Unit IV: Timer/Counter, Serial port and Interrupt Programming in 'C'

Programming 8051 Timers:

→ The 8051 has two timers: Timer 0 and Timer 1. They can be used either as timers or as event counters.

Basic registers of the timer:

→ Both Timer 0 and Timer 1 are 16-bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte.

Timer 0 registers:

→ The 16-bit register of Timer 0 is accessed as low byte and high byte.

→ The low byte register is called TL0 (Timer 0 low byte) and the high byte register is referred to as TH0 (Timer 0 high byte).

→ These registers can be accessed like any other register such as A, B, R0, R1, R2 etc.

Ex: MOV TL0, #4FH ; moves the value 4FH into TL0.

These registers can be read like any other register.

Ex: MOV R5, TH0 ; saves TH0 into R5.

Timer 1 registers:

→ Timer 1 is also 16 bits and its 16-bit register is split into two bytes, referred to as TL1 (Timer 1 low byte) and TH1 (Timer 1 high byte)

→ These registers are accessible in the same way as the registers of Timer 0.

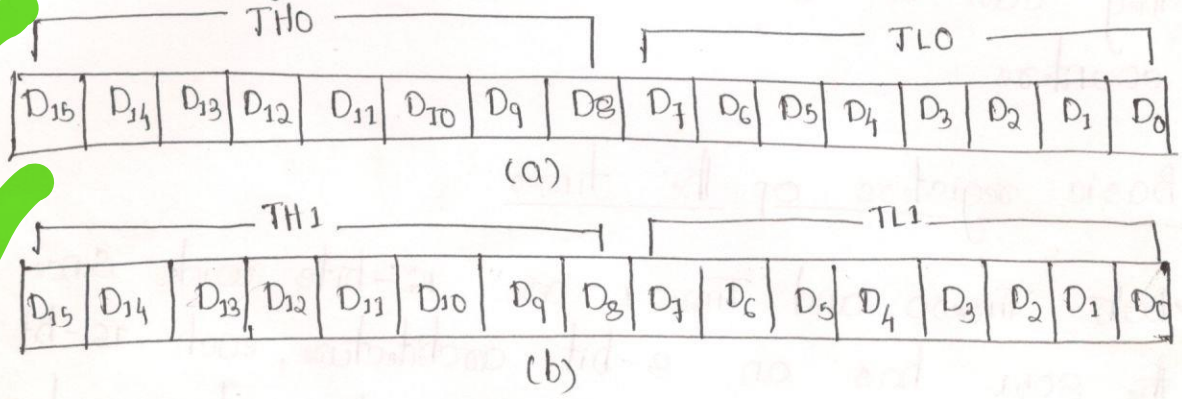
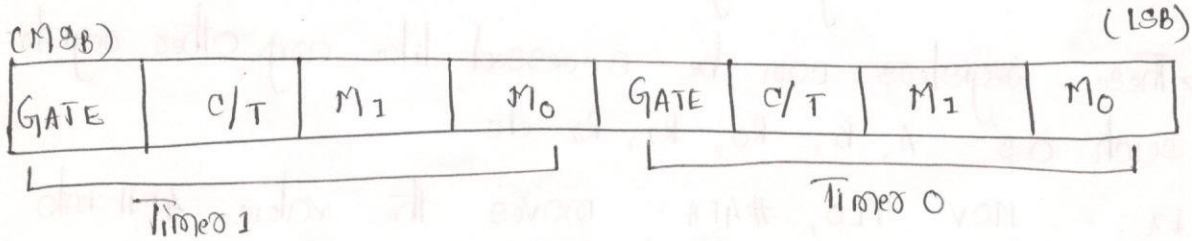


Fig. Timer 0 and Timer 1 registers.

TMOD register: (Timer mode register)

→ Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes.

→ TMOD is an 8-bit register in which the lower 4-bits are set aside for Timer 0 and the upper 4-bits for Timer 1.



GATE: hardware/software means to start or stop timer (Timer start bits)

GATE=0: TR bits (TR0) and (TR1) are used to

GATE=1: Hardware way of starting and stopping the timer by an external source ($\overline{INT0}$ and $\overline{INT1}$)

C/T: This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter.

C/T = 0: selects Timer

= 1: selects Counter

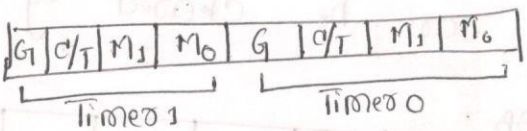
M_1 M_0: M_0 and M_1 select the timer mode.

M_1	M_0	Mode	Operating Mode
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto reload
1	1	3	split timer mode.

→ Mode 1 and mode 2 are the most widely used timer modes.

(1) Ex: Find the values of TMOD to operate as timers in the following modes.

(a) Mode 1 Timer 1



TMOD = 0001 0000 = 10H

The gate control bit and C/T are mode 0, and the unused timer (Timer 0 bit is also 0)

(b) Mode 2 Timer 0, Mode 2 Timer 1

TMOD = 0010 0010 = 22H

(c) Mode 0 Timer 1

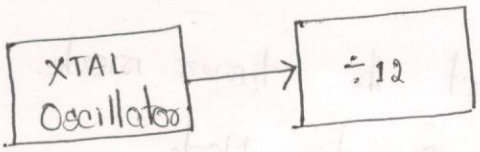
TMOD = 0000 0000 = 00H

(2) Find the timer's clock frequency and its period for various 8051 based systems with the following crystal frequencies.

(a) 12MHz

(b) 16 MHz

(c) 11.0592 MHz



(a) $1/12 \times 12\,MHz = 1\,MHz$ and $T = 1/1\,MHz = 1\,\mu s$

(b) $1/12 \times 16\,MHz = 1.333\,MHz$ and $T = 1/1.333\,MHz = 0.75\,\mu s$

(c) $1/12 \times 11.0592\,MHz = 921.6\,kHz$ and $T = 1/921.6\,kHz = 1.085\,\mu s$

TCON register: (Timer control register)

→ The timers are started by using instructions to set timer start bits TR0 and TR1, which are called timer run control bits.

→ They can be cleared by clearing these bits.

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

→ When a timer counts to its maximum value, it sets a flag TF0 or TF1.

→ While TMOD controls the timer modes, TCON register controls the timer/counter operations.

→ The lower 4-bits of TCON cater to interrupt

timer operations.

TF1 : Timer 1 overflow flag
TR1 : Timer 1 run control bit
TF0 : Timer 0 overflow flag
TR0 : Timer 0 run control bit.

Mode 1 Programming:

→ The following are the characteristics and operations of mode 1.

1. It is a 16-bit timer; therefore it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by making TR0 = 1 for Timer 0 and TR1 = 1 for Timer1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag).
   This timer flag can be monitored. When this timer flag is raised, the timer has to be stopped by making TR0 = 0 or TR1 = 0 for Timer 0 and Timer 1 respectively.
   Each timer has its own timer flag. TF0 for Timer 0 and TF1 for Timer 1.
4. After the timer reaches its limit and roll over, in order to repeat the process the registers TH and

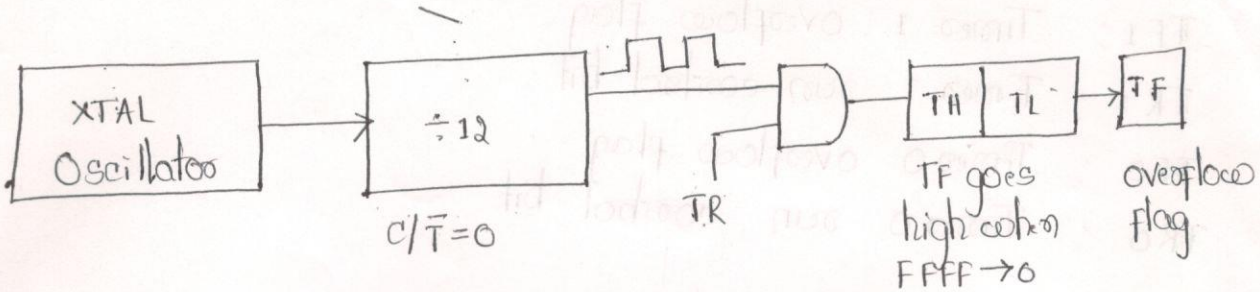


Fig. Timer in mode 1

Steps to program Timers in mode 1

(1) → Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.

