CHAPTER 9

AVR TIMER PROGRAMMING IN ASSEMBLY AND C

OBJECTIVES

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

- >> **List the timers of the ATmega328 and their associated registers**
- >> **Describe the Normal and CTC modes of the AVR timers**
- >> **Program the AVR timers in Assembly and C to generate time delays**
- >> **Program the AVR counters in Assembly and C as event counters**

Many applications need to count an event or generate time delays. So, there are counter registers in microcontrollers for this purpose. See Figure 9-1. When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred. When we want to generate time delays, we connect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter register represents how many ticks have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can calculate the tick period, and from the content of the counter register we will know how much time has elapsed.

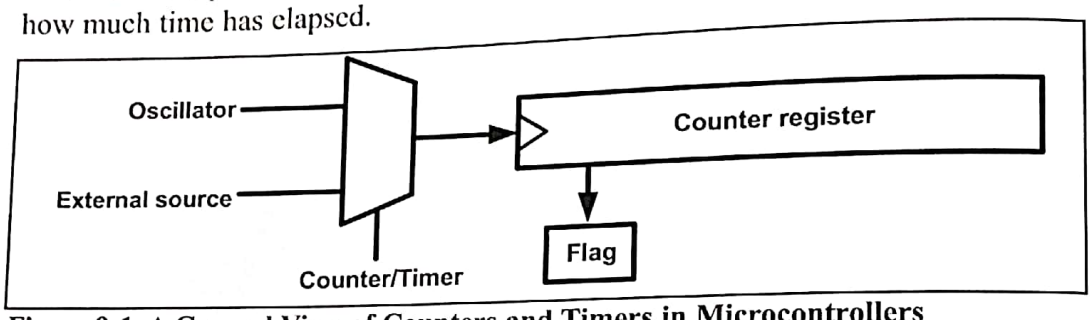


Figure 9-1. A General View of Counters and Timers in Microcontrollers

So, one way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number. For example, consider a microcontroller with an oscillator with frequency of 1 MHz; in the microcontroller, the content of the counter register increments once per microsecond. So, if we want a time delay of 100 microseconds, we should clear the counter and wait until it becomes equal to 100.

In microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software. The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set. For example, in a microcontroller with a frequency of 1 MHz, with an 8-bit counter register, if we want a time delay of 3 microseconds, we can load the counter register with $FD and wait until the flag is set after 3 ticks. After the first tick, the content of the register increments to $FE; after the second tick, it becomes $FF; and after the third tick, it overflows (the content of the register becomes $00) and the flag is set.

The AVR has one to six timers depending on the family member. They are referred to as Timers 0, 1, 2, 3, 4, and 5. They can be used as timers to generate a time delay or as counters to count events happening outside the microcontroller.

In the AVR some of the timers/counters are 8-bit and some are 16-bit. In ATmega328, there are three timers: Timer0, Timer1, and Timer2. Timer0 and Timer2 are 8-bit, while Timer1 is 16-bit. In this chapter we cover Timer0 and Timer2 as 8-bit timers, and Timer1 as a 16-bit timer.

If you learn to use the timers of ATmega328, you can easily use the timers of other AVRs. You can use the 8-bit timers like the Timer0 of ATmega328 and the 16-bit timers like the Timer1 of ATmega328.

SECTION 9.1: PROGRAMMING TIMERS 0, 1, AND 2

Every timer needs a clock pulse to tick. The clock source can be internal or external. If we use the internal clock source, then the frequency of the crystal oscillator is fed into the timer. Therefore, it is used for time delay generation and consequently is called a *timer*. By choosing the external clock option, we feed pulses through one of the AVR's pins. This is called a *counter*. In this section we discuss the AVR timer, and in the next section we program the timer as a counter.

Basic registers of timers

Examine Figure 9-2. In AVR, for each of the timers, there is a TCNTn (timer/counter) register. For example, in ATmega328 we have TCNT0, TCNT1, and TCNT2. The TCNTn register is a counter. Upon reset, the TCNTn contains zero. It counts up with each pulse. The contents of the timers/counters can be accessed using the TCNTn. You can load a value into the TCNTn register or read its value.

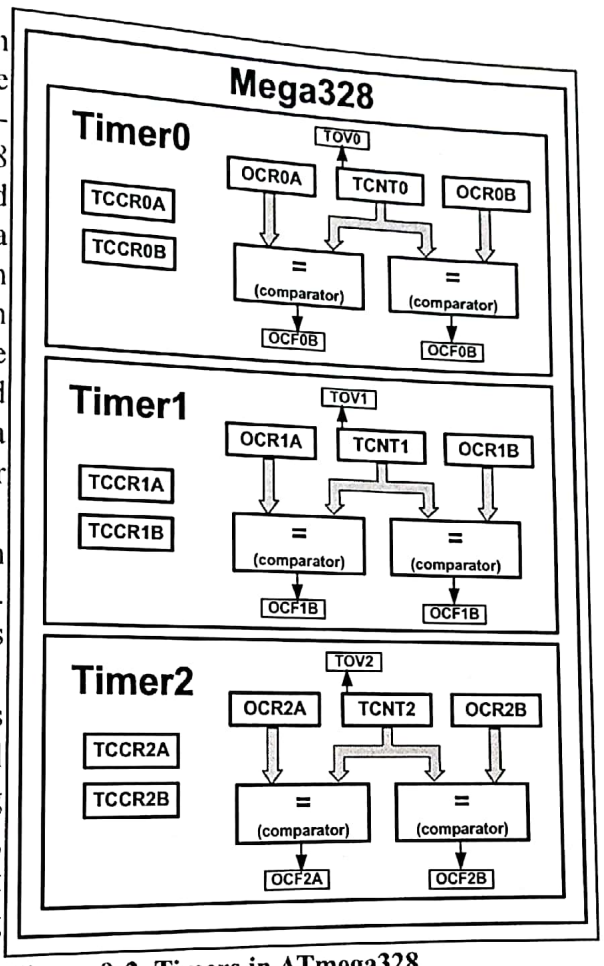


Figure 9-2. Timers in ATmega328

Each timer has a TOVn (Timer Overflow) flag, as well. When a timer overflows, its TOVn flag will be set.

Each timer also has TCCRn (timer/counter control register) registers for setting modes of operation. For example, you can specify Timer0 to work as a timer or a counter by loading proper values into the TCCR0A and TCCR0B registers.

Each timer also has some OCRn (Output Compare Register) registers. The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag) flag will be set.

The registers of Timer0 are located in the I/O register memory. Therefore, you can read or write from timer registers using IN and OUT instructions, like the other I/O registers. For example, the following instructions load TCNT0 with 25:

```
LDI R20,25        ;R20 = 25
OUT TCNT0,R20     ;TCNT0 = R20
```

or "`IN R19,TCNT0`" copies TCNT0 to R19.

The internal structure of the ATmega328 timers is shown in Figure 9-3. Next, we discuss each timer separately in more detail.

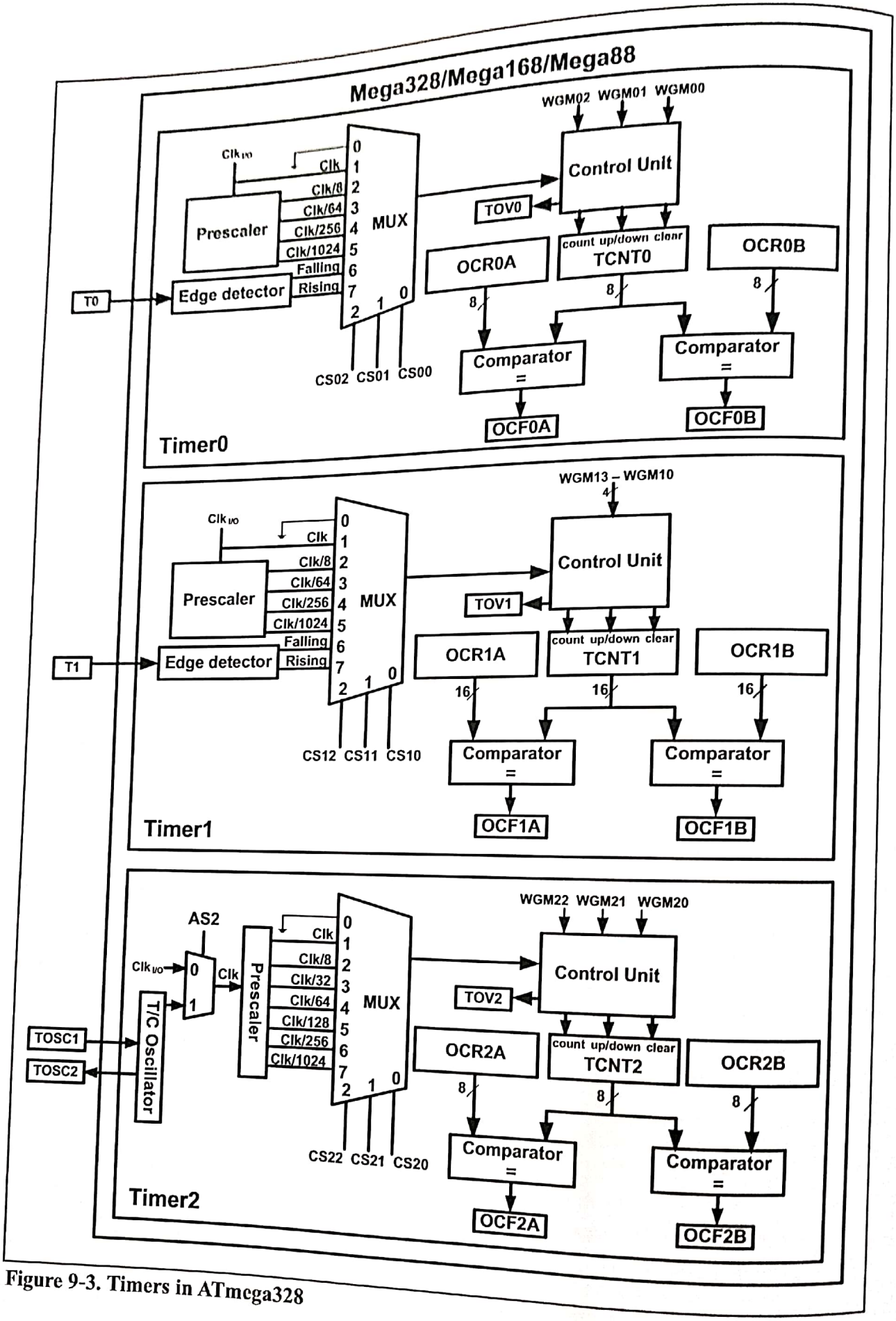


Figure 9-3. Timers in ATmega328

Timer0 programming

Timer0 is 8-bit in ATmega328; so, TCNT0 is 8-bit as shown in Figure 9-4.

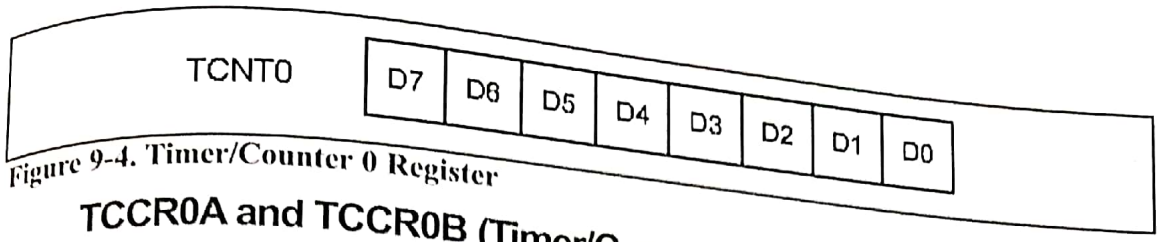


Figure 9-4. Timer/Counter 0 Register

TCCR0A and TCCR0B (Timer/Counter Control Register) registers

TCCR0A and TCCR0B are 8-bit registers used for control of Timer0. The bits for TCCR0A and TCCR0B are shown in Figures 9-5A and 9-5B.

CS02:CS00 (Timer0 clock source)

These bits in the TCCR0 register are used to choose the clock source. If CS02:CS00 = 000, then the counter is stopped. If CS02–CS00 have values between 001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation. See Figure 9-3 and then see Examples 9-1 and 9-2.

Bit	7	6	5	4	3	2	1	0
TCCR0A:	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

WGM02, WGM01, WGM00 Timer0 mode selector bits

Mode	WGM02 (TCCR0B)	WGM01 (TCCR0A)	WGM00	Timer/Counter Mode of Operation	Top	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	Phase Correct PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR0A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	Phase Correct PWM	OCR0A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR0A	BOTTOM	TOP

COM01:00 D7:D6 and D5:D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

Figure 9-5A. TCCR0A (Timer/Counter Control Register) Register

Bit	7	6	5	4	3	2	1	0
TCCR0B:	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
Read/Write	W	W	R	R	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

CS02:00

D2	D1	D0	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 9-5B. TCCR0B (Timer/Counter Control Register) Register

Example 9-1

Find the value for TCCR0A and TCCR0B if we want to program Timer0 in Normal mode, no prescaler. Use AVR's crystal oscillator for the clock source.

Solution:

To select no prescaler CS02–CS00=001, and to choose Normal mode WGM02–WGM00=000.

TCCR0A =	0	0	0	0	0	0	0
	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
TCCR0B =	0	0	0	0	0	0	0	1
	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

Example 9-2

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that no prescaler is used.

(a) 10 MHz (b) 8 MHz (c) 1 MHz

Solution:

(a) F = 10 MHz and T = 1/10 MHz = 0.1 µs
(b) F = 8 MHz and T = 1/8 MHz = 0.125 µs
(c) F = 1 MHz and T = 1/1 MHz = 1 µs

If CS02–CS00 are 110 or 111, the external clock source is used and it acts as a counter. We will discuss Counter in the next section.

WGM01:00

Timer0 can work in four different modes: Normal, phase correct PWM, CTC, and Fast PWM. The WGM00, WGM01, and WGM02 bits are used to choose one of them. We will discuss the PWM options in Chapter 16.

TIFR0 (Timer/counter Interrupt Flag Register) register

Each timer has a TIFRn register which contains the flags of the timer. Figure 9-6 shows the TIFR0 register which is related to Timer0.

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	OCF0B	OCF0A	TOV0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0A D1 Timer0 output compare A match flag
0 = compare match did not occur.
1 = compare match occurred.

OCF0B D2 Timer0 output compare B match flag
0 = compare match did not occur.
1 = compare match occurred.

Figure 9-6. TIFR0 (Timer/Counter Interrupt Flag Register)

TOV0 (Timer0 Overflow)

The TOV0 flag is set when the counter overflows. As we will see soon, when the timer rolls over from $FF to 00, the TOV0 flag is set to 1 and it remains set until the software clears it. See Figure 9-6.

The strange thing about this flag is that in order to clear it we need to write 1 to it. Indeed this rule applies to all flags of the AVR chip. In AVR, when we want to clear a given flag of a register we write 1 to it and 0 to the other bits. For example, the following program clears TOV0:

```
LDI    R20,0x01
OUT    TIFR0,R20     ;TIFR0 = 0b00000001
```

Normal mode

In this mode, the content of the timer/counter increments with each clock. It counts up until it reaches its max of 0xFF. When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0 (Timer Overflow). This timer flag can be monitored. See Figure 9-7.

