Timer/Counter

- Timer/ Counter definition
- Prescaler and interrupt in timer/counter.
- Timer/Counter in PIC16F887.
- Timer 0 as timer.
- Timer 0 as counter.
- Timer 1 as timer and an example of using Timer 1.

Prepared By-

Mohammed Abdul Kader

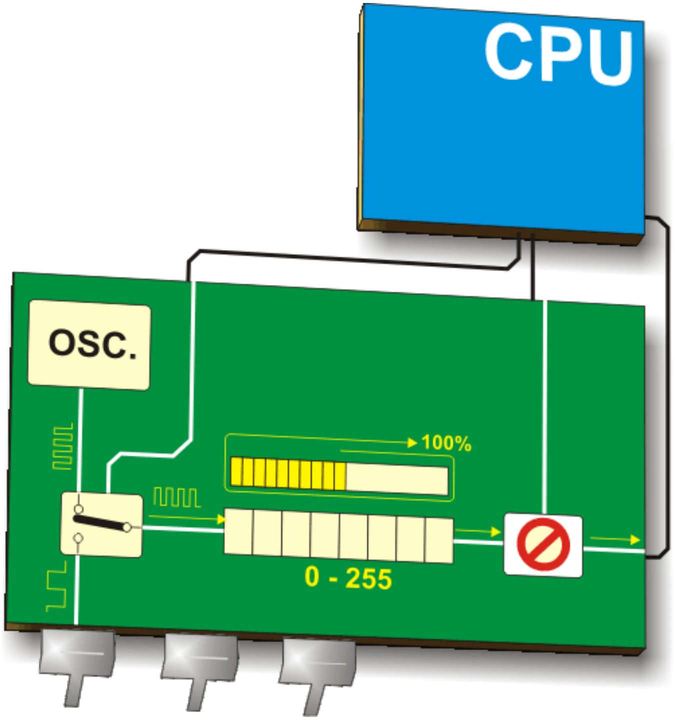
Assistant Professor, EEE, IIUC

Timers/Counters

Most programs use these miniature electronic ‘stopwatches’. These are commonly 8- or 16-bit SFRs the contents of which is automatically incremented by each coming pulse. Once a register is completely loaded, an interrupt may be generated!

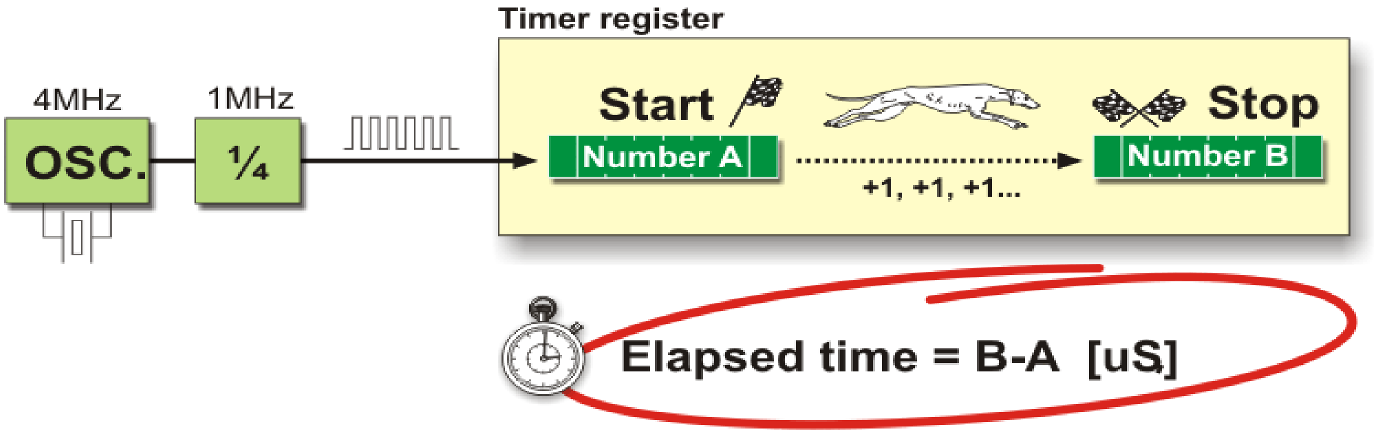
Timers: If it uses an internal quartz oscillator for its operation then it can be used to measure time between two events (if the register value is T1 at the moment measurement starts, and T2 at the moment it terminates, then the elapsed time is equal to the result of subtraction T2-T1).

Counters: If the timer receives pulses form the microcontroller input pin, then it turns into a counter. Obviously, it is the same electronic circuit able to operate in two different modes. The only difference is that in this case pulses to be counted come over the microcontroller input pin and their duration (width) is mostly undefined. This is why they cannot be used for time measurement, but for other purposes such as counting products on an assembly line, number of axis rotation, passengers etc. (depending on sensor in use).



HOW DOES THE TIMER OPERATE?

In practice, pulses generated by the quartz oscillator are once per each machine cycle, directly or via a prescaler, brought to the circuit which increments the number stored in the timer register. If one instruction (one machine cycle) lasts for four quartz oscillator periods then this number will be incremented a million times per second (each microsecond) by embedding quartz with the frequency of 4MHz.

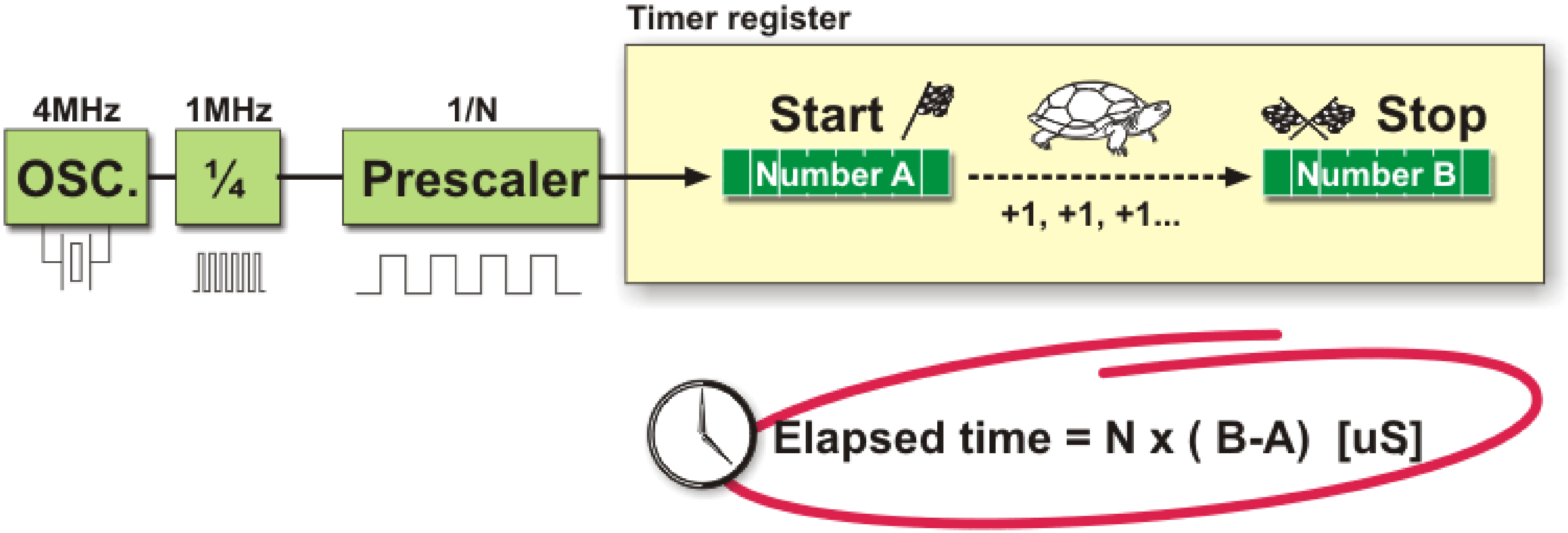


It is easy to measure short time intervals, **up to 256 microseconds**, in the way described above because it is the largest number that one register can store.

This restriction may be easily overcome in several ways such as by using a slower oscillator, registers with more bits, prescaler or interrupts. The first two solutions have some weaknesses so it is more recommended to use prescalers or interrupts.