(2) → Load registers TL and TH with initial count values

(3) → start the timer

(4) → Keep monitoring the timer flag (TF) with while (TF==0); to see if it is raised. Get out of the loop when TF becomes high.

(5) → stop the timer.

(6) → clear the TF flag for the next round

(7) → Go back to step 2 to load TH and TL again.

→ Fig. below shows the formula for delay calculation using mode 1 (16-bit) of the timer for a crystal frequency of XTAL = 11.0592 MHz.

(a) in hex

(FFFF − YYXX + 1) × 1.085 μs

where YYXX are TH, TL initial values respectively. Notice that

(b) in decimal

Convert YYXX values of the TH, TL register to decimal to get a NNNN decimal number, then

Ex: 1) Calculate the delay generated by the Timer0 given TH0 = 0FFH, TL0 = 0F2H. Assume that XTAL = 11.0592 MHz Timer 0 is operating in mode 1

solution:

i) Timer works with a clock foreguency of

$$f = \frac{1}{12} \times XTAL$$

$$= \frac{1}{12} \times 11.0592\,MHz$$

$$= 0.9216 \times 10^{+6}$$

$$f = 0.9216\,MHz$$

$$T = \frac{1}{f} = \frac{1}{0.9216 \times 10^{6}}$$

$$= 1.085 \times 10^{-6}\,sec$$

$$T = 1.085\,\mu sec$$

The delay is calculated as

FFFF − FFF2 = 0DH = 13d + 1 = 14d

Delay = 14 × 1.085 μsec = 15.19 μsec

max

Ex: 2) Calculate the delay generated by Timer 1 in mode 1 given TH1 = 00H and TL1 = 00H. Assume that XTAL = 11.0592MHz.

i) f = 1/12 × 11.0592MHz = 921.6 kHz

ii) T = 1/f = 1/921.6kHz = 1.085 μsec

iii) Delay = FFFF − 0000 = FFFF + 1 = 65,535 + 1 = 65,536

= 65,536 × 1.085 μsec

= 71.10ms

M1

Ex:3) Calculate the delay generated by Timer 0 in mode 0 Assume XTAL = 22 MHz. TH0 = 0FH and TL0 = 00H.

i) $f = \frac{1}{12} \times 22\,MHz = 1.833\,MHz$

ii) $T = \frac{1}{f} = 0.546\,\mu s$

iii) Mode-0 is 13-bit timer with TH storing the upper byte and lower five bits of TL storing the lower byte.

→ With a 13-bit timer, the maximum value of timer 1 register will be 1FFFH i.e., TH register is used fully and the lower 5 bits of the TL register i.e.,

0 0 0 0 0 0 0 0 ; 0 0 0 0 0 = 0000H

1 1 1 1 1 1 1 1 ; 1 1 1 1 1 = 1FFFH

TH (8-bits) ; TL (5 bits)

→ Here TH0 = 0FH and TL0 = 00H and the effective 13-bit number will be 0 0 0 0 1 1 1 1 | 0 0 0 0 0 = 01E0H

The delay is calculated as

1FFFH − 01E0H = 1E1FH = 7711

Delay = (7711 + 1)T = 7712 × 0.546 μs = 4.212 ms

Finding values to be loaded into the timer.

<u>Finding values to be loaded into the timer.</u>

→ Assuming that we know the amount of timer delay we need, following steps has to be followed to ... the values to be loaded into the timer

1. Divide the desired time delay by 1.085μs
2. Perform 65,536 − n, where 'n' is the decimal value we got in step 1
3. Convert the result of step 2 to hex, where yyzz is the initial hex value to be loaded into the timer's registers
4. Set TL = zz and TH = yy.

Ex.4) Assume that XTAL = 11.0592MHz. What value do we need to load into the timer's registers, if we want to have a time delay of 5ms

Sol: Desired time delay = 5ms

i) Divide it by 1.085μs: $\frac{5\text{ms}}{1.085\mu\text{s}}$ = 4608 clocks

ii) 65,536 − 4608 = 60,928

iii) 60928d = $\frac{EE}{YY}\ \frac{00}{XX}$ H

iv) Set TH = EE TL = 00.

Ex:5) Assume that XTAL = 22MHz. What value do we need to load into the timer's registers (mode 1), if we want to have a time delay of 15ms.

Sol: Desired time delay = 15ms

i) Divide it by 0.546μs: $\frac{15\text{ms}}{0.546\mu\text{s}}$ = 27,472 clocks

iii) $38064d = \underset{TH}{94}\ \underset{TL}{B0h}$

iv) TH = 94, TL = B0

Ex:6) Find the values to be loaded into the timer registers to generate a time delay of 3ms using timer 1 in mode 1. Assume XTAL = 22MHz

Sol: Desired time delay = 3ms

i) $\frac{3ms}{0.546\mu s} = 5495d$

ii) 65,536 − 5495 = 60041d

iii) $60041d = \underset{TH}{EA}\ \underset{TL}{89}$

iv) TH = EA , TL = 89

Generating a large time delay:

→ The size of the time delay depends on two factors

(a) Crystal frequency

(b) Timer's 16-bit register in mode 1 (also on 'C' compiler)

→ Both of these factors are beyond the control of the 8051 programmer.

→ Below example shows how to achieve large time delays

Ex:5) Generate a delay of 1ms using Timer 1 in mode 1.
Assume XTAL = 22 MHz.

Sol: i) $f = \frac{1}{12} \times XTAL = \frac{1}{12} \times 22\,MHz = 1.833\,MHz$

ii) $T = \frac{1}{f} = \frac{1}{1.833\,MHz} = 0.546\,\mu s$

Maximum delay is possible when both TH and TL stores values of 00

Then FFFF − 0000 = FFFF + 1 = 65,535 + 1 = 65,536

Delay = 65,536 × 0.546 μs = 35,782 μs = 35.75ms

→ The maximum possible delay is 35.75ms

→ To obtain 1 second delay, $n = \frac{1\,sec}{35.75ms}$

i.e., $n = 28$.

If we repeat 35.75ms 28 times we obtain 1 second delay

Mode 0:

→ Mode 0 is exactly like mode 1 except that it is a 13-bit timer instead of 16-bit

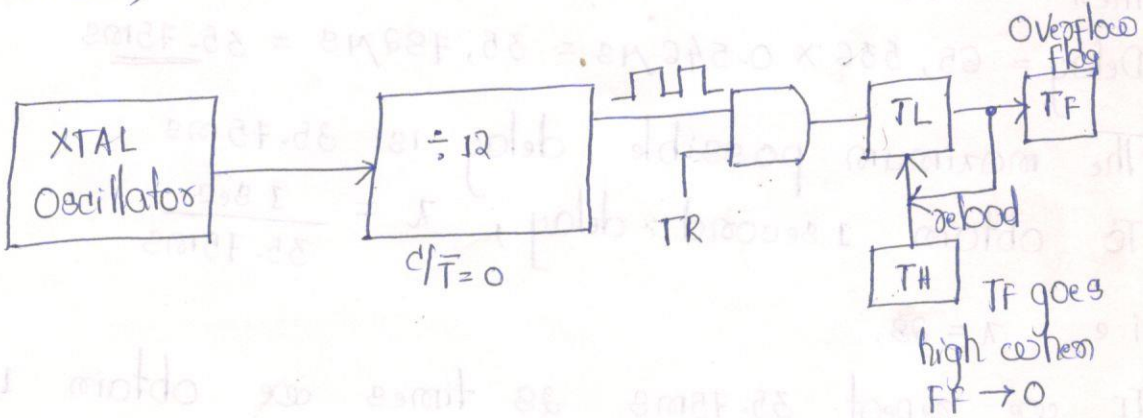
→ The 13-bit counter can hold values between 0000 to

maximum of 1FFFH, it rolls over to 0000 and TF is raised.

Mode 2 programming:

The following are the characteristics and operations of mode 2.