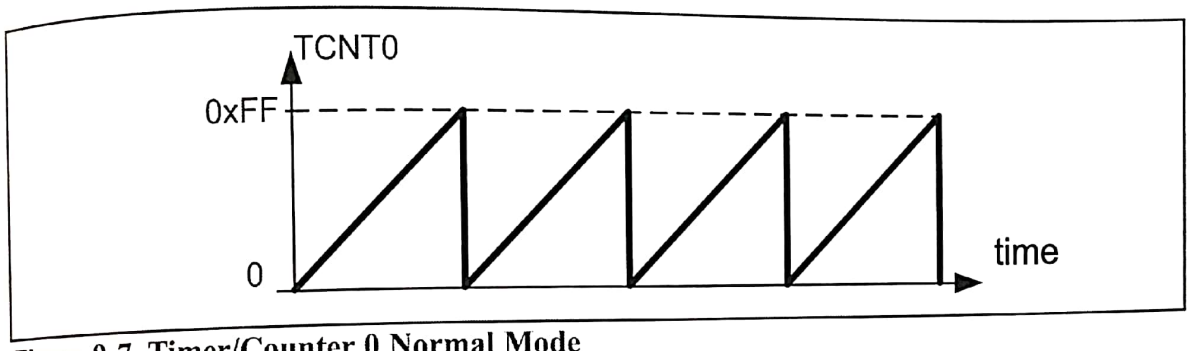


Figure 9-7. Timer/Counter 0 Normal Mode

Steps to program Timer0 in Normal mode

To generate a time delay using Timer0 in Normal mode, the following steps are taken:

1. Load the TCNT0 register with the initial count value.
2. Load the value into the TCCR0A and TCCR0B registers, indicating Normal mode and the porper prescaler option is to be used. When you select the clock source, the timer/counter starts to count, and each tick causes the content of the timer/counter to increment by 1.
3. Keep monitoring the timer overflow flag (TOV0) to see if it is raised. Get out of the loop when TOV0 becomes high.
4. Stop the timer by disconnecting the clock source, using the following instructions:

```
LDI    R20,0x00
OUT    TCCR0B,R20   ;timer stopped, mode=Normal
```

5. Clear the TOV0 flag for the next round.
6. Go back to Step 1 to load TCNT0 again.

To clarify the above steps, see Example 9-3.

Example 9-3

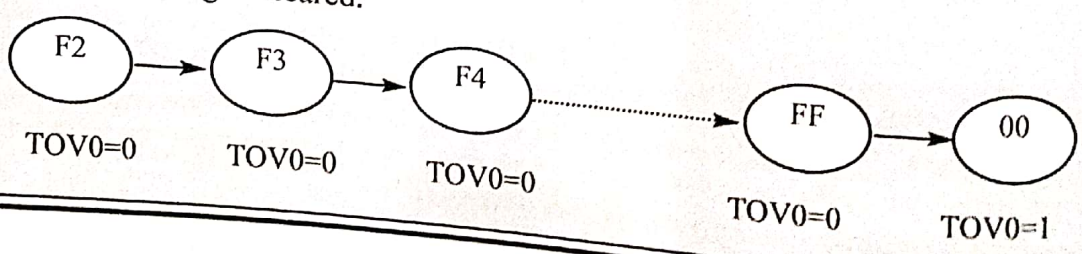
In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

```
.MACRO      INITSTACK          ;set up stack
      LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)
      OUT   SPL,R20
.ENDMACRO
      INITSTACK
      LDI   R16,1<<5     ;R16 = 0x20 (0010 0000 for PB5)
      SBI   DDRB,5       ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17    ;clear PORTB
BEGIN:RCALL DELAY        ;call timer delay
      EOR   R17,R16      ;toggle D5 of R17 by Ex-Oring with 1
      OUT   PORTB,R17    ;toggle PB5
      RJMP  BEGIN
;-------------------Time0 delay
DELAY:LDI   R20,0xF2     ;R20 = 0xF2
      OUT   TCNT0,R20    ;load timer0
      LDI   R20,0x00
      OUT   TCCR0A,R20   ;Normal mode
      LDI   R20,0x01
      OUT   TCCR0B,R20   ;Normal mode, internal clock, no prescaler
AGAIN:SBIS  TIFR0,TOV0   ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0B,R20   ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR0,R20    ;clear TOV0 flag by writing a 1 to TIFR0
      RET
```

Solution:

In the above program notice the following steps:

1. 0xF2 is loaded into TCNT0.
2. TCCR0A and TCCR0B are loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of F3, F4, F5, F6, F7, F8, F9, FA, FB, and so on until it reaches 0xFF. One more clock rolls it to 0, raising the Timer0 flag (TOV0 = 1). At that point, the "`SBIS TIFR0,TOV0`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The TOV0 flag is cleared.



To calculate the exact time delay and the square wave frequency generated on pin PB5, we need to know the XTAL frequency. See Examples 9-4 and 9-5.

Example 9-4

In Example 9-3, calculate the amount of time delay generated by the timer. Assume that XTAL = 16 MHz.

Solution:

We have 16 MHz as the timer frequency. As a result, each clock has a period of T = 1/16 MHz = 0.0625 µs. In other words, Timer0 counts up each 0.0625 µs resulting in delay = number of counts × 0.0625 µs.
The number of counts for the rollover is 0xFF – 0xF2 = 0x0D (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FF to 0 and raises the TOV0 flag. This gives 14 × 0.0625 µs = 0.875 µs for half the pulse.

Example 9-5

In Example 9-3, calculate the frequency of the square wave generated on pin PORTB.5. Assume that XTAL = 16 MHz.

Solution:

To get a more accurate timing, we need to add clock cycles due to the instructions.

```
                                    Cycles
      LDI   R16,1<<5
      SBI   DDRB,5
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY                   3
      EOR   R17,R16                 1
      OUT   PORTB,R17               1
      RJMP  BEGIN                   2
DELAY:LDI   R20,0xF2                1
      OUT   TCNT0,R20               1
      LDI   R20,0x00                1
      OUT   TCCR0A,R20              1
      LDI   R20,0x01                1
      OUT   TCCR0B,R20              1
AGAIN:SBIS  TIFR0,TOV0              1 / 2
      RJMP  AGAIN                   2
      LDI   R20,0x0                 1
      OUT   TCCR0B,R20              1
      LDI   R20,(1<<TOV0)           1
      OUT   TIFR0,R20               1
      RET                           4
                                    25
```

T = 2 × (14 + 25) × 0.0625 µs = 4.875 µs and F = 1 / T = 205.128 kHz.

(a) in hex	(b) in decimal
(FF - XX + 1) × 0.0625 µs where XX is the TCNT0, initial value. Notice that XX value is in hex.	Convert XX value of the TCNT0 register to decimal to get a NNN decimal number, then (256 - NNN) × 0.0625 µs

Figure 9-8. Timer Delay Calculation for XTAL = 16 MHz with No Prescaler

We can develop a formula for delay calculations using the Normal mode of the timer for a crystal frequency of XTAL = 16 MHz. This is given in Figure 9-8. The scientific calculator in the Accessories menu directory of Microsoft Windows can help you find the TCNT0 value. This calculator supports decimal, hex, and binary calculations. See Example 9-6.

Example 9-6

Find the delay generated by Timer0 in the following code, using both of the methods of Figure 9-8. Do not include the overhead due to instructions. (XTAL = 16 MHz)

```
       INITSTACK          ;add its definition from Example 9-3
       LDI    R16,(1<<5)
       SBI    DDRB,5      ;PB5 as an output
       LDI    R17,0
       OUT    PORTB,R17
BEGIN:RCALL DELAY
       EOR    R17,R16     ;toggle D5 of R17
       OUT    PORTB,R17   ;toggle PB5
       RJMP   BEGIN
DELAY:LDI    R20,0x3E
       OUT    TCNT0,R20   ;load timer0
       LDI    R20,0x00
       OUT    TCCR0A,R20  ;Normal mode
       LDI    R20,0x01
       OUT    TCCR0B,R20  ;Normal mode, internal clock, no prescaler
AGAIN:SBIS   TIFR0,TOV0   ;if TOV0 is set skip next instruction
       RJMP   AGAIN
       LDI    R20,0x00
       OUT    TCCR0B,R20          ;stop Timer0
       LDI    R20,(1<<TOV0)       ;R20 = 0x01
       OUT    TIFR0,R20           ;clear TOV0 flag
       RET
```

Solution:

(a) (FF - 3E + 1) = 0xC2 = 194 in decimal and 194 × 0.0625 µs = 12.125 µs.
(b) Because TCNT0 = 0x3E = 62 (in decimal) we have 256 - 62 = 194. This means that the timer counts from 0x3E to 0xFF. This plus rolling over to 0 goes through a total of 194 clock cycles, where each clock is 0.0625 µs in duration. Therefore, we have 194 × 0.125 µs = 12.125 µs as the width of the pulse.

Finding values to be loaded into the timer

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TCNT0 register. To calculate the values to be loaded into the TCNT0 registers, we can use the following steps:

1. Calculate the period of the timer clock using the following formula:
   $T_{clock} = 1/F_{Timer}$
   where F_{Timer} is the frequency of the clock used for the timer. For example, in no prescaler mode, $F_{Timer} = F_{oscillator}$. T_{clock} gives the period at which the timer increments.
2. Divide the desired time delay by T_{clock}. This says how many clocks we need.
3. Perform 256 – *n*, where *n* is the decimal value we got in Step 2.
4. Convert the result of Step 3 to hex, where *xx* is the initial hex value to be loaded into the timer's register.
5. Set TCNT0 = *xx*.

Look at Examples 9-7 and 9-8, where we use a crystal frequency of 16 MHz for the AVR system.

Example 9-7

Assuming that XTAL = 16 MHz, write a program to generate a square wave with a period of 6.25 µs on pin PORTB.3.

Solution:

For a square wave with T = 6.25 µs we must have a time delay of 3.125 µs. Because XTAL = 16 MHz, the counter counts up every 0.0625 µs. This means that we need 3.125 µs / 0.0625 µs = 50 clocks. 256 – 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

```
      INITSTACK          ;add its definition from Example 9-3
      LDI   R16,(1<<3)
      SBI   DDRB,3       ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D3 of R17
      OUT   PORTB,R17    ;toggle PB3
      RJMP BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0xCE
      OUT   TCNT0,R20    ;load Timer0
      LDI   R20,0x00
      OUT   TCCR0A,R20   ;Normal mode
      LDI   R20,0x01
      OUT   TCCR0B,R20   ;Normal mode, int clk, no prescaler
AGAIN:SBIS  TIFR0,TOV0   ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0B,R20   ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR0,R20    ;clear TOV0 flag
      RET
```

Example 9-8

Assuming that XTAL = 16 MHz, modify the program in Example 9-7 to generate a square wave of 32 kHz frequency on pin PORTB.3.

Solution:

Look at the following steps.
(a) T = 1 / F = 1 / 32 kHz = 31.25 µs the period of the square wave.
(b) 1/2 of it for the high and low portions of the pulse is 15.625 µs.
(c) 15.625 µs / 0.0625 µs = 250 and 256 - 250 = 6, which in hex is 0x06.
(d) TCNT0 = 0x06.

Using the Windows calculator to find TCNT0

The scientific calculator in Microsoft Windows is a handy and easy-to-use tool to find the TCNT0 value. Assume that we would like to find the TCNT0 value for a time delay that uses 135 clocks of 0.125 µs. The following steps show the calculation:

1. Bring up the scientific calculator in MS Windows and select decimal.
2. Enter 135.
3. Select hex. This converts 135 to hex, which is 0x87.
4. Select +/- to give -135 decimal (0x79).
5. The lowest two digits (79) of this hex value are for TCNT0. We ignore all the Fs on the left because our number is 8-bit data.

Prescaler and generating a large time delay

As we have seen in the examples so far, the size of the time delay depends on two factors, (a) the crystal frequency, and (b) the timer's 8-bit register. Both of

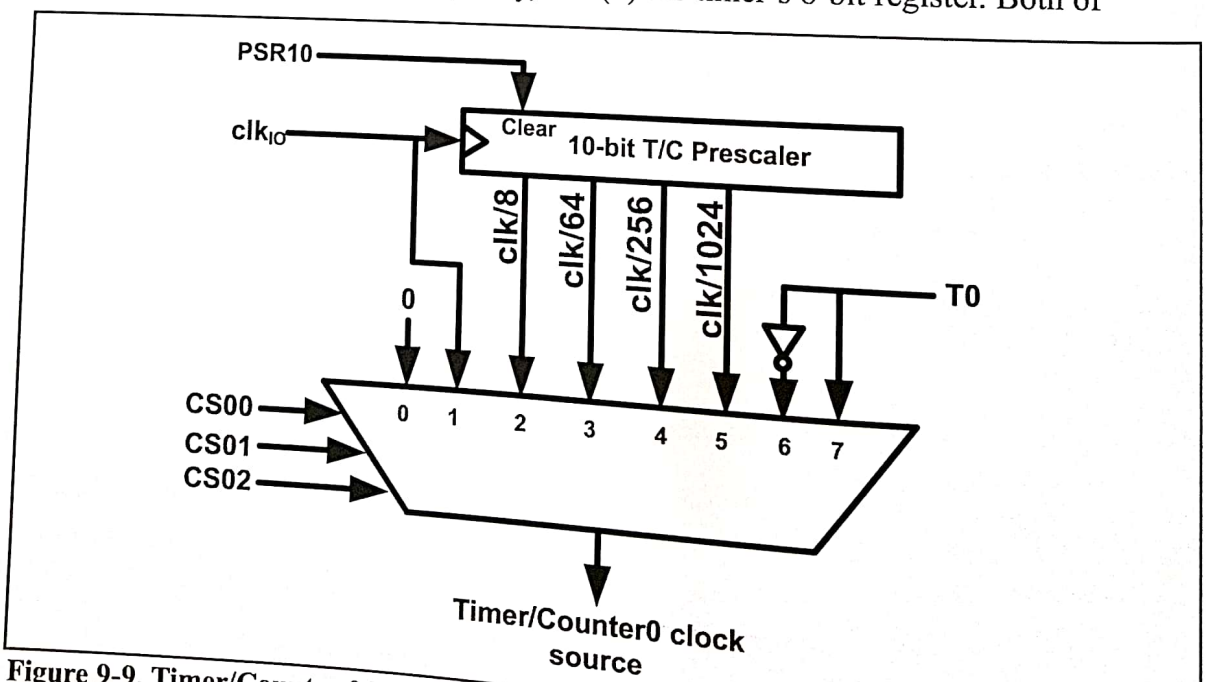


Figure 9-9. Timer/Counter 0 Prescaler

Example 9-9

Modify TCNT0 in Example 9-7 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

Solution:

To get the largest delay we make TCNT0 zero. This will count up from 00 to 0xFF and then roll over to zero.

```
      INITSTACK           ;add its definition from Example 9-3
      LDI   R16,(1<<3)
      SBI   DDRB,3        ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16       ;toggle D3 of R17
      OUT   PORTB,R17     ;toggle PB3
      RJMP  BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0x00
      OUT   TCNT0,R20     ;load Timer0 with zero
      LDI   R20,0x00
      OUT   TCCR0A,R20    ;Normal mode
      LDI   R20,0x01
      OUT   TCCR0B,R20    ;Normal mode, int clk, no prescaler
AGAIN:SBIS  TIFR0,TOV0    ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0B,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR0,R20     ;clear TOV0 flag
      RET
```

Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then will roll over to raise the TCNT0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 0.0625 μs = 16 μs. That gives us the smallest frequency of 1 / (2 × 16 μs) = 1 / (32 μs) = 31.250 kHz.

