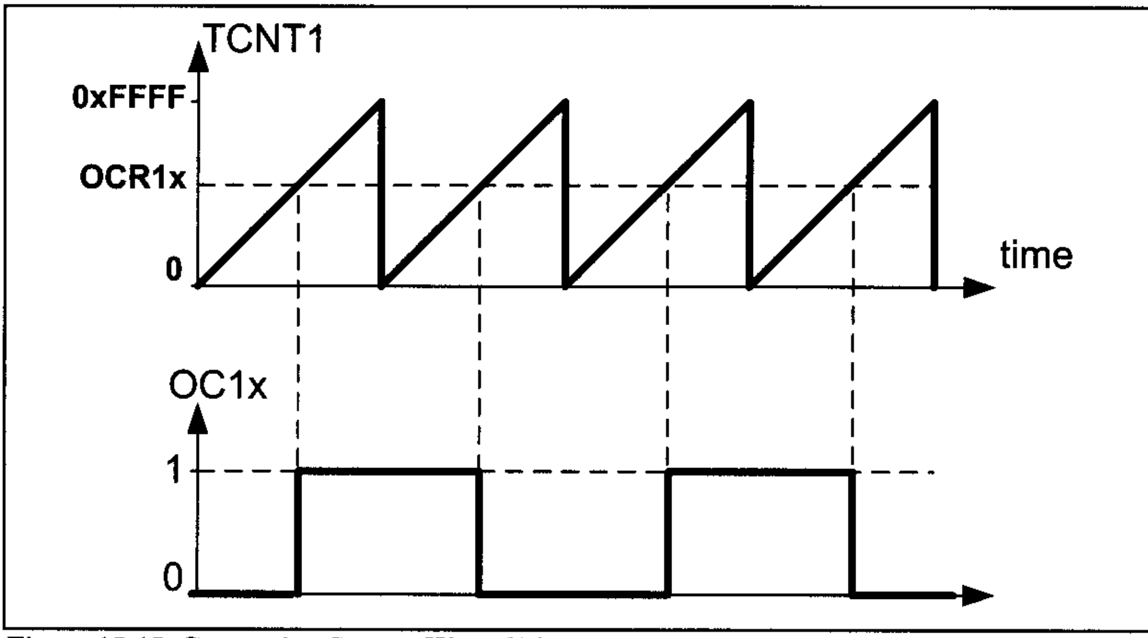


Figure 15-15. Generating Square Wave Using Normal Mode and Toggle Mode

Example 15-15

Find the value for TCCR1A and TCCR1B to program Timer1 as Normal mode and the OC1A generator as square wave generator and no prescaler.

Solution:

WGM13:10 = 0000 = Normal mode
COM1A1:0 = 01 = Toggle
CS12:10 = 001 = No prescaler
FOC1A = 1
FOC1B = 1

TCCR1A =

0	1	0	0	0	0	0	0
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

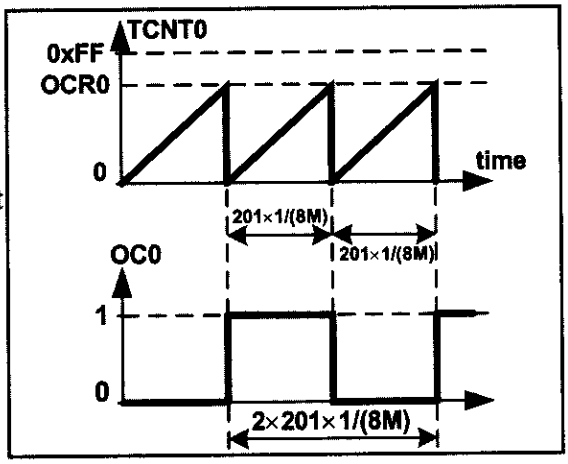
TCCR1B =

0	0	0	0	0	0	0	1
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Example 15-16

