7

8051 INTERRUPTS AND TIMERS/COUNTERS

Learning Objectives

babsol

boldsarb on

After you have completed this chapter, you should be able to

- Explain interrupts and their classification
- Explain 8051 interrupt structure

Understand the timers/counters of the 8051

Comprehend the operation of Timer 0 and Timer 1 in various modes

Write programs for timers/counters

7.1 ||| BASICS OF INTERRUPTS

Interrupt is an input to a processor, whereby an external device or a peripheral can inform the processor that It is ready for communication. When peripheral devices activate an interrupt signal, the processor branches to a program called *interrupt service routine.* This program is written by the user for performing tasks that the interrupting device wants the processor to execute. After executing the interrupt service routine, the processor returns to the main program as shown in Fig. 7.1.

When peripheral devices interrupt the processor, branching takes place to interrupt service subroutine. Before branching, the actions taken by the processor are as follows:

1. It completes the execution of current instruction.
2. Program status word register value is pushed onto the stack.
3. Program counter value is pushed onto the stack.
4. Interrupt flag is reset.
5. Program counter is loaded with Interrupt Service Subroutine (ISS) address.

Figure 7.1 *Interrupt service subroutine processing*

When program counter is loaded with interrupt service subroutine address, branching to ISS takes place. During execution of ISS, interrupts are disabled because interrupt flag is reset. The ISS is ended with RETI instruction. The execution of RETI instruction results in the following:

1. POP from the stack top to the program counter.
2. POP from the stack top to program status word register.

Thus, after executing the interrupt service routine, processor returns to the main program with program status word register value unchanged.

7.1.1 CLASSIFICATION OF INTERRUPTS

Interrupt is classified into two types-*external and internal interrupt*. Peripheral devices via the micro-controller interrupt pins initiate external interrupts. Internal interrupts are activated by the peripherals of the microcontroller and by the execution of interrupt instructions.

7.1.2 INTERRUPT MASKABILITY

Interrupt. requests are classified into two categories--*maskable* and *non-maskable interrupts*. Micro-processors and microcontrollers have the option to disable the interrupts. These types of interrupts are called maskable interrupts. Other types of interrupts, which the processor cannot disable, are called non-maskable interrupts.

7.1.3 INTERRUPT VECTOR

The term 'interrupt vector' refers to the starting address of the interrupt service routine. The processor needs to determine the starting address of the interrupt service routine before it can provide service. The interrupt vector can be determined by one of the following methods:

Vectored *Interrupts* In this method, the starting address of the interrupt service routine is predefined when the microcontroller is designed.

These types of interrupts are called vectored interrupts.

Non-*vectored Interrupts* In this method, when the microcontroller receives the interrupt signal from the external devices, the processor completes the current instruction and sends a signal called INTA interrupt acknowledge (active low). After receiving the INTA signal, external hardware sends the interrupt vector to the microcontroller. These types of interrupts are called *non-vectored interrupts*.

7.2 III 8051 INTERRUPT STRUCTURE

The 8051 provides five vectored interrupt sources. There are two external interrupts, external interrupt (INTO) and external interrupt 1 (INTI) and three internal interrupts, 2 timer interrupt and a serial port interrupt. When an interrupt is generated, the contents of pogram status word register and program counter (PC) are pushed onto the stack. Vector address as shown in Table 7.1 is loaded in the program counter. As it branches to interrupt service routine, the interrupt flag of that particular interrupt is cleared by the hardware. In 8051, the interrupt flags are IEO, IEI, TFO, TF1, RI and TI. The interrupt service routine is ended with RETI instruction. The RETI instruction will POP from the stack top to the program counter and program status word register, and set the interrupt flag of that particular interrupt. The processor will start executing from where the main program was interrupted.

*Interrupt **service** routines*

TABLE 7.1

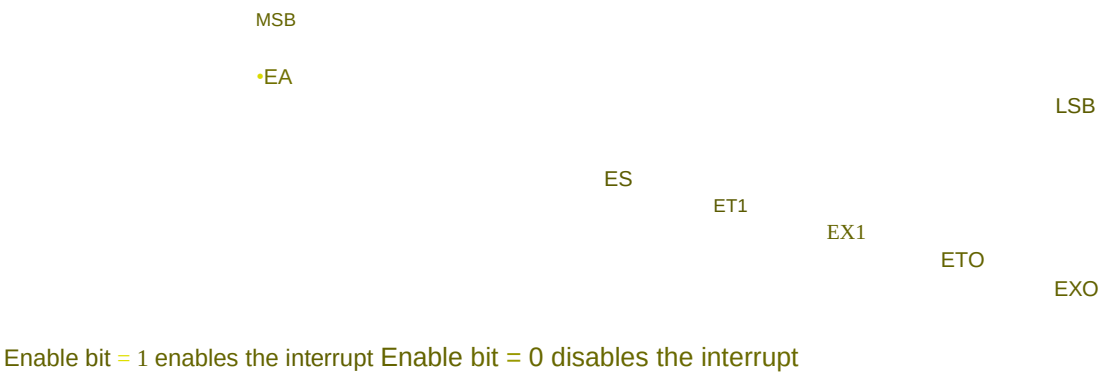
Interrupt Source	Vector Address	Priority within Level
External Interrupt 0	0003H	Highest
Timer 0 Interrupt	000BH	
External Interrupt 1	0013H	
Timer 1 Interrupt	001BH	
Serial Port Interrupt	0023H	Ldwest

Interrupt *Enable* Each of the interrupt sources can be individually enabled or disabled by setting or clearing a bit in Interrupt Enable (IE) register. Figure 7.2 shows the IE register in the 8051. This register also contains a global disable bit, which can be cleared to disable all the interrupts.

