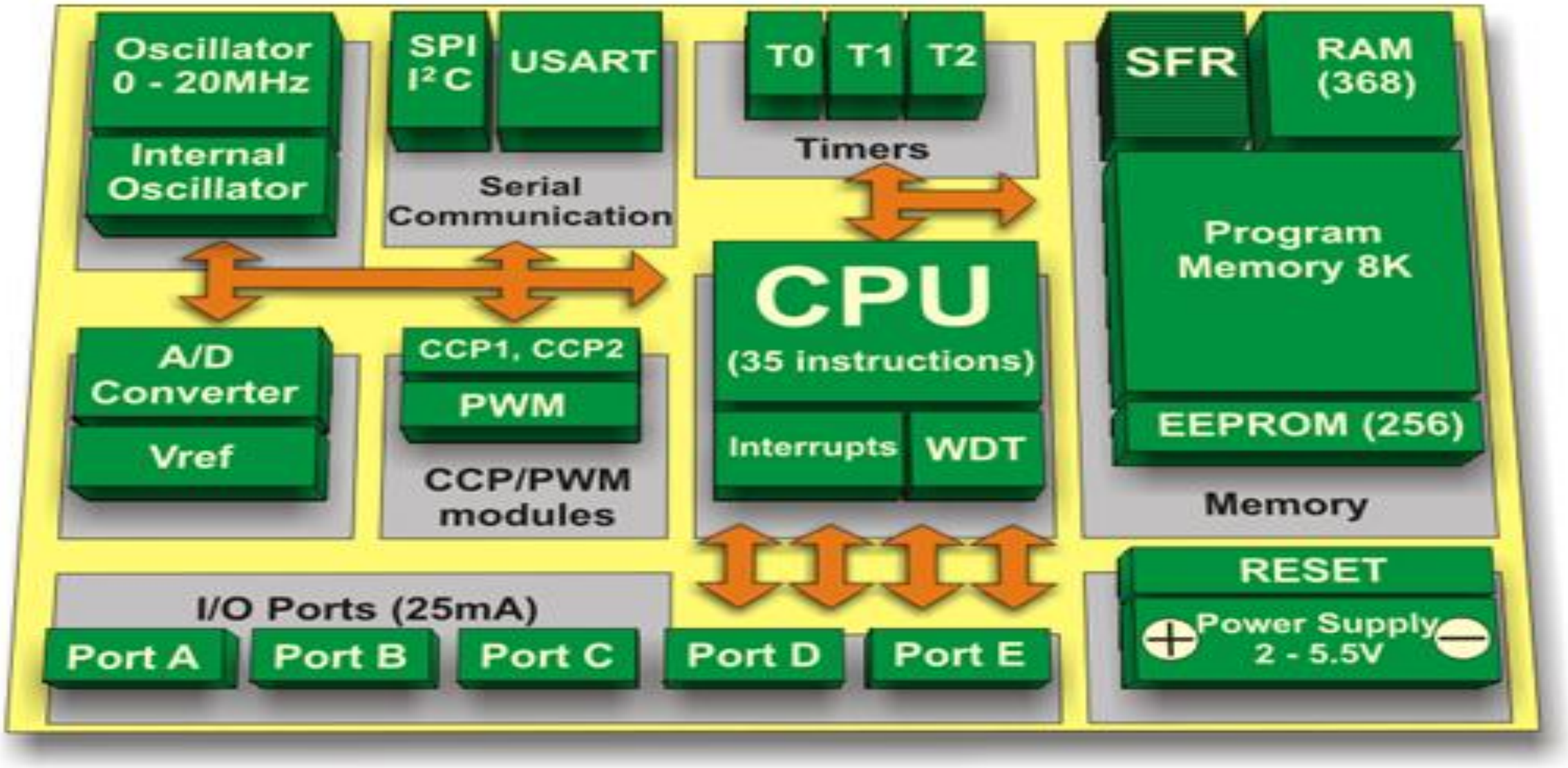
Unit 4
Timers and Serial Port

- A **timer** is a specialized type of clock which is used to measure time intervals.
- A timer that counts from zero upwards for measuring time elapsed is often called a **stopwatch**.
- It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.
- A **counter** is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal.
- It is used to count the events happening outside the microcontroller.
- In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.