1. It is an 8-bit timer, therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started by making TR=1. (TR1 – Timer 1 TR0 – Timer 0)
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH When it rolls over from FFH to 00, it sets high the TF (timer flag). (TF0 for Timer 0 and TF1 for Timer 1)



4. When the TL register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register.

To repeat the process TF is cleared without any

→ This makes mode 2 auto reload in contrast with mode 1 in which the programmer has to reload TH and TL.

Features of mode-2

→ 8-bit timer

→ Auto reloading capability

→ TH is loaded with initial count and a copy of it is given to TL. This reloading leaves TH unchanged still holding a copy of the original value.

→ Finds application in setting the baud rate in serial communication.

Steps to program in mode 2:

→ To generate a time delay using the timer's mode 2 take the following steps.

1. load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used, and select the timer mode (mode 2)
2. Load TH registers with the initial count value.
3. start the timer
4. Keep monitoring the timer flag (TF) with while (TF==0); instruction to see whether it is raised. Get out of the loop when TF goes high
5. Clear the TF flag.
6. Go back to step 4, since mode 2 is auto-reload.

Ex: 6) Calculate the time delay generated by Timer 1 in mode 2. Given TH1 = 05 and XTAL = 11.0592 MHz

Sol^{n}: i) $f = \frac{1}{12} \times XTAL = 921.6\,kHz$

ii) $T = \frac{1}{f} = 1.085\,\mu sec$

iii) TH = 05 i.e., $(256 - 05) \times 1.085\,\mu s = 251 \times 1.085\,\mu s = 272.33\,\mu s$

Counter Programming:

→ When timer/counter is used as a timer, the 8051's crystal is used as the source of the frequency.

→ When it is used as a counter, however, it is a pulse outside the 8051 that increments the TH, TL registers.

→ In counter mode, notice that the TMOD and TH, TL registers are the same as for the timer.

→ C/$\overline{T}$ = 1 for counter operation, and the source of the frequency is provided externally through pins T0 (Timer 0 input) and T1 (Timer 1 input)

These pins belong to ports.

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/Counter 0 external input
15	P3.5	T1	Timer/Counter 1 external input

→ For Timer 0 ... C/$\overline{T}$ ... pin P3.4 provides the

clock pulse coming from that pin.

→ For Timer 1, when $C/\overline{T} = 1$ each clock pulse coming in from $P_{3.5}$ makes the counter count up.

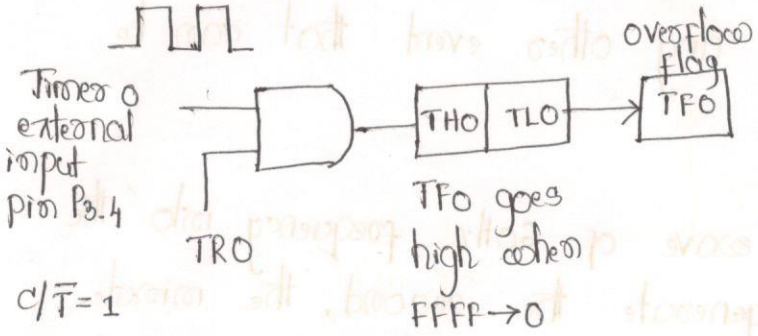


Fig. a. Timer 0 with external input (Mode 1)

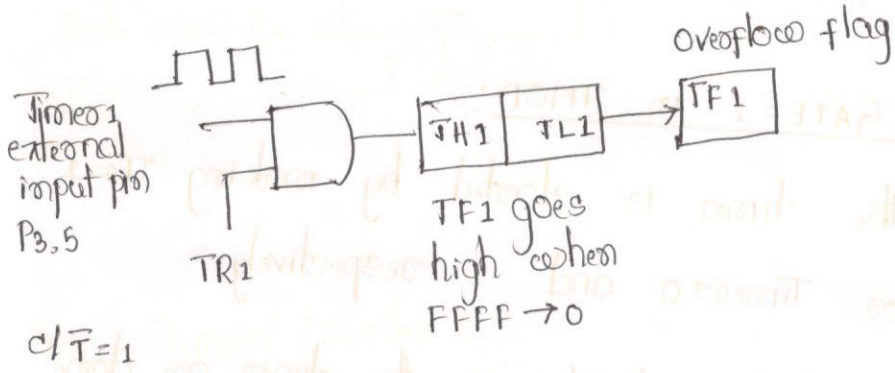


Fig. b. Timer 1 with external input (Mode 1)

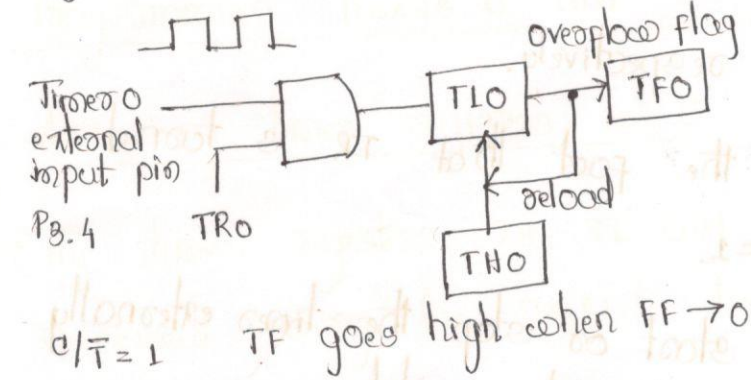
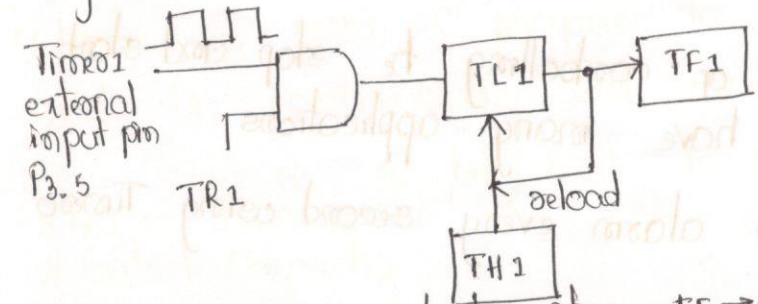


Fig. c. Timer 0 with external input (Mode 2)



Applications of counter:

→ Clock pulses fed to the external input pin (P3.4 or P3.5) could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

→ An external square wave of 60Hz frequency into the timer is fed to generate the second, the minute and the hour out of this input frequency and display the result on an LCD.

The case of GATE=1 in TMOD:

→ When GATE=0, the timer is started by making TR0=1 and TR1 = 1 for Timers 0 and 1 respectively.

→ If GATE = 1, the start and stop of the timer are done externally through pins P3.2 (INT0) and P3.3 (INT1) for Timers 0 and 1 respectively.

→ This is inspite of the fact that TR is turned on by making TR=1.

→ This allows us to start or stop the timer externally at any point via a simple switch.

→ This hardware way of controlling the stop and start of the timer can have many applications