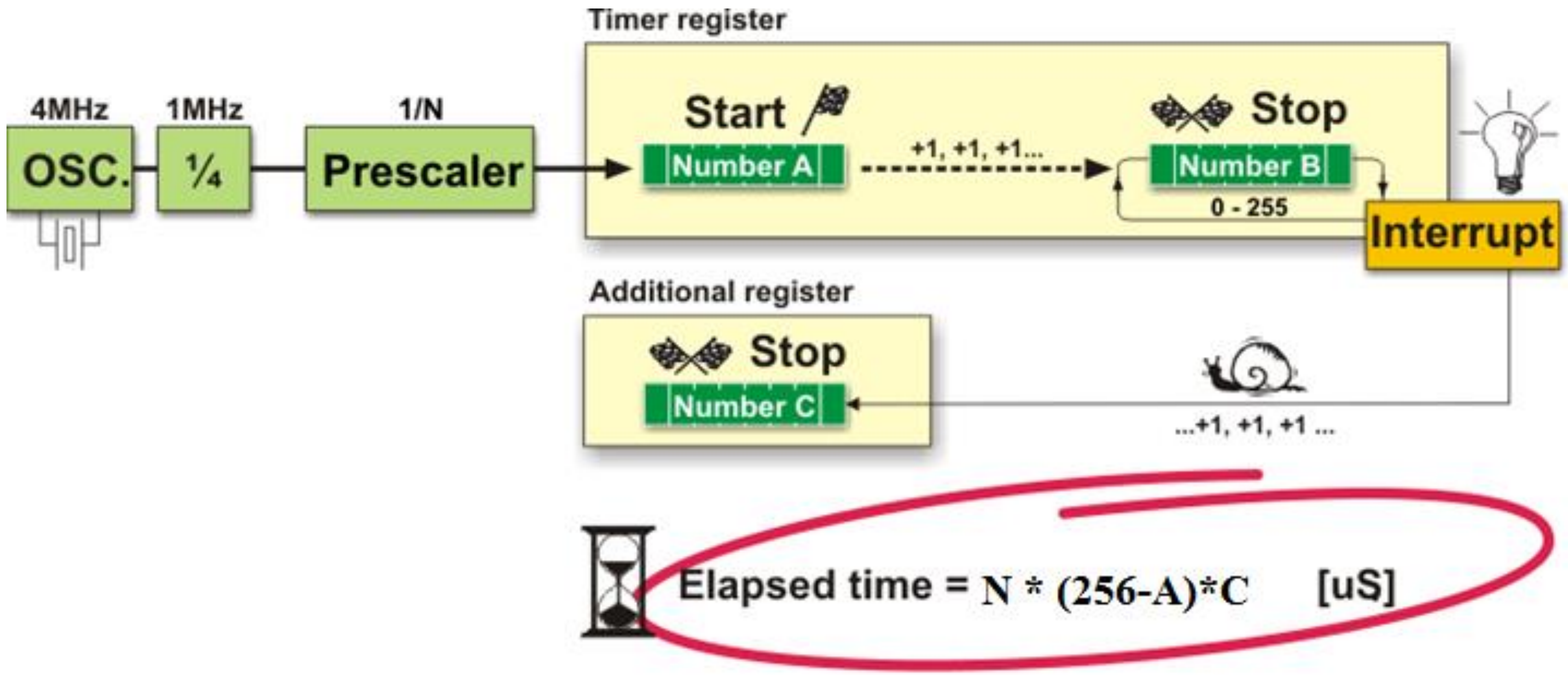
USING A PRESCALER IN TIMER OPERATION

A prescaler is an electronic device used to reduce frequency by a predetermined factor. In order to generate one pulse on its output, it is necessary to bring 1, 2 , 4 or more pulses on its input. Most microcontrollers have one or more prescalers built in and their division rate may be changed from within the program. The prescaler is used when it is necessary to measure longer periods of time. If one prescaler is shared by timer and watchdog timer, it cannot be used by both of them simultaneously



USING INTERRUPT IN TIMER OPERATION

If the timer register consists of 8 bits, the largest number it can store is 255. As for 16-bit registers it is the number 65.535. If this number is exceeded, the timer will be automatically reset and counting will start at zero again. This condition is called an overflow. If enabled from within the program, the overflow can cause an interrupt, which gives completely new possibilities. For example, the state of registers used for counting seconds, minutes or days can be changed in an interrupt routine. The whole process (except for interrupt routine) is automatically performed behind the scenes, which enables the main circuits of the microcontroller to operate normally.



Timers/counters in PIC16F887

Both PIC16f887 and 877A have three timers:-

- ✓ Timer 0,
- ✓ Timer 1, and
- ✓ Timer 2.

Timer 0 is 8-bit and can also be used as counter;
Timer 1 is 16-bit timer as well as a counter, whereas Timer 2 is 8-bit timer and can be used as the PWM time base for the PWM mode of the CCP module(s).

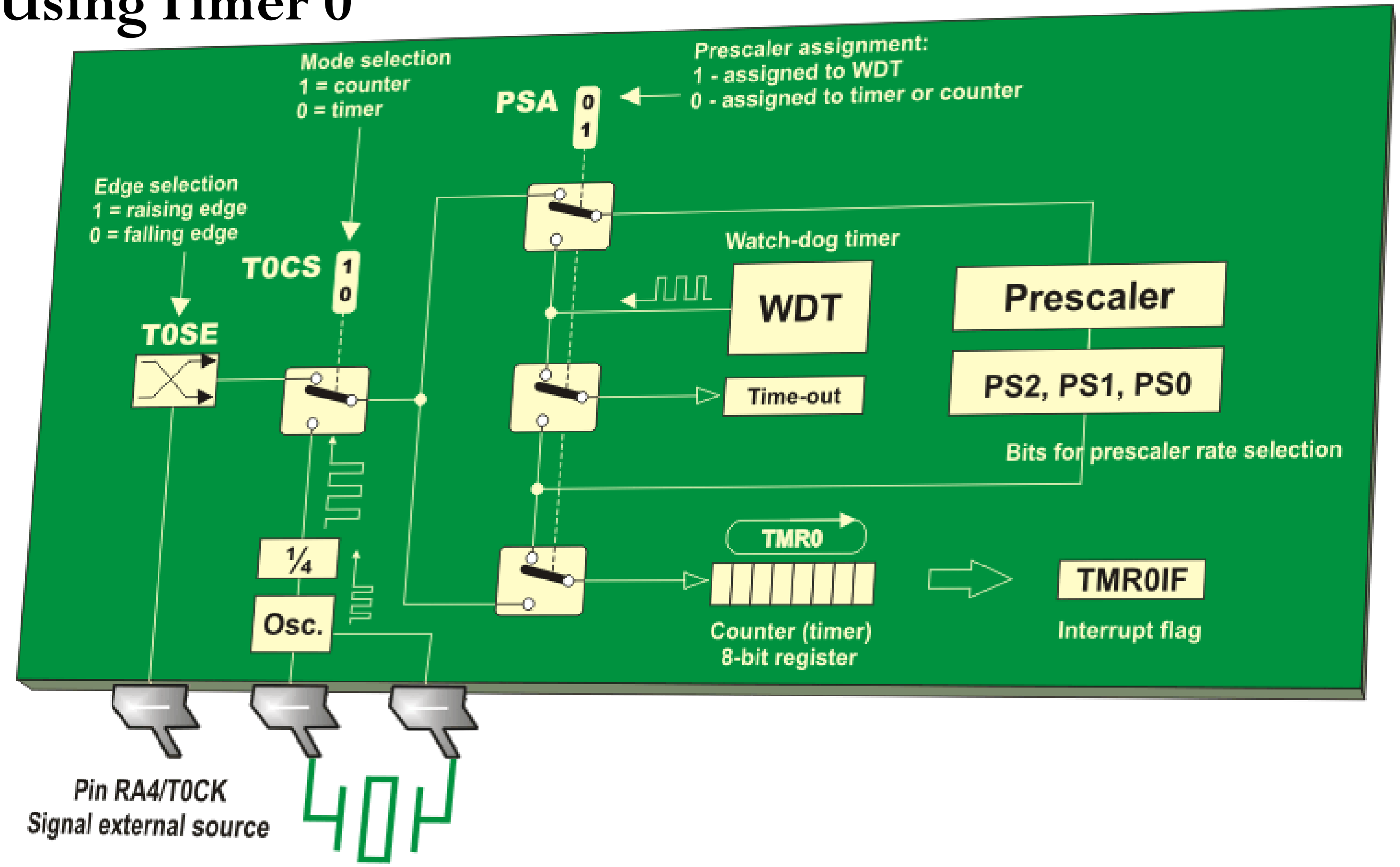
Timer 0

The timer TMR0 has a wide range of application in practice. Very few programs don't use it in some way. It is very convenient and easy to use for writing programs or subroutines for generating pulses of arbitrary duration, time measurement or counting external pulses (events) with almost no limitations.

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter.
- Readable and writable.
- 8-bit software programmable prescaler.
- Internal or external clock select.
- Interrupt on overflow from FFh to 00h.
- Edge select (rising or falling) for external clock.