Interrupt Priority Priority of the interrupt can be programmed by setting or clearing a bit in the Interrupt Priority (IP) register. When more than one interrupt is enabled, the user first programs the interrupt priority register. Figure 7.3 shows the Interrupt Priority (IP) register in the 8051.

019

If two interrupt requests of different priority levels are received simultaneously, the request of highest priority level is serviced. If interrupt requests of same priority levels are received simultaneously, an internal polling sequence as shown in Table 7.1, determines which interrupt should be serviced first. Thus, within each priority level there is a second priority structure determined by the polling sequence.



Enable bit = 1 enables the interrupt Enable bit = 0 disables the interrupt

Symbol	Position	Function
EA	IE.7	If EA = 0, disables all interrupts If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit
	IE.6	reserved*
	IE.5	reserved
ES	IE.4	Serial Port Interrupt Enable Bit
ET1	IE.3	Timer 1 Overflow Interrupt Enable Bit
EX1	JE.2	External Interrupt Enable Bit
ETO	IE.1	Timer 0 Overflow Interrupt *Enable* Bit
EXO	IE.O	External Interrupt 0 Enable Bit

*These reserved bits are used in other MCS-51 devices.

Figure 7.2 IE (*Interrupt Enable) Register in 8051*

LSB quasini

MSB

PS

PT1

PX1

PTO

PXOX

Priority bit = 1 Assigns high priority
Priority bit = 0 Assigns low priority

Symbol	Position	Function
	IP.7	reserved*
	IP.6	reserved*
	IP.5	reserved*
PS	IP.4	Serial Port Interrupt Priority Bit
PT1	IP.3	Timer 1 Interrupt Priority Bit
PX1	IP.2	External Interrupt 1 Priority Bit
PTO	IP.1	Timer 0 Interrupt Priority Bit
PXO	IP.O	External Interrupt 0 Priority Bit

*These reserved bits are used in other MCS-51 devices.

Figure **7.3** (*Interrupt Priority) Register* in *8051*

SECTION REVIEW

1. The 8051 has 5 vectored interrupt

sources. True/False? 2. Differentiate vectored and non-vectored interrupts.

3. Differentiate maskable and non-maskable interrupts.

4. List the steps taken by the processor after receiving the interrupt.

5. List the functions of RETI instructions.

6. The 8051 contains two external interrupts and 3 internal interrupts. True/False?

7. External interrupt 0 has the highest priority. True/False?

8. When the 8051 receives interrupt through INTI pin, it branches to 9. EXO bit of IE register enables external interrupt 0. True/False?

10. The 8051 has priority levels in interrupts.

address.

7.3 III TIMERS AND COUNTERS

In a microcontroller, physical time is represented by the count value of a timer. By interpreting the count value of a timer properly, many timer applications can be realised. There are many applications that require a dedicated timer system, including

- Time reference
- Time delay creation
- Pulse width. period and frequency measurement
- Periodic interrupt generation (to remind the processor to perform routine tasks)
- Waveform generation

A basic timer consists of a register that can be read from or written to by the processor and is driven by some frequency source. The register is usually of 8 or 16 bit. The timer/counter clock source is either microcontrollers clock or an external clock. The value of the register can be read or a new value can be written

during the processor operation. When the counter overflows (in 8 bit-FF to 00), an interrupt is generated.

7.4 III 8051 TIMERS/COUNTERS

The 8051 has two timers. Timer 0 and Timer 1. Both are 16 bit timers/counters. Both timers consist of two 8 bit registers. Timer 0 consists of two 8 bit registers-THO and TLO and Timer 1 consists of two 8 bit registers, THI and TLI. Both share the timer control (TCON) register and timer mode register (TMOD). The timer registers are as shown in Fig. 7.4.

A timer can be used to create time delay, or as a counter to count the external events happening outside the micrcontroller that occurred within a time interval. The only difference between counter and timer is the source of the clock pulse **as** shown in Fig. 7.5 The C/T bit in the TMOD register decides the source of the clock for the timer. If C/T = 1, then timer is used as counter and gets the clock from outside the microcontroller, else if C/T = 0, then timer gets the clock pulses from the crystal.