Ex: Turn ON/OFF the alarm every second using Timer0

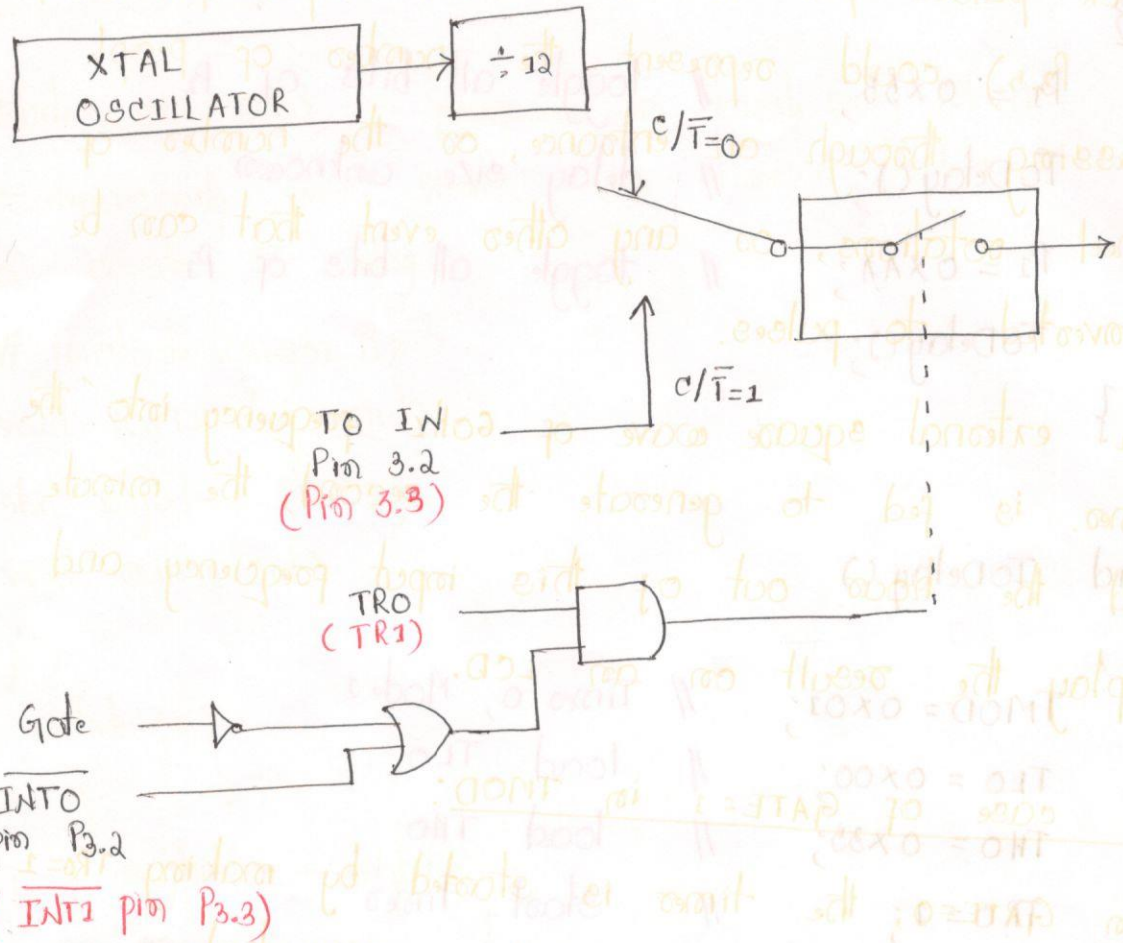


Fig. Timer/ Counter 0 (Timer/ Counter 1) with GATE = 1

Programming Timers 0 and 1 in 8051 C

Accessing timer registers in 'C'.

→ The timer registers TH, TL and TMOD can be directly accessed using the reg51.h header file.

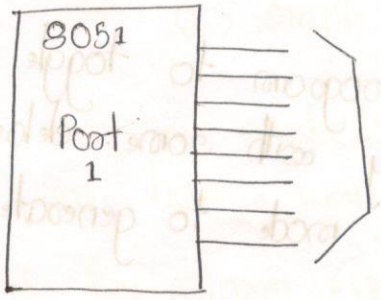
Ex:
1) Write an 8051 'C' program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.
XTAL = 11.0592 MHz

Sol^n:
```
#include <reg51.h>
```

```
    }
    while (1)                    // repeat forever
    {
        P1 = 0x55;               // toggle all bits of P1
        ToDelay();               // delay size unknown
        P1 = 0xAA;               // toggle all bits of P1
        ToDelay();
    }
}
void ToDelay()
{
    TMOD = 0x01;                 // Timer 0, Mode 1
    TL0 = 0x00;                  // load TL0
    TH0 = 0x35;                  // load TH0
    TR0 = 1;                     // start timer
    while (TF0 == 0);            // wait for TF0 to roll over
    TR0 = 0;                     // turn off T0
    TF0 = 0;                     // clear TF0
}
```

FFFF − 3500 = CAFFH = 51967 + 1 = 51969

51968 × 1.085 μs = 56.384 ms



Timers 0 and 1 delay using mode 1 (16-bit non auto-reload)

1) Write an 8051 'c' program to toggle only bit $P_{1.5}$ continuously every 50ms. Use Timer 0, mode 1 (16-bit) to create the delay.

```
# include <reg51.h>
void TOM1Delay (void);
sbit mybit = P1^5;
void main (void)
{
  while (1)
    {
      mybit = ~mybit;        // Toggle P1.5
      TOM1Delay ();
    }
}

void TOM1Delay (void)
{
      TMOD = 0x01;           // Timer 0, mode 1 (16-bit)
      TL0 = 0xFB;            // Load TL0
      TH0 = 0x4B;            // Load TH0
      TR0 = 1;               // start Timer
      while (TF0 == 0);      // wait for TF0 to roll over
      TR0 = 0;               // turn off T0
      TF0 = 0;               // clear TF0
}
```

Calculation to show the count needed to load into timer registers.

Required delay:

FFFF − 4BFEH = B401H = 46082 + 1 = 46082

Timer delay = 46082 × 1.085μs = 50ms

Calculation to show the amount of count to be loaded into the timer register.

i) 50ms: required delay

$$\frac{50ms}{1.085\mu s} = 46082$$

ii) 65,536 − 46082 = 19454

iii) 19454d = 4BFEH

iv) TH = 4B, TL = FE

2) Write an 8051 'C' program to toggle only pin P1.5 continuously every 250ms. Use Timer 0, mode 2 (8-bit auto reload to create the delay)

```
#include <reg51.h>
void T0M2Delay(void);
sbit mybit = P1^5;
void main(void)
{
    unsigned char x, y;
    while (1)
    {
      mybit = ~mybit;                    // Toggle P1.5
      for (x=0; x<250; x++)
      for (y=0; y<40; y++)
      T0M2Delay();
```

```
void T0M2Delay (void)
{
    TMOD = 0x02;          // Timer 0, mode 2 (8-bit auto reload)
    TH0 = -23;(EQ)        // load TH0
    TR0 = 1;              // start timer
    while (TF0 == 0);     // wait for TF0 to roll over
    TR0 = 0;              // turn off T0
    TF0 = 0;              // clear TF0
}
```

FF − EQ = 1CH = 22d

22+1 = 23d

23 × 1.085μs = 25μs

25μs × 250 × 40 / 100 loop = 250ms

3) Write an 8051 'C' program to create a frequency of 2500 Hz on pin P2.7. Use Timer 1, mode 2 to create the delay.

Sol:

```
#include <reg51.h>
void T1M2Delay (void);
sbit mybit = P2^7;
void main (void)
{
    unsigned char i;
    while (1)
    {
        mybit = ~mybit;
        T1M2Delay();
```

```
void T1M2Delay(void)
{
    TMOD = 0x20;
    TH1 = 48H
    TR1 = 1;
    while (TF1 == 0);
    TR1 = 0;
    TF1 = 0;
}
```

Calculation to obtain the count to be loaded into timer registers

Desired delay $= \frac{1}{F} = \frac{1}{2500Hz} = 400\mu s$

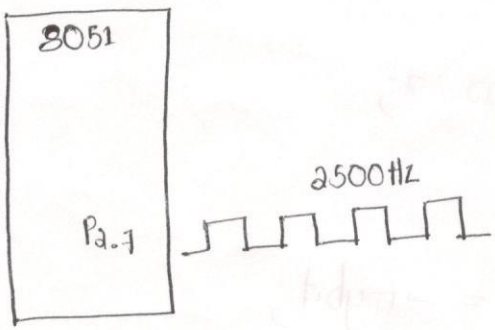
It is a square wave i.e., $T_{ON} + T_{OFF} = 400\mu s$

$\therefore T_{ON} = T_{OFF} = 400\mu s/2 = \underline{200\mu s}$

i) $\frac{200\mu s}{1.085\mu s} = 184$ clocks

ii) ~~1848 or 188~~ : $256 - 184 = 72d$

iii) $72d = \frac{48H}{TH}$



C programming of timers 0 and 1 as counters

→ A timer can be used as a counter if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source.