Assuming XTAL = 8 MHz, calculate the frequency of the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRD,5
      LDI   R22,0x40   ;COM1A = Toggle.
      OUT   TCCR1A,R22
      LDI   R22,0x01   ;WGM = Toggle, Mode = Normal, no prescaler
      OUT   TCCR1B,R22
      LDI   R22,HIGH(30000) ;the high byte
      OUT   OCR1AH,R22
      LDI   R22,LOW(30000)  ;the low byte
      OUT   OCR1AL,R22
HERE: RJMP  HERE
```



Solution:

From one compare match to the next one it takes 65,536 clocks and

$T_{timer\ clock} = 1/8\ \text{MHz} = 0.125\ \mu s$

$T_{wave} = 2 \times 65{,}536 \times 0.125\ \mu s = 16{,}384\ \mu s$

$F_{wave} = 1/16{,}384\ \mu s = 61.035\ \text{Hz}$

CTC mode is better than Normal mode for generating square waves, as the frequency of the wave can be easily adjusted by changing the top value (the value of the OCR1x register in mode 4, and ICR1 in mode12). See Figure 15-16. In CTC mode, when OCR1x (or ICR1 in mode 12) has a lower value, compare match occurs earlier and the period of the generated wave is smaller (higher frequency). When the OCR0 has a higher value, compare match occurs later and the period of the wave is longer (lower frequency). See Examples 15-17 through 15-19.

Example 15-17

Find the value for TCCR1A and TCCR1B to program Timer1 as CTC mode and the OC1A generator as square wave generator and no prescaler.

Solution:

WGM13:10 = 0100 = CTC
COM1A1:0 = 01 = toggle
CS12:10 = 001 = no prescaler

TCCR1A =	0	1	0	0	0	0	0	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B =	0	0	0	0	1	0	0	1
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

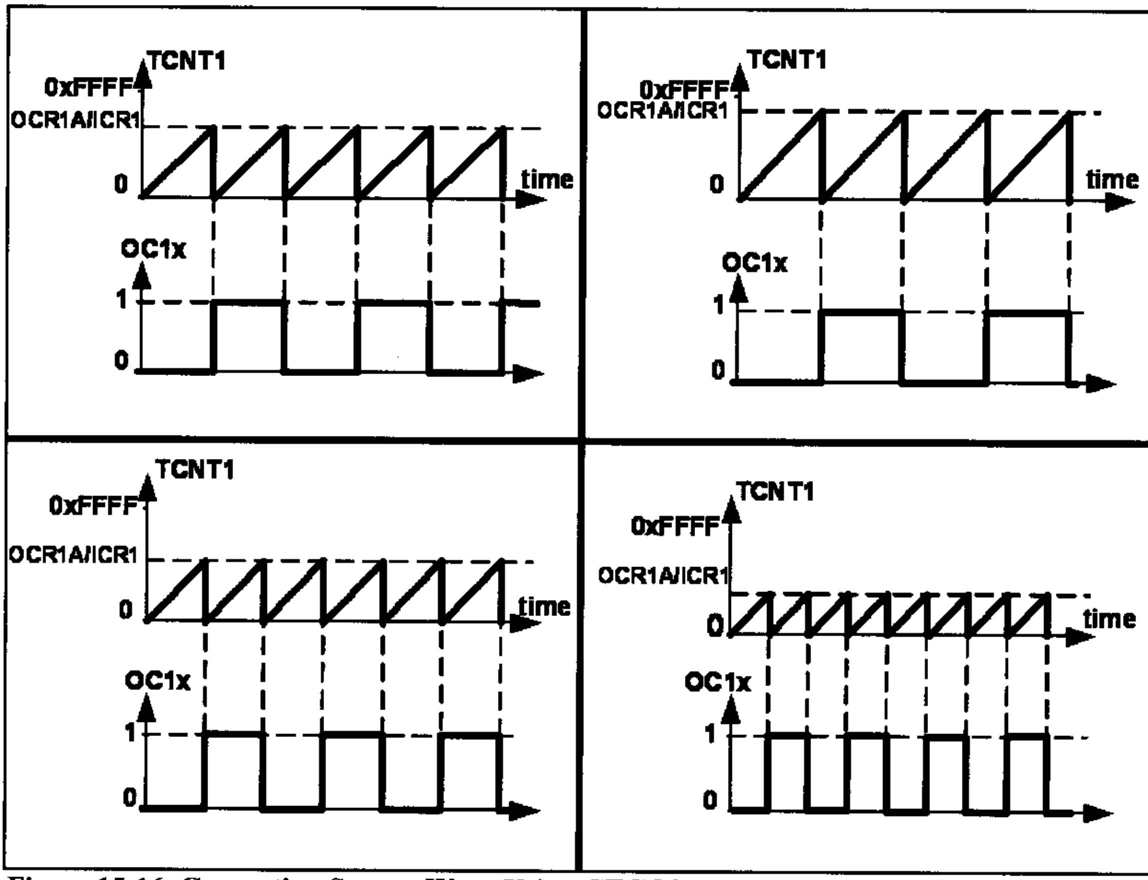


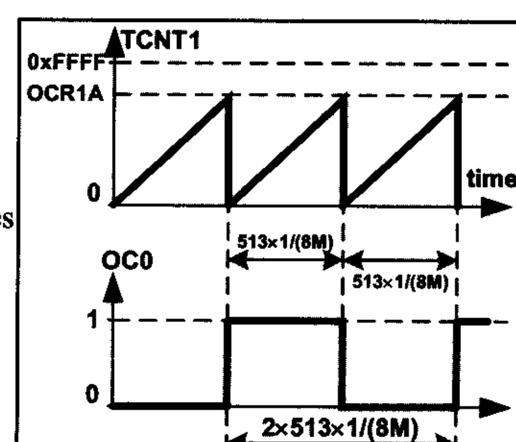
Figure 15-16. Generating Square Wave Using CTC Mode and Toggle Mode

Example 15-18

Assuming XTAL = 8 MHz, calculate the frequency of the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
        SBI   DDRD,5
        LDI   R22,0x40          ;COM1A = Toggle
        OUT   TCCR1A,R22
        LDI   R22,0x09          ;WGM = Toggle, Mode = CTC, no prescaler
        OUT   TCCR1B,R22
        LDI   R22,HIGH(512)
        OUT   OCR1AH,R22        ;TEMP = 0x02
        LDI   R22,LOW(512)
        OUT   OCR1AL,R22   ;OCR1A = 512
HERE:   RJMP  HERE
```

Solution:



From one compare match to the next one it takes 512 + 1 = 513 clocks and

$T_{\text{timer clock}} = 1 / 8\ \text{MHz} = 0.125\ \mu s$

$T_{\text{wave}} = 2 \times 513 \times 0.125\ \mu s = 128.25\ \mu s$

$F_{\text{wave}} = 1 / 128.25\ \mu s = 7797\ \text{Hz} = 7.797\ \text{kHz}$

Example 15-19

In Example 15-18, calculate the frequency of the wave generated in each of the following cases:
(a) OCR1A is loaded with 0x0500 (b) XTAL = 1 MHz and OCR1A is loaded with 0x5
(c) a prescaler option of 8 is chosen, XTAL = 4 MHz, OCR1A = 0x150
(d) a prescaler option of N is chosen, XTAL = F_{OSC}, OCR1A = X

Solution:

(a) 0x500 + 1 = 0x501 = 1281 clocks and $T_{timer\ clock}$ = 0.125 μs
➔ $T_{wave} = 2 \times 1281 \times 0.125\ \mu s = 320.25\ \mu s$ ➔ $F_{wave} = 1 / 320.25\ \mu s = 3122.56$ Hz

(b) 5 + 1 = 6 clocks and $T_{timer\ clock}$ = 1/1 MHz = 1 μs
➔ $T_{wave} = 2 \times 6 \times 1\ \mu s = 12\ \mu s$ ➔ $F_{wave} = 1 / 12\ \mu s$ = 83,333 Hz = 83.333 kHz

(c) 0x150 + 1 = 0x151 = 337 clocks and $T_{timer\ clock} = 8 \times 1 / 4$ MHz = 2 μs
➔ $T_{wave} = 2 \times 337 \times 2\ \mu s = 1348\ \mu s$ ➔ $F_{wave} = 1 / 2416\ \mu s = 741.8$ Hz

(d) X + 1 clocks and $T_{timer\ clock} = N \times 1/F_{OSC} = N / F_{OSC}$
➔ $T_{wave} = 2 \times (X + 1) \times N / F_{OSC}$ ➔ $F_{wave} = 1 / T_{wave} = F_{OSC} / [2N (X + 1)]$
The formula is the same as the one calculated in Example 15-7 d.

FOC1A (Force Output Compare) and FOC1B flags

Writing 1 to the FOC1A bit of the TCCR1A register forces the Waveform Generator A to act as if a compare match has occurred. Writing 1 to the FOC1B bit of the TCCR1A register forces Waveform Generator B to act as if a compare match has occurred. See Example 15-20.