Configuring/ Using Timer 0



In order to use TMR0 properly, it is necessary:

Step 1: To select mode:

- Timer mode is selected by the T0CS bit of the OPTION_REG register, (T0CS: 0=timer, 1=counter).
- When used, the prescaler should be assigned to the timer/counter by clearing the PSA bit of the OPTION_REG register. The prescaler rate is set by using the PS2-PS0 bits of the same register.
- When using interrupt, the GIE and TMR0IE bits of the INTCON register should be set.

Configuring/ Using Timer 0 (Cont.)

Step 2: Measuring and Counting

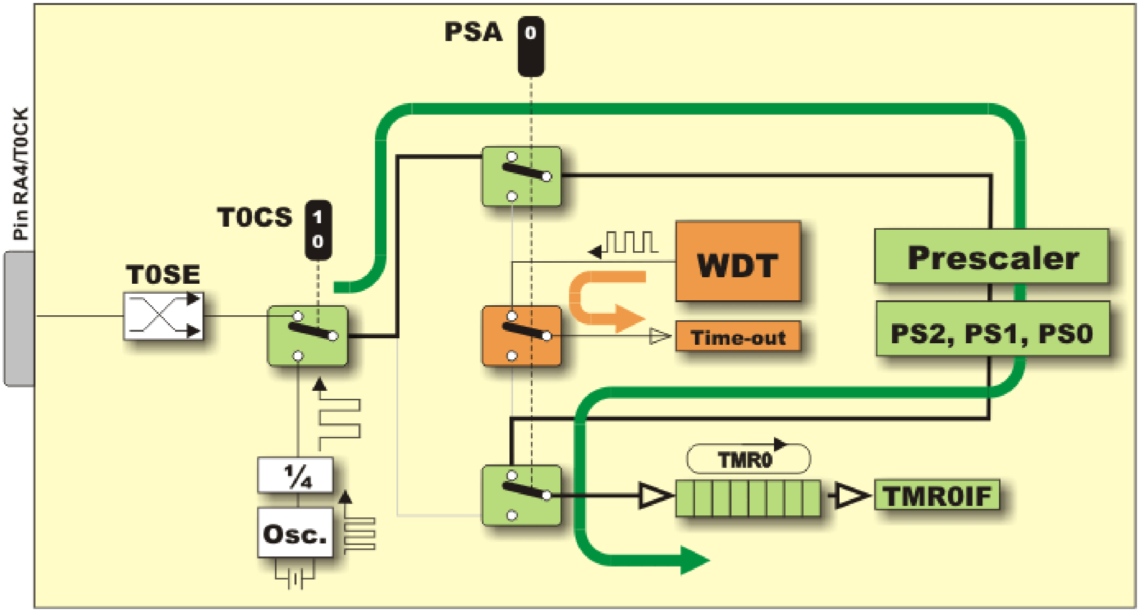
To measure time:

- Reset the TMR0 register or write some known value to it.
- Elapsed time (in microseconds when using 4MHz quartz) is measured by reading the TMR0 register.
- The flag bit TMR0IF of the INTCON register is automatically set every time the TMR0 register overflows. If enabled, an interrupt occurs.

To count pulses:

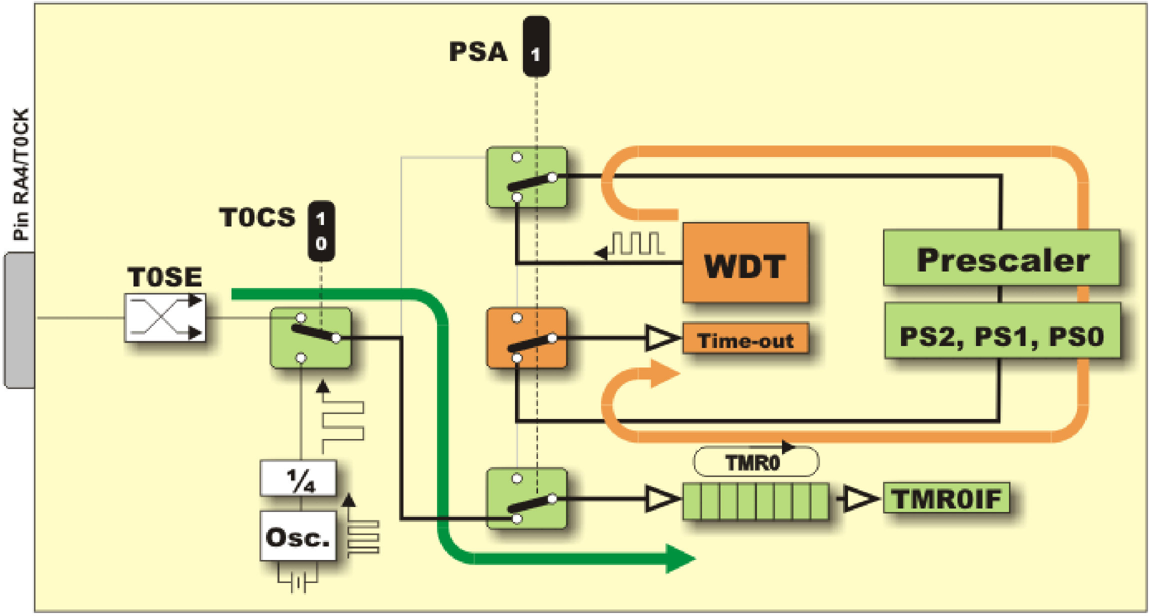
- The polarity of pulses are to be counted on the RA4 pin is selected by the TOSE bit of the OPTION_REG register (T0SE: 0=positive, 1=negative pulses).
- Number of pulses may be read from the TMR0 register. The prescaler and interrupt are used in the same manner as in timer mode.

Configuring/ Using Timer 0 (Cont.)

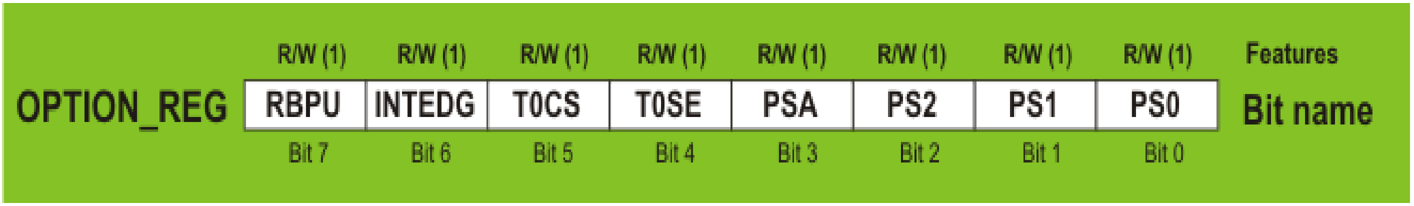


When PSA bit is cleared, prescaler is asigned to TMR0 timer/counter as ilustrated on the figure.

When PSA bit is set, prescaler is asigned to watch-dog timer as ilustrated on the figure:



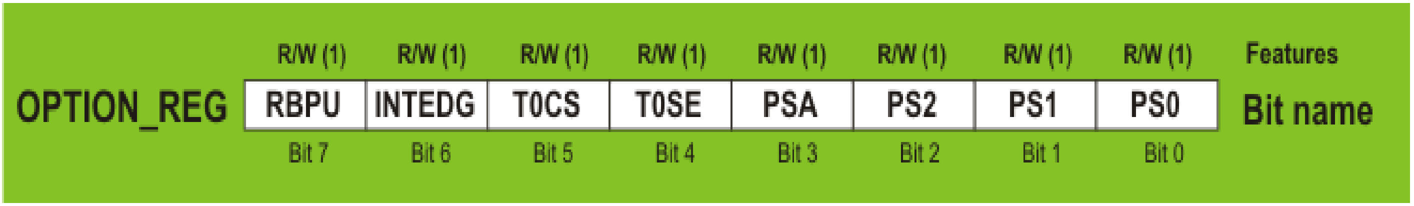
Option Register



	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- **RBPU - PORTB Pull-up enable bit**
  - 1 - PORTB pull-up resistors are disabled.
  - 0 - PORTB pins can be connected to pull-up resistors.
- **INTEDG - Interrupt Edge Select bit**
  - 0 - Interrupt on rising edge of the INT pin (0-1).
  - 1 - Interrupt on falling edge of the INT pin (1-0).
- **T0CS - TMR0 Clock Select bit**
  - 1 - Pulses are brought to TMR0 timer/counter input through the RA4 pin.
  - 0 - Timer uses internal cycle clock (Fosc/4).
- **T0SE - TMR0 Source Edge Select bit**
  - 0 - Increment on high-to-low transition on the TMR0 pin.
  - 1 - Increment on low-to-high transition on the TMR0 pin.
- **PSA - Prescaler Assignment bit**
  - 1 - Prescaler is assigned to the WDT.
  - 0 - Prescaler is assigned to the TMR0 timer/counter.