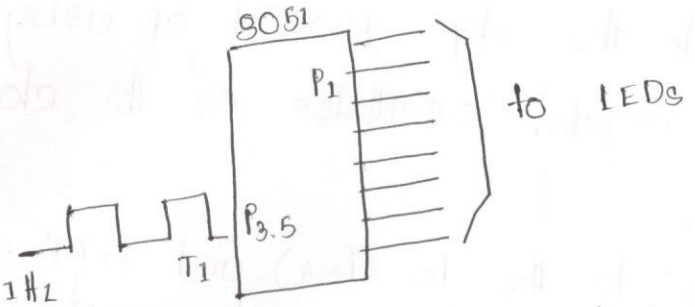
→ By feeding pulses to the T0 ($P_{3.4}$) and T1 ($P_{3.5}$) pins, we turn Timer 0 and Timer 1 into counter 0 and counter 1 respectively.

Ex: 1) Assume that a 1-Hz external clock is being fed into pin T1 ($P_{3.5}$). Write a 'C' program for counter 1 in mode 2 (8-bit auto reload) to count up and display the state of the TL1 count on P1. Start the count at 00H.

```
#include <reg51.h>
sbit T1 = P3^5;
void main (void)
{
   T1 = 1;                  // make T1 an input
   TMOD = 0x60;             // Counter 1 mode 2
   TH1 = 0;                 // set count to 0

   while (1)                // repeat forever
    {
      Do
        {
          TR1 = 1;          // start timer
          P1 = TL1;         // place value on pins
        }
      while (TF1 == 0);     // wait here
```

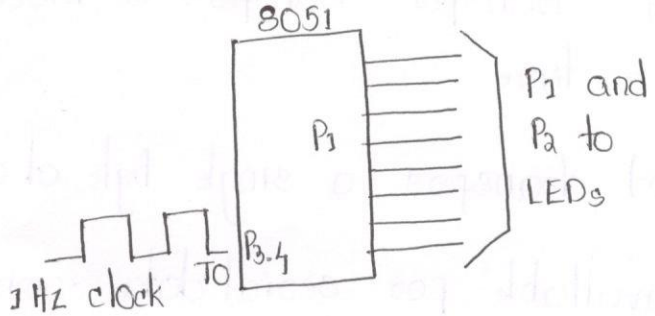
→ P_1 is connected to 8 leds. T_1 is connected to a 1-Hz external clock.



Ex: 2) Assume that a 1-Hz external clock is being fed into pin T0 ($P_{3.4}$). Write a 'C' program for counter 0 in mode 1 (16-bit) to count the pulses and display the TH0 and TL0 registers on P_2 and P_1 respectively.

Sol:

```
#include <reg51.h>
void main(void)
{
    T0 = 1;              // make T0 an input
    TMOD = 0x05;         // counter 0 mode 1
    TL0 = 0;             // set count to 0
    TH0 = 0;             // set count to 0
    while(1)             // repeat forever
    {
      do
        {
          TR0 = 1;       // start timer
          P1 = TL0       // place value on pins
          P2 = TH0;      //  "   "   "   "
        }
      while (TF0 == 0);  // wait here
      TR0 = 0;           // stop timer
      TF0 = 0;           // clear overflow flag
    }
}
```

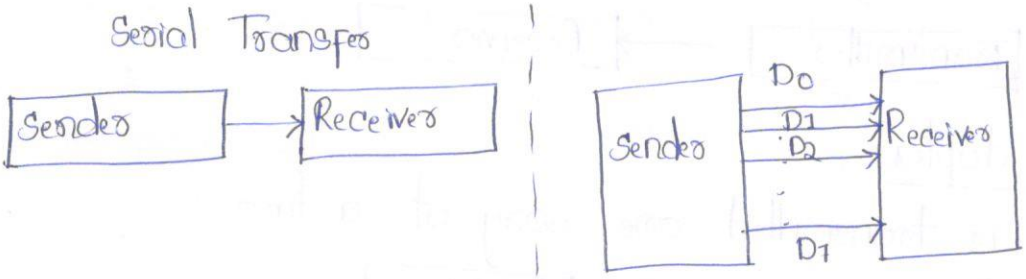


Serial Port Programming

Basics of Serial Communication:

→ Serial data transmission is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart.

→ Fig. below shows serial versus parallel data transfer



→ Serial communication uses a single data line instead of the 8-bit data line of parallel communication & hence it is cheaper.

→ For serial data transmission, byte must be converted into serial bits using a parallel-in-serial out shift register; then it can be transmitted over a single data line.

→ At the receiving end there must be a serial in parallel out shift register to receive the serial data and pack them into a byte.

→ Serial data communication uses two methods
i) asynchronous

→ The synchronous data transfer transfers a block of data (characters) at a time

→ Asynchronous method transfers a single byte at a time

→ Special chips are available for serial data communi-cations referred to as UART (Universal Asynchronous Receiver Transmitter) and USART (Universal Synchronous Asynchronous Receiver – Transmitter)

→ The 8051 chip has built in UART.

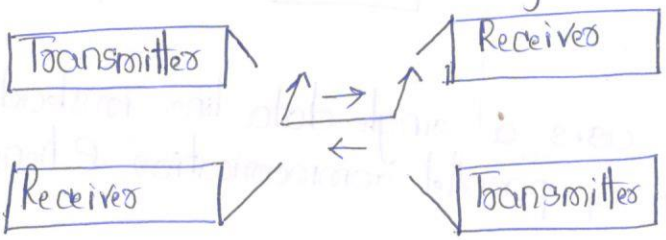
Transmission types

i) Simplex:

→ Computer only sends data. Ex: Printer.

ii) Half duplex:

→ Data is transmitted one way at a time



iii) Full duplex:

→ Data can go both ways at the same time, and it requires two wire conductors for the data lines, one for transmis-sion and one for reception in order to transfer and receive data simultaneously

Asynchronous serial communication and data framing

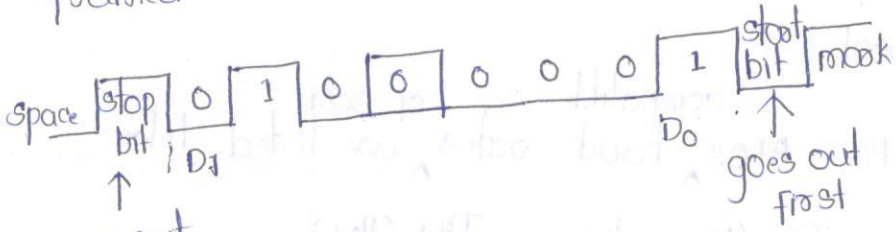
→ is widely used for character oriented transmissions, while block-oriented data transfers use the synchronous method.

→ In asynchronous method each character is placed between start and stop bits called framing

→ The start bit is always one bit, but the stop bit can be one or two bits.

→ The start bit is always a 0 (low) and the stop bit(s) is 1 (high)

→ Fig. below shows ASCII character 'A' (0100 0001) framed between start and stop bit



→ When there is no transfer, the signal is 1 (high), which is referred to as mark. The 0 (low) is referred to as space

→ The transmission begins with start bit followed by D_0, which is the LSB, then the rest of the bits until the MSB (D_7) and finally one stop bit indicating the end of the character 'A'.

Data transfer rate:

→ The rate of data transfer in serial data communication is stated in bps (bits per second) or baud rate.

RS232 standard:

RxD and TxD pins:

→ The 8051 has two pins that are used specifically for transferring and receiving data serially

→ These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1)

→ These pins are TTL compatible

8051 Serial port programming

→ To allow data transfer between the PC and an 8051 system without any error, we must make sure that the baud rate of the 8051 matches the baud rate of PC's COM port.

→ Some of the PC BIOS compatible baud rates of 8051 are listed below

Baud rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Table 1. Timer 1 TH1 values for various baud rates

→ Baud rate can be examined by going to the Windows HyperTerminal program and clicking on the Communication Settings option

Baud rate in the 8051:

→ The 8051 transfers and receives data serially at

→ The baud rate in the 8051 is programmable & is done with the help of Timer 1

→ Relationship between the crystal frequency and the baud rate in 8051 is

– 8051 divides the crystal frequency by 12 to get the machine cycle frequency.

– Assuming XTAL = 11.0592MHz; machine cycle frequency

$$f = \frac{1}{12} \times 11.0592\text{MHz} = 921.6\text{kHz}$$

– The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6kHz by 32 once more before it is used by Timer 1 to set the baud rate.

i.e., $\frac{921.6\text{kHz}}{32} = 28,800\text{Hz}$

– This value is used to find the Timer 1 value to set the baud rate.