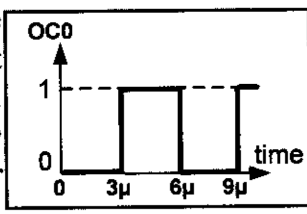
Example 15-20

Assuming XTAL = 1 MHz, draw the wave generated by the following program:

```
.INCLUDE "M32DEF.INC"
      SBI   DDRD,5
      LDI   R20,0x01
      OUT   TCCR1B,R20 ;Normal, timer stopped
      LDI   R20,0x48
L1:   OUT   TCCR1A,R20 ;toggle on match, FOC1A = 1
      RJMP  L1
```

Solution:

The wave generator is in toggle mode. So, it toggles on compare match. Setting the FOC1A bit causes the wave generator to act as if the compare match has occurred. So, the OC1A pin toggles. The execution of instructions "`OUT TCCR1A,R20`" and "`RJMP L1`" takes 1 and 2 clocks, respectively. So, toggle occurs after 1 + 2 = 3 clocks



Review Questions

1. True or false. In ATmega32, Timer1 has three waveform generators.
2. True or false. In CTC modes the TOP value is determined by OCR1A or ICR1.
3. True or false. We can associate each of the pins with each of the waveform generators.
4. True or false. In CTC modes we cannot change the frequency of the generated wave.

SECTION 15.3: INPUT CAPTURE PROGRAMMING

The Input Capture function is widely used for many applications. Among them are (a) recording the arrival time of an event, (b) pulse width measurement, and (c) period measurement. In ATmega32, Timer1 can be used as the Input Capture to detect and measure the events happening outside the chip. Upon detection of an event, the TCNT value is loaded into the ICR1 register, and the ICF1 flag is set.

As shown in Figure 15-17, there are two event sources: (1) the ICP1 pin, which is PORTD.6 in ATmega32, and (2) the output of the analog comparator. We can use the ACIC flag to select the event source. ACIC is a bit of the ACSR register, as shown in Figure 15-18.

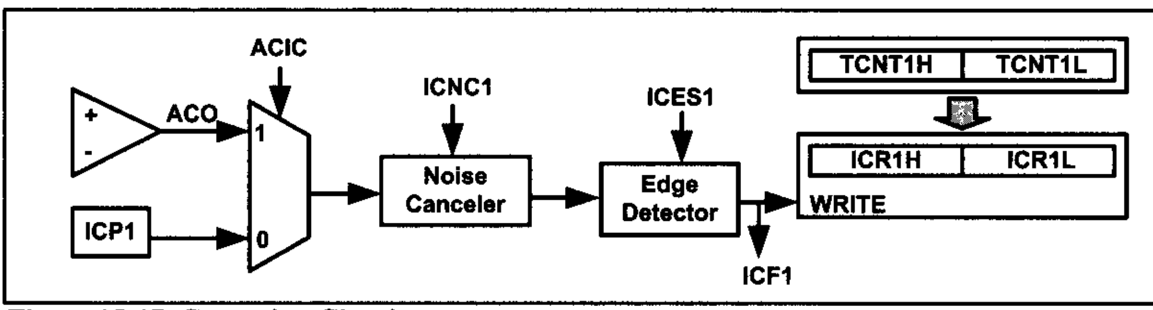


Figure 15-17. Capturing Circuit

ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

ACD (Analog Comparator Disable) When the bit is one, the power to the Analog Comparator is switched off, which reduces power consumption.

ACBG (Analog Comparator Bandgap Select) See the datasheet.

ACO (Analog Comparator Output) The output of the analog comparator is connected to the bit. ACO is read only. See Figure 15-17.
ACI (Analog Comparator Interrupt Flag)
ACIE (Analog Comparator Interrupt Enable)
ACIC (Analog Comparator Input Capture Enable) When the bit is one, the input capture is triggered by the Analog Comparator; otherwise, the ICP1 pin (PD6 in ATmega32) provides the capturing signal. See Figure 15-17.
ACIS1, ACIS0 (Analog Comparator Interrupt Mode Select) See the datasheet.

Figure 15-18. TCCR1B (Timer/Counter Control Register) Register, ICNC1, ICES1

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

ICNC1 (Input Capture Noise Canceller) Setting the bit activates the noise canceller. When the noise canceller is activated, each change is considered only if it persists for at least 4 successive system clocks. Notice that although activating the noise canceller prevents the detection of noises as signals, it causes 4 clocks of delay from the event occurrence to the load of the ICR1 register.

ICES1 (Input Capture Edge Select) Selects edge detection for the input capture function. When an edge is detected, the TCNT is loaded into the ICRx register. It also raises the ICFn (input capture flag) flag in the TIFR register.

0	Capture on falling edge
1	Capture on rising edge

WGM13:WGM12 D4 D3 Timer1 mode

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 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	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

CS12:CS10	D2D1D0	Timer1 clock selector
	0 0 0	No clock source (Timer/Counter stopped)
	0 0 1	clk (no prescaling)
	0 1 0	clk / 8
	0 1 1	clk / 64
	1 0 0	clk / 256
	1 0 1	clk / 1024
	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

Figure 15-19. TCCR1B (Timer/Counter Control Register) Register, ICNC1, ICES1

As shown in Figures 15-17 and 15-19, we use the TCCR1B register to select the type of edge detection and activate/deactivate the noise canceller unit.

Notice that the input capture unit does not work in the timer modes for which the ICR1 defines the top value (modes 8, 10, 12, 14). See Example 15-21.

Example 15-21

Using Figures 15-12 and 15-19, find TCCR1A and TCCR1B, for capturing on rising edge, no noise canceller, no prescaler, and timer mode = Normal.

Solution:

TCCR1A =

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
0	0	0	0	0	0	0	0

TCCR1B =

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
0	1	0	0	0	0	0	1

Steps to program the Input Capture function

We use the following steps to measure the edge arrival time for the Input Capture function.

1. Initialize the TCCR1A and TCCR1B for a proper timer mode (any mode other than modes 8, 10, 12, and 14), enable or disable the noise canceller, and select the edge (positive or negative) we want to measure the arrival time for.