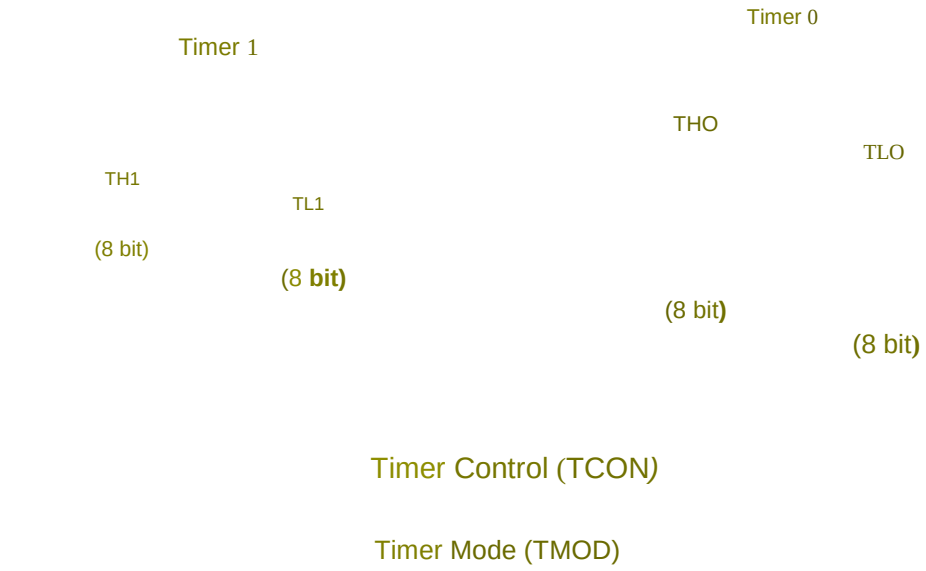
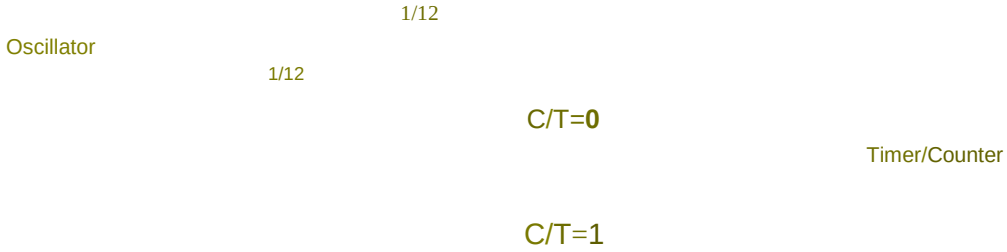


Figure 7.4 Timer registers



From Timer Control Logic

Pin Tx

Figure. 7.5 *Timer or Counter operation*

When used as a timer, the clock pulse is obtained from the oscillator through divide by 12 d circuit. When used as counter, pin TO (P3.4) supplies pulses to counter 0, and pin T1 (P3.5) supplies pulses to counter 1. If 12 MHz crystal is connected to the 8051, then the clock period will be equal to us. Registers are incremented after I us. In the counter function, the registers are incremented in response to a transition from 1 to 0, at external input pin P3.4 for counter 0 and P3.5 for counter 1. When the registers overflow from FFFF h to 0000 h, it sets a flag (TF flag) and generates an interrupt. Timer control register controls **the** timer/counter operation. Figure 7.6 shows TCON register. This register is an 8 bit register.

Bit O

Bit 7

TF1

TR1

TFO

TRO

IE1

IT1

IEO

ITO

TF1

Timer 1 overflow flag set when timer/counter overflows

TR1

Timer 1 run control bit

TFO

Timer 0 overflow flag set when timer/counter overflows

TRO

Timer 0 run control bit

IE1

Interrupt **1**

IT1

IEO

Timer 1 interrupt, ITI = 0 low level triggered, ITI = 1 falling edge trigger

Interrupt 0

ITO

Timer 0 interrupt, ITO = 0 low level triggered, ITO = 1 falling edge trigger

Figure 7.6 Timer *Control Register (TCON*)

TRO and TRI flags turn the timer ON or OFF. The upper 4 bit are used to store the TF and TR bits of both Timer 0 and Timer 1. The lower 4 bit are used for controlling the interrupt. TCON is a bit addressable register. Instructions SETB TCON.6 and CLR TCON.6 can be used to store 1 and 0 in TRI flag of TCON register. Figure 7.7 shows Timer mode control register (TMOD). The lower 4 bit are used for controlling Timer 0 and the upper 4 bit are used for controlling Timer 1. C/T selects timer or counter operation and M and MO select the mode.

Bit 7

Bit 0

GATE

CIT

M1

MO

GATE CΛT

M1

MO

Timer 1

Timer 0

GATE

CΛT

M1 MO Timer mode selection

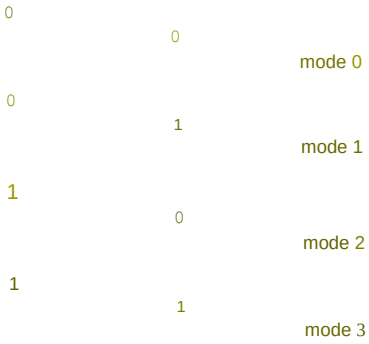
If GATE = 0, Timer 0 or 1 is enabled, If TR, or TR, control bit is set

If GATE = 1, Timer 0 or 1 is enabled, If INTO = 1 and TR2 = **1 or** INT1 = 1 and TR1 = 1

0 = limer mode, 1 = counter mode

M1

MO



EXAMPLE 7.1

Figure 7.7 Timer Mode *Control* Register *(TMOD)*

If an 8051-based system is controlled by the following crystal frequencies, find the timer clock frequency and its period.