– When Timer 1 is used to set the baud rate it must be programmed in mode 2, that is 8-bit auto-reload.

Ex: 1) With XTAL = 11.0592MHz, find the TH1 value needed to have the following baud rates
(a) 9600 (b) 2400 (c) 1200

Sol^n: With XTAL = 11.0592 MHz, we have:

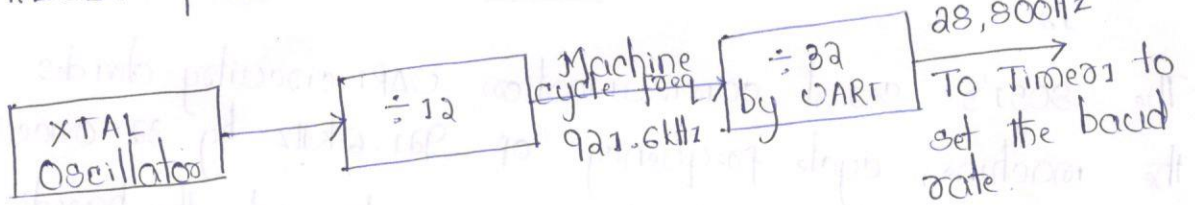
i) $f = \frac{1}{12} \times \text{XTAL} = \frac{1}{12} \times 11.0592\text{MHz} = 921.6\text{kHz}$

(a) 28,800/3 = 9600 where -3 = FD (hex) is loaded into TH1

(b) 28,800/12 = 2400 where -12 = F4 (hex) is loaded into TH1

(c) 28,800/24 = 1200 where -24 = E8 (hex) is loaded into TH1

Note: 1/12th the crystal frequency divided by 32 is the default value upon activation of the 8051 RESET pin.



SBUF register:

→ SBUF is an 8-bit register used solely for serial communication in the 8051.

→ For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register.

→ Similarly, SBUF holds the byte of data when it is received by the 8051's RxD line.

→ SBUF can be accessed like any other register in the 8051.

→ The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD pin.

→ similarly, when the bits are received serially via RxD, it converts it by eliminating the stop and

SCON (Serial Control) Register:

→ The SCON register is an 8-bit register used to program the start bit, stop bit and data bits of data framing.

→ Fig. below shows various bits of the SCON register

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

→ SM_0 and SM_1 determine the framing of data by specifying the number of bits per character, and the start and stop bits

SM_0	SM_1	
0	0	Serial Mode 0
0	1	Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

→ Except Mode 1, remaining modes are rarely used.

→ When Mode 1 is chosen, the data framing is 8 bits, 1 stop bit and 1 start bit, which makes it compatible with the COM port of IBM/compatible PCs.

→ Serial mode 1 allows the baud rate to be variable and is set by Timer 1 of the 8051.

→ In Serial Mode 1, for each character a total of 10 bits are transferred, where the first bit is the start bit, followed by 8 bits of data, and finally 1 stop bit.

SM_2: enables the multiprocessing capability of the

REN: (Receive Enable)

→ When the REN bit is high, it allows the 8051 to receive data on the RxD pin of the 8051.

→ As a result if we want the 8051 to both transfer and receive data, REN must be set to 1.

→ By making REN=0, the receiver is disabled.

TB8:

→ TB8 (transfer bit 8) is used for serial modes 2 and 3. TB8 = 0, since it is not used in our applications

RB8:

→ In serial mode1, this bit gets a copy of the stop bit when an 8-bit data is received.

→ This bit is rarely used anywhere.

→ In all our applications we will make RB8=0. Like TB8, the RB8 bit is also used in serial modes 2 and 3.

TI:

→ TI (Transmit interrupt) is an extremely important flag bit in the SCON register.

→ When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte.

→ TI bit is raised at the beginning of the stop bit.

RI: (Receive Interrupt)

→ Another important flag in the SCON register.

→ When the 8051 receives data serially via RxD it gets rid of the start and stop bits and places the byte in the SBUF register.

→ Then it raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost.

→ RI is raised halfway through the stop bit.

Programming the 8051 to transfer data serially:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2. (8-bit auto reload) to set the baud rate.
2. The TH1 is loaded with one of the values in Table 1 to set the baud rate for serial data transfer (assuming XTAL = 11.0592 MHz)
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start Timer 1
5. TI is cleared (TI = 0)
6. The character byte to be transferred serially is written into the SBUF register

completely.

8. To transfer the next character, go to step 5.

Importance of the TI flag:

→ To understand the importance of TI flag, look at the following sequence of steps that the 8051 goes through in transmitting a character via TxD.

1. The byte character to be transmitted is written into the SBUF register
2. The start bit is transferred.
3. The 8-bit character is transferred one bit a time
4. The stop bit is transferred, and TI flag is raised indicating that the last character was transmitted and it is ready to transfer the next character.
5. Monitoring of TI flag is necessary for not to overload the SBUF register
6. After SBUF is loaded with a new byte, the TI flag bit is cleared to 0 in order for this new byte to be transferred.

Programming the 8051 to receive data serially:

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto reload) to set the baud rate.
2. TH1 is loaded with one of the values in Table 1 to set the baud rate (assuming XTAL = 11.0592 MHz)
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bit data is framed with start and stop bits are receive enable is turned on.
4. TR1 is set to 1 to start Timer 1
5. RI is cleared (RI = 0)
6. The RI flag is monitored with [while[(RI==0);] to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte. Its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Importance of the RI flag bit:

→ In receiving bits via its RxD pin

1. It receives the start bit indicating that the next bit is the first bit of the character byte it is about to receive.

2. The 8-bit character is received one bit at a time. When the last bit is received, a byte is formed and placed in SBUF.

3. The stop bit is received, and 8051 makes RI = 1, indicating that an entire character byte has been received and must be picked up before it gets overwritten by an incoming character

4. By checking the RI flag bit when it is raised, we know that a character has been received and is sitting in the SBUF register. SBUF content is copied to safe place

5. After SBUF contents are copied into a safe place, the RI flag bit must be forced to '0' in order to allow the next received character byte to be placed in SBUF.

Doubling the baud rate in the 8051

→ There are two ways to increase the baud rate of data transfer in the 8051.

1. Use a higher-frequency crystal
2. change a bit in the PCON register, shown below

D7							D0
SMOD	-	-	-	GF1	GF0	PD	IDL

→ Option 1 is not feasible since the system crystal is fixed and also new crystal may not be compatible with the IBM PC serial COM port's baud rate.

→ When the 8051 is powered up, SMOD bit is zero. It can be set to high by software and thereby double the baud rate.

Baud rates for SMOD=0:

→ When SMOD=0, the 8051 divides 1/12 of the XTAL frequency by 32 and uses that frequency for Timer 1 to set the baud rate.

(XTAL = 11.0592 MHz)

Machine cycle freq = 11.0592 MHz / 12 = 921.6 kHz

921.6 kHz / 32 = 28,800 Hz since SMOD=0.

→ This is the frequency used by Timer 1 to set

Baud rates for SMOD=1:

→ When SMOD=1, 1/12 of XTAL is divided by 16 (instead of 32) and that is the frequency used by Timer 1 to set the baud rate.

(XTAL = 11.0592 MHz)

Machine cycle frequency = 11.0592 MHz / 12 = 921.6 kHz

921.6 kHz / 16 = 57,600 Hz since SMOD = 1.

→ This is the frequency used by Timer 1 to set the baud rate.

TH1 (Decimal)	(Hex)	SMOD=0	SMOD=1
-3	FD	9,600	19,200
-6	FA	4,800	9,600
-12	F4	2,400	4,800
-24	E8	1,200	2,400

Table 2: Baud rate comparison for SMOD=0 & SMOD=1

Ex: 1 If the crystal frequency is 22 MHz, what will be the baud rate if

(a) TH1 = -3 (b) TH1 = -12 each with SMOD=0 and SMOD=1.

i) SMOD=0.