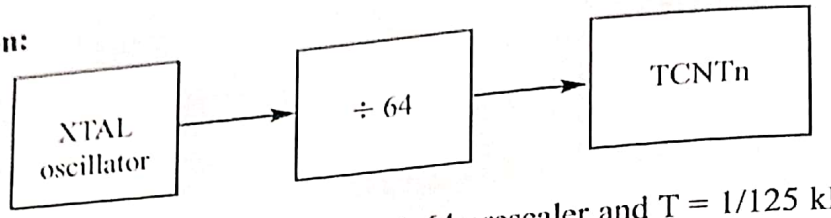
these factors are beyond the control of the AVR programmer. We saw in Example 9-9 that the largest time delay is achieved by making TCNT0 zero. What if that is not enough? We can use the prescaler option in the TCCR0B register to increase the delay by reducing the period. The prescaler option of TCCR0B allows us to divide the instruction clock by a factor of 8 to 1024 as was shown in Figure 9-5. The prescaler of Timer/Counter 0 is shown in Figure 9-9.

As we have seen so far, with no prescaler enabled, the crystal oscillator frequency is fed directly into Timer0. If we enable the prescaler bits in the TCCR0B register, however, then we can divide the clock before it is fed into Timer0. The lower 3 bits of the TCCR0B register give the options of the number we can divide by. As shown in Figure 9-9, this number can be 8, 64, 256, and 1024. Notice that the lowest number is 8 and the highest number is 1024. Examine Examples 9-10 through 9-14 to see how the prescaler options are programmed.

Example 9-10

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that a prescaler of 1:64 is used.
(a) 8 MHz (b) 16 MHz (c) 10 MHz

Solution:



(a) 1/64 × 8 MHz = 125 kHz due to 1:64 prescaler and T = 1/125 kHz = 8 µs
(b) 1/64 × 16 MHz = 250 kHz due to prescaler and T = 1/250 kHz = 4 µs
(c) 1/64 × 10 MHz = 156.2 kHz due to prescaler and T = 1/156 kHz = 6.4 µs

Example 9-11

Find the value for TCCR0B if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

Solution:

From Figure 9-5 we have TCCR0B = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0B=	0	0	0	0	0	0	1	1
	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

Example 9-12

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop. Assume XTAL = 16 MHz.

```
        INITSTACK           ;add its definition from Example 9-3
        LDI    R16,(1<<3)
        SBI    DDRB,3       ;PB3 as an output
        LDI    R17,0
        OUT    PORTB,R17
BEGIN:RCALL DELAY
        EOR    R17,R16      ;toggle D3 of R17
        OUT    PORTB,R17    ;toggle PB3
        RJMP   BEGIN
;----------------- Timer0 Delay
DELAY:LDI    R20,0x10
        OUT    TCNT0,R20    ;load Timer0
        LDI    R20,0x00
        OUT    TCCR0A,R20   ;Normal mode
        LDI    R20,0x03
        OUT    TCCR0B,R20   ;Timer0, Normal mode, int clk, prescaler 64
AGAIN:SBIS   TIFR0,TOV0     ;if TOV0 is set skip next instruction
        RJMP   AGAIN
        LDI    R20,0x0
```

Example 9-12 (Cont.)

```
        OUT     TCCR0B,R20   ;stop Timer0
        LDI     R20,1<<TOV0
        OUT     TIFR0,R20    ;clear TOV0 flag
        RET
```

Solution:

TCNT0 = 0x10 = 16 in decimal and 256 – 16 = 240. Now 240 × 64 × 0.0625 μs = 960 μs, or from Example 9-10, we have 240 × 4 μs = 960 μs.

Example 9-13

Assume XTAL = 16 MHz. (a) Find the clock period fed into Timer0 if a prescaler option of 1024 is chosen. (b) Show what is the largest time delay we can get using this prescaler option and Timer0.

Solution:

(a) 16 MHz × 1/1024 = 15625 Hz due to 1:1024 prescaler and T = 1/15625 Hz = 64 μs = 0.064 ms

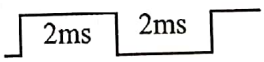
(b) To get the largest delay, we make TCNT0 zero. Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then roll over to raise the TOV0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 64 μs = 16,384 μs = 0.016384 seconds.

Example 9-14

Assuming XTAL = 16 MHz, write a program to generate a square wave of 250 Hz frequency on pin PORTB.3. Use Timer0, Normal mode, with prescaler = 256.

Solution:

Look at the following steps:
(a) T = 1 / 250 Hz = 4 ms, the period of the square wave.
(b) 1/2 of it for the high and low portions of the pulse = 2 ms



(c) (2 ms / 0.0625 μs) / 256 = 125 and 256 – 125 = 131 in decimal, and in hex it is 0x83.
(d) TCNT0 = 83 (hex)

```
        .MACRO INITSTACK          ;set up stack
        LDI    R20,HIGH(RAMEND)
        OUT    SPH,R20
        LDI    R20,LOW(RAMEND)
        OUT    SPL,R20
.ENDMACRO
        INITSTACK
        LDI    R16,(1<<3)
        SBI    DDRB,3        ;PB3 as an output
```

Example 9-14 (Cont.)

```
        LDI   R17,0
BEGIN:OUT   PORTB,R17   ;PORTB = R17
        CALL  DELAY
        EOR   R17,R16   ;Toggle D3 of R17
        RJMP  BEGIN

;---------------- Timer0 Delay
DELAY:LDI   R20,0x83
      OUT   TCNT0,R20   ;load Timer0
      LDI   R20,0x00
      OUT   TCCR0A,R20  ;Normal mode
      LDI   R20,0x04
      OUT   TCCR0B,R20  ;Normal mode, int clk, prescaler 256

AGAIN:SBIS  TIFR0,TOV0  ;if TOV0 is set skip next instruction
      RJMP  AGAIN

      LDI   R20,0x00
      OUT   TCCR0B,R20  ;stop Timer0
      LDI   R20,1<<TOV0
      OUT   TIFR0,R20   ;clear TOV0 flag
      RET
```

Assemblers and negative values

Because the timer is in 8-bit mode, we can let the assembler calculate the value for TCNT0. For example, in the "`LDI R20,-100`" instruction, the assembler will calculate the -100 = 9C and make R20 = 9C in hex. This makes our job easier. See Examples 9-15 and 9-16.

Example 9-15

Find the value (in hex) loaded into TCNT0 for each of the following cases.

(a)
```
LDI R20, -200
OUT TCNT0,R20
```

(b)
```
LDI R17,-60
OUT TCNT0,R17
```

(c)
```
LDI R25,-12
OUT TCNT0,R25
```

Solution:

You can use the Windows scientific calculator to verify the results provided by the assembler. In the Windows calculator, select decimal and enter 200. Then select hex, then +/- to get the negative value. The following is what we get.

Decimal	*2's complement (TCNT0 value)*
-200	0x38
-60	0xC4
-12	0xF4

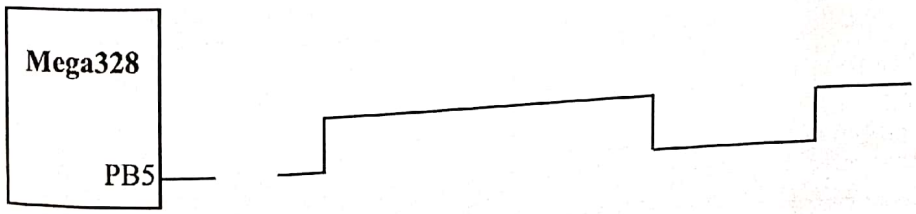
Example 9-16

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave. Assume XTAL = 16 MHz.

```
        LDI     R16,HIGH(RAMEND)
        OUT     SPH,R16
        LDI     R16,LOW(RAMEND)
        OUT     SPL,R16             ;initialize stack pointer

        SBI     DDRB,5              ;PB5 as an output
        LDI     R18,-150
BEGIN:  SBI     PORTB,5             ;PB5 = 1
        OUT     TCNT0,R18           ;load Timer0 byte
        CALL    DELAY
        OUT     TCNT0,R18           ;reload Timer0 byte
        CALL    DELAY
        CBI     PORTB,5             ;PB5 = 0
        OUT     TCNT0,R18           ;reload Timer0 byte
        CALL    DELAY
        RJMP    BEGIN

;----- Delay using Timer0
DELAY:  LDI     R20,0x00
        OUT     TCCR0A,R20  ;Normal mode
        LDI     R20,0x01
        OUT     TCCR0B,R20  ;start Timer0, Normal mode, int clk, no prescaler
AGAIN:  SBIS    TIFR0,TOV0  ;monitor TOV0 flag and skip if high
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0B,R20  ;stop Timer0
        LDI     R20,1<<TOV0
        OUT     TIFR0,R20   ;clear TOV0 flag bit
        RET
```

Solution:

For the TCNT0 value in 8-bit mode, the conversion is done by the assembler as long as we enter a negative number. This also makes the calculation easy. Because we are using 150 clocks, we have time for the DELAY subroutine = 150 × 0.0625 μs = 9.375 μs. The high portion of the pulse is twice the size of the low portion (66% duty cycle). Therefore, we have: T = high portion + low portion = 2 × 9.375 μs + 9.375 μs = 28.125 μs and frequency = 1 / 28.125 μs = 35.555 kHz.



Clear Timer0 on compare match (CTC) mode programming

Examining Figure 9-2 once more, we see the OCR0A register. The OCR0A register is used with CTC mode. As with the Normal mode, in the CTC mode, the timer is incremented with a clock. But it counts up until the content of the TCNT0 register becomes equal to the content of OCR0A (compare match occurs); then, the timer will be cleared and the OCF0A flag will be set when the next clock occurs. The OCF0A flag is located in the TIFR0 register. See Figure 9-10 and Examples 9-17 through 9-21.

Example 9-17

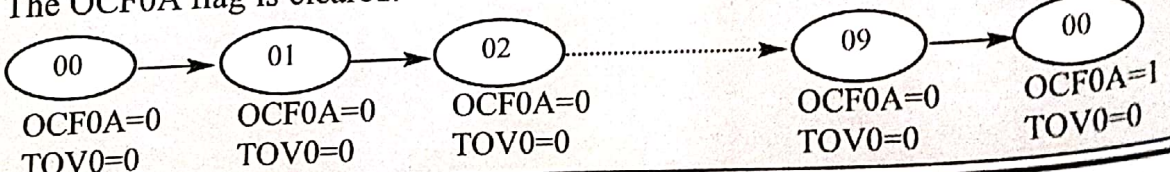
In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.3 bit. Timer0 is used to generate the time delay. Analyze the program.

```
        INITSTACK                ;add its definition from Example 9-3
        LDI    R16,(1<<3)        ;PB3 as an output
        SBI    DDRB,3
        LDI    R17,0             ;PORTB = R17
BEGIN:OUT    PORTB,R17
        RCALL  DELAY             ;toggle D3 of R17
        EOR    R17,R16
        RJMP   BEGIN
;---------------- Timer0 Delay
DELAY:LDI    R20,0
        OUT    TCNT0,R20
        LDI    R20,9             ;load OCR0A
        OUT    OCR0A,R20
        LDI    R20,(1<<WGM01)    ;CTC mode
        OUT    TCCR0A,R20
        LDI    R20,1             ;CTC mode, int clk
        OUT    TCCR0B,R20        ;if OCF0 is set skip next inst.
AGAIN:SBIS   TIFR0,OCF0A
        RJMP   AGAIN
        LDI    R20,0x0           ;stop Timer0
        OUT    TCCR0B,R20
        LDI    R20,1<<OCF0A      ;clear OCF0A flag
        OUT    TIFR0,R20
        RET
```

Solution:

In the above program notice the following steps:

1. 9 is loaded into OCR0A.
2. TCCR0 is loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of 00, 01, 02, 03, and so on until it reaches 9. One more clock rolls it to 0, raising the Timer0 compare match flag (OCF0A = 1). At that point, the "`SBIS TIFR0,OCF0A`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The OCF0A flag is cleared.



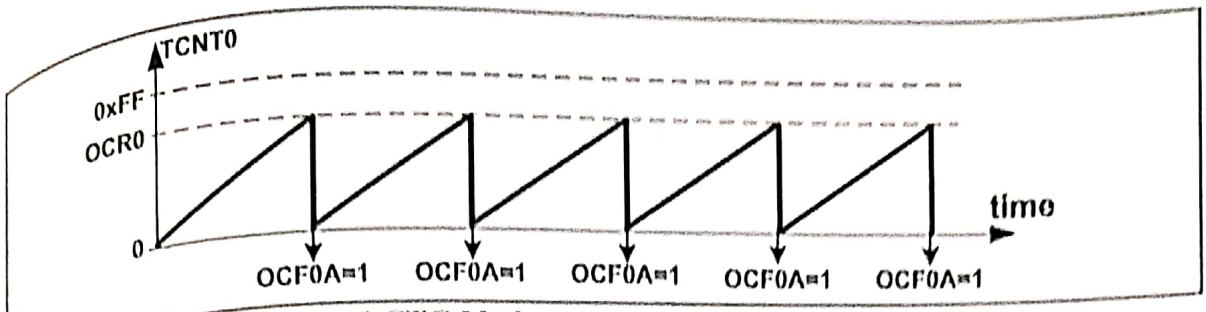


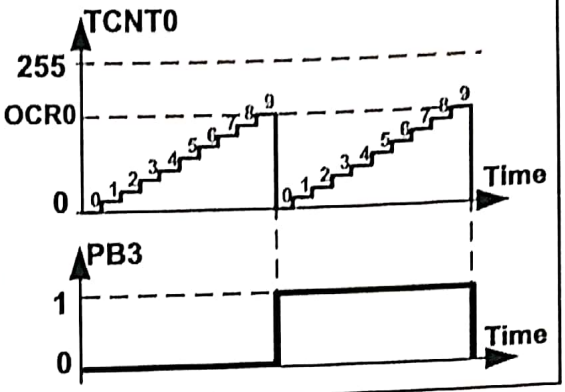
Figure 9-10. Timer/Counter 0 CTC Mode

Example 9-18

Find the delay generated by Timer0 in Example 9-17. Do not include the overhead due to instructions. (XTAL = 16 MHz)

Solution:

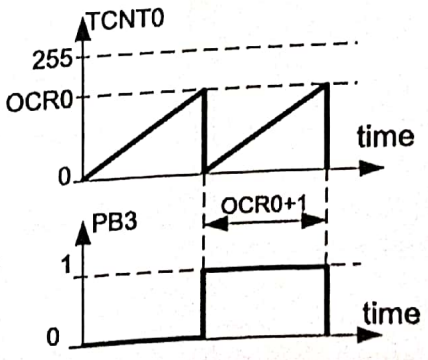
OCR0A is loaded with 9 and TCNT0 is cleared; Thus, after 9 clocks TCNT0 becomes equal to OCR0A. On the next clock, the OCF0 flag is set and the reset occurs. That means the TCNT0 is cleared after 9 + 1 = 10 clocks. Because XTAL = 16 MHz, the counter counts up every 0.0625 µs. Therefore, we have 10 × 0.0625 µs = 0.625 µs.