11 12 MHz

2. 18 MHz

3. 11.0592 MHz

SOLUTION

1. Timer clock frequencies = 1/12 **x** 12 MHz = 1 MHz

2,

Period T = 1/1MHz = 1 µs.

Timer clock frequencies **= 1/12 x** 18 MHz = 1.5 MHz

Period T = 1/1.5 MHz = 0.667 µs.

3. Timer clock frequencies = **1/12 x** 11.0592 MHz = 921.6 KHz

Period T = 1/921.6 KHz = 01.085 µs

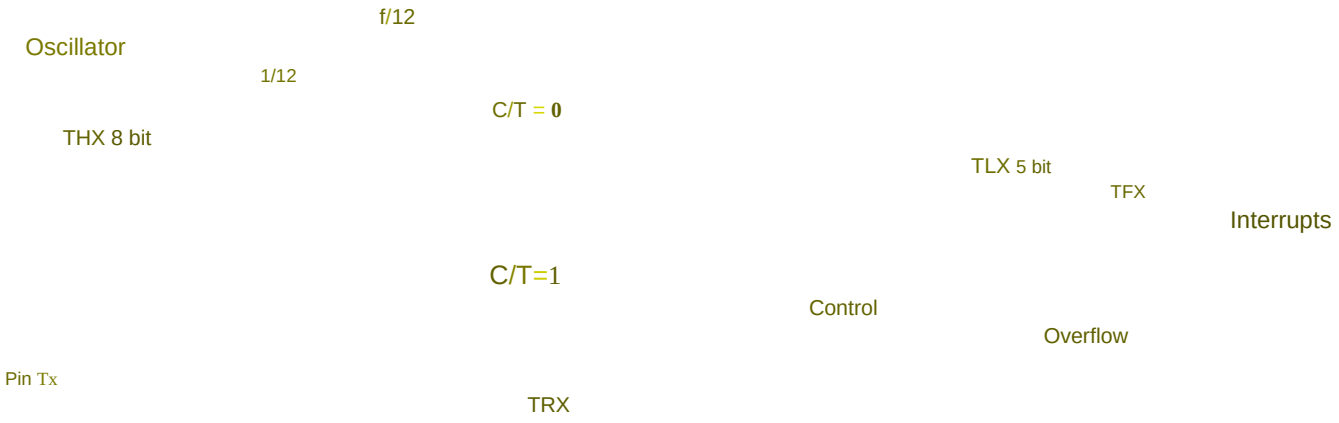
7.5 ||| TIMER/COUNTER OPERATION MODES
III

In addition to the timer or counter selection, both timer 0 and timer I have operating modes. Modes are selected by using Timer Mode Control Register (TMOD). Figure 7.7 shows TMOD register. The timers may operate in any one of the four modes that are determined by mode bits, MI and MO, in TMOD register.

MODE 0

1.13 bit

Figure 7.8 shows the mode 0 operation for timer 0 and timer 1. In this mode, timer register is configured as a 13 bit register. The 13 bit timer register consists of all 8 bit of THx and the lower 5 bit of TLX register. The upper 3 bit of TLX register are ignored. As the counter changes from all I's to O's, it sets the timer interrupt flag, TFx. The timer is enabled when TRX = 1, and either gate = 0 or INTx registers can hold values between 0000H to IFFFH in TH and TL. When the register reaches its maximum 1FFFH, it rolls over to 0000H i.e. when the register overflows, then it sets the timer interrupt flag (TF1 for timer 1 and TFO for timer 0). Example 7.2 gives the program to initialise timer 0 in mode 0)



Note: x = 0 Timer (

GATE

INTX

x = 1 Timer **1**

etion tem T

nnybetes

7

Figure 7.8 *Timer/Counter 0/1 in mode 0*

For mode 0, loading TLX and THX with values derived from the formula can create a specific time delay.

Time delay = [12 x [5110- Init Value]]/ Freq

Where Init Value = TLX + [256 × THx]

Where the values of THX and TLX are in decimal, the values must be converted to hexadecimal, then **it** must be loaded to THX and TLX registers.

EXAMPLE 7.2

III

Write 8051 program to initialise Timer o and Timer 1 in mode o. The external pin 12 (INTO) controls Timer o, and Timer 1 is fully controlled by TR1.