Machine cycle frequency = $\frac{22}{12}$ = 1833 kHz

(a) with TH1 = -3, the baud rate is $\frac{57,281}{3} = 19,093$

(b) with TH1 = -12, the baud rate is $\frac{57,281}{12} = 4773$

ii) SMOD = 1, the baud rates are doubled

(a) with TH1 = -3, the baud rate is 38,186

(b) with TH1 = -12, the baud rate is 9546

Serial port programming in 'C':

1) Write a 'C' program for the 8051 to transfer the letter "A" serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

```
#include <reg51.h>
void main (void)
{
    TMOD = 0x20;        // use Timer 1, 8-bit auto reload
    TH1 = 0xFA;         // 4800 baud rate
    SCON = 0x50;
    TR1 = 1;
    while (1)
    {
     SBUF = 'A';        // place value in buffer
     while (TI == 0);
     TI = 0;
    }
}
```

2) Write an 8051 'C' program to transfer the message "GIT" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

```
#include <reg51.h>
void SerTx (unsigned char);
void main (void)
{
    TMOD = 0x20;    // use Timer1, 8-bit auto reload
    TH1 = 0xFD;     // 9600 baud rate
    SCON = 0x50;
    TR1 = 1;        // start timer
    while (1)
    {
      SerTx ('G');
      SerTx ('I');
      SerTx ('T');
    }
}
void SerTx (unsigned char x)
{
    SBUF = x;       // place value in buffer
    while (TI == 0); // wait until transmitted.
    TI = 0;
}
```

8051 Interrupts:

Interrupts vs polling:

→ A single microcontroller can serve several devices. There are two ways to do that.

i) interrupts

ii) polling

→ In the interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal.

→ Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.

→ The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

→ In polling, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service.

→ After that, it moves on to monitor the next device until each one is serviced.

→ Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the microcontroller.

3) Program the 8051 in 'C' to receive bytes of data serially and put them in P1. Set the baud rate at 4800, 8-bit data, and 1 stop bit.

```
#include <reg51.h>
void main (void)
{
  unsigned char mybyte;
  TMOD = 0x20;          // use Timer 1, 8-bit auto-reload
  TH1 = 0xFA;           // 4800 baud rate
  SCON = 0x50;
  TR1 = 1;              // start timer
  while (1)             // repeat forever
  {
    while (RI == 0);    // wait to receive
    mybyte = SBUF;      // save value
    P1 = mybyte;        // write value to port
    RI = 0;
  }
}
```

$$\text{Baud rate} = \frac{K \times \text{XTAL frequency}}{32 \times 12 \times [256 - (TH1)]}$$

where $K = 1$; if SMOD = 0

$K = 2$; if SMOD = 1

Unit V: Interfacing I/O devices with 8051 Microcontroller – LCD, ADC, DAC, Stepper Motor, Keyboard.

1. LCD Interfacing: (Liquid Crystal Display): LCD operation

→ LCD is finding widespread use replacing LEDs (Seven Segment LEDs) for the following reasons

i. Declining prices of LCD
ii Ability to display numbers, characters and graphics
iii Incorporation of refreshing controller into the LCD
iv Ease of programming for characters and graphics.

LCD pin description:

→ The LCD discussed in this section has 14 pins.

1. RS (Register Select):

→ Two registers are available inside the LCD.
→ The RS pin is used for their selection as follows.
→ If RS=0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc.
→ If RS=1 the data register is selected, allowing the user to send data to be displayed on the LCD

2. R/W, read/write:

→ R/W input allows the user to write information to the LCD or read information from it
→ R/W = 1 when reading; R/W = 0 when writing

→

3. E, enable:

→ The enable pin is used by the LCD to latch information presented to its data pins.

→ When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins.

4. D0-D7:

→ The 8-bit data pins, D0-D7 are used to send information to the LCD or read the contents of the LCD's internal registers.

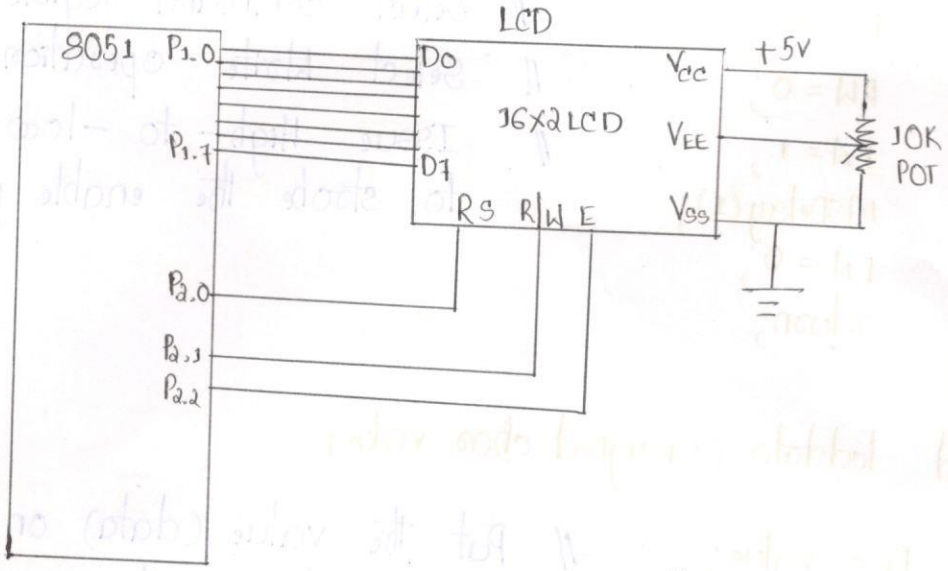
→ To display letters and numbers, we send ASCII codes for the letters A-Z, a-z and numbers 0-9 to these pins while making RS=1.

→ There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor, while making RS=0

→ Table below shows LCD command codes

Code (Hex)	Command to LCD Instruction Register
1	Clear Display screen
2	Return home
4	Decrement cursor
6	Increment cursor
E	Display ON, Cursor blinking
80	Force cursor to beginning of 1st line (80...8F)
C0	Force cursor to beginning of 2nd line (C0...CF)
38	2 lines and 5x7 matrix

Interfacing block diagram



Write an 8051 'C' program to send letters 'C', 'S' and 'E' to the LCD using delays

```
#include<reg51.h>
sbit RS = P2^0;
sbit RW = P2^1;
sbit EN = P2^2;
void main()
{
   lcdcmd(0x38);   // LCD 2 line, 5x7 matrix
   MSDelay(250);
   lcdcmd(0x0E);   // Display on, Cursor blinking
   MSDelay(250);
   lcdcmd(0x01);   // clear display
   MSDelay(250);
   lcdcmd(0x06);   // Increment cursor
   MSDelay(250);
   lcdcmd(0x86);   // LCD 1st line, 6th position
   MSDelay(250);
   lcddata('C');   // Send character 'C'
   MSDelay(250);
   lcddata('S');   // Send character 'S'
   MSDelay(250);
   lcddata('E');   // Send character 'E'
   MSDelay(250);
```

```
{
    P1 = value;          // Put the value (command) on pins
    RS = 0;              // Select command register
    RW = 0;              // Select Write operation
    EN = 1;              // Issue High-to-low pulse
    MSDelay(1);          //   to strobe the enable pin
    EN = 0;
    return;
}

void lcddata (unsigned char value)
{
    P1 = value;          // Put the value (data) on pins
    RS = 1;              // Select Data register
    RW = 0;              // Perform write operation
    EN = 1               // Issue High-to-low pulse to
    MSDelay(1);          //   strobe the enable pin.
    EN = 0;
    return;
}

void MSDelay (unsigned int itime)
{
    unsigned int i, j;
    for (i=0; i<itime; i++)
      for (j=0; j<1275; j++);
}
```

2. ADC Interfacing: (Analog-to-Digital converter) (0808/0809)

→ The ADC 0808/0809 chip allows us to monitor up to 8 different analog inputs using only a single chip.