Microcontroller Architecture

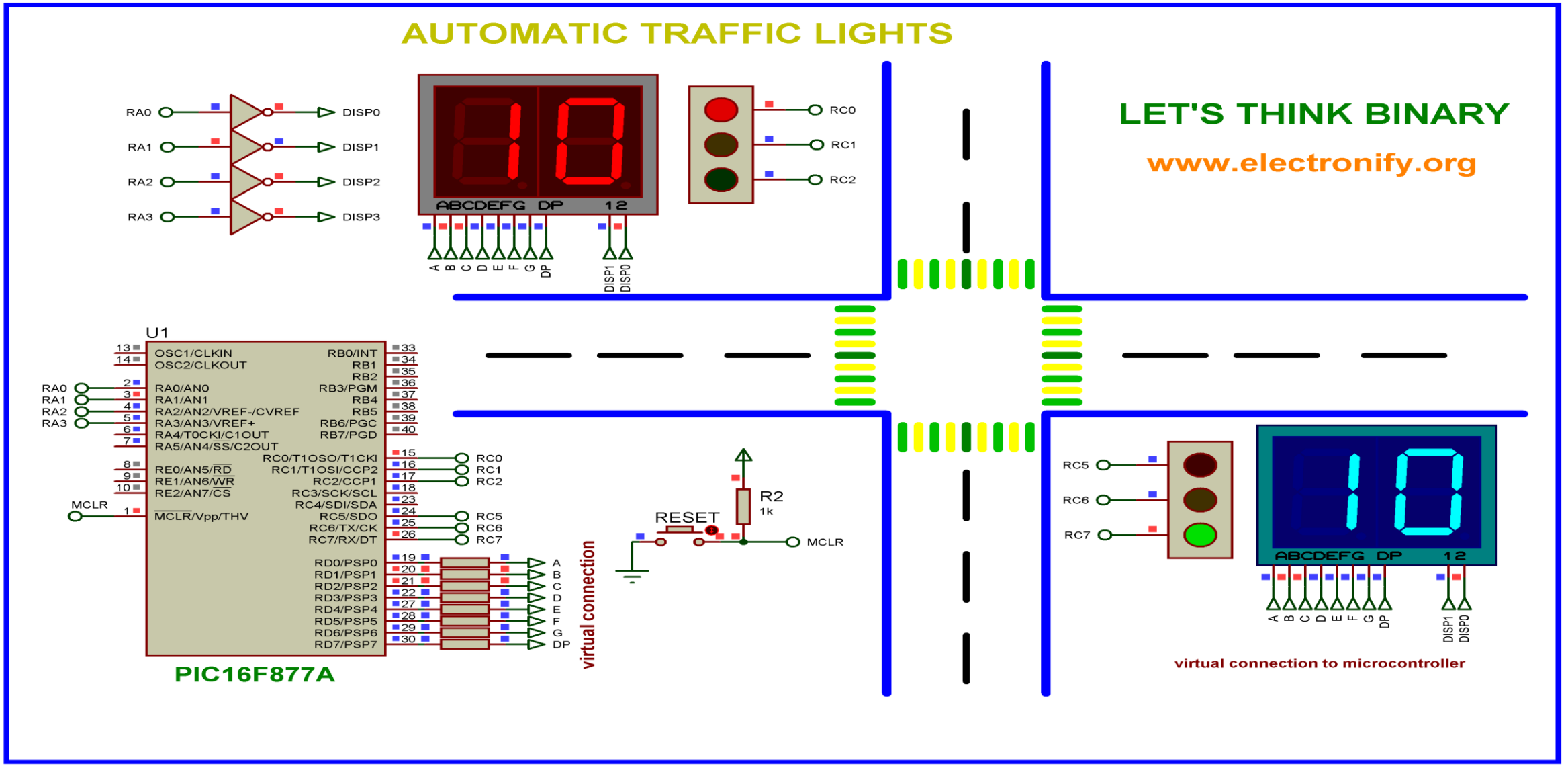


The principle of the timer

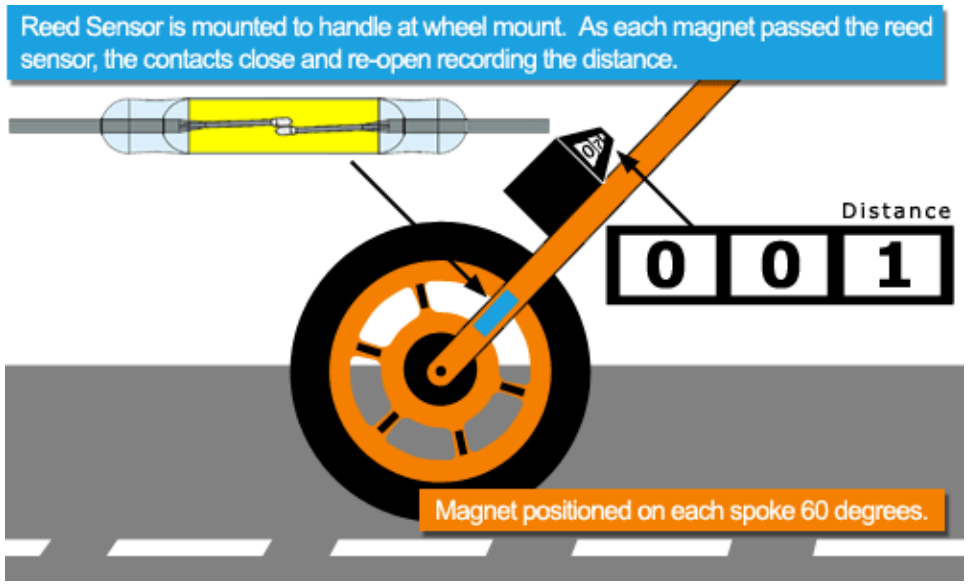
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**
1	0	0	0	**8**
1	0	0	1	**9**
1	0	1	0	**10**
1	0	1	1	**11**
1	1	0	0	**12**
1	1	0	1	**13**
1	1	1	0	**14**
1	1	1	1	**15**