SOLUTION

```
MOV TMOD, #08h; Timer 0 and 1 in mode 0 and timer 0 is controlled by
                pin INT 0
SETB TR1                    : Start timer 1
SETB TRO                    ; Start timer 0
END
```

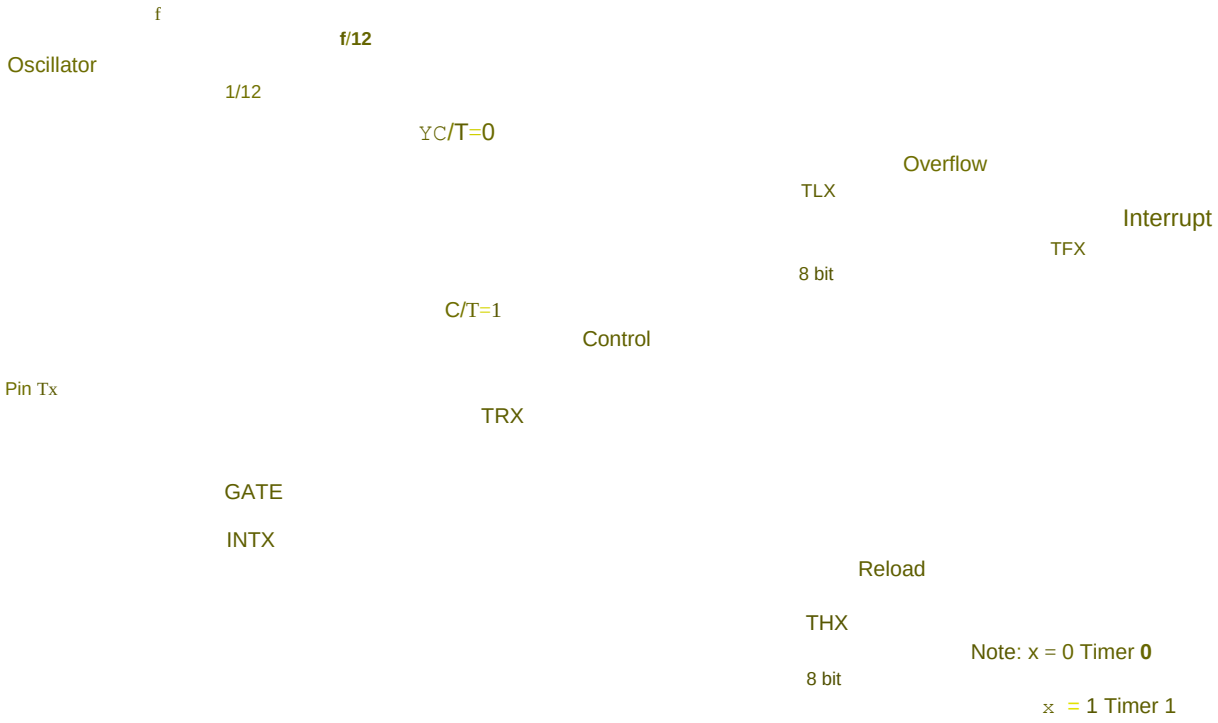
Mode 1 Mode 1 is the same as mode 0 except that the timer register uses all 16 bit. In this mode, THx and TLX are cascaded. For mode 1, loading TLX and THX with values derived from the formula can create a specific time delay.

Time delay = [12 x [65,536 - Init Value]]/ Freq

where Init Value = TLX + [256 × THx]

If the values of THX and TLX are in decimal, the values must be converted to Hexadecimal, then it must be loaded to THx and TLX registers.

Mode 2 The operation is same for timer 0 and timer i. In this mode, timer register is configured as TLX. When TLX overflows from FFh to 00H it, sets the flag TFX and it loads TLX with the contents of THx. The reload leaves THX unchanged. In this mode, timer 1or timer 0 supports the automatic reload operation. The timer control logic is same as mode 0 or mode I as shown in Fig. 7.9. Example 7.3 gives the program to initialise timer 0 in mode 2.



EXAMPLE 7.3

HI

Figure 7.9 Timer/*Counter 0/1 in* mode 2:8 *bit auto reload*

219mil

Write 8051 program to initialise Timer o in mode **2.** Load THo with preset value, 55H and load TLo with starter value, 55H.

```
SOLUTION

   MOV TMOD, #02h

   MOV THO, #55h
                        ; Load TMOD to operate timer 0 in mode 2
                        ; Load THO with preset value to be reloaded ;
                        Load TLO with starting value
                          Start timer 0
   MOV TLO, #55h

    SETB TRO

    END
```

bola somit all oil

pasupail lang

preset value

Mode 3 Timer 0 in mode 3 establishes TLO and THO as two separate registers as shown in Fig. 7.10. TLO uses Gate, TRO, INTO and TFO control bits of timer 0 and THO uses TRI and TF1 control bits of timer 1.

When TLO overflows from FF to 00, then it sets the timer flag TFO. If control bit TRI is set, THO receives the timer clock (oscillator divided by 12). When the counter THO overflows from FF to 00, then it sets the flag TF1.

Timer 1 may still be used in mode 0, mode 1 and mode 2, but it'neither interrupts the processor nor sets the flag. When timer 0 is in mode 3, timer I can be used for baud rate generation in serial communication.