Example 9-19

Find the delay generated by Timer0 in the following program. Do not include the overhead due to instructions. (XTAL = 16 MHz)

```
      LDI   R16,(1<<3)
      SBI   DDRB,3        ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
      LDI   R20,89
      OUT   OCR0A,R20     ;load Timer0
BEGIN:LDI   R20,(1<<WGM01)
      OUT   TCCR0A,R20    ;CTC mode
      LDI   R20,0x03
      OUT   TCCR0B,R20    ;CTC mode, prescaler = 64
AGAIN:SBIS  TIFR0,OCF0A   ;if OCF0 flag is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0B,R20    ;stop Timer0 (This line can be omitted)
      LDI   R20,1<<OCF0A
      OUT   TIFR0,R20     ;clear OCF0 flag
      EOR   R17,R16       ;toggle D3 of R17
      OUT   PORTB,R17     ;toggle PB3
      RJMP  BEGIN
```



Solution:

Due to prescaler = 64 each timer clock lasts 64 × 0.0625 µs = 4 µs. OCR0A is loaded with 89; thus, after 90 clocks OCF0A is set. Therefore we have 90 × 4 µs = 360 µs.

Example 9-20

Assuming XTAL = 16 MHz, write a program to generate a delay of 12.8 ms. Use Timer0, CTC mode, with prescaler = 1024.

Solution:

Due to prescaler = 1024 each timer clock lasts 1024 × 0.0625 µs = 64 µs. Thus, in order to generate a delay of 12.8 ms we should wait 12.8 ms / 64 µs = 200 clocks. Therefore the OCR0A register should be loaded with 200 - 1 = 199.

```
DELAY:LDI    R20,0
      OUT    TCNT0,R20
      LDI    R20,199
      OUT    OCR0A,R20          ;load OCR0A
      LDI    R20,(1<<WGM01)
      OUT    TCCR0A,R20
      LDI    R20,0x05
      OUT    TCCR0B,R20         ;CTC mode, prescaler = 1024
AGAIN:SBIS   TIFR0,OCF0A        ;if OCF0A is set skip next inst.
      RJMP   AGAIN
      LDI    R20,0x0
      OUT    TCCR0B,R20         ;stop Timer0
      LDI    R20,1<<OCF0A
      OUT    TIFR0,R20          ;clear OCF0 flag
      RET
```

Example 9-21

Assuming XTAL = 16 MHz, write a program to generate a delay of 1 ms.

Solution:

As XTAL = 16 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	16 MHz	1/16 MHz = 0.0625 µs	1 ms/0.0625 µs = 16000
8	16 MHz/8 = 2 MHz	1/2 MHz = 0.5 µs	1 ms/0.5 µs = 2000
64	16 MHz/64 = 250 kHz	1/250 kHz = 4 µs	1 ms/4 µs = **250**
256	16 MHz/256 = 62.5 kHz	1/62.5 kHz = 16 µs	1 ms/16 µs = **62.5**
1024	16 MHz/1024 = 15.625 kHz	1/15.625 kHz= 64 µs	1 ms/64 µs = **15.625**

From the above calculation we can only use the options Prescaler = 64, Prescaler = 256, or Prescaler = 1024. We should use the option Prescaler = 64 since we cannot use a decimal point. To wait 250 clocks we should load OCR0A with 250 - 1 = 249.

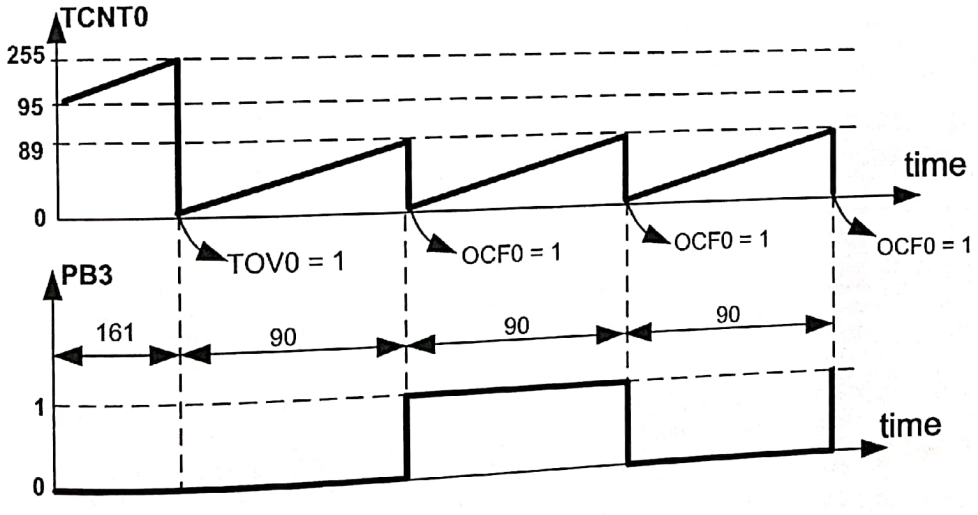
```
DELAY:LDI    R20,0
      OUT    TCNT0,R20          ;TCNT0 = 0
      LDI    R20,249
      OUT    OCR0A,R20          ;OCR0 = 249
      LDI    R20,(1<<WGM01)
      OUT    TCCR0A,R20         ;CTC mode
      LDI    R20,0x03
      OUT    TCCR0B,R20         ;CTC mode, prescaler = 64
AGAIN:SBIS   TIFR0,OCF0A        ;if OCF0A is set skip next instruction
      RJMP   AGAIN
      LDI    R20,0x0
      OUT    TCCR0B,R20         ;stop Timer0
      LDI    R20,1<<OCF0A
      OUT    TIFR0,R20          ;clear OCF0A flag
      RET
```

Notice that the comparator checks for equality; thus, if we load the OCR0A register with a value that is smaller than TCNT0's value, the counter will miss the compare match and will count up until it reaches the maximum value of $FF and rolls over. This causes a big delay and is not desirable in many cases. See Example 9-22.

Example 9-22

In the following program, how long does it take for the PB3 to become one? Do not include the overhead due to instructions. (XTAL = 16 MHz)

```
        SBI   DDRB,3              ;PB3 as an output
        CBI   PORTB,3             ;PB3 = 0
        LDI   R20,89
        OUT   OCR0A,R20           ;OCR0 = 89
        LDI   R20,95
        OUT   TCNT0,R20           ;TCNT0 = 95
        LDI   R16,(1<<3)
BEGIN:  LDI   R20,(1<<WGM01)
        OUT   TCCR0A,R20          ;CTC mode
        LDI   R20,1
        OUT   TCCR0B,R20          ;CTC mode, prescaler = 1
AGAIN:  SBIS  TIFR0,OCF0A         ;if OCF0A flag is set skip next inst.
        RJMP  AGAIN
        LDI   R20,0x00
        OUT   TCCR0B,R20          ;stop Timer0 (This line can be omitted)
        LDI   R20,1<<OCF0A
        OUT   TIFR0,R20    ;clear OCF0A flag
        EOR   R17,R16      ;toggle D3 of R17
        OUT   PORTB,R17    ;toggle PB3
        RJMP  BEGIN
```



Solution:

Since the value of TCNT0 (95) is bigger than the content of OCR0A (89), the timer counts up until it gets to $FF and rolls over to zero. The TOV0 flag will be set as a result of the overflow. Then, the timer counts up until it becomes equal to 89 and compare match occurs. Thus, the first compare match occurs after 161 + 90 = 251 clocks, which means after 251 × 0.0625 µs = 15.6875 µs. The next compare matches occur after 90 clocks, which means after 90 × 0.0625 µs = 5.625 µs.

Timer2 programming

See Figure 9-12. Timer2 is an 8-bit timer. Therefore it works the same way as Timer0. But there are three differences between Timer0 and Timer2:

1. Timer2 can be used as a real time counter. To do so, we should connect a crystal of 32.768 kHz to the TOSC1 and TOSC2 pins of AVR and set the AS2 bit. See Figure 9-13. For more information about this feature, see the AVR datasheet.

2. The TCNT2, TCCR2A, and TCCR2B are in extended I/O memory. So, they can be accessed using STS and LDS instructions.

3. In Timer0, when CS02–CS00 have values 110 or 111, Timer0 counts the external events. But in Timer2, the multiplexer selects between the different scales of the clock. In other words, the same values of the CS bits can have different meanings for Timer0 and Timer2. See Figure 9-12A. Compare Figure 9-11 with Figure 9-5 and examine Examples 9-23 through 9-25.

Bit	7	6	5	4	3	2	1	0
	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

WGM20, WGM21 Timer2 mode selector bits

Mode	WGM22 (TCCR2B)	WGM21 (TCCR2A)	WGM20	Timer/Counter Mode of Operation	Top	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	Phase Correct PWM	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCR2A	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	Phase Correct PWM	OCR2A	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCR2A	BOTTOM	TOP

COM21:20 D5 D4 Compare Output Mode:
These bits control the waveform generator (See Chapter 15).

Figure 9-11A. TCCR2A (Timer/Counter Control Register) Register

Bit	7	6	5	4	3	2	1	0
TCCR2B:	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20
Read/Write	W	W	R	R	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

CS22:20 D2 D1 D0 Timer2 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 / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

FOC2A, FOC2B D7, D6 Force compare match (Discussed in Chapter 15)

Figure 9-11B. TCCR2B (Timer/Counter Control Register) Register

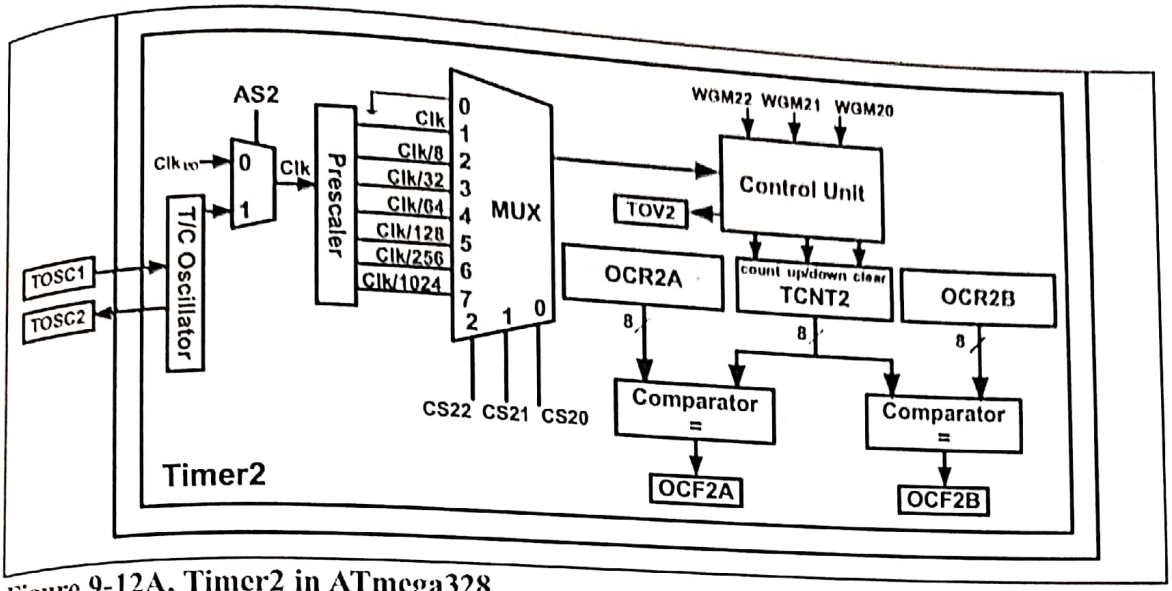


Figure 9-12A. Timer2 in ATmega328

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	OCF2B	OCF2A	TOV2
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV2 D0 Timer2 overflow flag bit
0 = Timer2 did not overflow.
1 = Timer2 has overflowed (going from $FF to $00).

OCF2A D1 Timer2 output compare A match flag
0 = compare match did not occur.
1 = compare match occurred.

OCF2B D2 Timer2 output compare B match flag
0 = compare match did not occur.
1 = compare match occurred.

Figure 9-12B. TIFR2 (Timer/Counter Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 When it is zero, Timer2 is clocked from $clk_{I/O}$. When it is set, Timer2 works as RTC.

Figure 9-13. ASSR (Asynchronous Status Register)

Example 9-23

Find the value for TCCR2B if we want to program Timer2 in Normal mode with a prescaler of 64 using internal clock for the clock source.

Solution:

From Figure 9-11 we have TCCR2 = 0000 0100; XTAL clock source, prescaler of 64.

TCCR2B=	0	0	0	0	0	1	0	0
	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20

Compare the answer with Example 9-11.

Example 9-24

Using a prescaler of 64, write a program to generate a delay of 960 µs. Assume XTAL = 16 MHz.

Solution:

Timer clock = 16 MHz/64 = 250 kHz → Timer Period = 1 / 250 kHz = 4 µs → Timer Value = 960 µs / 4 µs = 240

```
;---------------- Timer2 Delay
DELAY:LDI   R20,-240     ;R20 = 0x10
      STS   TCNT2,R20    ;load Timer2
      LDI   R20,0x00
      STS   TCCR2A,R20
      LDI   R20,0x04
      STS   TCCR2B,R20   ;Timer2, Normal mode, int clk, prescaler 64
AGAIN:SBIS  TIFR2,TOV2   ;if TOV2 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      STS   TCCR2B,R20   ;stop Timer2
      LDI   R20,1<<TOV2
      OUT   TIFR2,R20    ;clear TOV2 flag
      RET
```

Compare the above program with the DELAY subroutine in Example 9-12. There are three differences between the two programs:

1. The register names are different. For example, we use TCNT2 instead of TCNT0.
2. The values of TCCRnB are different for the same prescaler.
3. TCNT2, TCCR2A, and TCCR2B are accessed using STS.

Example 9-25

Using CTC mode, write a program to generate a delay of 4 ms. Assume XTAL = 16 MHz.

Solution:

As XTAL = 16 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	16 MHz	1/16 MHz = 0.0625 µs	4 ms / 0.0625 µs = 64k
8	16 MHz/8 = 2 MHz	1/2 MHz = 0.5 µs	4 ms / 0.5 µs = 8000
32	16 MHz/32 = 500 kHz	1/500 kHz = 2 µs	4 ms / 2 µs = 2000
64	16 MHz/64 = 250 kHz	1/250 kHz = 4 µs	4 ms / 4 µs = 1000
128	16 MHz/128 = 125 kHz	1/125 kHz = 8 µs	4 ms / 8 µs = 500
256	16 MHz/256 = 62.5 kHz	1/62.5 kHz = 16 µs	4 ms / 16 µs = **250**
1024	16 MHz/1024 = 15.625 kHz	1/15.625 kHz= 64 µs	4 ms / 64 µs = **62.5**

From the above calculation we can only use options Prescaler = 256 or Prescaler = 1024. We should use the option Prescaler = 256 since we cannot use a decimal point. To wait 250 clocks we should load OCR2 with 250 – 1 = 249.

Example 9-25 (Cont.)

TCCR2 =

0	0	0	0	0	1	1	0
FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20

```
;---------------- Timer2 Delay
DELAY:LDI   R20,0
      STS   TCNT2,R20          ;TCNT2 = 0
      LDI   R20,249
      STS   OCR2A,R20          ;OCR2A = 249
      LDI   R20,(1<<WGM21)
      STS   TCCR2A,R20
      LDI   R20,0x06
      STS   TCCR2B,R20         ;CTC mode,prescaler = 256
AGAIN:SBIS  TIFR2,OCF2A        ;if OCF2A is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      STS   TCCR2B,R20         ;stop Timer2
      LDI   R20,1<<OCF2A
      OUT   TIFR2,R20          ;clear OCF2A flag
      RET
```

Timer1 programming

Timer1 is a 16-bit timer and has lots of capabilities. Next, we discuss Timer1 and its capabilities.

