Timer 1 is also conceptually very different from Timer 0. Timer 0 is usually stopped, started, reset, and so on in its normal use. Timer 1, on the other hand, is usually left running. This creates some considerable differences in its use. These differences will be discussed in detail in the sections that follow, covering the special uses of Timer 1.

Timer 1 Prescaler and Selector

In spite of its many special features, Timer 1 is still a binary up-counter whose count speed or timing intervals depend on the clock signal applied to its input, just as Timer 0 was. As with all peripherals in the microcontroller, Timer 1 is controlled through a control register.

Timer/counter control register 1, TCCR1 (the ATMega16 timer control register for Timer 1), is actually composed of two registers, TCCR1A and TCCR1B. TCCR1A controls the compare modes and the pulse width modulation modes of Timer 1. These will be discussed later in the section. TCCR1B controls the prescaler and input multiplexer for Timer 1, as well as the input capture modes. Figure 2–20 shows the bit definition for the bits of TCCR1B.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ICNC1	ICES1		WGM 13	WGM 12	CS12	CS11	CS10

Bit	***Function***
ICNC1	Input Capture Noise Canceller (1 = enabled)
ICES1	Input Capture Edge Select (1 = rising edge, 0 = falling edge)
WGM 1x	Output waveform control. See TCCR1A
CS12	Counter Input Select Bits Exactly the same definition as for Timer 0
CS11	
CS10	

Figure 2–20 *TCCR1B Bit Definitions*

TCCR1B *counter select bits* control the input to Timer 1 in exactly the same manner as the counter select bits of Timer 0. In fact, the three select bits provide clock signals that are absolutely identical to those of Timer 0.

Timer 1 Input Capture Mode

Measuring a time period with Timer 0 involves starting the timer at the beginning of the event, stopping the timer at the end of the event, and finally reading the time of the event from the timer counter register. The same job with Timer 1 is handled differently because Timer 1 is always running. To measure an event, the time on Timer 1 is captured or held at

the beginning of the event, the time is also captured at the end of the event, and the two are subtracted to find the time that it took for the event to occur. You would do much the same if you are trying to find out how long it took you walk from your history class to the bookstore. You would note the time when you left history class, again note the time when you arrived at the bookstore, and subtract the two times to determine how long it took you get to the bookstore. In Timer 1, these tasks are managed by the *input capture register* (ICR1).

ICR1 is a 16-bit register (made up of ICR1H and ICR1L) that will capture the actual reading of Timer 1 when the microcontroller receives a certain signal. The signal that causes a capture to occur can be either a rising or a falling edge applied to the *input capture pin,* ICP, of the microcontroller. As shown in Figure 2–20, the choice of a rising or falling edge trigger for the capture is controlled by the *input capture edge select bit,* ICES1. Setting ICES1 will allow ICR1 to capture the Timer 1 time on a rising edge, and clearing it will allow ICR1 to capture the time on a falling edge.

As is probably obvious by now, since there is only one capture register available to Timer 1, the captured contents must be read out as soon as they are captured to prevent the next capture from overwriting and destroying the previous reading. In order to accomplish this, an interrupt is provided that occurs whenever new data is captured into ICR1. Each time the capture interrupt occurs, the program must determine whether the interrupt signals the beginning or the ending of an event that is being timed so that it can treat the data in ICR1 appropriately.

Timer 1 also provides an input noise canceller feature to prevent miscellaneous unwanted spikes in the signal applied to the ICP from causing a capture to occur at the wrong time. When the noise canceller feature is active, the ICP must remain at the active level (high for a rising edge, or low for a falling edge) for four successive samples before the microcontroller will treat the trigger as legitimate and capture the data. This prevents a noise spike from triggering the capture register. Setting the *input capture noise canceller bit,* ICNC1, in TCCR1B enables the noise canceller feature (refer to Figure 2–20).

The hardware and software shown in Figures 2–21 and 2–22, respectively, demonstrate the use of the input capture register. The goal of this hardware and software is to measure the period of a square wave applied to the ICP of the microcontroller and to output the result, in milliseconds, on port C.

The software in Figure 2–22 has several features worth noting. A **#define** statement is used to connect a meaningful name with the output port.

The ISR for the Timer 1 overflow does nothing more than increment an overflow counter when the overflow occurs during a period measurement. The number of overflows is used in the calculation for the period.

The *Input Capture Event* ISR occurs every time the input waveform is a rising edge. At this instant, the ISR reads the input capture register to get the time that was captured when the