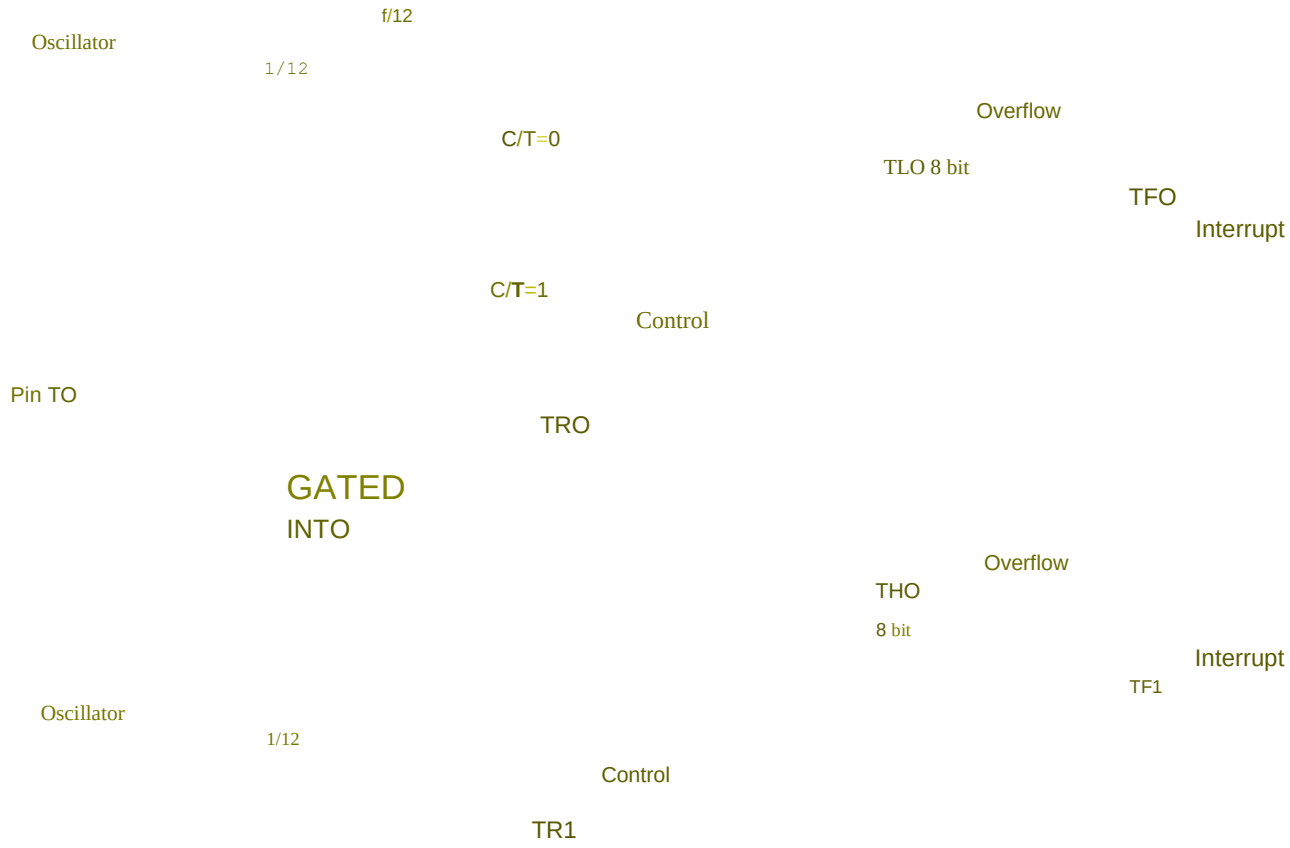


Figure 7.10 Timer/*Counter 0/1* in *mode 3*: *Two* 8 *bit counters*

SECTION REVIEW |||

1. List the applications that require 8051 timers.

2. The 8051 has

1

number of timers.

3. Timer 0 and I are 16 bit timer/counter.

True/False? **T**

4. Timer 0 and 1 share TCON register and TMOD register. True/False? T

5. If C/T 0, then the timer gets the clock pulses from outside the microcontroller. True/False?

6.

register controls the timer/counter operation.

7. What is the significance of IT1 bit of TCON register?

8. List the special function registers of timer/counter.

9. TCON is a bit addressable register. True/False?

10.

bit selects timer/counter operation.

11. Timer 1 is enabled if gate = 0 and TRI bit is set. True/False?

12. Timer 0 is enabled if gate = 1,

and

bit are set.

13. If an 8051 is operated with 18 MHz crystal frequency, then the timer clock frequency is

MHz.

14. If an 8051 system is operated with 11.0925 MHz crystal frequency, then the timer clock period is 1.085 μs. True/False?

15. In mode 0, timer register is configured as 13.

bit register.

16. In mode 1, timer register is configured as 16-bit register. True/False? T

17. In timer/counter, which mode supports automatic reload operation? 3

18. In mode 3, when THO register overflows,

19. Timer/counter operates in

5

timer flag is set.

TEI

number of modes.

20. Differentiate timer/counter operations.

7.6 II PROGRAMMING 8051 TIMERS

EXAMPLE 7.4

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a subroutine to create a time delay of 20 ms.

SOLUTION

Step 1: If C/T = 0, then clock source to timer 1 = oscillator /12

= 12 MHz/12 = 1 MHz.

Step 2: For 20 ms, 20 ms/1 µs = 20,000.

us

Step 3: Timer register (TH1 and TL1) value = (65,536-20,000)

(65,536-20,000) = 45,536 d B1E0 H.