Since Timer1 is a 16-bit timer its 16-bit register is split into two bytes. These are referred to as TCNT1L (Timer1 low byte) and TCNT1H (Timer1 high byte). See Figure 9-15. Timer1 also has three control registers named TCCR1A (Timer/counter 1 control register), TCCR1B, and TCCR1C. The TOV1 (timer overflow) flag bit goes HIGH when overflow occurs. Timer1 also has the prescaler options of 1:1, 1:8, 1:64, 1:256, and 1:1024. See Figure 9-14 for the Timer1 block diagram and Figures 9-17 and 9-18 for TCCR1 register options. There are two OCR registers in Timer1: OCR1A and OCR1B. There are two separate flags for each of the OCR registers, which act independently of each other. Whenever TCNT1 equals OCR1A, the OCF1A flag will

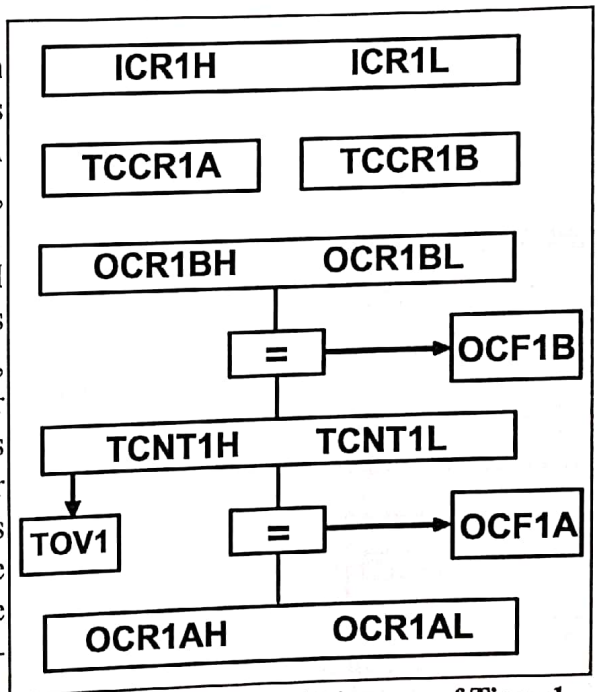


Figure 9-14. Simplified Diagram of Timer1

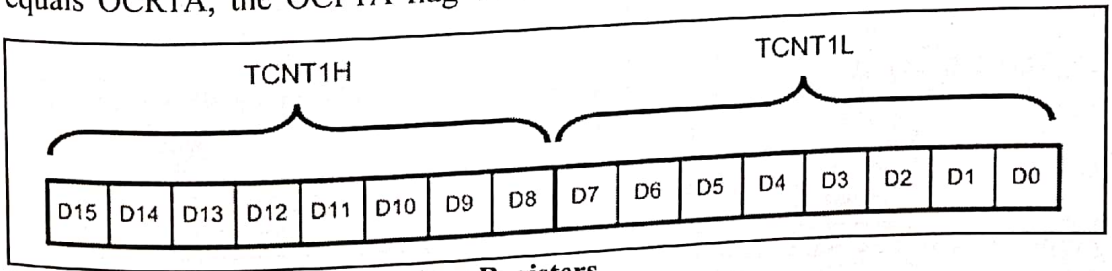


Figure 9-15. Timer1 High and Low Registers

be set on the next timer clock. When TCNT equals OCR1B, the OCF1B flag will be set on the next clock. As Timer1 is a 16-bit timer, the OCR registers are 16-bit registers as well and they are made of two 8-bit registers. For example, OCR1A is made of OCR1AH (OCR1A high byte) and OCR1AL (OCR1A low byte). For a detailed view of Timer1 see Figure 9-3.

The TIFR1 register contains the TOV1, OCF1A, and OCF1B flags. See Figure 9-16.

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	OCF1B	OCF1A	TOV1
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV D0 Timer1 overflow flag bit
0 = Timer1 did not overflow.
1 = Timer1 has overflowed (going to $00).

OCFA D1 Timer1 output compare A match flag
0 = compare match did not occur.
1 = compare match occurred.

OCFB D2 Timer1 output compare B match flag
0 = compare match did not occur.
1 = compare match occurred.

Figure 9-16. TIFR1 (Timer/Counter Interrupt Flag Register)

There is also an auxiliary register named ICR1, which is used in operations such as capturing. ICR1 is a 16-bit register made of ICR1H and ICR1L, as shown in Figure 9-19.

WGM13:10

The WGM13, WGM12, WGM11, and WGM10 bits define the mode of Timer1, as shown in Figure 9-17B. Timer1 has 16 different modes. One of them (mode 13) is reserved (not implemented). In this chapter, we cover mode 0 (Normal mode) and mode 4 (CTC mode). The other modes will be covered in Chapters 15 and 16.

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	-	-	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 (discussed in Chapter 15)

COM1B1:COM1B0 D5 D4 Compare Output Mode for Channel B (discussed in Chapter 15)

WGM11:10 D1 D0 Timer1 mode (discussed in Figure 9-17B)

Figure 9-17A. TCCR1A (Timer 1 Control) Register

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 Canceler
0 = Input Capture is disabled.
1 = Input Capture 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

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 9-17B. TCCR1B (Timer 1 Control) Register

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

FOC1A,FOC1B D7, D6 Force Output Compare (discussed in Chapter 15)

Figure 9-18. TCCR1C (Timer 1 Control) Register

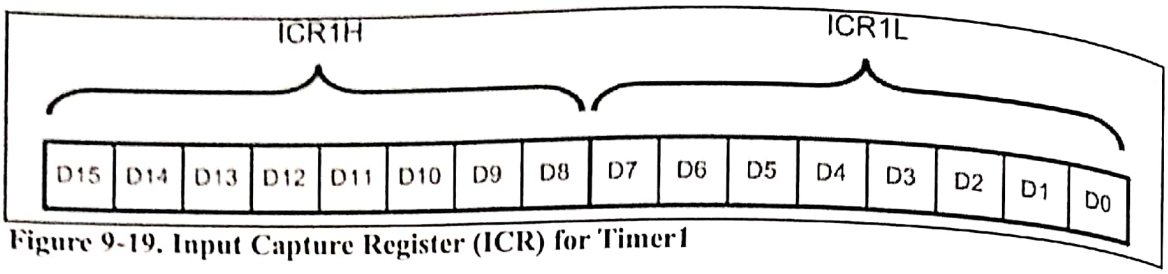


Figure 9-19. Input Capture Register (ICR) for Timer1

Timer1 operation modes

Normal mode (WGM13:10 = 0000)

In this mode, the timer counts up until it reaches $FFFF (which is the maximum value) and then it rolls over from $FFFF to 0000. When the timer rolls over from $FFFF to 0000, the TOV1 flag will be set. See Figure 9-20 and Examples 9-26 and 9-27. In Example 9-27, a delay is generated using Normal mode.

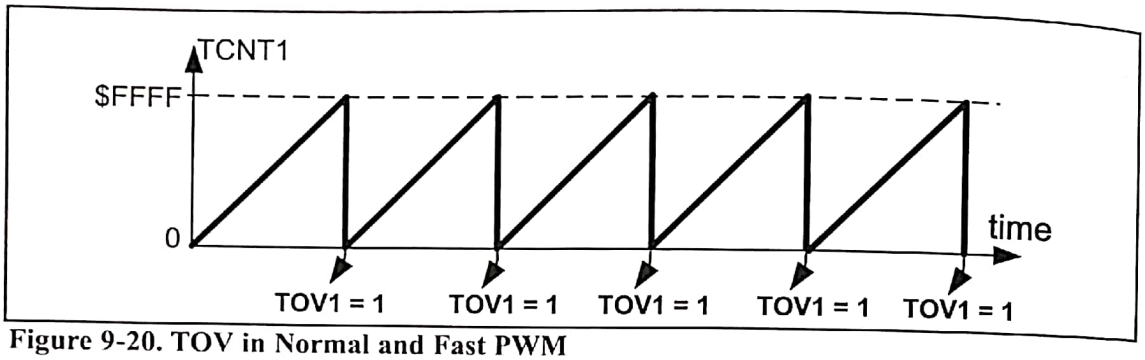


Figure 9-20. TOV in Normal and Fast PWM

CTC mode (WGM13:10 = 0100)

In mode 4, the timer counts up until the content of the TCNT1 register becomes equal to the content of OCR1A (compare match occurs); then, the timer will be cleared when the next clock occurs. The OCF1A flag will be set as a result of the compare match as well. See Figure 9-21 and Examples 9-28 and 9-29.

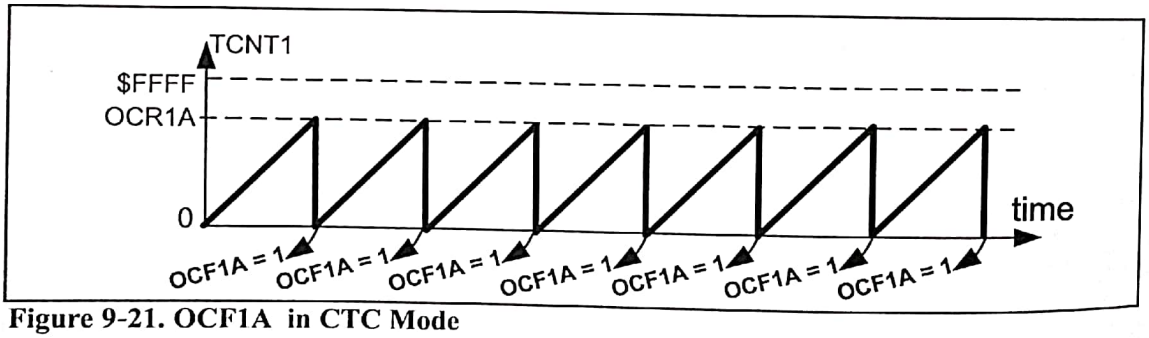


Figure 9-21. OCF1A in CTC Mode

Example 9-26

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 0 (Normal), with no prescaler. Use AVR's crystal oscillator for the clock source.

Solution:
TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 0001 WGM13 = 0, WGM12 = 0, oscillator clock source, no prescaler

Example 9-27

Find the frequency of the square wave generated by the following program if XTAL = 16 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
        INITSTACK               ;add ILS definition from Example 9-3
        LDI     R16,0x20
        SBI     DDRB,5          ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17       ;PB5 = 0
BEGIN:  RCALL   DELAY
        EOR     R17,R16         ;toggle D5 of R17
        OUT     PORTB,R17       ;toggle PB5
        RJMP    BEGIN
;--------------- Timer1 delay
DELAY:  LDI     R20,0xD8
        STS     TCNT1H,R20      ;TCNT1H = 0xD8
        LDI     R20,0xF0
        STS     TCNT1L,R20      ;TCNT1L = 0xF0
        LDI     R20,0x00
        STS     TCCR1A,R20      ;WGM11:10 = 00
        LDI     R20,0x01
        STS     TCCR1B,R20      ;WGM13:12 = 00, Normal mode, prescaler = 1
AGAIN:  SBIS    TIFR1,TOV1      ;if TOV1 is set skip next instruction
        RJMP    AGAIN
        LDI     R20,0x00
        STS     TCCR1B,R20      ;stop Timer1
        LDI     R20,(1<<TOV1)
        OUT     TIFR1,R20       ;clear TOV1 flag
        RET
```

Solution:

WGM13:10 = 0000 = 0x00, so Timer1 is working in mode 0, which is Normal mode, and the top is 0xFFFF.
FFFF + 1 - D8F0 = 0x2710 = 10,000 clocks, which means that it takes 10,000 clocks. As XTAL = 16 MHz each clock lasts 1/(16M) = 0.0625 μs and delay = 10,000 × 0.0625 μs = 625 μs = 0.625 ms and frequency = 1 / (0.625 ms × 2) = 800 Hz.
In this calculation, the overhead due to all the instructions in the loop is not included.

Notice that instead of using hex numbers we can use HIGH and LOW directives, as shown below:

```
        LDI     R20,HIGH (65536-10000)      ;load Timer1 high byte
        STS     TCNT1H,R20  ;TCNT1H = 0xD8
        LDI     R20,LOW (65536-10000)       ;load Timer1 low byte
        STS     TCNT1L,R20  ;TCNT1L = 0xF0
```

or we can simply write it as follows:

```
        LDI     R20,HIGH (-10000)           ;load Timer1 high byte
        STS     TCNT1H,R20  ;TCNT1H = 0xD8
        LDI     R20,LOW (-10000)            ;load Timer1 low byte
        STS     TCNT1L,R20  ;TCNT1L = 0xF0
```

Example 9-28

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 4 (CTC, Top = OCR1A), no prescaler. Use AVR's crystal oscillator for the clock source.

Solution:

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 1001 WGM13 = 0, WGM12 = 1, oscillator clock source, no prescaler

Example 9-29

Find the frequency of the square wave generated by the following program if XTAL = 16 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
       SBI    DDRB,5              ;PB5 as an output
BEGIN:SBI     PORTB,5             ;PB5 = 1
       RCALL  DELAY
       CBI    PORTB,5             ;PB5 = 0
       RCALL  DELAY
       RJMP   BEGIN
;--------------- Timer1 delay
DELAY:LDI     R20,0x00
       STS    TCNT1H,R20
       STS    TCNT1L,R20          ;TCNT1 = 0
       LDI    R20,0
       STS    OCR1AH,R20
       LDI    R20,159
       STS    OCR1AL,R20          ;OCR1A = 159 = 0x9F
       LDI    R20,0x0
       STS    TCCR1A,R20          ;WGM11:10 = 00
       LDI    R20,0x09
       STS    TCCR1B,R20          ;WGM13:12 = 01,CTC mode, prescaler = 1
AGAIN:SBIS    TIFR1,OCF1A         ;if OCF1A is set skip next instruction
      RJMP    AGAIN
      LDI     R20,1<<OCF1A
      OUT     TIFR1,R20           ;clear OCF1A flag
      LDI     R19,0
      STS     TCCR1B,R19          ;stop timer
      STS     TCCR1A,R19
      RET
```

Solution:

WGM13:10 = 0100 = 0x04 therefore, Timer1 is working in mode 4, which is a CTC mode, and max is defined by OCR1A.
159 + 1 = 160 clocks
XTAL = 16 MHz, so each clock lasts 1/(16M) = 0.0625 µs.
Delay = 160 × 0.0625 µs = 10 µs and frequency = 1 / (10 µs × 2) = 50 kHz.
In this calculation, the overhead due to all the instructions in the loop is not included.

Accessing 16-bit registers

The AVR is an 8-bit microcontroller, which means it can manipulate data 8 bits at a time, only. But some Timer1 registers, such as TCNT1, OCR1A, ICR1, and so on, are 16-bit; in this case, the registers are split into two 8-bit registers, and each one is accessed individually. This is fine for most cases. For example, when we want to load the content of SP (stack pointer), we first load one half and then the other half, as shown below:

```
LDI   R16, 0x12
OUT   SPL, R16
LDI   R16, 0x34
OUT   SPH, R16    ;SP = 0x3412
```

In 16-bit timers, however, we should read/write the entire content of a register at once, otherwise we might have problems. For example, imagine the following scenario:

The TCNT1 register contains 0x15FF. We read the low byte of TCNT1, which is 0xFF, and store it in R20. At the same time a timer clock occurs, and the content of TCNT1 becomes 0x1600; now we read the high byte of TCNT1, which is now 0x16, and store it in R21. If we look at the value we have read, R21:R20 = 0x16FF. So, we believe that TCNT1 contains 0x16FF, although it actually contains 0x15FF.