2. Initialize the ACSR to select the desired event source.

3. Monitor the ICF1 flag in TIFR to see if the edge has arrived. Upon the arrival of the edge, the TCNT1 value is loaded into the ICR1 register automatically by the AVR. Example 15-22 shows how the Input Capture function works. The Input Capture function is widely used to measure the period or the pulse width of an incoming signal.

Example 15-22

Assuming that clock pulses are fed into pin ICP1, write a program to read the TCNT1 value on every rising edge. Place the result on PORTA and PORTB.

Solution:

```
.INCLUDE "M32DEF.INC"
      LDI   R16,0xFF
      OUT   DDRA,R16     ;PORTA as output
      OUT   DDRB,R16     ;PORTB as output
      OUT   PORTD,R16    ;activate pull-up
BEGIN:LDI   R20,0x00
      OUT   TCCR1A,R20   ;timer mode = Normal
      LDI   R20,0x41
      OUT   TCCR1B,R20  ;rising edge, no prescaler, no noise canceller
L1:   IN    R21,TIFR
      SBRS  R21,ICF1     ;skip next if ICF1 flag is set
      RJMP  L1           ;jump L1
      OUT   TIFR,R21     ;clear ICF1
      IN    R22,ICR1L    ;TEMP = ICR1H, R22 = ICR1L
      OUT   PORTA,R22    ;PORTA = R22
      IN    R22,ICR1H    ;R22 = TEMP = ICR1H
      OUT   PORTB,R22    ;PORTB = R22
      RJMP  BEGIN        ;jump begin
```

Note: Upon the detection of each rising edge, the TCNT1 value is loaded into ICR1. Also notice that we clear the ICF1 flag bit.

Measuring period

We can use the following steps to measure the period of a wave.

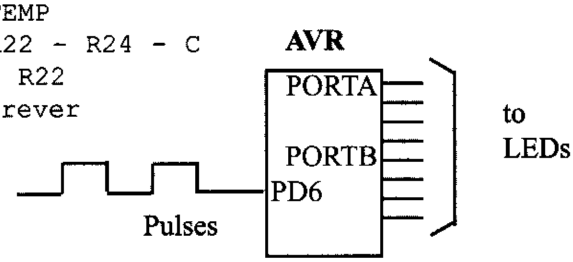
1. Initialize the TCCR1A and TCCR1B.
2. Initialize the ACSR to select the desired event source.
3. Monitor the ICF1 flag in TIFR to see if the edge has arrived. Upon the arrival of the edge, the TCNT1 is loaded into the ICR1 register automatically by the AVR.
4. Save the ICR1.
5. Monitor the ICF1 flag in TIFR to see if the second edge has arrived. Upon the arrival of the edge, the TCNT is loaded into the ICR1 register automatically by the AVR.
6. Save the ICR1 for the second edge. By subtracting the second edge value from the first edge value we get the time. See Examples 15-23 and 15-24. Also see Figure 15-20.

Example 15-23

Assuming that clock pulses are fed into pin PORTD.6, write a program to measure the period of the pulses. Place the binary result on PORTA and PORTB.

Solution:

```
.INCLUDE "M32DEF.INC"
      LDI    R16,0xFF
      OUT    DDRA,R16     ;PORTA as output
      OUT    DDRB,R16     ;PORTB as output
      OUT    PORTD,R16
BEGIN:LDI    R20,0x00
      OUT    TCCR1A,R20   ;timer mode = Normal
      LDI    R20,0x41
      OUT    TCCR1B,R20   ;rising edge, no prescaler, no noise canceller
L1:   IN     R21,TIFR
      SBRS   R21,ICF1     ;skip next instruction if ICF1 flag is set
      RJMP   L1           ;jump L1
      IN     R23,ICR1L    ;R23 = ICR1L, TEMP = ICR1H (first edge value)
      IN     R24,ICR1H    ;R24 = ICR1H
      OUT    TIFR,R21     ;ICF1 = 0
L2:   IN     R21,TIFR
      SBRS   R21,ICF1     ;skip next if ICF1 flag is set
      RJMP   L2
      OUT    TIFR,R21     ;clear ICF1
      IN     R22,ICR1L    ;R22 = ICR1L, TEMP = ICR1H (second edge value)
      SUB    R22,R23      ;Period = Second edge - First edge
      OUT    PORTA,R22    ;PORTA = R22
      IN     R22,ICR1H    ;R22 = TEMP
      SBC    R22,R24      ;R22 = R22 - R24 - C
      OUT    PORTB,R22    ;PORTB = R22
L3:   RJMP   L3           ;wait forever
```



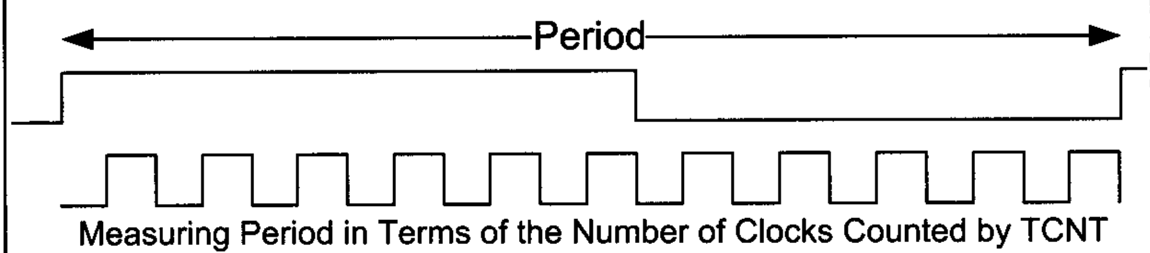


Figure 15-20. Using Input Capture to Measure Period

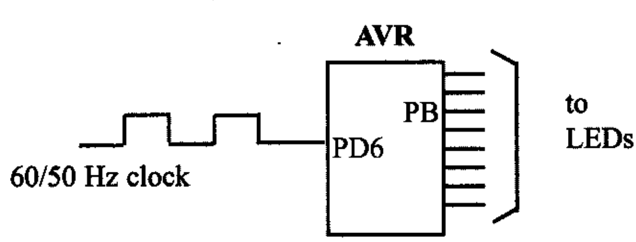
Example 15-24

The frequency of a pulse is between 50 Hz and 60 Hz. Assume that a pulse is connected to ICP1 (pin PD6). Write a program to measure its period and display it on PORTB. Use the prescaler value that gives the result in a single byte. Assume XTAL = 8 MHz.