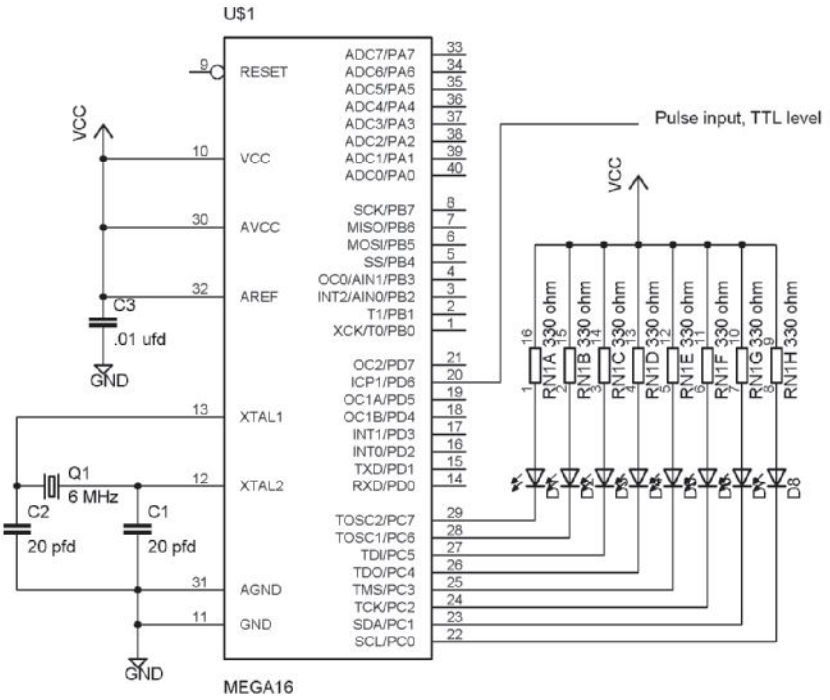

Figure 2–21 *Period Measurement Hardware*

```
#include <mega16.h>

/*define Port C as output for pulse width*/
#define period_out PORTC

unsigned char ov_counter;   /*counter for timer 1 overflow*/
unsigned int starting_edge, ending_edge;  /*storage for times*/
unsigned int clocks;   /*storage for actual clock counts in the pulse*/

/*Timer 1 overflow ISR*/
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
   ++ov_counter;   /*increment counter when overflow occurs*/
}

/*Timer 1 input capture ISR*/
interrupt [TIM1_CAPT] void timer1_capt_isr(void)
{
```

Figure 2–22 *Period Measurement Software (Continues)*

```
   /*combine the two 8-bit capture registers into the 16-bit count*/
   ending_edge = 256*ICR1H + ICR1L; /*get end time for period*/
   clocks = (unsigned long)ending_edge
            + ((unsigned long)ov_counter * 65536)
            - (unsigned long)starting_edge;
   period_out = ~(clocks / 750);  /*output milliseconds to Port C*/
   /*clear overflow counter for this measurement*/
   ov_counter = 0;
   /*save end time to use as starting edge*/
   starting_edge = ending_edge;
}

void main(void)
{
   DDRC=0xFF;  /*set Port C for output*/
   TCCR1A = 0;  /*disable all waveform functions*/
   TCCR1B = 0xC2;/*Timer 1 input to clock/8, enable input capture*/
   TIMSK = 0x24; /*unmask timer 1 overflow and capture interrupts*/

   #asm("sei") /*enable global interrupt bit*/

   while (1)
   {
      ; /*do nothing here*/
   }
}
```

Figure 2–22 *Period Measurement Software (Continued)*

rising edge occurred. This number is the actual count of clock ticks that was present when the input capture event occurred.

Because ICR1 is composed of two separate 8-bit registers (ICR1H and ICR1L), they must be combined into an integer-sized number to use in the period calculation. This is accomplished by:

```
ending_edge = 256*ICR1H + ICR1L; /*get end time for period*/
```

Multiplying the 8-bit result is exactly the same as shifting the number form ICR1H eight bits to the left, making the more significant byte of the integer count. The result of this multiplication is added to the value from ICR1L to form the complete integer result. This number is the timer count at the *end* of the period being timed. The count from the start of the period is the same as the ending count from the previous period.

The total number of clock ticks that occurred during the measurement period may then be calculated as follows:

```
clocks = (unsigned long) ending_edge
         + ((unsigned long) ov_counter * 65536)
         - (unsigned long) starting_edge;
```

The second line of the equation accounts for any overflows that occurred during the measurement period.

The timer is being clocked by Fclk/8 (6 MHz / 8 = 750 kHz), which means that for every millisecond that has elapsed, 750 counts have occurred. Finally, dividing the total number of counts by 750 produces the actual number of milliseconds in the measurement period.

Timer 1 Output Compare Mode

The output compare mode is used by the microcontroller to produce output signals. The outputs may be square or asymmetrical waves, and they may be varying in frequency or symmetry. Output compare mode, for instance, would be appropriate if you attempt to program a microcontroller to play your school's fight song. In this case, the output compare mode would be used to generate the musical notes that make up the song.

Output compare mode is sort of the antithesis of input capture mode. In input capture mode, an external signal causes the current time in a timer to be captured or held in the input capture register. In output compare mode, the program loads an *output compare register*. The value in the output compare register is compared to the value in the timer/counter register, and an interrupt occurs when the two values match. This interrupt acts as an alarm clock to cause a processor to execute a function relative to the signal it is producing, exactly when it is needed.

In addition to generating an interrupt, output compare mode can automatically set, clear, or toggle a specific output port pin. For Timer 1, the output compare modes are controlled by *timer counter control register 1A*, TCCR1A. Figure 2–23 shows the bit definitions for TCCR1A that relate to output compare operation.

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

COM1A0 & COM1A1 Control the compare mode function for compare resister A
COM1B0 & COM1B1 Control the compare mode function for compare resister B

Control Bit Definitions:

COM1x1	*COM1x0*	*Function. ('x' is 'A' or 'B' as appropriate)*
0	0	No Output
0	1	Compare match toggles the OC1x line
1	0	Compare match clears the OC1x line to 0
1	1	Compare match sets the OC1x line to 1

Figure 2–23 *TCCR1A Bit Definitions for Output Compare Mode*