

COURSE MATERIAL ON EMBEDDED C PROGRAMMING ON TIMER DATE 22/092020

What is a Timer

A timer is a clock that controls the sequence of an event while counting in fixed intervals of time. It is used for producing precise time delay. Secondly, it can be used to repeat or initiate an action after/at a known period of time. This feature is very commonly used in several applications. An example could be setting up an alarm which triggers at a point of time or after a period of time.

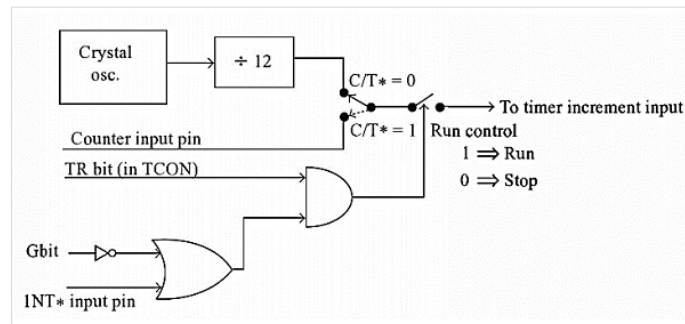
Timers in a controller: Why to use them

Most microcontrollers have built-in timers that not only generate time delays, but can also be used as counters to count an action or event. The value of a counter increases by one every time its corresponding action or event occurs. Timers in a controller are inbuilt chips that are controlled by special function registers (SFRs) assigned for Timer operations. These SFRs are used to configure Timers in different modes of operations.

While working with microcontrollers, it is more than often required to generate time delays. There are two possible ways of generating time delays. First is by using the code, like using *for* or *while* loops in a C program. However, the delays provided by the software **are not very precise**. The other method is to use Timers. Timers provide **time delays** that are very precise and accurate.

8051 microcontroller has two Timers designated as Timer0 and Timer1. Each of these timers is assigned a 16-bit register. The value of a Timer register increases by one every time a timer counts. Timer takes a time period of one machine cycle to count one. (Machine cycle is a unit that refers to the time required by the microcontroller to execute instructions.) This means that the **maximum number of times** a timer can count without repeating is 2^{16} , i.e., **65536**. So the maximum allowed counts in value of Timer registers can be from 0000H to FFFFH.

Since **8051 is an 8 bit controller**, the registers of 8051 Timers are accessed as two different registers; one for lower byte and other for higher byte. For example, register of Timer0 is accessed as **TL0** for lower byte and **TH0** for higher byte. Similarly **TL1** and **TH1** are registers assigned to Timer 1.



In the following table, we will see the bit details and their different operations for high or low value.

Bit Details	High Value(1)	Low Value(0)
C/ \bar{T}	Configure for the Counter operations	Configure for the Timer operations
Gate (G)	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high.	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high and $\overline{INT0}$ or $\overline{INT1}$ is high.

8051 Timer Issues

While using 8051 Timers certain factors need to be considered, like whether the Timer is to be used for time keeping or for counting; whether the source for time generation is external clock or the controller itself; how many bits of Timer register are to be used or left unused.

How a Timer functions

The registers of Timers are loaded with some initial value. The value of a Timer register increases by one after every machine cycle. One machine cycle duration is the 1/12th of the frequency of the crystal attached to the controller.

For example, if the frequency of the crystal is 12 MHz, then the frequency for Timer will be 1MHz (1/12 of crystal frequency) and hence the time ($T = 1/f$) taken by the Timer to count by one is **1 μ s** (1/1MHz). Similarly if an 11.0592 MHz crystal is used, operating frequency of Timer is 921.6 KHz and the time period is **1.085 μ s**.

If no value is loaded into the Timer, it starts counting from 0000H.

When the Timer reaches FFFFH, it reloads to 0000H. This **roll over** is communicated to the controller by raising a flag corresponding to that Timer, i.e., a flag bit is raised (set high) when the timer starts counting from 0000H again. **TF0** and **TF1** are the Timer flags corresponding to Timers 0 and 1. These flags must be cleared (set low) by software every time they are raised. The Timer may terminate updating register values after a roll over or continue with its operation.

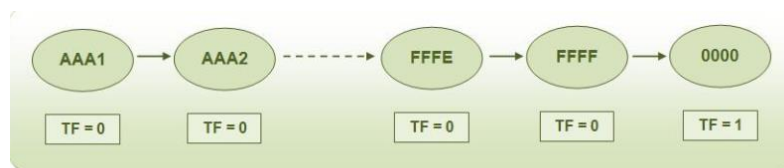


Fig. 2: Image showing Status of Timer Flag on Roll Over

Starting or stopping a Timer

For every Timer, there is a corresponding Timer control bit which can be set or cleared by the program to start or stop the Timer. **TR0** and **TR1** are the control bits for Timers 0 and 1 respectively. Setting the control bit would start the Timer.