Solution:

8 MHz × 1/1024 = 7812.5 Hz due to prescaler and T = 1/7812.5 Hz = 128 μs.
The frequency of 50 Hz gives us the period of 1/50 Hz = 20 ms. So, the output is 20 ms/128 μs = 156.
The frequency of 60 Hz gives us the period of 1/60 Hz = 16.6 ms. So, the output is 16.6 ms/128 μs = 130.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,0xFF
      OUT   DDRB,R16     ;PORTB as output
      OUT   PORTD,R16
BEGIN:LDI   R20,0x00
      OUT   TCCR1A,R20   ;timer mode = Normal
      LDI   R20,0x45
      OUT   TCCR1B,R20   ;rising edge, prescaler = 1024, no noise canc.
L1:   IN    R21,TIFR
      SBRS  R21,ICF1     ;skip next instruction if ICF1 flag is set
      RJMP  L1           ;jump L1
      IN    R16,ICR1L    ;R16 = ICR1L (first edge value)
      OUT   TIFR,R21     ;ICF1 = 0
L2:   IN    R21,TIFR
      SBRS  R21,ICF1     ;skip next if ICF1 flag is set
      RJMP  L2
      IN    R22,ICR1L    ;R22 = ICR1L, TEMP = ICR1H (second edge value)
      SUB   R22,R16      ;period = second edge - first edge
      OUT   PORTB,R22    ;PORTB = R22
      OUT   TIFR,R21     ;clear ICF1
L3:   RJMP  L3           ;wait forever
```



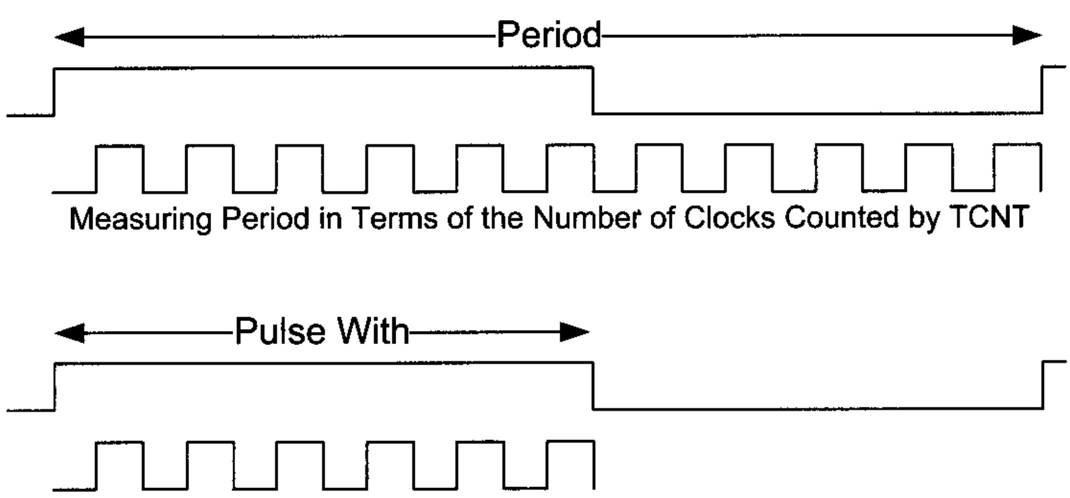


Figure 15-21. Using Input Capture to Measure Period and Pulse Width

Measuring pulse width

We can use the following steps to measure the pulse width of a wave.

1. Initialize TCCR1A and TCCR1B, and select capturing on rising edge.
2. Initialize ACSR to select the desired event source.
3. Monitor the ICF1 flag in TIFR to see if the edge has arrived. Upon the arrival of the edge, the TCNT1 value is loaded into the ICR1 register automatically by the AVR.
4. Save the ICR1 and change the capturing edge to the falling edge.
5. Monitor the ICF1 flag in TIFR to see if the second edge has arrived. Upon the arrival of the edge, the TCNT value is loaded into the ICR1 register automatically by the AVR.
6. Save the ICR1 for the second edge. Subtract the second edge value from the first edge value to get the time.

See Figure 15-21 and Examples 15-25 through 15-27 to see how it is done.

Example 15-25

Using Figure 15-19, find TCCR1B for no noise canceller, prescaler = 1024, and timer in Normal mode: (a) for capturing on rising edge (b) for capturing on falling edge

Solution:

(a) for capturing on rising edge

TCCR1B =

0	1	0	0	0	1	0	1
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

(b) for capturing on falling edge

TCCR1B =

0	0	0	0	0	1	0	1
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

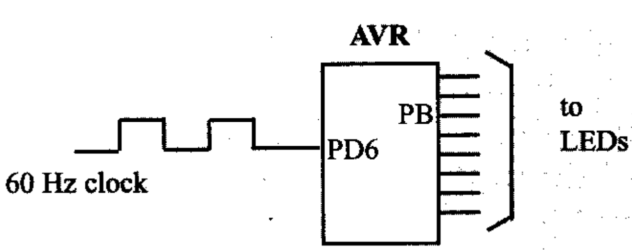
Example 15-26

Assume that a 60-Hz frequency pulse is connected to ICP1 (pin PD6). Write a program to measure its pulse width. Use the prescaler value that gives the result in a single byte. Display the result on PORTB. Assume XTAL = 8 MHz.

Solution:

The frequency of 60 Hz gives us the period of 1/60 Hz = 16.6 ms.
Now, 8 MHz × 1/1024 = 7812.5 Hz due to prescaler and T = 1/7812.5 Hz = 128 μs for TCNT. That means we get the value of 130 (1000 0010 binary) for the period since 16.6 ms / 128 μs = 130. Now the pulse width can be anywhere between 1 to 129.