This problem exists in many 8-bit microcontrollers. But the AVR designers have resolved this issue with an 8-bit register called TEMP, which is used as a buffer. See Figure 9-22. When we write or read the high byte of a 16-bit register, the value will be written into the TEMP register. When we write into the low byte of a 16-bit register, the content of TEMP will be written into the high byte of the 16-bit register as well. For example, consider the following program:

```
LDI   R16, 0x15
STS   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
LDI   R16, 0xFF
STS   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
```

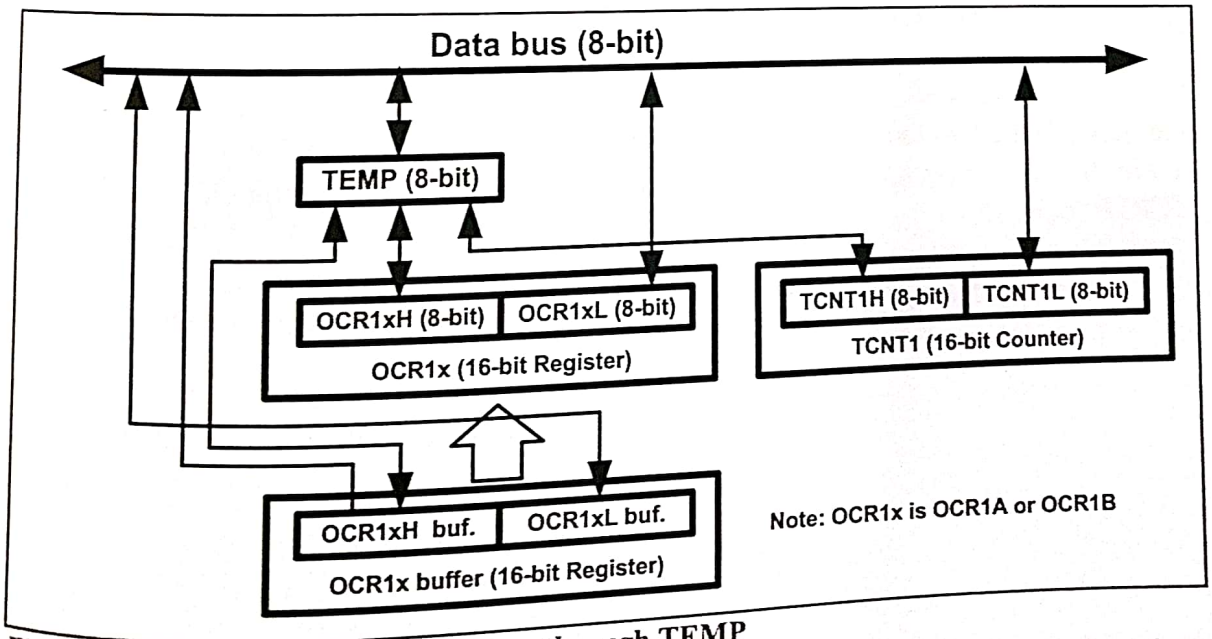


Figure 9-22. Accessing 16-bit Registers through TEMP

After the execution of "STS TCNT1H, R16", the content of R16, 0x15, will be stored in the TEMP register. When the instruction "STS TCNT1L, R16" is executed, the content of R16, 0xFF, is loaded into TCNT1L, and the content of the TEMP register, 0x15, is loaded into TCNT1H. So, 0x15FF will be loaded into the TCNT1 register at once.

Notice that according to the internal circuitry of the AVR, we should first write into the high byte of the 16-bit registers and then write into the lower byte. Otherwise, the program does not work properly. For example, the following code:

```
LDI   R16, 0xFF
STS   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
LDI   R16, 0x15
STS   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
```

does not work properly. This is because, when the TCNT1L is loaded, the content of TEMP will be loaded into TCNT1H. But when the TCNT1L register is loaded, TEMP contains garbage (improper data), and this is not what we want.

When we read the low byte of 16-bit registers, the content of the high byte will be copied to the TEMP register. So, the following program reads the content of TCNT1:

```
LDS   R20,TCNT1L       ;R20 = TCNT1L, TEMP = TCNT1H
LDS   R21,TCNT1H       ;R21 = TEMP of Timer1
```

We must pay attention to the order of reading the high and low bytes of the 16-bit registers. Otherwise, the result is erroneous.

Notice that reading the OCR1A and OCR1B registers does not involve using the temporary register. You might be wondering why. It is because the AVR microcontroller does not update the content of OCR1A nor OCR1B unless we update them. For example, consider the following program:

```
LDS   R20,OCR1AL       ;R20 = OCR1L
LDS   R21,OCR1AH       ;R21 = OCR1H
```

The above code reads the low byte of the OCR1A and then the high byte, and between the two readings the content of the register remains unchanged. That is why the AVR does not employ the TEMP register while reading the OCR1A / OCR1B registers.

Examine Examples 9-29 through 9-31 to see how to generate time delay in different modes.

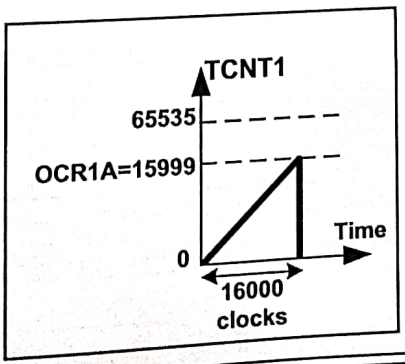
Example 9-30

Assuming XTAL = 16 MHz, write a program that toggles PB5 once per millisecond.

Solution:

XTAL = 16 MHz means that each clock takes 0.0625 µs. Now for 1 ms delay, we need 1 ms/0.0625 µs = 16000 clocks = 0x3E80 clocks. We initialize the timer so that after 16000 clocks the OCF1A flag is raised, and then we will toggle the PB5.

```
      LDI    R16,HIGH(RAMEND)
      OUT    SPH,R16
      LDI    R16,LOW(RAMEND)
      OUT    SPL,R16              ;initialize the stack
      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
;-----------------------Timer1 delay
DELAY_1ms:
      LDI    R20,0x00
      STS    TCNT1H,R20           ;TEMP = 0
      STS    TCNT1L,R20           ;TCNT1L = 0, TCNT1H = TEMP

      LDI    R20,HIGH(16000-1)
      STS    OCR1AH,R20           ;TEMP = 0x1F
      LDI    R20,LOW(16000-1)
      STS    OCR1AL,R20           ;OCR1AL = 0x3F, OCR1AH = TEMP

      LDI    R20,0x0
      STS    TCCR1A,R20           ;WGM11:10 = 00
      LDI    R20,0x09
      STS    TCCR1B,R20           ;WGM13:12 = 01, CTC mode, CS = 1
AGAIN:
      SBIS   TIFR1,OCF1A          ;if OCF1A is set skip next instruction
      RJMP   AGAIN
      LDI    R19,0
      STS    TCCR1B,R19           ;stop timer
      STS    TCCR1A,R19
      LDI    R20,1<<OCF1A
      OUT    TIFR1,R20            ;clear OCF1A flag
      RET
```



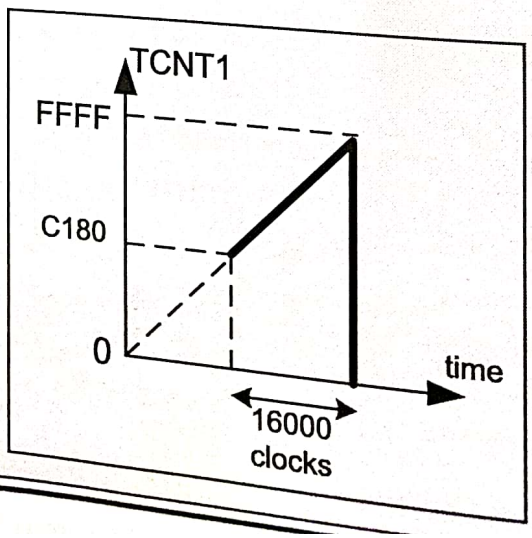
Example 9-31

Rewrite Example 9-30 using the TOV1 flag.

Solution:

To wait 1 ms we should load the TCNT1 register so that it rolls over after 16000 = 0x3E80 clocks. In Normal mode the top value is 0xFFFF = 65535. 65535 + 1 - 16000 = 49536 = 0xC180. Thus, we should load TCNT1 with 49536, or 0xC180 in hex, or we can simply use 65536 - 16000, as shown below:

```
      LDI   R16,HIGH(RAMEND)  ;initialize stack pointer
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16
      SBI   DDRB,5            ;PB5 as an output
      SBI   PORTB,5           ;PB5 = 1
BEGIN:RCALL DELAY_1ms
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY_1ms
      RJMP  BEGIN
;----------------------Timer1 delay
DELAY_1ms:
      LDI   R20,HIGH(65536-16000)   ;R20 = high byte of 49536
      STS   TCNT1H,R20              ;TEMP = 0xE0
      LDI   R20,LOW(65536-16000)    ;R20 = low byte of 49536
      STS   TCNT1L,R20        ;TCNT1L = 0xC1, TCNT1H = TEMP
      LDI   R20,0x0
      STS   TCCR1A,R20        ;WGM11:10 = 00
      LDI   R20,0x1
      STS   TCCR1B,R20        ;WGM13:12 = 00, Normal mode, CS = 1
AGAIN:
      SBIS  TIFR1,TOV1        ;if OCF1A is set skip next instruction
      RJMP  AGAIN
      LDI   R19,0
      STS   TCCR1B,R19        ;stop timer
      STS   TCCR1A,R19
      LDI   R20,1<<TOV1
      OUT   TIFR1,R20
      RET                     ;clear TOV1 flag
```



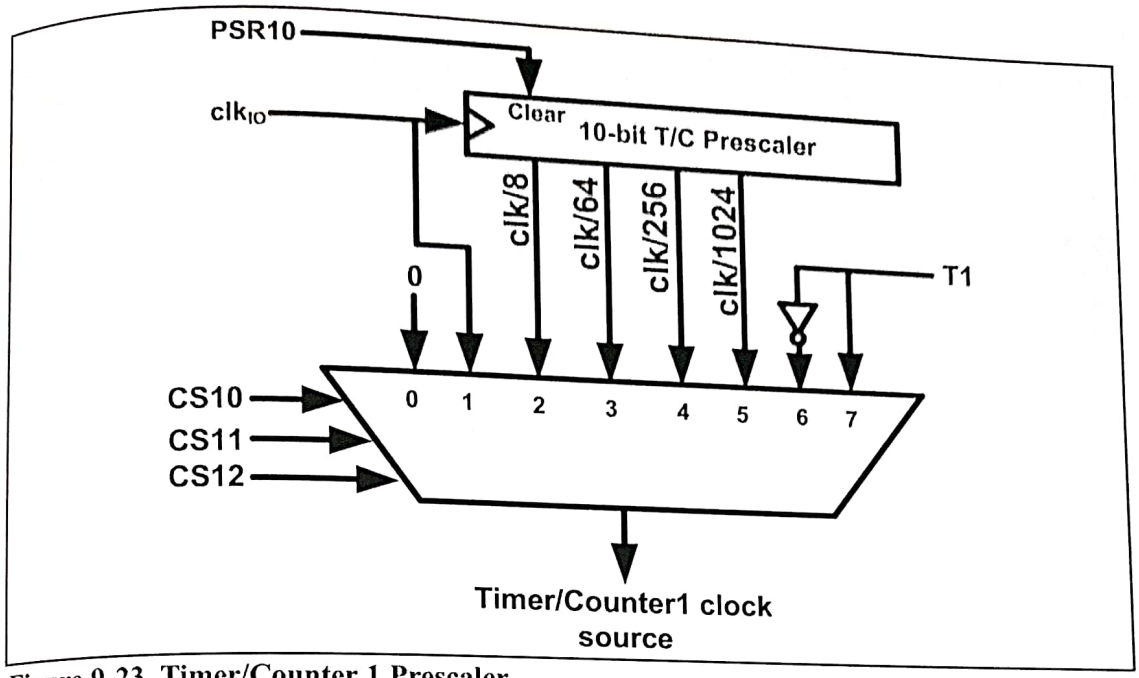


Figure 9-23. Timer/Counter 1 Prescaler

Generating a large time delay using prescaler

As we have seen in the examples so far, the size of the time delay depends on two factors: (a) the crystal frequency, and (b) the timer's 16-bit register. Both of these factors are beyond the control of the AVR programmer. We can use the prescaler option in the TCCR1B register to increase the delay by reducing the period. The prescaler option of TCCR1B allows us to divide the instruction clock by a factor of 8 to 1024, as was shown in Figure 9-16. The prescaler of Timer/Counter 1 is shown in Figure 9-23.

As we have seen so far, with no prescaler enabled, the crystal oscillator frequency is fed directly into Timer1. If we enable the prescaler bit in the TCCR1B register, then we can divide the instruction clock before it is fed into Timer1. The lower 3 bits of the TCCR1B register give the options of the number we can divide the clock by before it is fed to timer. As shown in Figure 9-23, this number can be 8, 64, 256, or 1024. Notice that the lowest number is 8, and the highest number is 1024. Examine Examples 9-32 and 9-33 to see how the prescaler options are programmed.

Review Questions

1. How many timers do we have in the ATmega328?
2. True or false. Timer0 is a 16-bit timer.
3. True or false. Timer1 is a 16-bit timer.
4. True or false. The TCCR0A register is a bit-addressable register.
5. In Normal mode, when the counter rolls over it goes from ____ to ____.
6. In CTC mode, the counter rolls over when the counter reaches____.
7. To get a 5-ms delay, what numbers should be loaded into TCNT1H and TCNT1L using Normal mode and the TOV1 flag? Assume that XTAL = 8 MHz.
8. To get a 20-µs delay, what number should be loaded into the TCNT0 register using Normal mode and the TOV0 flag? Assume that XTAL = 1 MHz.

Example 9-32

An LED is connected to PB5. Assuming XTAL = 16 MHz, write a program that toggles the LED once per second.

Solution:

As XTAL = 16 MHz, the different outputs of the prescaler are as follows:

Scaler	Timer Clock	Timer Period	Timer Value
None	16 MHz	1/16 MHz = 0.0625 µs	1 s/0.0625 µs = 16 M
8	16 MHz/8 = 2 MHz	1/2 MHz = 0.5 µs	1 s/0.5 µs = 2 M
64	16 MHz/64 = 250 kHz	1/250 kHz = 4 µs	1 s/4 µs = 250,000
256	16 MHz/256 = 62.5 kHz	1/62.5 kHz = 16 µs	1 s/16 µs = 62,500
1024	16 MHz/1024 = 15625 Hz	1/15625 Hz = 64 µs	1 s/64 µs = 15,625

From the above calculation we can use options 256 or 1024. We choose 1024 in this Example.

```
      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_1s
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY_1s
      RJMP BEGIN
;------------------- Timer1 delay
DELAY_1s:
      LDI   R20,HIGH (15625-1)
      STS   OCR1AH,R20        ;TEMP = $3D (since 15624 = $3D08)
      LDI   R20,LOW (15625-1)
      STS   OCR1AL,R20        ;OCR1AL = $08 (since 15624 = $3D08)
      LDI   R20,0
      STS   TCNT1H,R20        ;TEMP = 0x00
      STS   TCNT1L,R20        ;TCNT1L = 0x00, TCNT1H = TEMP
      LDI   R20,0x00
      STS   TCCR1A,R20        ;WGM11:10 = 00
      LDI   R20,0x5
      STS   TCCR1B,R20        ;WGM13:12=00, Normal mode, CS=CLK/1024
AGAIN:SBIS  TIFR1,OCF1A       ;if OCF1A is set skip next instruction
      RJMP  AGAIN
      LDI   R19,0
      STS   TCCR1B,R19
      STS   TCCR1A,R19        ;stop timer
      LDI   R20,1<<OCF1A
      OUT   TIFR1,R20
      RET                     ;clear OCF1A flag
```

Example 9-33

Assuming XTAL = 16 MHz, write a program to generate 1 Hz frequency on PB5.