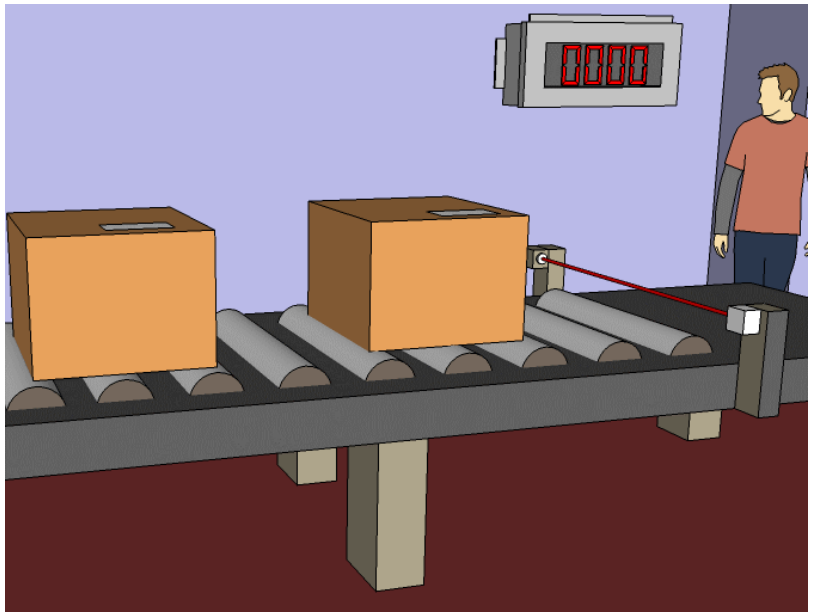
2^4	2^3	2^2	2^1	2^0	
16	8	4	2	1	
0	0	0	0	0	0 0

Application of the Timers



Application of the Counters





TIMERS / COUNTERS Registers

- **Four-SFR's connected** with **TIMER/COUNTER** operation
  - **TMOD** – Timer Mode Register
  - **TCON** – Timer Control Register
  - **TH0, TL0** – Timer/Counter - 0
  - **TH1, TL1** – Timer/Counter - 1
- **Two pins** of **8051 connected** with **Timer/counter**.
  - **T0** – Timer 0 external input – **P3.4**
  - **T1** – Timer 1 external input – **P3.5**
- **INT0** and **INT1** are also used for controlling the timer/counters.

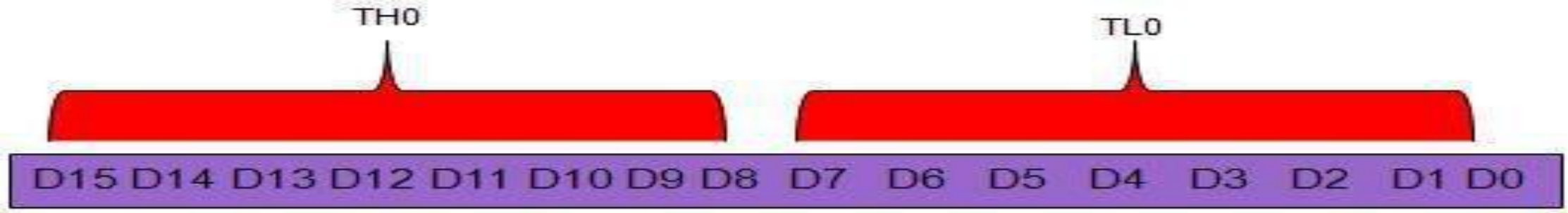
Timers of 8051 and their Associated Registers

The 8051 has two timers, Timer 0 and Timer 1. They can be used as timers or as event counters. Both Timer 0 and Timer 1 are 16-bit wide. Since the 8051 follows an 8-bit architecture, each 16 bit is accessed as two separate registers of low-byte and high-byte.