```
.INCLUDE "M32DEF.INC"
      LDI    R16,0xFF
      OUT    DDRB,R16      ;PORTB as output
      OUT    PORTD,R16
BEGIN:LDI    R20,0x00
      OUT    TCCR1A,R20    ;timer mode = Normal
      LDI    R20,0x45
      OUT    TCCR1B,R20    ;rising edge, prescaler = 1024, no noise canc.
L1:   IN     R21,TIFR
      SBRS   R21,ICF1      ;skip next instruction if ICF1 flag is set
      RJMP   L1            ;jump L1
      IN     R16,ICR1L     ;R16 = ICR1L (rising edge value)
      OUT    TIFR,R21      ;ICF1 = 0 (for next round)
      LDI    R20,0x05
      OUT    TCCR1B,R20    ;falling edge, prescaler = 1024, no noise canc.
L2:   IN     R21,TIFR
      SBRS   R21,ICF1      ;skip next if ICF1 flag is set
      RJMP   L2
      IN     R22,ICR1L     ;R22 = ICR1L, TEMP = ICR1H (falling edge value)
      SUB    R22,R16       ;pulse width = falling edge - rising edge
      OUT    PORTB,R22     ;PORTB = R22
      OUT    TIFR,R21      ;clear ICF1 (for next round)
L3:   RJMP   L3            ;wait forever
```



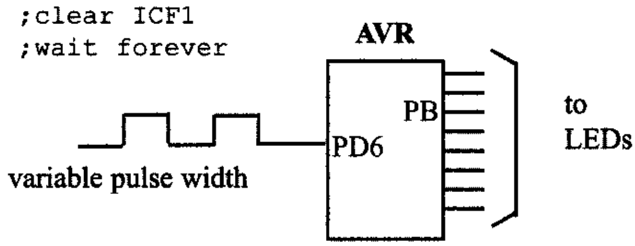
Example 15-27

Assume that a temperature sensor is connected to pin PD6. The temperature provided by the sensor is proportional to pulse width and is in the range of 1 μs to 250 μs. Write a program to measure the temperature if 1 μs is equal to 1 degree. Use the prescaler value that gives the result in a single byte. Display the result on PORTB. Assume XTAL = 8 MHz.

Solution:

8 MHz × 1 / 8 = 1 MHz = 1,000,000 Hz due to prescaler and T = 1/1,000,000 Hz = 1 μs for TCNT. That means we get the values between 1 and 65,536 μs for the TCNT, but since the pulse width never goes beyond 250 μs we should be able to display the temperature value on PORTB.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,0xFF
      OUT   DDRB,R16     ;PORTB as output
      OUT   PORTD,R16
BEGIN:LDI   R20,0x00
      OUT   TCCR1A,R20   ;timer mode = Normal
      LDI   R20,0x42
      OUT   TCCR1B,R20   ;rising edge, prescaler 8, no noise canceller
L1:   IN    R21,TIFR     ;stay here for ICP rising
      SBRS  R21,ICF1     ;skip next instruction if ICF1 flag is set
      RJMP  L1           ;jump L1
      IN    R16,ICR1L    ;R16 = ICR1L
      OUT   TIFR,R21     ;ICF1 = 0
      LDI   R20,0x02
      OUT   TCCR1B,R20   ;falling edge, prescaler 8, no noise canceller
L2:   IN    R21,TIFR     ;stay here for ICP falling edge
      SBRS  R21,ICF1     ;skip next if ICF1 flag is set
      RJMP  L2
      IN    R22,ICR1L    ;R22 = ICR1L, TEMP = ICR1H
      SUB   R22,R16      ;period = falling edge - rising edge
      OUT   PORTB,R22    ;PORTB = R22
      OUT   TIFR,R21     ;clear ICF1
L3:   RJMP  L3           ;wait forever
```



Analog comparator

As shown in Figure 15-17, when the ACIC bit is set, the analog comparator provides the trigger signal for the input capture unit. The analog comparator is an op-amp that compares the voltage of AIN1 (PORTB.3 in ATmega32) with AIN0 (PORTB.2 in ATmega32). If the voltage of AIN1 is higher than AIN0, the comparator's output is 1; otherwise, its output is 0. For more information, see the datasheet of the ATmega32.

Review Questions

1. True or false. In the ATmega32, only Timer1 has the Input Capture function.
2. True or false. TCNT1 is also used by the Input Capture function.
3. True or false. Activating the noise canceller causes the capturing to occur instantly when an event rises.
4. Indicate the registers used by the Input Capture function.
5. True or false. The Input Capture function can capture the timing of an incoming pulse on the rising edge only.

SECTION 15.4: C PROGRAMMING

Examples 15-28 through 15-42 show the C versions of the earlier programs.

Example 15-28 (C version of Example 15-2)

Write a program that (a) after 4 external clocks turns on an LED connected to the OC0 pin, and (b) toggles the OC0 pin every 4 pulses.

Solution:

```
(a)
#include "avr/io.h"
int main ( )
{
      DDRB &= ~(1<<0);          //PB0(T0) pin as input
      DDRB = DDRB|(1<<3);       //PB3(OC0) pin as output
      OCR0 = 3;
      TCNT0 = 0;                //load timer with 0
      TCCR0 = 0x36;             //external clock, Normal mode, set OC0
      while (1);
      return 0;
}
```

```
(b)
#include "avr/io.h"
int main ( )
{
      DDRB &= ~(1<<0);          //PB0(T0) pin as input
      DDRB = DDRB|(1<<3);       //PB3(OC0) pin as output
      OCR0 = 3;
      TCNT0 = 0;                //load timer with 0
      TCCR0 = 0x1E;             //external clock, CTC mode, set OC0
      while (1);
      return 0;
}
```

Example 15-29 (C version of Example 15-4)

Rewrite the program of Example 15-4 using C.

Solution:

```
#include "avr/io.h"
int main ( )
{
      DDRB = DDRB|(1<<3);   //PB3(OC0) = output
      TCCR0 = 0x11;  //COM01:00=Toggle, Mode=Normal, no prescaler
      OCR0 = 100;
      while (1);
      return 0;
}
```

Example 15-30 (C version of Example 15-6)

Rewrite the program of Example 15-6 using C.

Solution:

```
#include "avr/io.h"
int main ( )
{
      DDRB = DDRB|(1<<3);   //PB3(OC0) = output
      TCCR0 = 0x19;   //COM01:00=Toggle, Mode=CTC, no prescaler
      OCR0 = 200;
      while (1);
}
```

Example 15-31 (C version of Example 15-8)

Rewrite the program of Example 15-8 using C.

Solution:

```
#include "avr/io.h"
int main ( )
{
      DDRB |= (1<<3);         //PB3 = output
      while (1)
      {
            OCR0 = 99;
            TCCR0 = 0x19;      //CTC, no prescaler, set on match
            while ((TIFR&(1<<OCF0)) == 0);
            TIFR = (1<<OCF0);     //clear OCF0
            OCR0 = 69;
            TCCR0 = 0x39;      //CTC, no prescaler, set on match
            while ((TIFR&(1<<OCF0)) == 0);
            TIFR = (1<<OCF0);     //clear OCF0
      }
      return 0;
}
```

Example 15-32 (C version of Example 15-9)

Rewrite the program of Example 15-9 using C.