$TR0 = 1;$ starts Timer 0

$TR1 = 1;$ starts Timer 1

Clearing the control bit would stop the Timer.

$TR0 = 0$; stops Timer 0

$TR1 = 0;$ stops Timer1

Configuring a Timer

A register called **TMOD** is used for configuring the Timers for the desired operation. TMOD is an 8-bit register with following bit configuration:



The lower four bits (TMOD.0 – TMOD.3) are used to configure Timer 0 while the higher four bits (TMOD.4 – TMOD.7) are for Timer 1. When **GATE** is high, the corresponding Timer is enabled only when there is an interrupt at corresponding INTx pin of AT89C51 controller and Timer control bit is high. Otherwise only setting Timer control bit is sufficient to start the Timer.

If C/T is low, Timer is used for time keeping, i.e., Timer updates its value automatically corresponding to 8051 clock source. When C/T is high, Timer is used as counter, i.e., it updates its value when it receives pulse from outside the 8051 controller.

M1 and M0 bits decide the Timer modes. There are four Timer modes designated as Modes 0, 1, 2 and 3.

Modes 1 and 2 are most commonly used while working with Timers.

TMOD = 0x01; sets the mode1 of Timer0 used for timing

TMOD = 0x20; sets the mode2 of Timer1 used for timing

Different modes of a Timer

There are four Timer modes designated as Modes 0, 1, 2 and 3. A particular mode is selected by configuring the M1 & M0 bits of TMOD register.

Mode	M1	M0	Operation
Mode 0	0	0	13-bit Timer
Mode 1	0	1	16-bit Timer
Mode 2	1	0	8-bit Auto Reload
Mode 3	1	1	Split Timer Mode

(i) Mode 0 : 13-bit Timer

Mode 0 is a 13 bit Timer mode and uses 8 bits of high byte and 5 bit prescaler of low byte. The value that the Timer can update in mode0 is from 0000H to 1FFFH. The 5 bits of lower byte append with the bits of higher byte. The Timer rolls over from 1FFFH to 0000H to raise the Timer flag.

(ii) Mode 1 : 16-bit Timer

Mode1 is one of the most commonly used Timer modes. It allows all 16 bits to be used for the Timer and so it allows values to vary from 0000H to FFFFH.

If a value, say YYXXH, is loaded into the Timer bytes, then the delay produced by the Timer will be equal to the product :

$$[(\text{FFFFH} - \text{YYXXH} + 1) \times (\text{period of one timer clock})] .$$

It can also be considered as follows: convert YYXXH into decimal, say NNNNN, then delay will be equal to the product :

$$[(65536 - \text{NNNNN}) \times (\text{period of one timer clock})] .$$

The period of one timer clock is 1.085 μ s for a crystal of 11.0592 MHz frequency as discussed above.

Now **to produce a desired delay**, divide the required delay by the Timer clock period. Assume that the division yields a number NNNNN. This is the number of times Timer must be updated before it stops. Subtract this number from 65536 (binary equivalent of FFFFH) and convert the difference into hex. This will be the initial value to be loaded into the Timer to get the desired delay.

The calculator application in Windows can be a handy tool to carry out these calculations.

Programming 8051 Timers

The programming of 8051 Timers can be done by using either polling method or by using interrupt. In polling, the microcontroller keeps monitoring the status of Timer flag. While doing so, it does no other operation and consumes all its processing time in checking the Timer flag until it is raised on a rollover. In interrupt method controller responds to only when the Timer flag is raised. The interrupt method prevents the wastage of controller's processing time unlike polling method.

Polling is mostly used for time delay generation and interrupt method is more useful when waveforms are to be generated or some action has to be repeated in fixed delays.

(i) Polling Method

The polling method involves the following algorithm:

1. Configure the Timer mode by passing a hex value into the TMOD register. This will tell the controller about which Timer is to be used; the mode of Timer; operation (to be used as timer or counter); and whether external interrupt is required to start Timer.
2. Load the initial values in the Timer low TLx and high THx byte. (x = 0/1)
3. Start the Timer by setting TRx bit.
4. Wait while the Timer flag TFx is raised.
5. Clear the Timer flag. The Timer flag is raised when Timer rolls over from FFFFH to 0000H. If the Timer is not stopped, it will start updating from 0000H in case of modes 0 & 1 while with initial value in case of mode 2. If TFx is not cleared, controller will not be able to detect next rollover.