→ The 8 analog input channels are multiplexed and selected according to the Table given below. using three address pins A, B and C

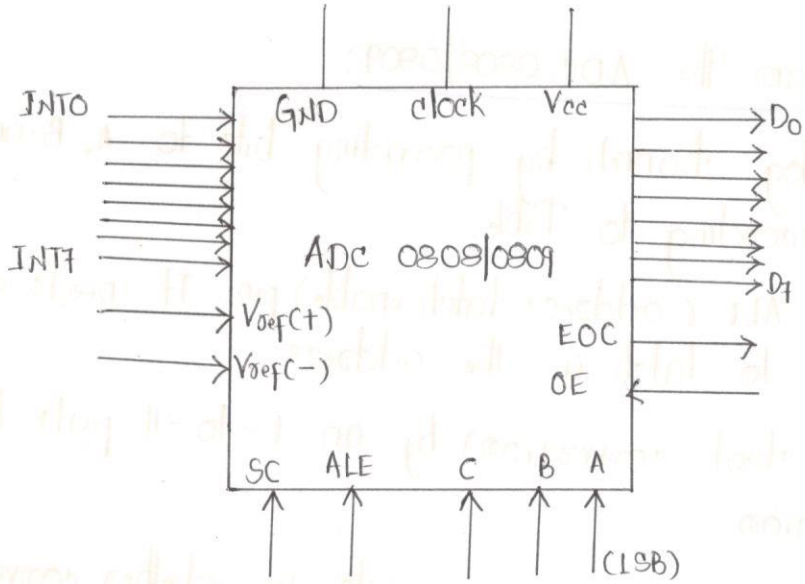


Fig. ADC 0808/0809

Table: ADC 0808/0809 Analog channel Selection

Selected Analog channel	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

→ In ADC, $V_{ref}(+)$ and $V_{ref}(-)$ set the reference voltage. If $V_{ref}(-) = GND$ and $V_{ref}(+) = 5V$, the step size is

$$5V/256 = 19.53mV.$$

→ A, B and C address lines are used to select IN0-IN7, and activate ALE to latch in the address.

→ SC is for start conversion (WRITE). EOC is for end-of-conversion and OE is for output enable (READ)

→ The EOC and OE are same as INTR and RD pins respectively.

Steps to program the ADC 0808/0809:

1. Select an analog channel by providing bits to A, B and C addresses according to Table.
2. Activate the ALE (address latch enable) pin. It needs an L-to-H pulse to latch in the address.
3. Activate SC (start conversion) by an L-to-H pulse to initiate conversion.
4. Monitor EOC (End-of-conversion) to see whether conversion is finished. H-to-L output indicates that the data is converted and is ready to be picked up.
5. Activate OE (output Enable) to read data out of the ADC chip. An L-to-H pulse to the OE pin will bring digital data out of the chip.

Programming ADC 0808/0809 in 'C':

```
#include <reg51.h>
sbit ALE = P2^4;
sbit OE = P2^5;
sbit SC = P2^6;
sbit EOC = P2^7;
sbit ADDR_A = P2^0;
sbit ADDR_B = P2^1;
sbit ADDR_C = P2^2;
```

```
void main()
{
  unsigned char value;
  P1 = 0xFF;              // make P1 as input
  EOC = 1;                // make EOC an input
  ALE = 0;                // clear ALE
  OE = 0;                 // clear OE
  SC = 0;                 // clear SC
  while(1)
    {
      ADDR_C = 0;         // C=0
      ADDR_B = 0;         // B=0
      ADDR_A = 1;         // A=1, select channel 1
      MSDelay(1);         // Delay
      ALE = 1;
      MSDelay(1);
      SC = 1;
      MSDelay(1);
      ALE=0;
      SC = 0;             // start conversion
      while(EOC==1);      // wait for data conversion
      while(EOC==0);
      OE = 1;             // enable RD
      MSDelay(1)
      value = P1;         // get the data
      OE = 0;             // clear OE for next round
    }
}
```

3. DAC Interfacing (D to A converter): (0808)

→ The digital-to-analog converter is a widely used device to convert digital pulses to analog signals.

→ DAC 0808 is a R-2R ladder type DAC and it achieves much higher degree of precision.

→ The resolution of DAC is defined as the number of data bit inputs.

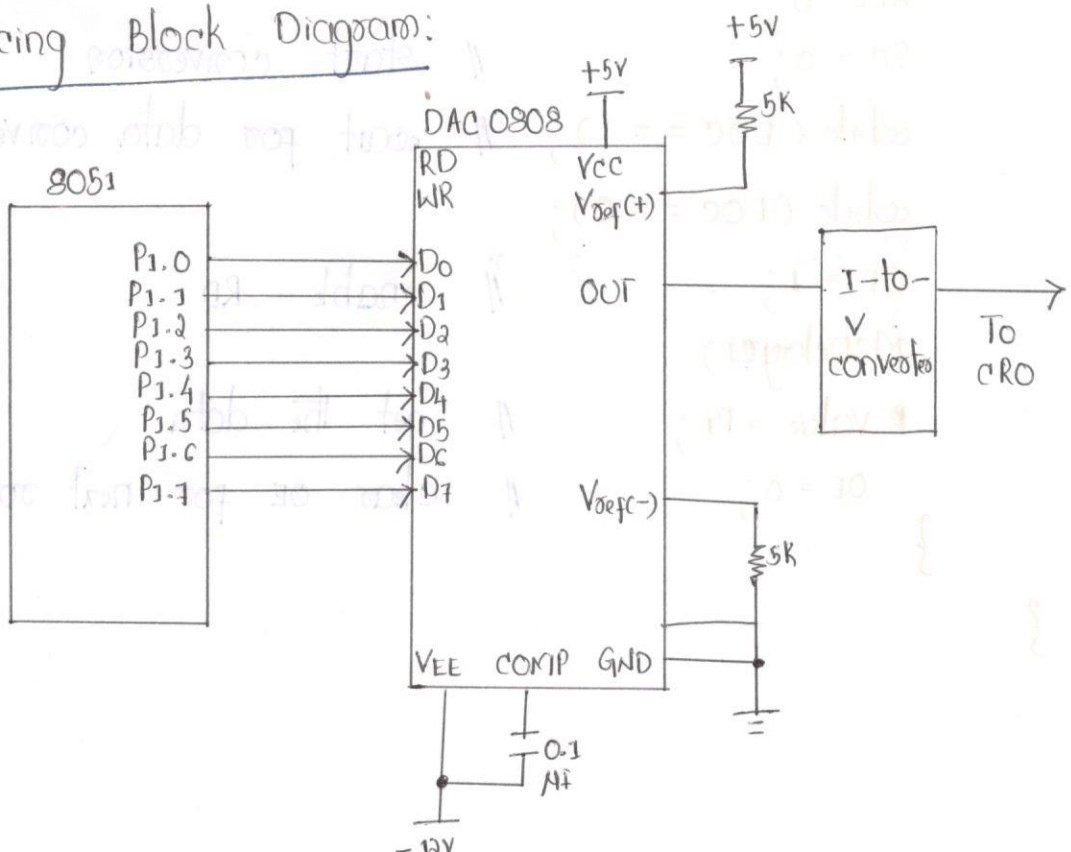
→ DAC 0808 is 8-bit DAC and it provides $2^8 = 256$ discrete voltage / current levels.

→ The output DAC 0808 is current and it is given by

$$I_{out} = I_{ref}\left(\frac{D_7}{2} + \frac{D_6}{4} + \frac{D_5}{8} + \frac{D_4}{16} + \frac{D_3}{32} + \frac{D_2}{64} + \frac{D_1}{128} + \frac{D_0}{256}\right)$$

→ where D_0 is LSB, D_7 is MSB for the inputs and I_{ref} is the input current that must be applied to pin 14.

Interfacing Block Diagram:



Programming DAC in 'C': (To generate sine wave)

```
#include <reg51.h>
DAC
void main()
{
  unsigned char Wave[12]={128,192,238,255,238,192,128,64,
                          17,0,17,64};

  unsigned char i;
  while(1)
   {
    for(i=0; i<12; i++)
      {
        P1 = Wave[i];
      }
   }
}
```

Vout = 5V + 5. sinθ where θ = 0, 30, 60, 90 . . . 360

Values sent to DAC (decimal) (voltage Magnitude x 25.6)

Exercise Programs:

1. Write an 8051 'c' program to interface DAC with 8051 Microcontroller to generate the following waveforms
   i) Square / Rectangular
   ii) Ramp
   iii) Triangular
   iv) Staircase.