Option Register (Cont.)



	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- **PS2, PS1, PS0 - Prescaler Rate Select bit**
  - Prescaler rate is adjusted by combining these bits. As seen in the table, the same combination of bits gives different prescaler rate for the timer/counter and watch-dog timer, respectively.

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Note: When TMR0 is set in counter mode and prescaler is assigned to WDT then prescaler of counter is set in 1:1.

Calculating Count, Fout, and TMR0 values

Using system (internal) clock cycle:

If using INTERNAL crystal as clock, the division is performed as follow:
PIC TIMER0 formula for internal clock

$$\boxed{fout = \frac{fclk}{4 * \mathrm{Prescaler} * (256 - \mathrm{TMR0}) * \mathrm{Count}}} \quad where \quad \boxed{Tout = \frac{1}{fout}}$$

Here, Fout– The output frequency after the division.

Tout – The Cycle Time after the division.

4 - The division of the original clock (4 MHz) by 4, when using internal crystal as clock (and not external oscillator).

Count - A numeric value to be placed to obtain the desired output frequency - Fout.

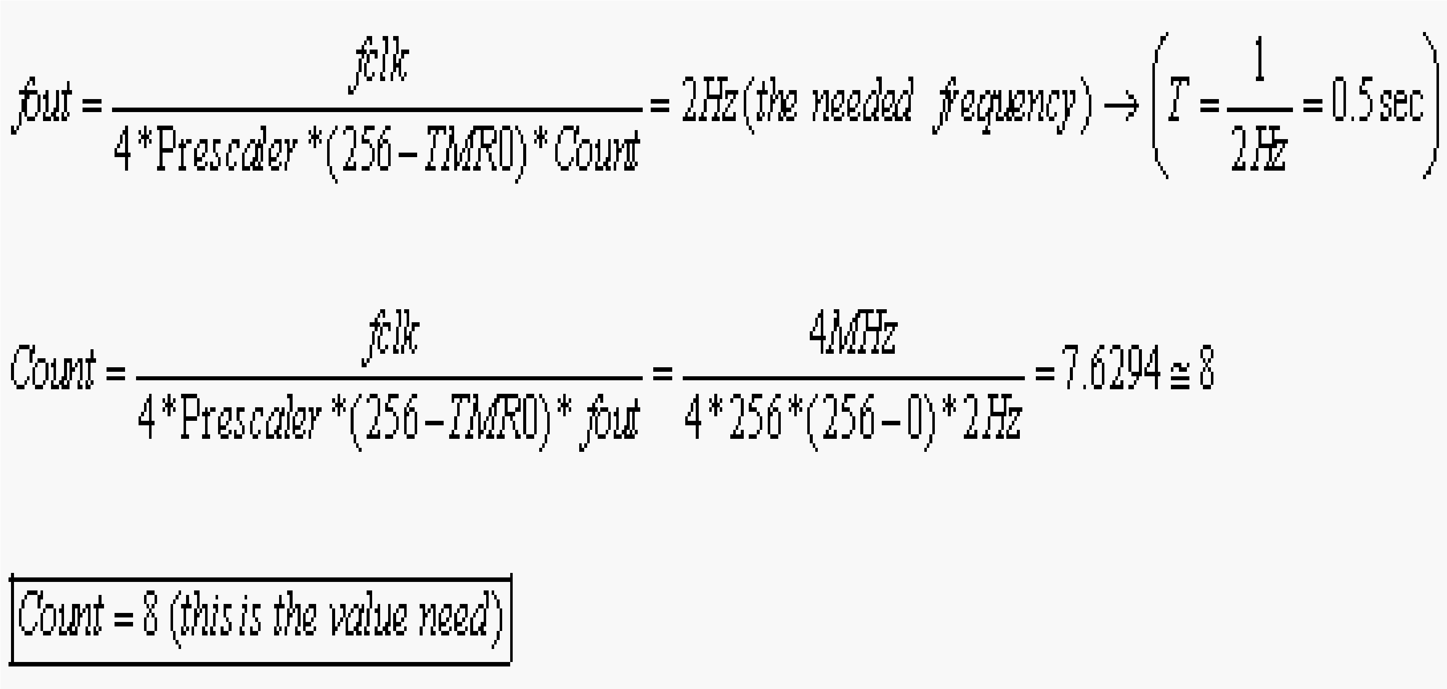
(256 - TMR0) - The number of times in the timer will count based on the register TMR0.

Calculating Count, Fout, and TMR0 values (Cont.)

An example of INTERNAL crystal as clock

Suppose we want to create a delay of 0.5 second in the program using Timer0. What is the value of Count?

Calculation: First, let's assume that the frequency division by the Prescaler will be 1:256. Second, let's set TMR0=0. Thus:



$$fout=\frac{fclk}{4*Prescaler*(256-TMR0)*Count}=2Hz(\text{the needed frequency})\rightarrow\left(T=\frac{1}{2Hz}=0.5\sec\right)$$

$$Count=\frac{fclk}{4*Prescaler*(256-TMR0)*fout}=\frac{4MHz}{4*256*(256-0)*2Hz}=7.6294\cong 8$$

$$\boxed{Count=8\ (\text{this is the value need})}$$

Calculating Count, Fout, and TMR0 values (Cont.)

Using external clock (from RA4 pin): If using EXTERNAL clock source (oscillator), the division is performed as follow:

PIC TIMER0 formula for external clock

$$\boxed{fout = \frac{fclk}{\text{Prescaler} * (256 - \text{TMR0}) * \text{Count}}} \quad where \quad \boxed{Tout = \frac{1}{fout}}$$

In this case there is no division by 4 of the original clock. We use the external frequency as it is.

Example: What is the output frequency - Fout, when the external oscillator is 100kHz and Count=8?

Calculation: First, let's assume that the frequency division by the Prescaler will be 1:256. Second, let's set TMR0=0. Thus:

$$fout = \frac{fclk}{Prescaler * (256 - TMR0) * Count} = \frac{100kHz}{256 * (256 - 0) * 8} = 0.19Hz$$

$$Tout = \frac{1}{fout} = \frac{1}{0.19} = 5.243\sec$$

$$\boxed{fout = 0.19Hz \ (Output \ \ frequency) \rightarrow \ \ Tout = 5.243\sec \ (Delay \ on \ the \ output)}$$

Problem: Timer0 as timer (Precise time delay by using TMR0)

Write a program to invert the status of PORTB in every 1 second later. Use timer 0 to create time delay. Suppose, the clock of external crystal oscillator is 8MHz.

Solution:

I f we choose prescler as 1:128, initial count (TMR0)131, count value 125, then for 8MHz clock we get 1sec time delay from following equation.

$$f_{out} = \frac{1}{T} = \frac{f_{clk}}{4 \times Prescaler \times (256 - TMR0) \times Count}$$

