## Module – 4

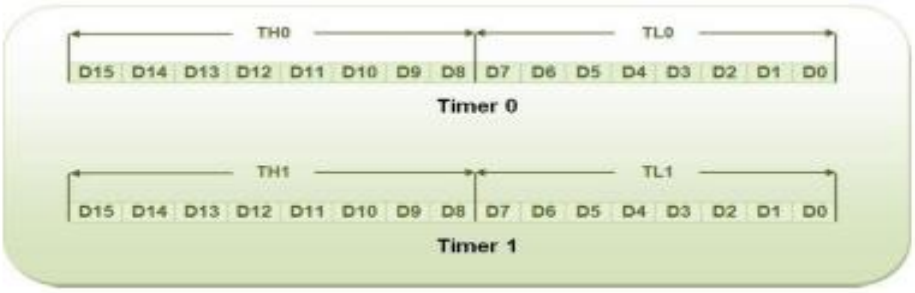
# 8051 Timers and Serial Port

Timers/Counters are used generally for

- Time reference
- Creating delay
- Wave form properties measurement
- Periodic interrupt generation
- The 8051 has two timers/counters, they can be used either as Timers to generate a time delay or as Event counters to count events happening outside the microcontroller

8051 has two timers, Timer 0 and Timer 1.

- Timer 0 and Timer 1 are 16 bits.
- 8051 has an 8-bit architecture, each 16-bits timer is accessed as two separate registers of low byte and high byte.
- The low byte register is called TL0/TL1 and The high byte register is called TH0/TH1.
- Accessed like any other register.




Timer in 8051 is used as timer, counter and baud rate generator. Timer always counts up irrespective of whether it is used as timer, counter, or baud rate generator: Timer is always incremented by the microcontroller. The time taken to count one digit up is based on master clock frequency.

If Master CLK=12 MHz,
Timer Clock frequency = Master CLK/12 = 1 MHz
Timer Clock Period = 1micro second
This indicates that one increment in count will take 1 microsecond.
The two timers in 8051 share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).

The following are timer related SFRs in 8051.

| SFR Name | Description | SFR Address |
|---|---|---|
| *TH0* | *Timer 0 High Byte* | *8Ch* |
| *TL0* | *Timer 0 Low Byte* | *8Ah* |
| *TH1* | *Timer 1 High Byte* | *8Dh* |
| *TL1* | *Timer 1 Low Byte* | *8Bh* |
| *TCON* | *Timer Control* | *88h* |
| *TMOD* | *Timer Mode* | *89h* |

## TMOD Register

- Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes
- TMOD is an 8-bit register
- The lower 4 bits are for Timer 0, the upper 4 bits are for Timer 1
- In each case, the lower 2 bits are used to set the timer mode, the upper 2 bits to specify the operation.

| GATE | C/T | M1 | M0 | GATE | C/T | M1 | M0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| TIMER 1 | | | | TIMER 0 | | | |

| | |
|---|---|
| GATE | When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control). |
| C/T | Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin). |
| M1 | Mode selector bit (NOTE 1). |
| M0 | Mode selector bit (NOTE 1). |

**Note 1 :**

| M1 | M0 | | OPERATING MODE |
|---|---|---|---|
| 0 | 0 | 0 | 13-bit Timer |
| 0 | 1 | 1 | 16-bit Timer/Counter |
| 1 | 0 | 2 | 8-bit Auto-Reload Timer/Counter |
| 1 | 1 | 3 | (Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit Timer and is controlled by Timer 1 control bits. |
| 1 | 1 | 3 | (Timer 1) Timer/Counter 1 stopped. |

## TCON (timer control) register

- TCON (timer control) register is an 8bit register

**TCON : Timer/Counter Control Register (Bit Addressable)**

| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| TF1 | TCON.7 | Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine. |
| TR1 | TCON.6 | Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF. |
| TF0 | TCON.5 | Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine. |
| TR0 | TCON.4 | Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF. |
| IE1 | TCON.3 | External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed. |
| IT1 | TCON.2 | Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt. |
| IE0 | TCON.1 | External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed. |
| IT0 | TCON.0 | Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt. |