Solution:

With 1 Hz we have T = 1 / F = 1 / 1 Hz = 1 second, half of which is high and half low. Thus we need a delay of 0.5 second duration.
Since XTAL = 16 MHz, the different outputs of the prescaler are as follows:

Scaler	Timer Clock	Timer Period	Timer Value
None	16 MHz	1/8 MHz = 0.0625 µs	0.5 s/0.0625 µs = 4 M
8	16 MHz/8 = 2 MHz	1/2 MHz = 0.5 µs	0.5 s/0.5 µs = 1 M
64	16 MHz/64 = 250 kHz	1/250 kHz = 4 µs	0.5 s/4 µs = 125,000
256	16 MHz/256 = 62.5 kHz	1/62.5 kHz = 16 µs	0.5 s/16 µs = 31,250
1024	16 MHz/1024 = 15625 Hz	1/15625 kHz = 64 µs	0.5 s/64 µs = 7812.5

From the above calculation we can use only options 256 or 1024. We should use option 256 since we cannot use a decimal point.

```
      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_0_5s
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY_0_5s
      RJMP BEGIN
;------------------- Timer1 delay
DELAY_0_5s:
      LDI   R20,HIGH (31250-1)
      STS   OCR1AH,R20        ;TEMP = $7A (since 31249 = $7A11)
      LDI   R20,LOW (31250-1)
      STS   OCR1AL,R20        ;OCR1AL = $11 (since 31249 = $7A11)
      LDI   R20,0x00
      STS   TCNT1H,R20        ;TEMP = 0x00
      STS   TCNT1L,R20        ;TCNT1L = 0x00, TCNT1H = TEMP
      LDI   R20,0x00
      STS   TCCR1A,R20        ;WGM11:10 = 00
      LDI   R20,0x4
      STS   TCCR1B,R20        ;WGM13:12 = 00, Normal mode, CS = CLK/256
AGAIN:SBIS  TIFR1,OCF1A       ;if OCF1A is set skip next instruction
      RJMP  AGAIN
      LDI   R19,0
      STS   TCCR1B,R19        ;stop timer
      STS   TCCR1A,R19
      LDI   R20,1<<OCF1A
      OUT   TIFR1,R20         ;clear OCF1A flag
      RET
```

In the previous section, we used the timers of the AVR to generate time delays. The AVR timer can also be used to count, detect, and measure the time of events happening outside the AVR. The use of the timer as an event counter is covered in this section. When the timer is used as a timer, the AVR's crystal is used as the source of the frequency. When it is used as a counter, however, it is a pulse outside the AVR that increments the TCNTx register. Notice that, in counter mode, registers such as TCCR, OCR0, and TCNT are the same as for the timer discussed in the previous section; they even have the same names.

CS00, CS01, and CS02 bits in the TCCR0 register

Recall from the previous section that the CS bits (clock selector) in the TCCR0 register decide the source of the clock for the timer. If CS02:00 is between 1 and 5, the timer gets pulses from the crystal oscillator. In contrast, when CS02:00 is 6 or 7, the timer is used as a counter and gets its pulses from a source outside the AVR chip. See Figure 9-24. Therefore, when CS02:00 is 6 or 7, the TCNT0 counter counts up as pulses are fed from pin T0 (Timer/Counter 0 External Clock input). In ATmega328, T0 is the alternative function of PORTD.4. In the case of

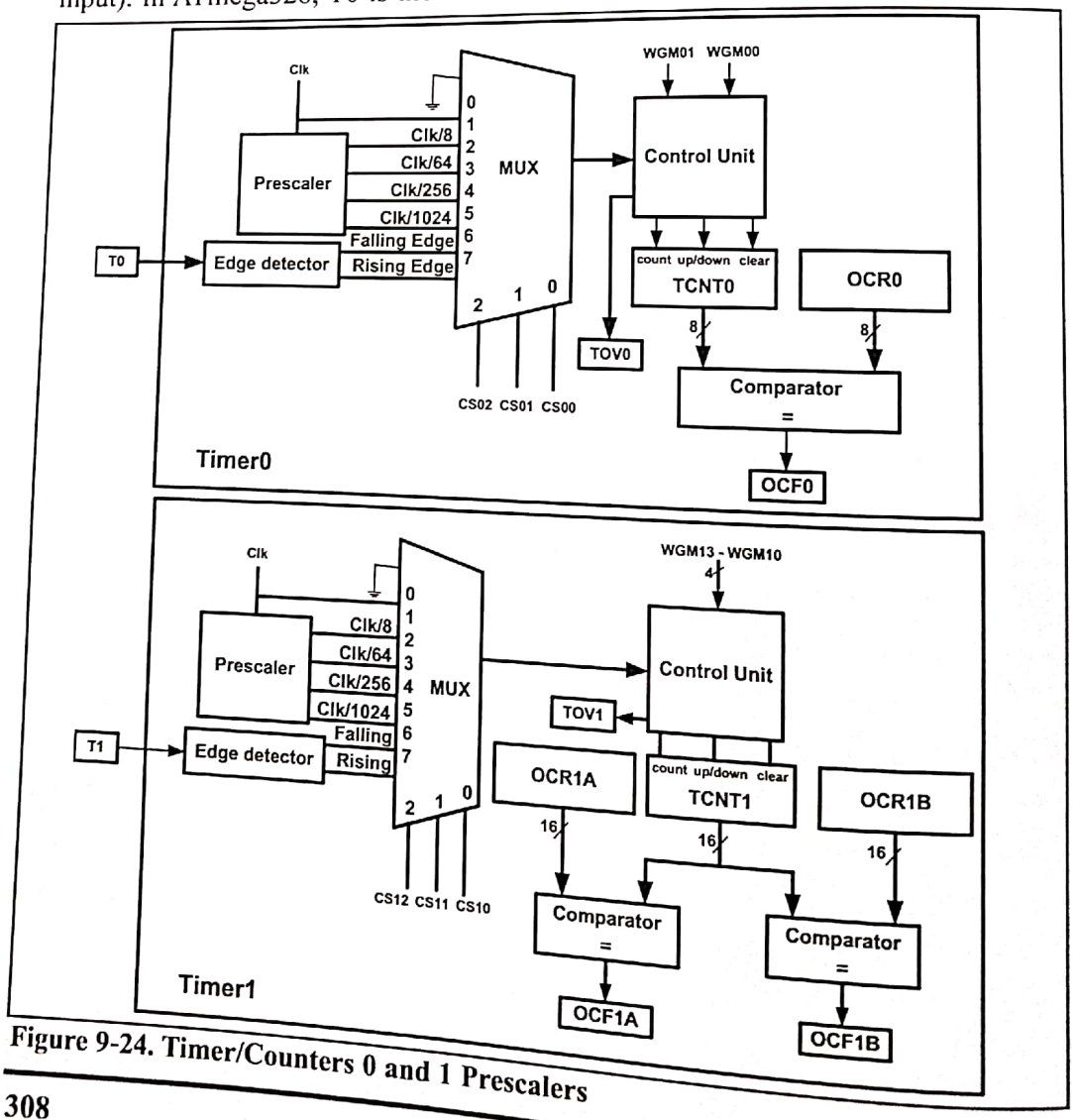


Figure 9-24. Timer/Counters 0 and 1 Prescalers

Example 9-34

Find the value for TCCR0 if we want to program Timer0 as a Normal mode counter. Use an external clock for the clock source and increment on the positive edge.

Solution:

TCCR0 = 0000 0111 Normal, external clock source, no prescaler

Timer0, when CS02:00 is 6 or 7, pin T0 provides the clock pulse and the counter counts up after each clock pulse coming from that pin. Similarly, for Timer1, when CS12:10 is 6 or 7, the clock pulse coming in from pin T1 (Timer/Counter 1 External Clock input) makes the TCNT1 counter count up. When CS12:10 is 6, the counter counts up on the negative (falling) edge. When CS12:10 is 7, the counter counts up on the positive (rising) edge. In ATmega328, T1 is the alternative function of PORTD.5. See Example 9-34.

In Example 9-35, we are using Timer0 as an event counter that counts up as clock pulses are fed into PD4. These clock pulses could represent the number of people passing through an entrance, or of wheel rotations, or any other event that can be converted to pulses.

In Example 9-35, the TCNT0 data was displayed in binary. In Example

Example 9-35

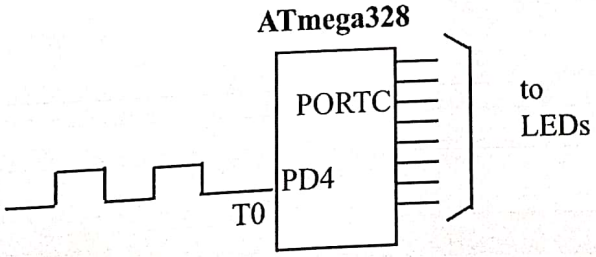
Assuming that a 1 Hz clock pulse is fed into pin T0 (PB0), write a program for Counter0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.

Solution:

```
      CBI   DDRD,4            ;make T0 (PD4) input
      LDI   R20,0xFF
      OUT   DDRC,R20          ;make PORTC output
      LDI   R20,0x00
      OUT   TCCR0A,R20
      LDI   R20,0x06
      OUT   TCCR0B,R20        ;counter, falling edge
AGAIN:
      IN    R20,TCNT0
      OUT   PORTC,R20         ;PORTC = TCNT0
      RJMP  AGAIN             ;keep doing it
```

Note: In ATmega328, PORTC has only 6 pins. To monitor bits 6 and 7 of TCNT0, you can shift right R20 six times and then send it out on another port.

PORTC is connected to LEDs and input T0 (PD4) to 1 Hz pulse.



9-36, the TCNT0 register is extended to a 16-bit counter using the TOV0 flag. See Examples 9-37 and 9-38.

As another example of the application of the counter, we can feed an external square wave of 60 Hz frequency into the timer. The program will generate the second, the minute, and the hour out of this input frequency and display the result on an LCD. This will be a nice looking digital clock, although not a very accurate one.

Before we finish this section, we need to state an important point. You might think monitoring the TOV and OCR flags is a waste of the microcontroller's time. You are right. There is a solution to this: the use of interrupts. Using interrupts enables us to do other things with the microcontroller. When a timer Interrupt flag such as TOV0 is raised it will inform us. This important and powerful feature of the AVR is discussed in Chapter 10.

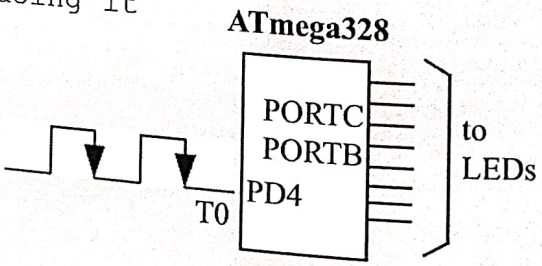
Example 9-36

Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTB.

Solution:

```
      LDI   R19,0             ;R19 = 0
      CBI   DDRD,4            ;make T0 (PD4) input
      LDI   R20,0xFF
      OUT   DDRC,R20          ;make PORTC output
      OUT   DDRB,R20          ;make PORTB output
      LDI   R20,0x00
      OUT   TCCR0A,R20
      LDI   R20,0x06
      OUT   TCCR0B,R20        ;counter, falling edge
AGAIN:
      IN    R20,TCNT0
      OUT   PORTC,R20         ;PORTC = TCNT0
      SBIS  TIFR0,TOV0
      RJMP  AGAIN             ;keep doing it
      LDI   R16,1<<TOV0       ;clear TOV0 flag
      OUT   TIFR0, R16
      INC   R19               ;R19 = R19 + 1
      OUT   PORTB,R19         ;PORTB = R19
      RJMP  AGAIN             ;keep doing it
```

PORTC and PORTB are connected to LEDs
and input T0 (PD4) to 1 Hz pulse.

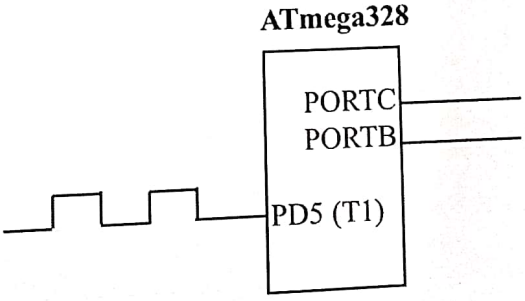


Example 9-37

Assuming that clock pulses are fed into pin T1 (PD5), write a program for Counter1 in Normal mode to count the pulses on falling edge and display the state of the TCNT1 count on PORTC and PORTB.

Solution:

```
      CBI   DDRD,5            ;make T1 (PD5) input
      LDI   R20,0xFF
      OUT   DDRC,R20          ;make PORTC output
      OUT   DDRB,R20          ;make PORTB output
      LDI   R20,0x00
      STS   TCCR1A,R20
      LDI   R20,0x06
      STS   TCCR1B,R20        ;counter, falling edge

AGAIN:
      LDS   R20,TCNT1L        ;R20 = TCNT1L, TEMP = TCNT1H
      OUT   PORTC,R20         ;PORTC = TCNT1L
      LDS   R20,TCNT1H        ;R20 = TEMP
      OUT   PORTB,R20         ;PORTB = TCNT1H
      SBIS  TIFR1,TOV1
      RJMP  AGAIN             ;keep doing it
      LDI   R16,1<<TOV1       ;clear TOV1 flag
      OUT   TIFR1, R16
      RJMP  AGAIN             ;keep doing it
```



Example 9-38

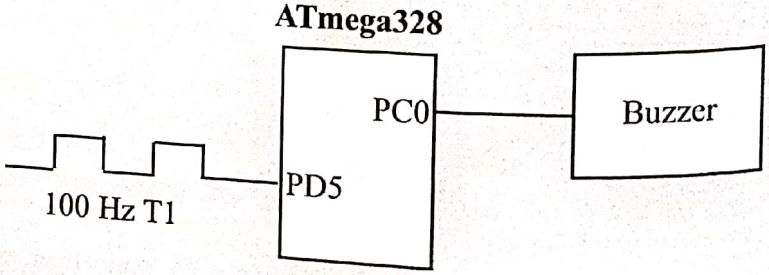
Assuming that clock pulses are fed into pin T1 (PD5) and a buzzer is connected to pin PORTC.0, write a program for Counter 1 in CTC mode to sound the buzzer every 100 pulses.

Solution:

To sound the buzzer every 100 pulses, we set the OCR1A value to 99 (63 in hex), and then the counter counts up until it reaches OCR1A. Upon compare match, we can sound the buzzer by toggling the PORTC.0 pin.

```
        CBI   DDRD,5          ;make T1 (PD5) input
        SBI   DDRC,0          ;PC0 as an output
        LDI   R16,0x1
        LDI   R17,0

        LDI   R20,0x0
        STS   TCCR1A,R20
        LDI   R20,0x0E
        STS   TCCR1B,R20      ;CTC, counter, falling edge
AGAIN:
        LDI   R20,0
        STS   OCR1AH,R20      ;TEMP = 0
        LDI   R20,99
        STS   OCR1AL,R20      ;ORC1L = R20, OCR1H = TEMP
L1:     SBIS  TIFR1,OCF1A
        RJMP  L1              ;keep doing it
        LDI   R20,1<<OCF1A    ;clear OCF1A flag
        OUT   TIFR1, R20

        EOR   R17,R16         ;toggle D0 of R17
        OUT   PORTC,R17       ;toggle PC0
        RJMP  AGAIN           ;keep doing it
```

PC0 is connected to a buzzer and input T1 to a pulse.