1=

Algorithm to create a delay of 20 ms is as follows:

Step 1: Configure timer 1 to operate in mode 1 and choose oscillator/12 as the clock input (C/T = 0), Gate = 0 and TF1 = 0.

Step 2: Place value B1E0h, into timer 1 register and wait until the overflow flag is set to 1.

PROGRAM

Main Program

LCALL DELAY

Subroutine

```
DELAY: MOV TMOD, #10h
       CLR TF1
       CLR ET1
       MOV TH1, #Blh MOV TL1,
       #EOh SETB TR1 ho
                        ; Set up timer 1 in mode 1 ;
                        Clear timer 1 overflow flag
                        ; Clear timer 1 interrupt flag
                        ; Store upper byte of count
                        ; Store lower byte of count ; Enable
                        timer 1
WAIT: JNB TF1, WAIT; Wait until TF1 is set to 1
      RET
```

EXAMPLE 7.5

III

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a subroutine **to** create a time delay of 1 second.

SOLUTION

Step 1: If C/T = 0, then clock source to timer 1 = oscillator/12 = 12 MHz/12 = 1 MHz.

Step 2: For 50 ms, 50 ms/1 µs = 50,000.

Step 3: Timer register (TH1 and TL1) value =

d = 3CAF H.

Step 4: For 1 second, 1 second/50 ms times, we get 1 second.

(65,536-50,000)

15,536

20, if it is repeated 20 a

Algorithm to create a delay of 1 second is as follows:

Step 1: Configure timer 1 to operate in mode 1 and choose oscillator/12 *as* the the clock input (C/T = 0), Gate = 0 and TF1 = 0.

Step 2: Place value 3CAF into timer 1 register and wait until the overflow flag is set to *1*.

Step 3: Repeat step 20d (14 h) times.

PROGRAM

Main Program

Subroutine

```
DELAY1: MOV R1, #14h; Repeat the delay 20 times MOV TMOD, #10h
                                                ;Setup timer 1 in mode 1
        CLR ET1
LOOP2:  CLR TF1
LCALL DELAY1
        MOV TH1, #3C h
        MOV TL1, #0AF h
        SETB TRI
WAIT:   JNB TF1, WAIT
        DJNZ R1, LOOP2
```

```
RET
                                    ; Disable timer 1 interrupt
                                    ; Clear timer 1 overflow flag
                                    ; Place 15536 into timer register

                                    ; Enable timer 1 to operate
                                    ; Wait until TF1 is set
```

EXAMPLE 7.6

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a program to generate 2 KHz square wave on port 1.0 (pin **1)**.

SOLUTION

To generate a square wave on port 1.0 (pin 1) is as follows

Step 1: If C/T 0, then clock source to timer 1 = oscillator/12 = 12

MHz/12 = 1 MHz.

250 µs.

Step 2: Period of square wave: T = 1/f = 1/2KHz = 500 us. Step 3: Duration of high and low portion of the pulse is Step 4: Timer register value is: 250 ps/1 us = 250 and (65,536 - 250)

= 65,286 d = FF06 h is loaded to TH and TL register.

PROGRAM

```
MOV TMOD, #10h
CLR ET1
LOOP: CLR TF1
                                    ; Setup timer 1 in mode 1
                                    ; Disable timer 1 interrupt
                                    ; Clear timer 1 overflow flag
```

```
MOV TH1, #OFFh
MOV TL1, #06h
SETB TR1
WAIT: JNB TF1, WAIT
CLR TR1
CPL P1.0
SJMP LOOP
```



```
; Place 65,286 into timer register

; Enable timer 1 to operate
; Wait until TF1 is set
: Stop timer 1
; Complement P1.0 to get high and low
```

EXAMPLE 7.7

11

1

Assume an oscillator running at 12 MHz controls an 8051 micrcontroller. Write a program to generate

KHz

square wave on port 1.2 (pin 3) using timer o in auto reload mode.

4

SOLUTION

To generate a square wave on port 1.2 (pin 3) is as follows

Step 1: If C/T = 0, then clock source to timer 1 = oscillator/12 = 12 MHz/12 = 1 MHz.

Step 2: Period of square wave: T = 1/f = 1/4KHZ = 250 µs.

Step 3: Duration of high and low portion of the pulse = 125 µs.

Step 4: Timer register value is : 125 µs/1µs = 125 and (256 - 125)

131 d (83 h) is loaded to THO and TLO register .

PROGRAM

```
MOV TMOD, #02h
CLR TFO
CLR ETO
LOOP: MOV THO, #83h
MOV TLO, #83h
SETB TRO
                        ; Setup timer 0 in auto reload mode
                        ; Clear timer 0 overflow flag
                        ; Disable timer 0 interrupt
                        ; Place 83 h into timer register

                        ; Enable timer 0 to operate
BACK: JNB TFO, BACK; Wait until TFO is set
CLR TRO
CPL P1.2
CLR TFO
                        ; Stop timer 1
                        ;
                              Complement P1.2 to get high and low
                        ; Clear timer flagl
```

SJMP BACK

EXAMPLE 7.8

Assume an oscillator running at 11.0592 MHz controls an 8051 microcontroller. Write a program to generate 2 KHz square wave on port 1.3 (pin 4) using *timer* o interrupt.