## TIMER MODES

Timers can operate in four different modes. They are as follows

**Timer Mode-0:** In this mode, the timer is used as a 13-bit UP counter as follows.

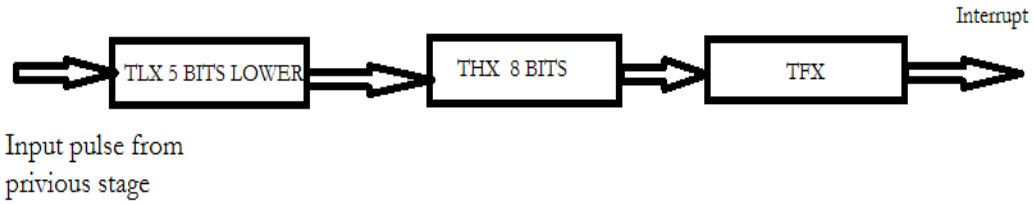



**Fig. Operation of Timer on Mode-0**

The lower 5 bits of TLX and 8 bits of THX are used for the 13-bit count. Upper 3 bits of TLX are ignored. When the counter rolls over from all 0's to all 1's, TFX flag is set and an interrupt is generated. The input pulse is obtained from the previous stage. If TR1/0 bit is 1 and Gate bit is 0, the counter continues counting up. If TR1/0 bit is 1 and Gate bit is 1, then the operation of the counter is controlled by input. This mode is useful to measure the width of a given pulse fed to input.

**Timer Mode-1:** This mode is similar to mode-0 except for the fact that the Timer operates in 16-bit mode.

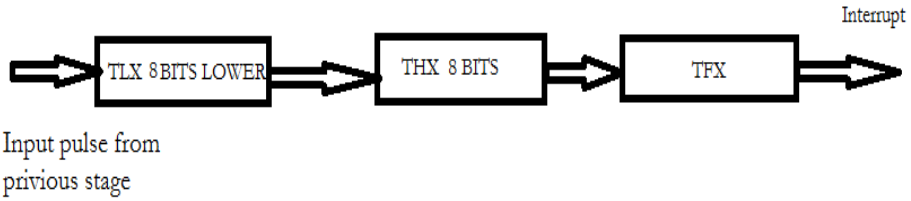



**Fig: Operation of Timer in Mode 1**

**Timer Mode-2:** (Auto-Reload Mode): This is an 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example, if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling.

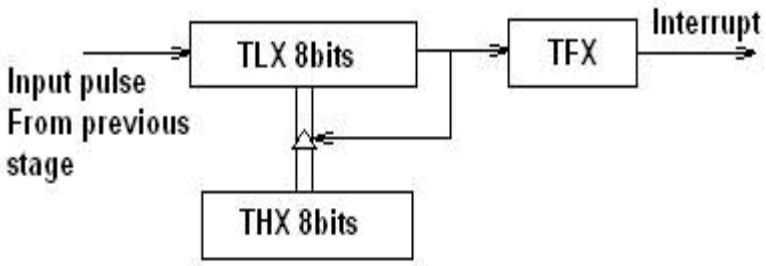



**Fig: Operation of Timer in Mode 2**

**Timer Mode-3:** Timer 1 in mode-3 simply holds its count. The effect is same as setting TR1=0. Timer0 in mode-3 establishes TL0 and TH0 as two separate counters.

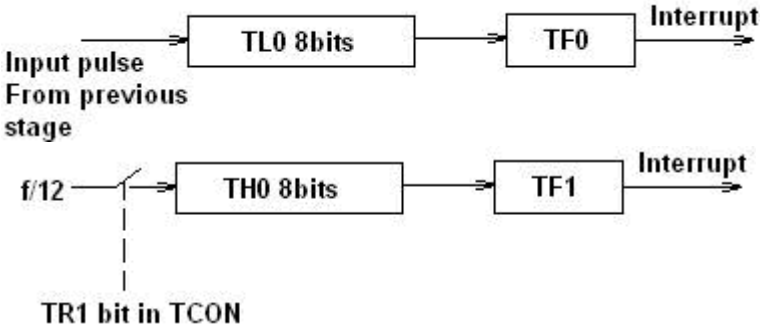



**Fig: Operation of Timer in Mode 3**

Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits (TL0).

## PROGRAMMING 8051 TIMERS IN ASSEMBLY

In order to program 8051 timers, it is important to know the calculation of initial count value to be stored in the timer register. The calculations are as follows.

In any mode, Timer Clock period = 1/Timer Clock Frequency.

= 1/ (Master Clock Frequency/12)

### 1. Mode 1 (16 bit timer/counter)

Value to be loaded in decimal = 65536 – (Delay required/Timer clock period)

Convert the answer into hexadecimal and load onto THx and TLx register.

*(65536D = FFFFH+1)*

### 2. Mode 0 (13 bit timer/counter)

Value to be loaded in decimal = 8192 – (Delay required/Timer clock period)

Convert the answer into hexadecimal and load onto THx and TLx register.