$$T = \frac{4 \times Prescaler \times (256 - TMR0) \times Count}{f_{clk}}$$

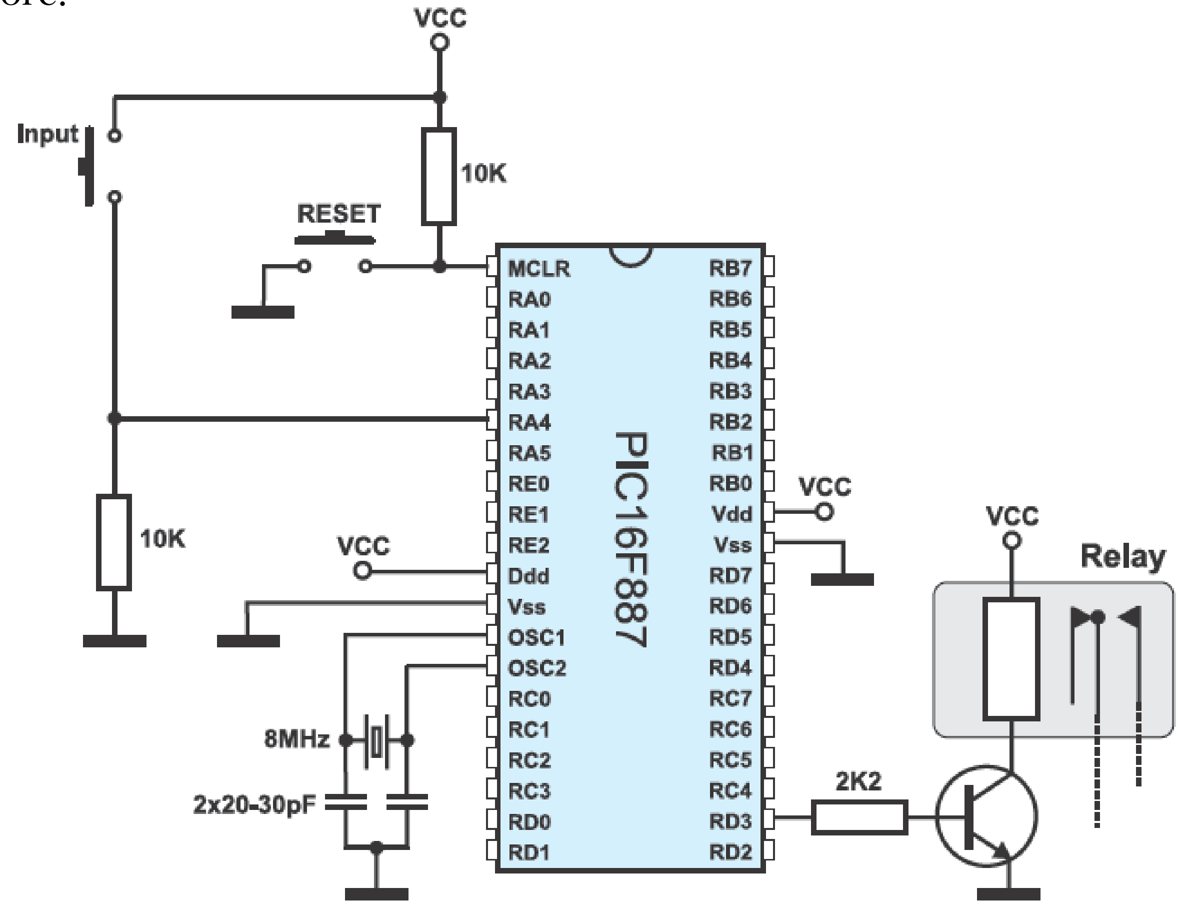
$$T = \frac{4 \times 128 \times (256 - 131) \times 125}{8 \times 10^6} = 1\ sec$$

```
unsigned count;
void interrupt()
{
count++;
INTCON.T0IE=1;
INTCON.T0IF=0;  // clear timer 0 overflow flag bit
TMR0=131;
}
void main() {
ANSEL=0;
ANSELH=0;
OPTION_REG=0b00000110; // prescaler 1:128
TMR0=131;
INTCON=0xA0;
TRISB=0;
PORTB=0xFF;
while(1)
{
if(count==125){ PORTB=~PORTB; count=0;}
}
}
```

Problem: Timer0 as Counter.

There is a schematic in figure below, while the program is on the next page. Timer TMR0 is used as a counter. The counter input is connected to a push button so that any button press causes timer TMR0 to count one pulse. When the number of pulses matches the number stored in the TEST register, a logic one (5V) appears on the pin PORTD.3. This voltage activates an electromechanical relay, and this bit is called 'RELAY' in the program, therefore.

In this example, the TEST register stores number 5. Of course, it can be any number obtained either by computing or defined as a constant. Besides, the microcontroller can activate some other device instead of relay, while the sensor can be used instead of the push button. This example illustrates one of the most common applications of the microcontroller in the industry; when something is performed as many times as needed, then something else should be turned on or off.



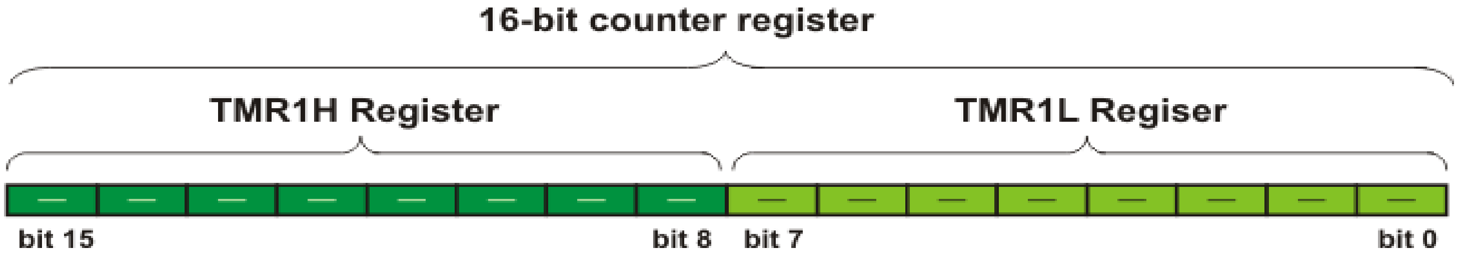
```
void main() {
   char TEST = 5;           // Constant TEST = 5
   enum outputs {RELAY = 3}; // Constant RELAY = 3

   ANSEL = 0;               // All I/O pins are configured as digital
   ANSELH = 0;
   PORTA = 0;               // Reset port A
   TRISA = 0xFF;            // All portA pins are configured as inputs
   PORTD = 0;               // Reset port D
   TRISD = 0b11110111;      // Pin RD3 is configured as an output
   OPTION_REG.F5 = 1;       // Counter TMR0 receives pulses through the RA4 pin
   OPTION_REG.F3 = 1;       // Prescaler rate is 1:1

   TMR0 = 0;                // Reset timer/counter TMR0
   do {
      if (TMR0 == TEST)     // Does the number in timer match constant TEST?
      (PORTD.RELAY = 1);    // Numbers match. Set the RD3 bit (output RELAY)
   }
   while (1);               // Remain in endless loop
}
```

Timer 1

Timer TMR1 module is a 16-bit timer/counter, which means that it consists of two registers (TMR1L and TMR1H). It can count up 65.535 pulses in a single cycle, i.e. before the counting starts from zero.



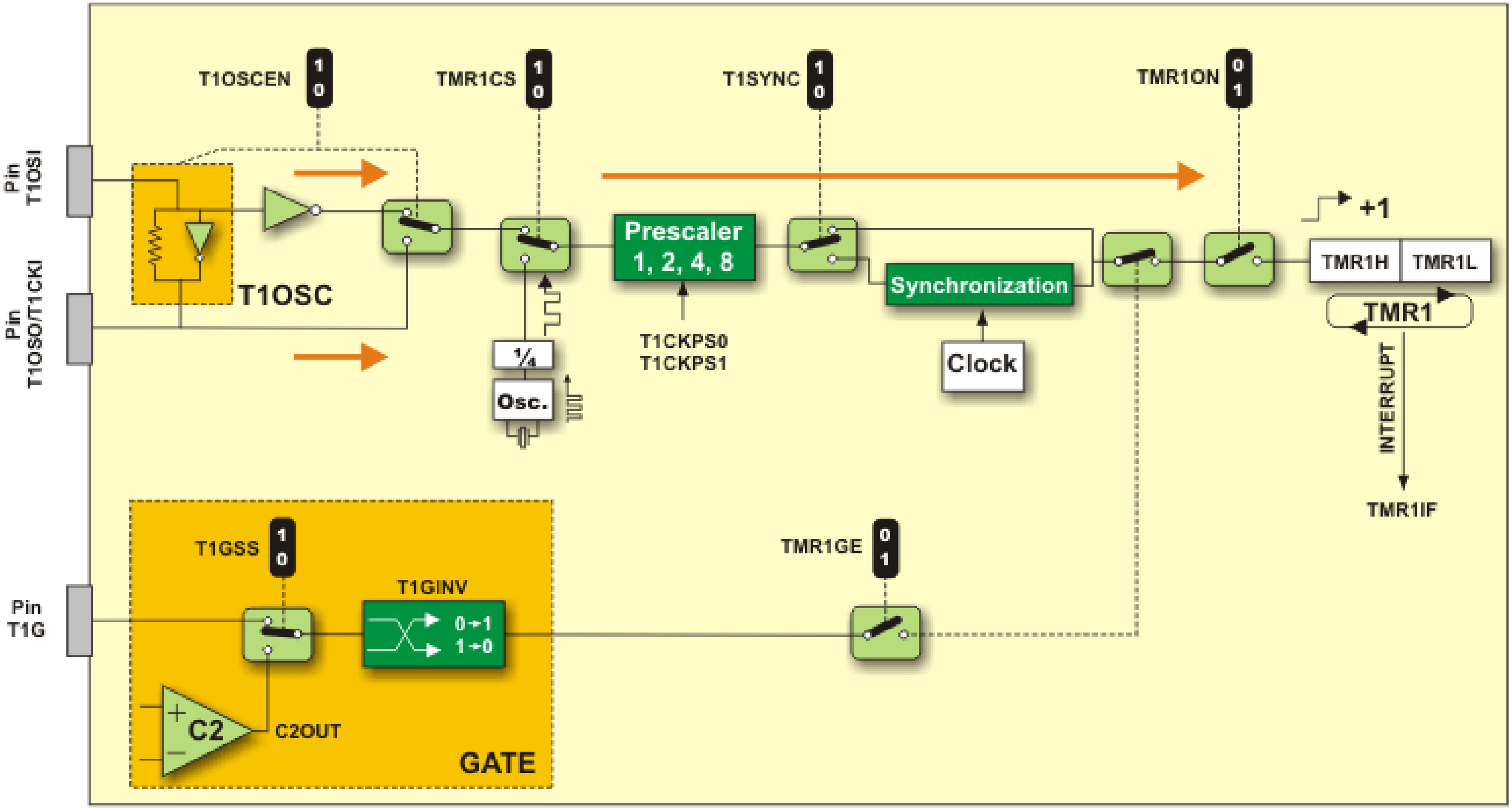
Similar to the timer TMR0, these registers can be read or written to at any moment. In case an overflow occurs, an interrupt is generated if enabled.