Review Questions

1. Which resource provides the clock pulses to AVR timers if CS02:00 = 6?
2. For Counter 0, which pin is used for the input clock?
3. To allow PD5 to be used as an input for the Timer1 clock, what must be done, and why?
4. Do we have a choice of counting up on the positive or negative edge of the clock?

SECTION 9.3: PROGRAMMING TIMERS IN C

In Chapter 7 we showed some examples of C programming for the AVR. In this section we show C programming for the AVR timers. As we saw in the examples in Chapter 7, the general-purpose registers of the AVR are under the control of the C compiler and are not accessed directly by C statements. All of the SFRs (Special Function Registers), however, are accessible directly using C statements. As an example of accessing the SFRs directly, we saw how to access ports PORTB–PORTD in Chapter 7.

In C we can access timer registers such as TCNT0, OCR0, and TCCR0 directly using their names. See Example 9-39.

Example 9-39

Write a C program to toggle all the bits of PORTB continuously with some delay. Use Timer0, Normal mode, and no prescaler options to generate the delay.

Solution:

```
#include "avr/io.h"
void T0Delay ( );
int main ( )
{
      DDRB = 0xFF;        //PORTB output port

      while (1)
      {
            PORTB = 0x55;      //repeat forever
            T0Delay ( );       //delay size unknown
            PORTB = 0xAA;      //repeat forever
            T0Delay ( );
      }
}

void T0Delay ( )
{
      TCNT0 = 0x20;                    //load TCNT0
      TCCR0A = 0;
      TCCR0B = 0x01;                   //Timer0, Normal mode, no prescaler
      while ((TIFR0&(1<<TOV0))==0); //wait for Timer0 to roll over
      TCCR0B = 0;
      TIFR0 = 1<<TOV0;                 //clear TOV0
}
```

Calculating delay length using timers

As we saw in the last two sections, the delay length depends primorily on two factors: (a) the crystal frequency, and (b) the prescaler factor. A third factor in the delay size is the C compiler because various C compilers generate different hex code sizes, and the amount of overhead due to the instructions varies by compiler. Study Examples 9-40 through 9-42 and verify them using an oscilloscope.

Example 9-40

Write a C program to toggle only the PORTB.4 bit continuously every 35 µs. Use Timer0, Normal mode, and 1:8 prescaler to create the delay. Assume XTAL = 16 MHz.

Solution:

XTAL = 16MHz ➔ $T_{machine\ cycle}$ = 1/16 MHz
Prescaler = 1:8 ➔ T_{clock} = 8 × 1/16 MHz = 0.5 µs
35 µs/0.5 µs = 70 clocks ➔ 1 + 0xFF – 70 = 0x100 – 0x46 = 0xBA = **186**

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

void T0Delay ( );

int main ( )
{
      DDRB = 0xFF;      //PORTB output port

      while (1)
      {
            T0Delay ( );            //Timer0, Normal mode
            PORTB = PORTB ^ 0x10;   //toggle PORTB.4
      }
}

void T0Delay ( )
{
      TCNT0 = 186;      //load TCNT0
      TCCR0A = 0;
      TCCR0B = 0x02;    //Timer0, Normal mode, 1:8 prescaler
      while ((TIFR0&(1<<TOV0))==0); //wait for TOV0 to roll over

      TCCR0B = 0;               //turn off Timer0
      TIFR0 = (1<<TOV0);        //clear TOV0
}
```

Example 9-41

Write a C program to toggle only the PORTB.4 bit continuously every 1 ms. Use Timer1, Normal mode, and no prescaler to create the delay. Assume XTAL = 16 MHz.

Solution:

XTAL = 16 MHz → $T_{machine\ cycle}$ = 1/16 MHz = 0.0625 µs
Prescaler = 1:1 → T_{clock} = 0.0625 µs
1 ms/0.0625 µs = 16,000 clocks = 0x3E80 clocks

1 + 0xFFFF – 0x3E80 = 0xC180

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

void T1Delay ( );

int main ( )
{
      DDRB = 0xFF;              //PORTB output port

      while (1)
      {
            PORTB ^= (1<<4);    //toggle PB4
            T1Delay ( );        //delay size unknown
      }
}


void T1Delay ( )
{
      TCNT1H = 0xC1;     //TEMP = 0xC1
      TCNT1L = 0x80;

      TCCR1A = 0x00;     //Normal mode
      TCCR1B = 0x01;     //Normal mode, no prescaler

      while ((TIFR1&(0x1<<TOV1))==0);     //wait for TOV1 to roll over

      TCCR1B = 0;
      TIFR1 = 0x1<<TOV1;       //clear TOV1
}
```

Example 9-42 (C version of Example 9-32)

Write a C program to toggle only the PORTB.4 bit continuously every second. Use Timer1, Normal mode, and 1:1024 prescaler to create the delay. Assume XTAL = 16 MHz.

Solution:

XTAL = 16 MHz → $T_{machine\ cycle}$ = 1/16 MHz = 0.0625 µs = T_{clock}

Prescaler = 1:1024 → T_{clock} = 1024 × 0.0625 µs = 64 µs

1 s/64 µs = 15,625 clocks = 0x3D09 clocks → 1 + 0xFFFF – 0x3D09 = **0xC2F7**

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

void T1Delay ( );

int main ( )
{
      DDRB = 0xFF;          //PORTB output port

      while (1)
      {
            PORTB = PORTB ^ (1<<PB4); //toggle PB4
            T1Delay ( );
      }
}

void T1Delay ( )
{
      TCNT1H = 0xC2;     //TEMP = 0x85
      TCNT1L = 0xF7;

      TCCR1A = 0x00;     //Normal mode
      TCCR1B = 0x05;     //Normal mode, 1:256 prescaler

      while ((TIFR1&(0x1<<TOV1))==0);     //wait for TF0 to roll over

      TCCR1B = 0;
      TIFR1 = 0x1<<TOV1;                  //clear TOV1
}
```

Note: In the above program, we can also load the TCNT1 by writing the following:
TCNT1 = 0xC2F7;

C programming of Timers 0 and 1 as counters

In Section 9.2 we showed how to use Timers 0 and 1 as event counters. Timers can be used as counters if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source. By feeding pulses to the T0 (PD4) and T1 (PD5) pins, we use Timer0 and Timer1 as Counter 0 and Counter 1, respectively. Study Examples 9-43 and 9-44 to see how Timers 0 and 1 are programmed as counters using C language.

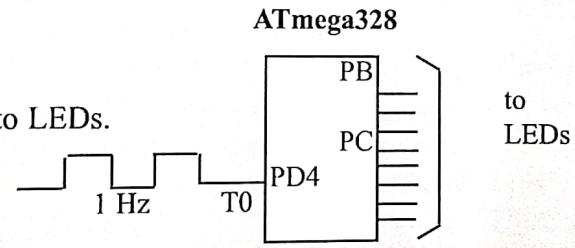
Example 9-43 (C version of Example 9-36)

Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTD.

Solution:

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

int main ( )
{
      PORTD = 1<<4;              //pull up PD4(T0)
      DDRC = 0xFF;               //PORTC as output
      DDRB = 0xFF;               //PORTB as output

      TCCR0A = 0x06;             //output clock source
      TCCR0B = 0x00;
      TCNT0 = 0x00;

      while (1)
      {
            do
            {
                  PORTC = TCNT0;
            }while((TIFR0&(0x1<<TOV0))==0);//wait for TOV0 to set

            TIFR = 1<<TOV0;          //clear TOV0
            PORTB ++;                //increment PORTB
      }
}
```

PORTC and PORTB are connected to LEDs.
T0 (PD4) is connected to a 1-Hz external clock.



SUMMARY

The AVR has one to six timers/counters depending on the family member. When used as timers, they can generate time delays. When used as counters, they can serve as event counters.

Some of the AVR timers are 8-bit and some are 16-bit. The 8-bit timers are accessed as TCNTn (like TCNT0 for Timer0), whereas 16-bit timers are accessed as two 8-bit registers (TCNTnH, TCNTnL).

Each timer has its own TCCR (Timer/Counter Control Register) register, allowing us to choose various operational modes. Among the modes are the prescaler and timer/counter options. When the timer is used as a timer, the AVR crystal is used as the source of the frequency; however, when it is used as a counter, it is a pulse outside of the AVR that increments the TCNT register.

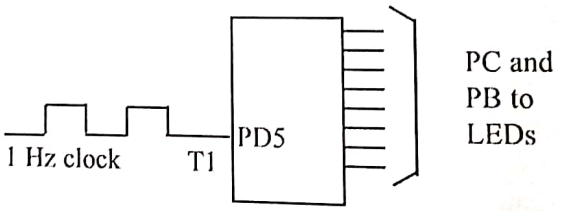
This chapter showed how to program the timers/counters to generate delays and count events using Normal and CTC modes.

Example 9-44 (C version of Example 9-37)

Assume that a 1-Hz external clock is being fed into pin T1 (PD5). Write a C program for Counter1 in rising edge mode to count the pulses and display the TCNT1H and TCNT1L registers on PORTB and PORTC, respectively.

Solution:

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

int main ( )
{
      PORTD = 1<<5;              //activate pull-up of PB0
      DDRC = 0xFF;               //PORTC as output
      DDRB = 0xFF;               //PORTB as output

      TCCR1A = 0x00;             //output clock source
      TCCR1B = 0x06;             //output clock source

      TCNT1 = 0x00;              //set count to 0

      while (1)                  //repeat forever
      {
            do
            {
                  PORTC = TCNT1L;
                  PORTB = TCNT1H;        //place value on pins
            }while((TIFR1&(0x1<<TOV1))==0);//wait for TOV1

            TIFR1 = 0x1<<TOV1;           //clear TOV1
      }
}
```



PROBLEMS

SECTION 9.1: PROGRAMMING TIMERS 0, 1, AND 2

1. How many timers are in the ATmega328?
2. Timer0 of the ATmega328 is ____-bit, accessed as _____.
3. Timer1 of the ATmega328 is ____-bit, accessed as _____ and _____.
4. Timer0 supports the highest prescaler value of _____.
5. Timer1 supports the highest prescaler value of _____.
6. The TCCR0 register is a(n) ___-bit register.
7. What is the job of the TCCR0A register?

8. True or false. TCCR0B is a bit-addressable register.
9. True or false. TIFR0 is a bit-addressable register.
10. Find the TCCR0A and TCCR0B values for Normal mode, no prescaler, with the clock coming from the AVR's crystal.
11. Find the frequency and period used by the timer if the crystal attached to the AVR has the following values:
    (a) XTAL = 8 MHz
    (b) XTAL = 16 MHz
    (c) XTAL = 1 MHz
    (d) XTAL = 10 MHz
12. Which registers hold the TOV0 (timer overflow flag) and TOV1 bits?
13. Indicate the rollover value (in hex and decimal) of the timer for each of the following cases:
    (a) Timer0 and Normal mode
    (b) Timer1 and Normal mode
14. Indicate when the TOVx flag is raised for each of the following cases:
    (a) Timer0 and Normal mode
    (b) Timer1 and Normal mode
15. True or false. Both Timer0 and Timer1 have their own timer overflow flags.
16. True or false. Both Timer0 and Timer1 have their own timer compare match flags.
17. Assume that XTAL = 8 MHz. Find the TCNT0 value needed to generate a time delay of 20 µs. Use Normal mode, no prescaler mode.
18. Assume that XTAL = 8 MHz. Find the TCNT0 value needed to generate a time delay of 5 ms. Use Normal mode, and the largest prescaler possible.
19. Assume that XTAL = 1 MHz. Find the TCNT1H,TCNT1L value needed to generate a time delay of 2.5 ms. Use Normal mode, no prescaler mode.
20. Assume that XTAL = 1 MHz. Find the OCR0 value needed to generate a time delay of 0.2 ms. Use CTC mode, no prescaler mode.
21. Assume that XTAL = 1 MHz. Find the OCR1H,OCR1L value needed to generate a time delay of 2 ms. Use CTC mode, and no prescaler mode.
22. Assuming that XTAL = 8 MHz, and we are generating a square wave on pin PB7, find the lowest square wave frequency that we can generate using Timer1 in Normal mode.
23. Assuming that XTAL = 8 MHz, and we are generating a square wave on pin PB2, find the highest square wave frequency that we can generate using Timer1 in Normal mode.
24. Repeat Problems 22 and 23 for Timer0.
25. Assuming that TCNT0 = $F1, indicate which states Timer0 goes through until TOV0 is raised. How many states is that?
26. Program Timer0 to generate a square wave of 2 kHz. Assume that XTAL = 16MHz.
27. Program Timer1 to generate a square wave of 6 kHz. Assume that XTAL = 16MHz.
28. State the differences between Timer0 and Timer1.
29. Find the value (in hex) loaded into R16 in each of the following:
    (a) LDI R16,-12
    (b) LDI R16,-22
    (c) LDI R16,-34
    (d) LDI R16,-92
    (e) LDI R16,-120
    (f) LDI R16,-104

SECTION 9.2: COUNTER PROGRAMMING

30. To use a timer as an event counter we must set the _____ bits in the TCCR register to _____.
31. Can we use both Timer0 and Timer1 as event counters?
32. For Counter 0, which pin is used for the input clock?
33. For Counter 1, which pin is used for the input clock?
34. Program Timer1 to be an event counter. Use Normal mode, and display the binary count on PORTC and PORTD continuously. Set the initial count to 20,000.
35. Program Timer0 to be an event counter. Use Normal mode and display the binary count on PORTC continuously. Set the initial count to 20.

SECTION 9.3: PROGRAMMING TIMERS IN C

36. Program Timer0 in C to generate a square wave of 1 kHz. Assume that XTAL = 1 MHz.
37. Program Timer1 in C to generate a square wave of 1 kHz. Assume that XTAL = 8 MHz.
38. Program Timer0 in C to generate a square wave of 3 kHz. Assume that XTAL = 16 MHz.
39. Program Timer1 in C to generate a square wave of 3 kHz. Assume that XTAL = 10 MHz.
40. Program Timer1 in C to be an event counter. Use Normal mode and display the binary count on PORTB and PORTD continuously. Set the initial count to 20,000.
41. Program Timer0 in C to be an event counter. Use Normal mode and display the binary count on PORTD continuously. Set the initial count to 20.

ANSWERS TO REVIEW QUESTIONS

SECTION 9.1: PROGRAMMING TIMERS 0, 1, AND 2

1. 3
2. False
3. True
4. False
5. Max ($FFFF for 16-bit timers and $FF for 8-bit timers), 0000
6. OCR1A
7. $10000 – (5000 × 8) = 25536 = 63C0, TCNT1H = 0x64 and TCNT1L = 0xC0
8. XTAL = 1 MHz ➔ $T_{machine\ cycle}$ = 1/1 M = 1 µs ➔ 20 µs / 1 µs = 20
   −20 = $100 – 20 = 256 – 20 = 236 = 0xEC

SECTION 9.2: COUNTER PROGRAMMING

1. External clock (falling edge)
2. PORTB.0 (T0)
3. DDRB.0 must be cleared to turn the output circuit off and use the pin as input.
4. Yes