*(8192D = 1FFFH+1)*

**3. Mode 2 (8 bit auto reload)**

Value to be loaded in decimal = 256 – (Delay required/Timer clock period)
Convert the answer into hexadecimal and load onto THx register. Upon starting the timer this value from THx will be reloaded to TLx register. *(256D = FFH+1)*

**Steps for programming timers in 8051**

**Mode 1:**

- Load the TMOD value register indicating which timer (0 or 1) is to be used and which timer mode is selected.
- Load registers TL and TH with initial count values.
  Start the timer by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer 1.
- Keep monitoring the timer flag (TF) with the "JNB TFx, target" instruction to see if it is raised. Get out of the loop when TF becomes high.
- Stop the timer with the instructions "CLR TR0" or "CLR TR1", for timer 0 and timer 1, respectively.
- Clear the TF flag for the next round with the instruction "CLR TF0" or "CLR TF1", for timer 0 and timer 1, respectively.
- Go back to step 2 to load TH and TL again.

**Mode 0:**

The programming techniques mentioned here are also applicable to counter/timer mode 0. The only difference is in the number of bits of the initialization value.

**Mode 2:**

- Load the TMOD value register indicating which timer (0 or 1) is to be used; select timer mode 2.
- Load TH register with the initial count value. As it is an 8-bit timer, the valid range is from 00 to FFH.
- Start the timer.
- Keep monitoring the timer flag (TFx) with the "JNB TFx, target" instruction to see if it is raised. Get out of the loop when TFx goes high.
- Clear the TFx flag.
- Go back to step 4, since mode 2 is auto-reload.

**Example 4-1**

Indicate which mode and which timer are selected for each of the following.
(a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

**Solution:**

We convert the value from hex to binary. From Figure 9-3 we have:
(a) TMOD = 00000001, mode 1 of timer 0 is selected.
(b) TMOD = 00100000, mode 2 of timer 1 is selected.

(c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected

**Example 4-2**

Find the timer's clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.


1/12 × 11.0529 MHz = 921.6 MHz;

T = 1/921.6 kHz = 1.085 us

**Example 4-3**

In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program

```
        MOV TMOD,#01     ;Timer 0, mode 1(16-bit mode)
HERE:   MOV TL0,#0F2H    ;TL0=F2H, the low byte
        MOV TH0,#0FFH    ;TH0=FFH, the high byte
        CPL P1.5         ;toggle P1.5
        ACALL DELAY
        SJMP HERE
DELAY:  SETB TR0         ;start the timer 0
AGAIN:  JNB  TF0,AGAIN   ;monitor timer flag 0 ;until it rolls over
        CLR  TR0         ;stop timer 0
        CLR  TF0         ;clear timer 0 flag
        RET
```

In the above program notice the following step.

1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3,FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1).

At that point, the JNB instruction falls through.

7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends, and the process is repeated.
Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated

**Example 4-4**
In Example 9-4, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.
**Solution:**
The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency. As a result, each clock has a period of T =1/921.6kHz = 1.085us. In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts × 1.085us.
The number of counts for the roll over is FFFFH – FFF2H = 0DH (13decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag. This gives 14× 1.085us = 15.19us for half the pulse. For the entire period it is T = 2× 15.19us = 30.38us as the time delay generated by the timer.

**Example 4-5**
The following program generates a square wave on P1.5 continuously using timer 1 for a time delay. Find the frequency of the square wave if XTAL = 11.0592 MHz. In your calculation do not include the overhead due to Instructions in the loop.

```
        MOV  TMOD,#10    ;Timer 1, mod 1 (16-bitmode)
AGAIN:  MOV  TL1,#34H    ;TL1=34H, low byte of timer
        MOV  TH1,#76H    ;TH1=76H, high byte timer
        SETB TR1         ;start the timer 1
BACK:   JNB  TF1,BACK    ;till timer rolls over
        CLR  TR1         ;stop the timer 1
        CPL  P1.5        ;comp. p1. to get hi, lo
        CLR  TF1         ;clear timer flag 1
        SJMP AGAIN       ;is not auto-reload
```

**Solution:**
Since FFFFH – 7634H = 89CBH + 1 = 89CCH and 89CCH = 35276 clock count and 35276 × 1.085 us = 38.274 ms for half of the square wave. The frequency = 13.064Hz. Also notice that the high portion and low portion of the square wave pulse are equal. In the above calculation, the overhead due to all the instruction in the loop is not included.