The timer TMR1 module may operate in one of two basic modes, that is as a timer or a counter. Unlike the TMR0 timer, both of these modes have additional functions.

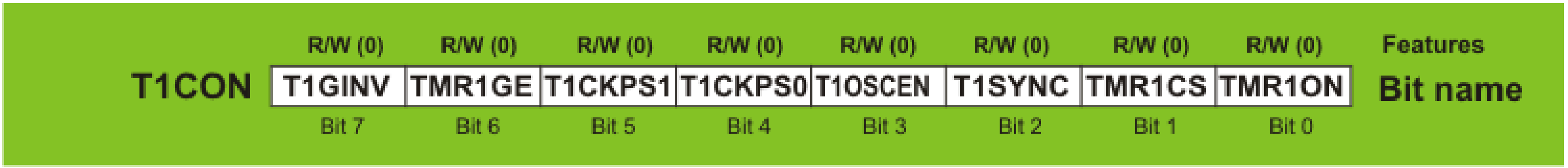
The TMR1 timer has following features:

- 16-bit timer/counter register pair;
- Programmable internal or external clock source;
- 3-bit prescaler;
- Optional LP oscillator;
- Synchronous or asynchronous operation;
- Timer TMR1 gate control (count enable) via comparator or T1G pin;
- Interrupt on overflow;
- Wake-up on overflow (external clock); and
- Time base for Capture/Compare function.

Configuring Timer 1



T1CON Register



	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Legend

R/W	Readable/Writable bits
(0)	After reset, bit is cleared

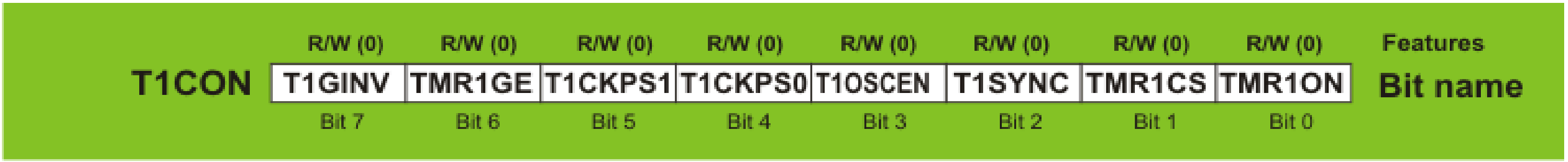
T1GINV - Timer1 Gate Invert bit acts as logic state inverter on the T1G pin gate or the comparator C2 output (C2OUT) gate. It enables the timer to mea sure time whilst the gate is high or low.

- 1 - Timer 1 counts when the T1G pin or bit C2OUT gate is high (1).
- 0 - Timer 1 counts when the T1G pin or bit C2OUT gate is low (0).

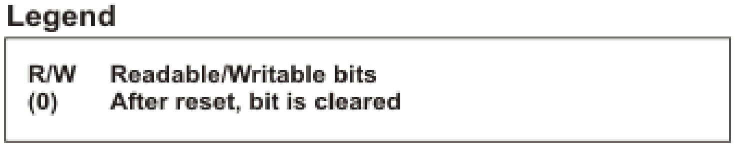
TMR1GE - Timer1 Gate Enable bit determines whether the T1G pin or comparator C2 output (C2OUT) gate will be active or not. This bit is functional only in the event that the timer TMR1 is on (bit TMR1ON = 1). Otherwise, this bit is ignored.

- 1 - Timer TMR1 is on only if Timer1 gate is not active.
- 0 - Gate has no influence on the timer TMR1.

T1CON Register (Cont.)



	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	



Legend

R/W	Readable/Writable bits
(0)	After reset, bit is cleared

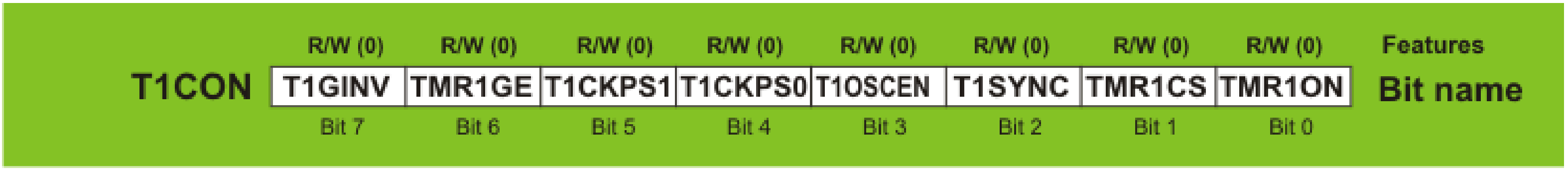
T1CKPS1,T1CKPS0 - Determine the rate of the prescaler assigned to the timer TMR1.

T1CKPS1	T1CKPS0	PRESCALER RATE
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

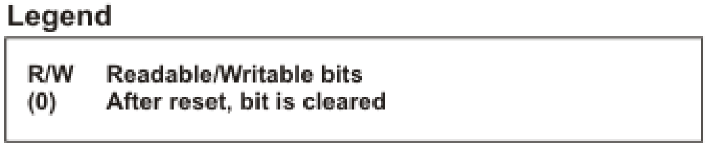
T1OSCEN - LP Oscillator Enable Control bit

- 1 - LP oscillator is enabled for timer TMR1 clock (oscillator with low power consumption and frequency 32.768 kHz).
- 0 - LP oscillator is off.

T1CON Register (Cont.)



	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	



Legend

R/W	Readable/Writable bits
(0)	After reset, bit is cleared

T1SYNC - Timer1 External Clock Input Synchronization Control bit enables synchronization of the LP oscillator input or T1CKI pin input with the microcontroller internal clock. This bit is ignored while counting pulses from the main oscillator (bit TMR1CS = 0).

- 1 - Do not synchronize external clock input.
- 0 - Synchronize external clock input.

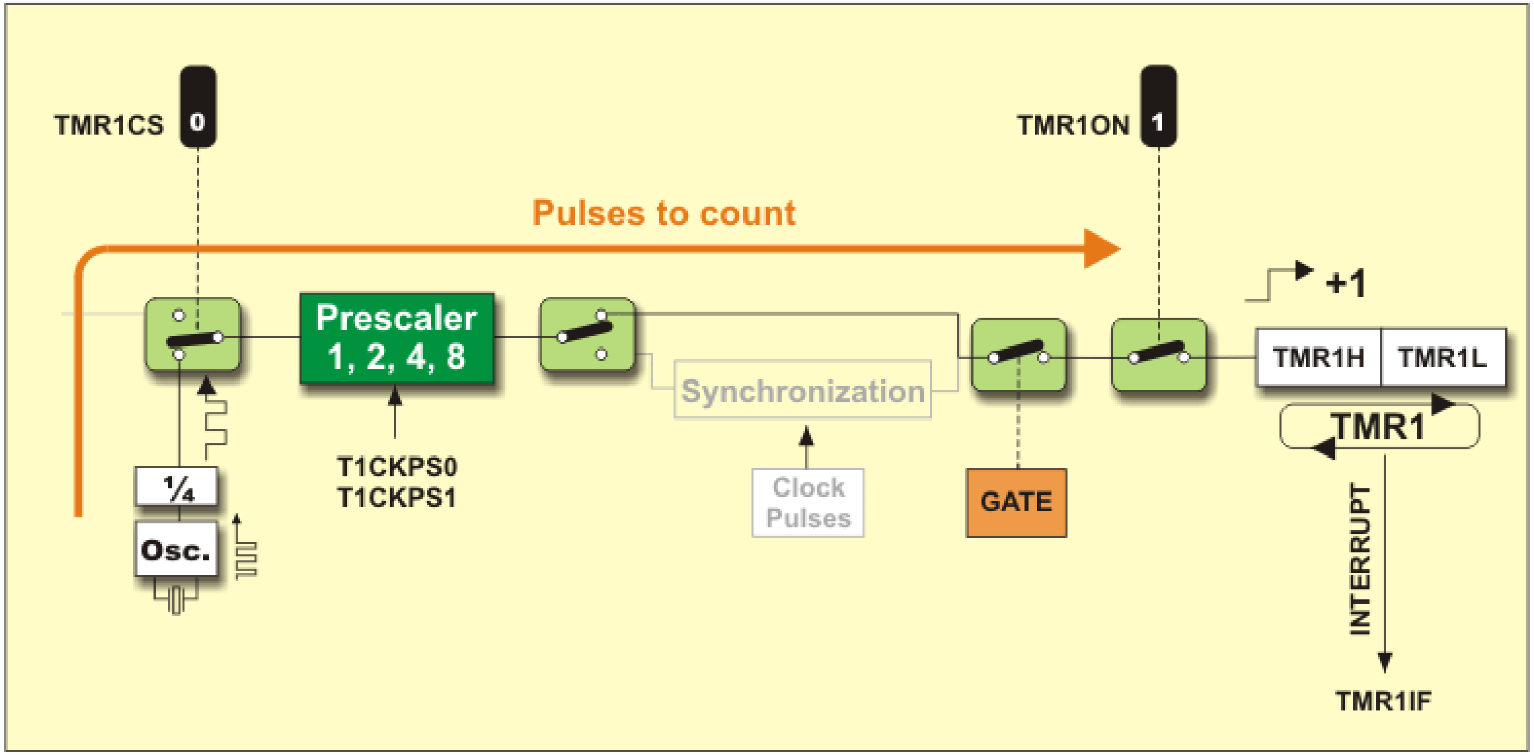
TMR1CS - Timer TMR1 Clock Source Select bit