6. Stop the Timer by clearing TRx bit. If TRx bit is not cleared the Timer will restart updating from 0000H after the rollover in case of modes 0 and 1 while with initial value in case of mode 2.

(ii) Interrupt Method

The interrupt method makes use of a register called Interrupt Enable (IE) register. An 8051 microcontroller has 6 hardware interrupts. The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller skips temporarily whatsoever it was doing and responds to the interrupt.

The controller starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler which is a piece of code that tells the processor or controller what to do on receipt of an interrupt. After the execution of ISR, controller returns to whatever it was doing earlier (before the interrupt was received).

The Interrupt Enable register has the following bits which enable or disable the hardware interrupts of 8051 microcontroller.

IE :	EA	-	ET2	ES	ET1	EX1	ET0	EX0
------	----	---	-----	----	-----	-----	-----	-----

When EA (IE.7) is set (=1), interrupts are enabled. When clear (EA=0), interrupts are disabled and controller does not respond to any interrupts.

ET0, ET1 & ET2 (IE.3, IE.4 & IE.5) are Timer interrupt bits. When set (high) the timer are enabled and when cleared (low) they are disabled. (8052 controllers have three Timers, so ET2 is its Timer 2 interrupt bit.) The ISR corresponding to these interrupts are executed when the TFX flags of respective Timers are raised. For more details on other IE register bits, refer Interrupts with 8051.

Note that IE register is bit addressable.

Timer programming using Timer interrupts involves following algorithm.

1. Configure the Timer mode by passing a hex value to TMOD register.
2. Load the initial values in the Timer low TLx and high THx byte.
3. Enable the Timer interrupt by passing hex value to IE register or setting required bits of IE register. For example,

IE = 0x82; enables Timer 0 interrupt

IE = 0x88; enables Timer 1 interrupt

Timer/Counter Calculations and Program using 8051 Microcontroller:

The delay is the one of the important factors in the application software development. The timers and counters are hardware components of the microcontroller, that are used in many applications to provide the accurate time delay with count pulses. Both the tasks are implemented by the software technique.

PROGRAM TO FIND THE DELAY USING TIMER OF 8051 IN EMBEDDED C

```
#include <reg51.h>
void timerdelay(void);
sbit mybit = P1^0;
void main()
{
while (1)
{
mybit = ~mybit; //toggle the bit between 0 and 1
timerdelay();
}
}

void timerdelay(void)
{
TMOD = 0X01; // TIMER 0 AND MODE 1
TL0=0X01; //LOAD THE TIMER
TH0=0X02;
TR0=1; //START THE TIMER
while(TF0==0)//WAIT UNTIL THE TIMER REACHED FFFF AND TF0 SET
TR0=0;
TF0=0;
}
```

PROGRAM TO FIND THE DELAY OF 500 msec USING TIMER OF 8051 IN EMBEDDED C

Consider the clock of 11.059 Mhz

This is divided by 12 as per internal structure of timer in 8051

$$11.059/12 = 0.921 \text{ MHz}$$

Therefore the one time for one clock will be $1/0.921 = 1.085$ microsecond

Now we want delay of 500 msec

So we have to obtain the count corresponding to this delay

Count = desired delay/one timer clock period

$$= 500\text{msec}/1.085\text{microsec}$$

$$= 460829 \text{ but this exceeds the } 65526$$

So let us take 50 msec first

$$= 50/1.085$$

= 46083 this is in decimal so convert it to Hex

It is b403 H

TH0=B4 TL0 =03

But the time delay for this will be only 50 msec.

For 500 msec repeat this for 10 so that $10 \times 50 = 500\text{msec}$

Program on next page

```

#include <reg51.h>
void timerdelay(void);
void main()
{
    unsigned char x ;
    P1=0X55;
    While(1)
    {
        P1 = ~P1; //toggle the bit between 0 and 1
        for (x=0;x<=10;x++)
            timerdelay();
    }
}

void timerdelay(void)
{
    TMOD = 0X01; // TIMER 0 AND MODE 1
    TL0=0X03; //LOAD THE TIMER
    TH0=0Xb4;
    TR0=1; //START THE TIMER
    while(TF0==0)//WAIT UNTIL THE TIMER REACHED FFFF AND TF0 SET
        TR0=0;
    TF0=0;
}

```