**Example 4-6**

Write a program to continuously generate a square wave of 2 kHz frequency on pin P1.5 using timer 1. Assume the crystal oscillator frequency to be 12 MHz.

The period of the square wave is T = 1/(2 kHz) = 500 μs. Each half pulse = 250 μs.
The value n for 250 μs is: 250 μs /1 μs = 250 65536 - 250 = FF06H. TL = 06H and TH = 0FFH.

```
        MOV TMOD,#10      ;Timer 1, mode 1
AGAIN:  MOV TL1,#06H      ;TL0 = 06H
        MOV TH1,#0FFH     ;TH0 = FFH
        SETB TR1          ;Start timer 1
BACK:   JNB TF1,BACK      ;Stay until timer rolls over
        CLR TR1           ;Stop timer 1
        CPL P1.5          ;Complement P1.5 to get Hi, Lo
        CLR TF1           ;Clear timer flag 1
        SJMP AGAIN        ;Reload timer
```

**Example4-6**

Write a program segment that uses timer 1 in mode 2 to toggle P1.0 once whenever the counter reaches a count of 100. Assume the timer clock is taken from external source P3.5 (T1).
The TMOD value is 60H The initialization value to be loaded into TH1 is 256 - 100 = 156 = 9CH

```
        MOV  TMOD, #60h ;Counter1, mode 2, C/T'= 1
        MOV  TH1, #9Ch  ;Counting 100 pulses
        SETB P3.5       ;Make T1 input
        SETB TR1        ;Start timer 1
BACK:   JNB  TF1, BACK  ;Keep doing it if TF = 0
        CPL  P1.0       ;Toggle port bit
        CLR  TF1        ;Clear timer overflow flag
        SJMP BACK       ;Keep doing it
```

# SERIAL COMMUNICATION

The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

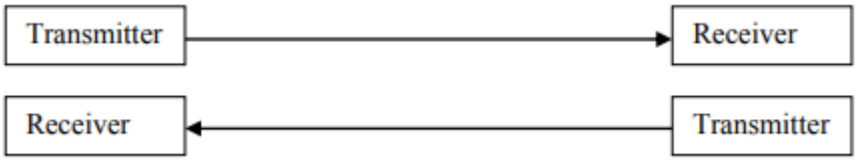
Basics of serial data communication

Communication Links

1. **Simplex communication link**: In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.


2. **Half duplex communication link**: In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.

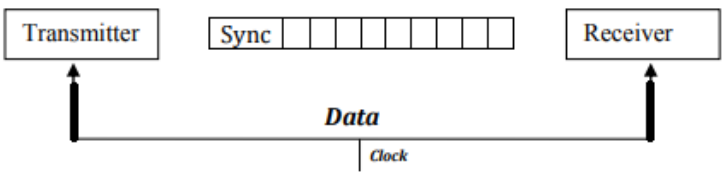

3. **Full duplex communication link**: If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.



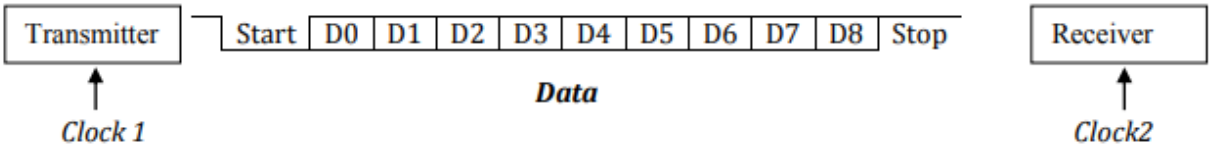

**Types of Serial communication:**

Serial data communication uses two types of communication.

1. Synchronous serial data communication: In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In Synchronous serial data communication a block of data is transmitted at a time.



2. Asynchronous Serial data transmission: In this, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data and then stop bit. For error checking purpose parity bit is included just prior to stop bit. In Asynchronous serial data communication a single byte is transmitted at a time.




**Baud rate**: The rate at which the data is transmitted is called baud or transfer rate. The baud rate is the reciprocal of the time to send one bit. In asynchronous transmission, baud rate is not equal to number of bits per second. This is because; each byte is preceded by a start bit and followed by parity and stop bit. For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second. For bit transmission time = 1 second/ 9600 = 0.104 ms.

## 8051 SERIAL COMMUNICATION

Three special function registers support serial communication.