- 1 - Count pulses on the T1CKI pin (on the rising edge 0-1).
- 0 - Count pulses of the microcontroller internal clock.

TMR1ON - Timer1 On bit

- 1 - Enable timer TMR1.
- 0 - Stop timer TMR1.

TMR1 IN TIMER MODE

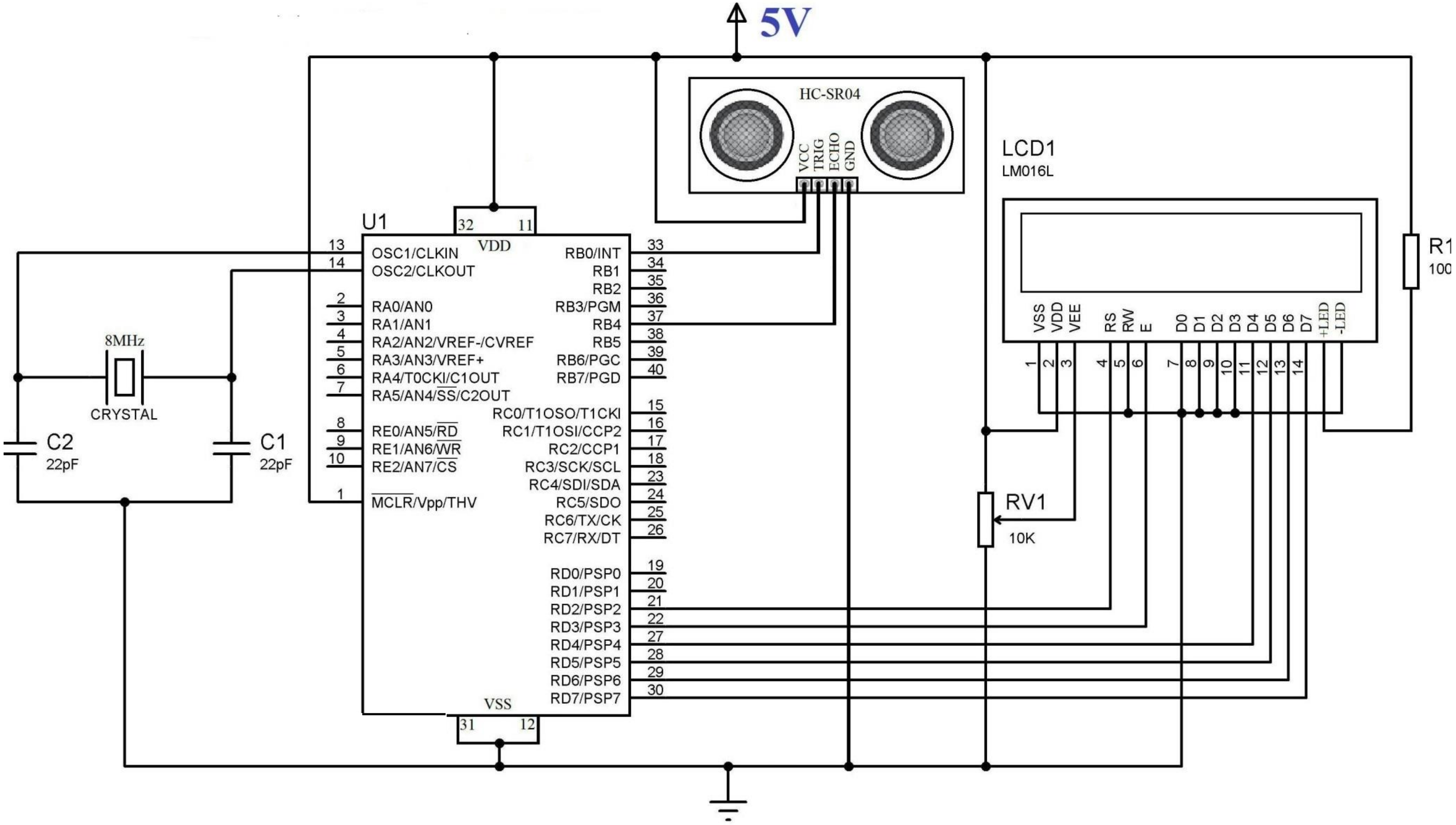


In order to select this mode, it is necessary to clear the TMR1CS bit. After this, the 16-bit register will be incremented on every pulse generated by the internal oscillator. If the 4MHz quartz crystal is in use, it will be incremented every microsecond.

In this mode, the T1SYNC bit does not affect the timer because it counts internal clock pulses. Since the whole electronics uses these pulses, there is no need for synchronization.

Example of using Timer 1: Measuring Distance by Ultrasonic Sensor

Write code to find the distance of an object using following circuit diagram.

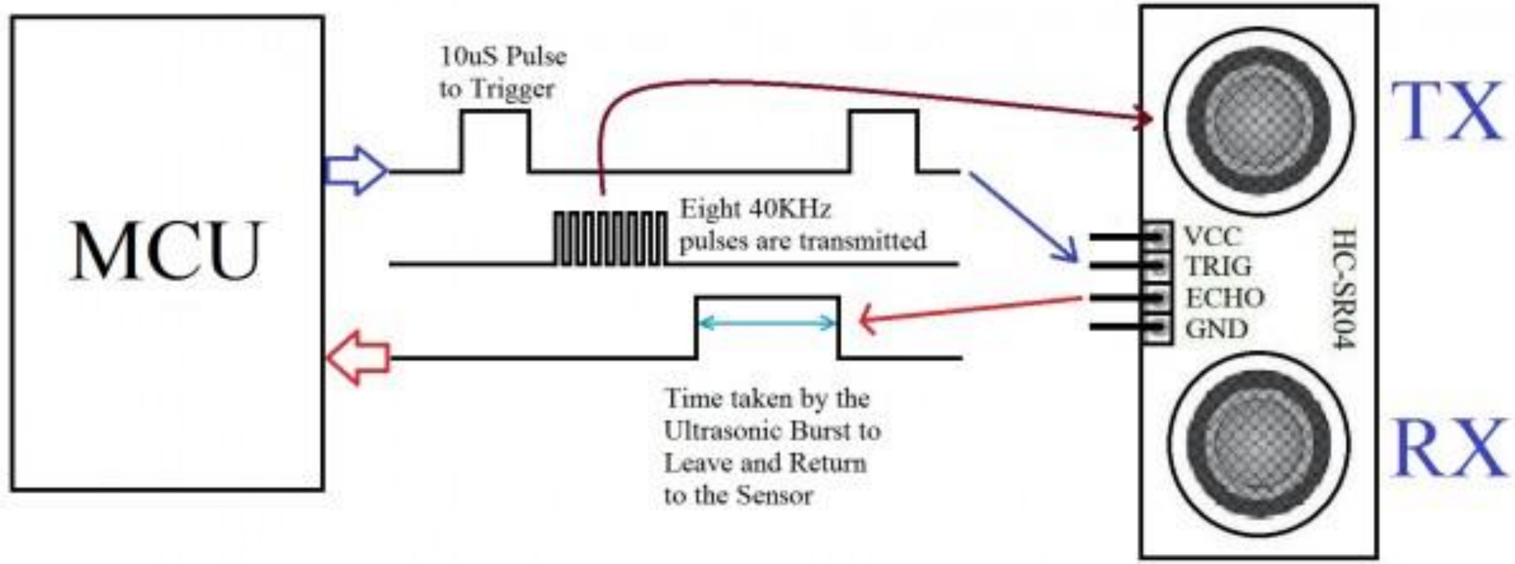


Courtesy: www.electrosome.com

Example of using Timer 1: Measuring Distance by Ultrasonic Sensor (Cont.)