Solution:

```
#include "avr/io.h"
#include "avr/interrupt.h"

int main ( )
{
	DDRB = DDRB |(1<<3);  //PB3 = output
	OCR0 = 69;
	TCCR0 = 0x19;      //CTC, no prescaler, toggle on match
	TIMSK = (1<<OCIE0);   //enable compare match interrupt
	sei();                //enable interrupts
	while(1);

	return 0;
}

ISR(TIMER0_COMP_vect)
{
	const unsigned char waveTable [] = {24,49,39,34};
	static unsigned char index = 0;

	OCR0 = waveTable[ index] ;
	index ++;

	if(index >= 4)
		index = 0;
}
```

Example 15-33 (C version of Example 15-10)

Rewrite the program of Example 15-10 using C.

Solution:

```
#include "avr/io.h"

int main ( )
{
	DDRB = DDRB |(1<<3);        //PB3 = output

	while(1)
	  TCCR0 = 0x98; //CTC, timer stopped, toggle on match, FOC0=1

	return 0;
}
```

Example 15-34 (C version of Example 15-12)

Rewrite the program of Example 15-12 using C.

Solution:

```
#include "avr/io.h"

int main ( )
{
      DDRD = DDRD |(1<<7);  //PD7(OC2) = output
      TCCR2 = 0x19;    //COM21:20=Toggle, Mode=CTC, no prescaler
      OCR2 = 200;

      while (1);
}
```

Example 15-35 (C version of Example 15-13)

Rewrite the program of Example 15-13 using C.

Solution:

```
#include "avr/io.h"

void delay_1ms ( );

int main ()
{
      DDRB = (1<<5);
      while (1)
      {
            PORTB = PORTB ^ (1<<5);
            delay_1ms ( );
      }
      return 0;
}
void delay_1ms ( )
{
      ICR1H = 0x27;
      ICR1L = 0x0F;   //ICR1L = 0x0F, ICR1H = TEMP
      TCNT1H = 0;
      TCNT1L = 0;
      TCCR1A = 0x02;  //WGM11:10 = 10

      TCCR1B = 0x19;  //WGM13:12 = 11, CS = CLK, mode = 14
      while((TIFR&(1<<ICF1)) == 0);
      TIFR = (1<<ICF1);
      TCCR1B = 0;
      TCCR1A = 0;      //stop timer
}
```

Example 15-36 (C version of Example 15-18)

Rewrite the program of Example 15-18 using C.

Solution:

```
#include "avr/io.h"
int main ()
{
      DDRD = (1<<5);
      TCCR1A = 0x40;   //COM1A = Toggle
      TCCR1B = 0x09;   //WGM = Toggle, Mode = CTC, no prescaler
      OCR1AH = 0x02;   //TEMP = 0x02
      OCR1AL = 0x00;   //OCR1A = 0x200 = 512
      while (1);
      return 0;
}
```

Example 15-37 (C version of Example 15-20)

Rewrite the program of Example 15-20 using C.

Solution:

```
#include "avr/io.h"
int main ()
{
      DDRD = DDRD |(1<<5);
      TCCR1B = 0x01;  //Normal, timer stopped
      while (1)
        TCCR1A = 0x48;      //toggle on match, FOC1A = 1
}
```

Example 15-38 (C version of Example 15-22)

Assuming that clock pulses are fed into pin ICP1, write a program to read the TCNT1 value on every rising edge. Place the result on PORTA and PORTB.

Solution:

```
#include "avr/io.h"
int main ( )
{
      DDRA = 0xFF;     //port A as output
      DDRB = 0xFF;     //port B as output
      PORTD = 0xFF;    //activate pull-up
      while(1) {
        TCCR1A = 0;    //Mode = Normal
        TCCR1B = 0x41;//rising edge, no scaler, no noise canceller
        while ((TIFR&(1<<ICF1)) == 0);
        TIFR = (1<<ICF1);   //clear ICF1
        PORTA = ICR1L;
        PORTB = ICR1H;
      }
      return 0;
}
```

Example 15-39 (C version of Example 15-23)

Assuming that clock pulses are fed into pin PORTD.6, write a program to measure the period of the pulses. Place the binary result on PORTA and PORTB.

Solution:

```
#include "avr/io.h"
int main ( )
{
      unsigned int t;
      DDRA = 0xFF;      //PORTA as output
      DDRB = 0xFF;      //PORTB as output
      PORTD = 0xFF;     //activate pull-up
      TCCR1A = 0;       //Mode = Normal
      TCCR1B = 0x41; //rising edge, no scaler, no noise canceller
      while ((TIFR&(1<<ICF1)) == 0);
      t = ICR1;
      TIFR = (1<<ICF1);      //clear ICF1
      while ((TIFR&(1<<ICF1)) == 0);
      t = ICR1 - t;
      PORTA = t;        //the low byte
      PORTB = t>>8;     //the high byte
      while (1);
      return 0;
}
```

Example 15-40 (C version of Example 15-24)

The frequency of a pulse is either 50 Hz or 60 Hz. Assume that a the pulse is connected to ICP1 (pin PD6). Write a program to measure its period and display it on PORTB. Use the prescaler value that gives the result in a single byte. Assume XTAL = 8 MHz.

Solution:

```
#include "avr/io.h"
int main ( )
{
      unsigned char t1;
      DDRB = 0xFF;      //PORTB as output
      PORTD = 0xFF;
      TCCR1A = 0;       //Timer Mode = Normal
      TCCR1B = 0x45; //rising edge, prescaler=1024, no noise canc.

      TIFR = (1<<ICF1);      //clear ICF1
      while ((TIFR&(1<<ICF1)) == 0); //wait while ICF1 is clear
      t1 = ICR1L;            //first edge value
      TIFR = (1<<ICF1);      //clear ICF1
      while ((TIFR&(1<<ICF1)) == 0); //wait while ICF1 is clear
      PORTB = ICR1L - t1;   //period = second edge - first edge
      TIFR = (1<<ICF1);      //clear ICF1
      while (1);             //wait forever
}
```

Example 15-41 (C version of Example 15-26)

Assume that a 60-Hz frequency pulse is connected to ICP1 (pin PD6). Write a program to measure its pulse width. Use the prescaler value that gives the result in a single byte. Display the result on PORTB. Assume XTAL = 8 MHz.