1. SBUF Register: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.

2. SCON register: The contents of the Serial Control (SCON) register are shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).

| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| **SM0** | SCON.7 | Serial port mode specifier |
| **SM1** | SCON.6 | Serial port mode specifier |
| **SM2** | SCON.5 | Used for multiprocessor communication |
| **REN** | SCON.4 | Set/cleared by software to enable/disable reception |
| **TB8** | SCON.3 | Not widely used |
| **RB8** | SCON.2 | Not widely used |
| **TI** | SCON.1 | Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW |
| **RI** | SCON.0 | Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW |

*Note: Make SM2, TB8, and RB8 =0*

**SM0, SM1:**They determine the framing of data by specifying the number of bits per character, and the start and stop bits

| SM0 | SM1 | |
|---|---|---|
| 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 |

**SM2:** This enables the multiprocessing capability of the 8051

**REN (receive enable) :** It is a bit-addressable register  When it is high, it allows 8051 to receive data on RxD pin  If low, the receiver is disable

**TI (transmit interrupt) :** When 8051 finishes the transfer of 8-bit character  It raises 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) :** When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register  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.

3. **PCON register:** The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission

**PCON : Power Control Register (Not Bit Addressable)**

| SMOD | – | – | – | GF1 | GF0 | PD | IDL |
|---|---|---|---|---|---|---|---|

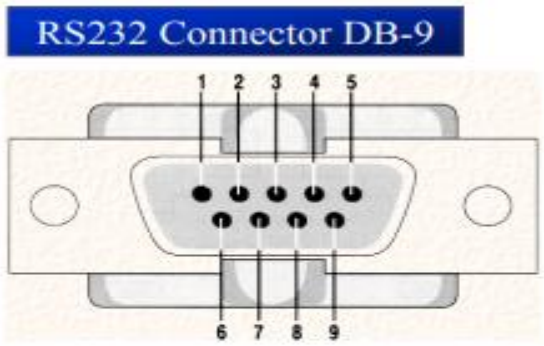
| | | |
|---|---|---|
| SMOD | PCON.7 | Double baud rate bit. If SMOD = 1, the baud rate is doubled when the serial part is used in mode 1, 2 and 3. |
| - | PCON.6 | Not implemented, reserved for futur used* |
| - | PCON.5 | Not implemented, reserved for futur used* |
| - | PCON.4 | Not implemented, reserved for futur used* |
| GF1 | PCON.3 | General purpose bit. |
| GF0 | PCON.2 | General purpose bit. |
| PD | PCON.1 | Power Down bit. If set, the oscillator is stopped. A reset or an interrupt (83C154 and 83C154D only) can cancel this mode (Note 1). |
| IDL | PCON.0 | IDLE bit. If set the activity CPU is stopped. A reset or an interrupt can cancel this mode (See Note 1). |

## SERIAL COMMUNICATION MODES

1. **Mode 0:** In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.
2. **Mode 1:** In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate. Baud rate = [2smod/32] x Timer 1 overflow Rate = [2smod/32] x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]]
3. **Mode 2**: This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit. Baud rate = [2smod/64] x Oscillator Clock Frequency.
4. **Mode 3:** This is similar to mode 2 except baud rate is calculated as in mode 1

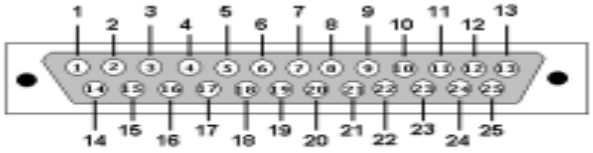
**RS-232 standards:** To allow compatibility among data communication equipment made by various manufactures, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible. In RS232, a logic one (1) is represented by -3 to -25V and

referred as MARK while logic zero (0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers. In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.