Ultrasonic Sensor

- Provide TRIGGER signal, atleast 10µS High Level (5V) pulse.
- The module will automatically transmit eight 40KHz ultrasonic burst.
- If there is an obstacle in-front of the module, it will reflect the ultrasonic burst.
- If the signal is back, ECHO output of the sensor will be in HIGH state (5V) for a duration of time taken for sending and receiving ultrasonic burst. Pulse width ranges from about 150µS to 25mS and if no obstacle is detected, the echo pulse width will be about 38ms.


MCU
10uS Pulse to Trigger
Eight 40KHz pulses are transmitted
Time taken by the Ultrasonic Burst to Leave and Return to the Sensor
TX
RX
VCC
TRIG
ECHO
GND
HC-SR04

Example of using Timer 1: Measuring Distance by Ultrasonic Sensor (Cont.)

Configuring Timer 1 as timer mode

Since we will use Timer1 module as a Timer, we should use internal clock (Fosc/4), ie **TMR1CS = 0**. Prescaler is used to divide the internal clock (Fosc/4). Here we can set Prescaler as 2, ie **T1CKPS1 = 0** & **T1CKPS0 = 1**. **T1SYNC bit is ignored** when TMR1CS = 0. As we are using internal clock (Fosc/4) we can disable oscillator, ie **T1OSCEN = 0**. **TMR1ON** bit can be used to ON or OFF timer as per our requirements.
Thus we can initialize timer as : **T1CON = 0x10**
To TURN ON the Timer : **T1CON.F0 = 1** or **TMR1ON = 1**
To TURN OFF the Timer : **T1CON.F0 = 0** or **TMR1ON = 0**

Fosc is the oscillator frequency, here we are using 8MHz crystal hence **Fosc = 8MHz**.
Time = (TMR1H:TMR1L)*(1/Internal Clock)*Prescaler
Internal Clock = Fosc/4 = 8MHz/4 = 2MHz
Therefore, **Time = (TMR1H:TMR1L)*2/(2000000) = (TMR1H:TMR1L)/1000000**

Example of using Timer 1: Measuring Distance by Ultrasonic Sensor (Cont.)

Distance Calculation:

Distance = Speed * Time

Let **d** be the distance between Ultrasonic Sensor and Target

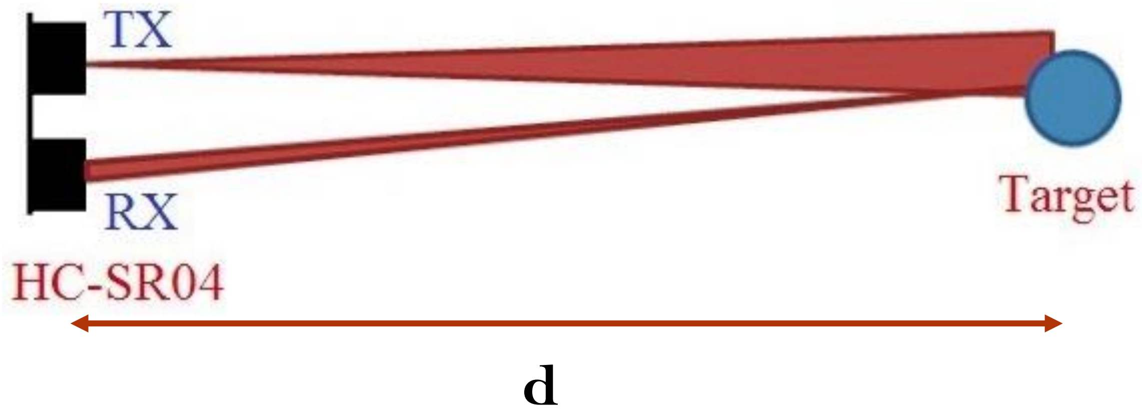
Total distance traveled by the ultrasonic burst : **2d** (forward and backward)

Suppose, Speed of Sound in Air : **340 m/s = 34000 cm/s**

Thus, **d = (34000*Time)/2**, where Time = (TMR1H:TMR1L)/(1000000)

Therefore, **d = (TMR1H:TMR1L)/58.82 cm**

TMR1H:TMR1L = TMR1L | (TMR1H<<8)



Example of using Timer 1: Measuring Distance by Ultrasonic Sensor (Cont.)

Program without using interrupt

```
sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// End LCD module connections

void main()
{
 int a;
 char txt[7];
```

```
Lcd_Init();
 Lcd_Cmd(_LCD_CLEAR);  // Clear display

Lcd_Cmd(_LCD_CURSOR_OFF);    // Cursor off

 TRISB = 0b00010000; //RB4 as Input PIN (ECHO)

 T1CON = 0x10;   //Initialize Timer Module

 while(1)
 {
  TMR1H = 0;       //Sets the Initial Value of Timer
  TMR1L = 0;       //Sets the Initial Value of Timer

   PORTB.F0 = 1;           //TRIGGER HIGH
   Delay_us(10);          //10uS Delay
   PORTB.F0 = 0;           //TRIGGER LOW

   while(!PORTB.F4);         //Waiting for Echo
T1CON.F0 = 1;           //Timer Starts
while(PORTB.F4);          //Waiting for Echo goes LOW
  T1CON.F0 = 0;             //Timer Stops

   a = (TMR1L | (TMR1H<<8));   //Reads Timer Value
   a = a/58.82;             //Converts Time to Distance
   if(a>=2 && a<=400)        //Check whether the result is valid or
not
   {
    IntToStr(a,txt);
    Ltrim(txt);
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Distance = ");
    Lcd_Out(1,12,txt);
    Lcd_Out(1,15,"cm");
   }
   else
   {
   Lcd_Cmd(_LCD_CLEAR);
   Lcd_Out(1,1,"Out of Range");
   }
  }
}
```

Example of using Timer 1: Measuring Distance by Ultrasonic Sensor (Cont.)

Program by Using PORTB On-Change Interrupt

```
// LCD module connections
sbit LCD_RS at RD2_bit;
sbit LCD_EN at RD3_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;
sbit LCD_RS_Direction at TRISD2_bit;
sbit LCD_EN_Direction at TRISD3_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
// End LCD module connections
int a;
//Interrupt function will be
automatically executed on Interrupt
```

```
void interrupt()
{
 if(INTCON.RBIF == 1)  //Makes sure that it is PORTB On-
                                          Change Interrupt
  {
   INTCON.RBIE = 0;         //Disable On-Change Interrupt
   if(PORTB.F4 == 1)        //If ECHO is HIGH
    T1CON.F0 = 1;           //Start Timer
   if(PORTB.F4 == 0)        //If ECHO is LOW
   {
    T1CON.F0 = 0;                //Stop Timer
    a = (TMR1L | (TMR1H<<8))/58.82;
   }
  }
 INTCON.RBIF = 0; //Clear PORTB On-Change Interrupt
                                                 flag
 INTCON.RBIE = 1;  //Enable PORTB On-Change Interrupt
}
```

```
void main()
{
  char txt[7];
  Lcd_Init();
  Lcd_Cmd(_LCD_CLEAR);                // Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF);           // Cursor off
 TRISB = 0b00010000;
  INTCON.GIE = 1;                     //Global Interrupt Enable
  INTCON.RBIF = 0;                    //Clear PORTB On-Change Interrupt Flag
  INTCON.RBIE = 1;                    //Enable PORTB On-Change Interrupt
 T1CON = 0x10;                        //Initializing Timer Module
  while(1)
  {
   TMR1H = 0;                         //Setting Initial Value of Timer
   TMR1L = 0;                         //Setting Initial Value of Timer
    a = 0;
    PORTB.F0 = 1;                     //TRIGGER HIGH
    Delay_us(10);                     //10uS Delay
    PORTB.F0 = 0;                     //TRIGGER LOW
    Delay_ms(100);                    //Waiting for ECHO
 a = a + 1;                     //Error Correction Constant
   if(a>2 && a<400)                    //Check whether the result is valid or not
   {
    IntToStr(a,txt);
    Ltrim(txt);
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Distance = ");
    Lcd_Out(1,12,txt);
    Lcd_Out(1,15,"cm");
   }
   else
   {
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Out of Range");
   }
   Delay_ms(400);
  }
}
```