SOLUTION

To generate a square wave on pin 4 of port 1.3 is as follows Step 1:
If C/T = 0, then clock source to timer 1 = oscillator/12
11.0592 MHz/12 = 0.9216 MHz.
Step 2: Period of square wave: T = *1*/f = 1/2KHz = 500 µs.
Step 3: Duration of high and low portion of the pulse = 250 µs. Step 4: Timer register value is: 250 µs/1.085 us = 230 and (65,536
230) = 65,306 d = FF1A h is loaded to TH and TL register.

PROGRAM

```
    ORG 0000
    LJMP MAIN
    ORG 000B
    CLR TRO
    CPL P1.3
    MOV TLO, #1Ah
    MOV THO, #OFFh
    RETI
    ORG 0030
                                    ; Bypass interrupt vector table
                                    ; ISR for timer 0
                                    ; Disable timer 0
                                    ; Complement P1.3 to get high and low


                                    ; Return from interrupt service routine

MAIN: MOV TMOD, #00000001B; Timer 0 in mode 1
```

```
        MOV TLO, #1Ah
        MOV THO, #OFFh
        CLR TFO
        MOV IE, #82h
    LOOP1:SETB TRO
    HERE: SJMP HERE
        AJMP LOOP1
```

; Place 65306 into timer register

; Clear TF flag

; Enable timer 0 interrupt

; Start timer 0

EXAMPLE 7.9

11

1

Assume an oscillator running at 11.0592 MHz controls an 8051 micrcontroller. Write a program to generate 1 KHz square wave from port 1.1 using timer o.

SOLUTION

To generate a square wave from port 1.1 is as follows

Step 1: If C/T 0 then clock source to timer 1

to timer 1 = oscillator /12 =

11.0592 MHz/12 = 0.9216 MHz.

Step 2: Period of square wave: T = 1/f = 1/1 KHz= 1 ms.

Step 3: Duration of high and low portion of the square wave = 500 µs.

Step us

4: Timer register value is: 500 us/1.085 µs = 461 (to nearest whole number) and (65,536 - 461)

loaded to TH and

TL register.

65,075 d = FF1A h is

C PROGRAM

```
#include <Intel\8051.h>
#define on 1
#define off 0
```

MA15049 J

```
bit Squarewavepin
                    P1.1; //Pin 1 of port 1
void delay1KHz(); //Delay on () returns nothing and takes nothing
main() {
                    // Start the program
  TMOD = 0x01;
  While (1){
```

```
  }
   Squarewavepin
                    // Timer 0: Gate = 0, C/T = 0, M1 = 0, M0 = 1; mode 1
                              // Do for ever
                     on;
   delay 1KHz ( );
   Squarewavepin off;
   delay 1KHz ( );
     }

void delay1KHz ( ){
   THO =OX FF;
```

```
    TLO = 0X1A;
    TRO on;
    While (!TFO);
    TRO = off;
    TFO off;
}
                                    // P1.1 set to 1
// Wait for on time // P1.1 set
to 0 // Wait for off time // While
                                                ()
     // Main ()
```

```
//Set TRO of TCON to run timer 0
//Wait for timer 0 to set the flag TFO? //Stop the
timer 0
//Clear the TFO

//Delay ( )
```

EXAMPLE 7.10

Assume an oscillator running at 11.0592 MHz controls an 8051 microcontroller. Write a program to generate 5 KHz square wave from port 1.5 using timer **1**.

SOLUTION

Solution to generate a square wave from port 1.5 is as follows

Step 1: If C/T = 0, then clock source to timer 1 = oscillator /12

11.0592 MHz/12 = 0.9216 MHz.

1/5 KHz = 0.2 ms.

Step 2: Period of square wave: T = 1/f = 1/5 KHz
=

Step 3: Duration of high and low portion of the square wave -

ms.

= 0.1

Step 4: Timer register value is: 100 us/1.085 µs = 92 (to nearest whole number) and (65,536 - 92) = 65,444 d = FF5A h is loaded to

TH and TL register.

C PROGRAM

```
#include <Intel\8051.h>
#define on 1
#define off 0
bit Squarewavepin P1.5;
void delay1KHz ( );

    main() {
                          //Pin 5 of portl
  //Delay on() returns nothing and takes nothing
                          // Start the program
   TMOD = 0100; // Timer0 : Gate = 0, C/T =0, M1 = 0, MO = 1; mode 1
       While (1) {

  Squarewavepin on; delay 5KHz (
        ); Squarewavepin = off;
        delay 5KHz ();
          }
      }
                                  // Do for ever
                                       // P1.5 set to 1
```

```
                                        // Wait for on time
                                             // P1.5 set to 0
                                        // Wait for off time
                                             // While ()
                                             // Main ()
void delay5KHz ( )
            {
          TH1 = 0XFF;
          TL1 = 0X5A;
          TR1 on;
          While (!TF1);

          TR1 = off;
          TF1 = off;
          }
                                             //Set TRO of TCON to run timer 1
   //Wait for timer 1 to set the flag TF1
                                             //Stop the timer 1
                                             //Clear the TF1
                                                  //Delay ()
```