Solution:

```
#include "avr/io.h"
int main ( )
{
     unsigned char t1;
     DDRB = 0xFF;          //Port B as output
     PORTD = 0xFF;
     TCCR1A = 0;           //Timer Mode = Normal
     TCCR1B = 0x45; //rising edge, prescaler=1024, no noise canc.
     while ((TIFR&(1<<ICF1)) == 0);
     t1 = ICR1L;           //first edge value
     TIFR = (1<<ICF1);     //clear ICF1 flag
     TCCR1B = 0x05;        //falling edge
     while ((TIFR&(1<<ICF1)) == 0);
     PORTB = ICR1L - t1;   //pulse width = falling - rising
     TIFR = (1<<ICF1);     //clear ICF1 flag
     while (1);            //wait forever
     return 0;
}
```

Example 15-42 (C version of Example 15-27)

Assume that a temperature sensor is connected to pin PD6. The temperature provided by the sensor is proportional to pulse width and is in the range of 1 μs to 250 μs. Write a program to measure the temperature if 1 μs is equal to 1 degree. Use the prescaler value that gives the result on PORTB. Assume XTAL = 8 MHz.

Solution:

```
#include "avr/io.h"
int main ( )
{
     unsigned char t1;
     DDRB = 0xFF;          //Port B as output
     PORTD = 0xFF;
     TCCR1A = 0;           //Timer Mode = Normal
     TCCR1B = 0x42;  //rising edge, prescaler = 8, no noise canc.
     while ((TIFR&(1<<ICF1)) == 0);
     t1 = ICR1L;
     TIFR = (1<<ICF1);     //clear ICF1 flag
     TCCR1B = 0x02;        //falling edge
     while ((TIFR&(1<<ICF1)) == 0);
     PORTB = ICR1L - t1;   //pulse width = falling - rising
     TIFR = (1<<ICF1);     //clear ICF1 flag
     while (1);            //wait forever
     return 0;
}
```

SUMMARY

This chapter began by describing the pulse wave generating features of the AVR family. We discussed how to generate square waves and pulses using CTC mode. We discussed how to generate waves using Timer0 as an 8-bit timer and Timer1 as a 16-bit timer. We also described the input capture feature. We used the input capture feature of AVR to measure the pulse width and period of incoming pulses.

PROBLEMS

SECTION 15.1: WAVE GENERATION USING 8-BIT TIMERS

1. True or false. The ATmega32 has only one 8-bit timer.
2. True or false. In the ATmega32, Timer0 has a 16-bit register accessible as TCNT0L and TCNT0H.
3. True or false. Each waveform generator has a single pin.
4. Give the pin used for Timer2 waveform generator in the ATmega32.
5. Using Timer0, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 80 kHz. Assume XTAL = 8 MHz.
6. Using Timer0, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 5 kHz. Assume XTAL = 1 MHz.
7. Using Timer0 and CTC mode, write a program that generates a square wave with a frequency of 625 Hz. Assume XTAL = 8 MHz.
8. Using Timer0 and CTC mode, write a program that generates a square wave with a frequency of 3125 Hz. Assume XTAL = 16 MHz.

SECTION 15.2: WAVE GENERATION USING TIMER1

9. True or false. In the ATmega32, Timer1 has two waveform generator channels.
10. Give the number of waveform generators in the ATmega32.
11. Using Timer1, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 1 kHz. Assume XTAL = 8 MHz.
12. Using Timer1, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 5 kHz. Assume XTAL = 8 MHz.
13. Using Timer1 and CTC mode, write a program that generates a square wave with a frequency of 50 Hz. Assume XTAL = 8 MHz.
14. Using Timer1 and CTC mode, write a program that generates a square wave with a frequency of 20 Hz. Assume XTAL = 16 MHz.

SECTION 15.3: INPUT CAPTURE PROGRAMMING

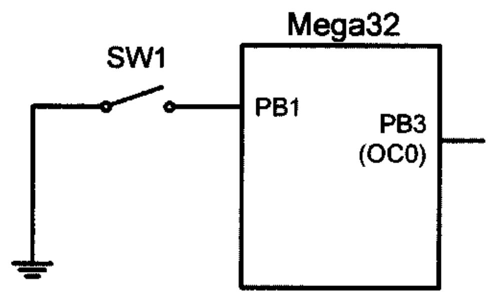
15. What is the use of capturing?
16. True or false. In the ATmega32, all of the timers have the capturing capability.
17. True or false. To use capture mode, we must make the ICP pin an output pin.
18. Which timers can be used for the capture mode?
19. Find the value for the TCCR1B register in capture mode if we want to capture

on the falling edge.

20. Find the value for the TCCR1B register in capture mode if we want to capture on the rising edge while the noise canceller is active.

SECTION 15.4: C PROGRAMMING

21. Using Timer0, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 50 kHz. Assume XTAL = 8 MHz.
22. Using Timer2, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 20 kHz. Assume XTAL = 1 MHz.
23. Using Timer2, prescaler = 256, and CTC mode, write a program that generates a square wave with a frequency of 100 Hz. Assume XTAL = 8 MHz.
24. Using Timer0, prescaler = 64, and CTC mode, write a program that generates a square wave with a frequency of 95 Hz. Assume XTAL = 1 MHz.
25. As shown in the Figure, a switch is connected to PB1. Using CTC mode and prescaler = 1024, write a program in which, if the switch is closed, the waveform generator creates a 60 Hz wave; otherwise, it generates a wave with a frequency of 50 Hz. Assume XTAL = 8 MHz.



26. Using Timer1, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 3 kHz. Assume XTAL = 8 MHz.
27. Using Timer1, no prescaler, and CTC mode, write a program that generates a square wave with a frequency of 44 kHz. Assume XTAL = 8 MHz.

ANSWERS TO REVIEW QUESTIONS

SECTION 15.1: WAVE GENERATION USING 8-BIT TIMERS

1. True
2. True
3. False
4. PB3 (PORTB.3)

SECTION 15.2: WAVE GENERATION USING TIMER1

1. False
2. True
3. False
4. False

SECTION 15.3: INPUT CAPTURE PROGRAMMING

1. True
2. True
3. False
4. ICR, TCNT
5. False