**RS232 DB-9 Pins**

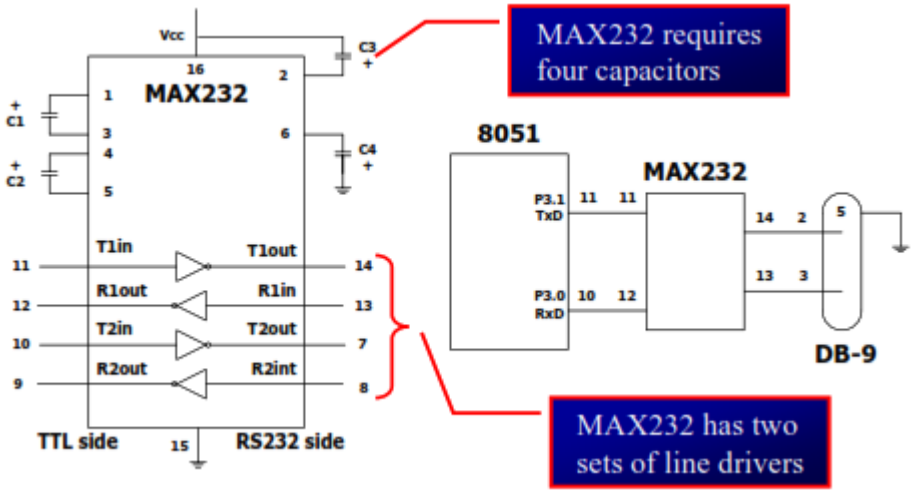
| Pin | Description |
|---|---|
| 1 | Data carrier detect (-DCD) |
| 2 | Received data (RxD) |
| 3 | Transmitted data (TxD) |
| 4 | Data terminal ready (DTR) |
| 5 | Signal ground (GND) |
| 6 | Data set ready (-DSR) |
| 7 | Request to send (-RTS) |
| 8 | Clear to send (-CTS) |
| 9 | Ring indicator (RI) |




*DB25 Male Connector*

**The 8051 connection to MAX232 is as follows.**

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD, RXD. Pin 11 of the 8051 (P3.1) assigned to TXD and pin 10 (P3.0) is designated as RXD. These pins TTL compatible; therefore they require line driver (MAX 232) to make them RS232 compatible. MAX 232 converts RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 is that it uses a +5V power source which is the same as the source voltage for the 8051. The typical connection diagram between MAX 232 and 8051 is shown below



## SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of the 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 5.1 to set the baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an8-bit data is framed with start and stop bits.
4. TR1 is set to 1 start timer 1.
5. TI is cleared by the "CLR TI" instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferred completely.
8. To transfer the next character, go to step 5.

**Example 4.7**

With XTAL = 11.0592 MHz, find the TH1 value needed to have thefollowing baud rates. (a) 9600 (b) 2400 (c) 1200

**Solution:**

The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz,and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 toset baud rate.

(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

**Example 4.8**

Write a program for the 8051 to transfer letter 'A' serially at 4800- baud rate, 8 bit data, 1 stop bit continuously.

**Solution:**

```
        MOV  TMOD, #20H     ;timer 1,mode 2(auto reload)
        MOV  TH1, #-6       ;4800 baud rate
        MOV  SCON, #50H     ;8-bit, 1 stop, REN enabled
        SETB TR1            ;start timer 1
AGAIN:  MOV  SBUF, #"A"     ;letter "A"to transfer
HERE:   JNB  TI, HERE       ;wait for the last bit
        CLR  TI             ;clear TI for next char
        SJMP AGAIN          ;keep sending A
```

**Example 4.9**

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

```
                MOV   TMOD, #20H        ;timer 1,mode 2(auto reload)
                MOV   TH1, #-3          ;9600 baud rate
                MOV   SCON, #50H        ;8-bit, 1 stop, REN enabled
                SETB  TR1               ;start timer 1
AGAIN:          MOV   A, #"Y"           ;transfer "Y"
                ACALL TRANS
                MOV   A, #"E"           ;transfer "E"
                ACALL TRANS
                MOV   A, #"S"           ;transfer "S"
                ACALL TRANS
                SJMP  AGAIN             ;keep doing it serial data transfer subroutine
TRANS:          MOV   SBUF,A            ;load SBUF
HERE:           JNB   TI,HERE           ;wait for the last bit
                CLR   TI                ;get ready for next byte
                RET
```

**Example 4.10**

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

**Solution:**

```
#include <reg51.h>
void main(void)
{
        TMOD=0x20;              //use Timer 1, mode 2
        TH1=0xFA;               //4800 baud rate
        SCON=0x50;
        TR1=1;
        while (1) {
                        SBUF='A';       //place value in buffer
                        while (TI==0);
                        TI=0;
                  }
}
```

**Example 4.11**

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

**Solution:**

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void)
{
        TMOD=0x20;          //use Timer 1, mode 2
        TH1=0xFD;           //9600 baud rate
        SCON=0x50;
        TR1=1;              //start timer
        While (1) {
                    SerTx('Y');
                    SerTx('E');
                    SerTx('S');
                }
}
void SerTx(unsigned char x)
{
        SBUF=x;             //place value in buffer
        While (TI==0);      //wait until transmitted
        TI=0;
}
```