Timer 0 Register

The 16-bit register of Timer 0 is accessed as low- and high-byte. The low-byte register is called TL0 (Timer 0 low byte) and the high-byte register is called TH0 (Timer 0 high byte). These registers can be accessed like any other register.

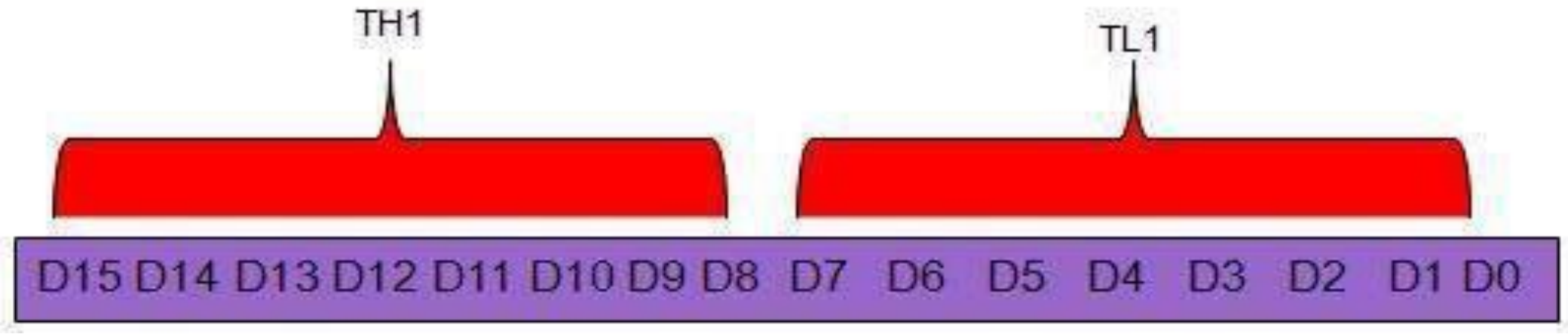
For example, the instruction MOV TL0, #4H moves the value into the low-byte of Timer #0.



Timer 1 Register

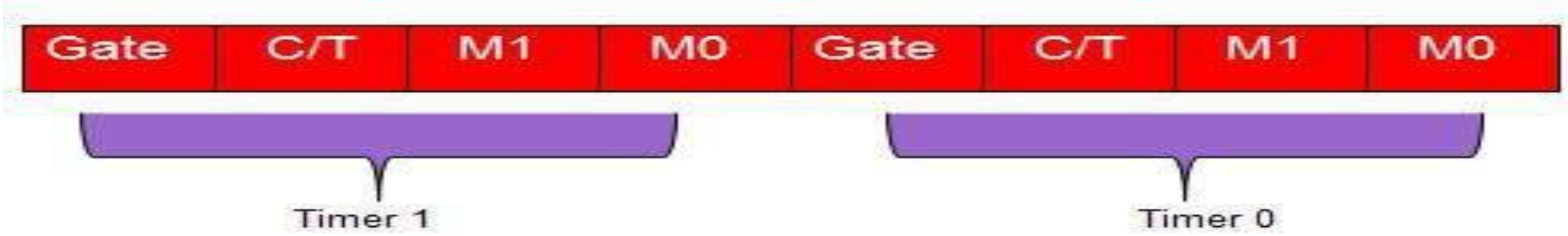
The 16-bit register of Timer 1 is accessed as low- and high-byte. The low-byte register is called TL1 (Timer 1 low byte) and the high-byte register is called TH1 (Timer 1 high byte). These registers can be accessed like any other register.

For example, the instruction **MOV TL1, #4H** moves the value into the low-byte of Timer 1.



TMOD (Timer Mode) Register

Both Timer 0 and Timer 1 use the same register to set the various timer operation modes. It is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper four bits for Timers. In each case, the lower 2 bits are used to set the timer mode in advance and the upper 2 bits are used to specify the location.



Gate – When set, the timer only runs while INT(0,1) is high.
C/T – Counter/Timer select bit.
M1 – Mode bit 1.
M0 – Mode bit 0.

Timer Mode (GATE)

- Every timer has a means of starting and stopping.
- Some timers do this by software, some by hardware, and some have both software and hardware controls.
- 8051 timers have both software and hardware controls. The start and stop of a timer is controlled by software using the instruction
- **SETB TR1** and **CLR TR1** for timer 1, and **SETB TR0** and **CLR TR0** for timer 0.
- The **SETB** instruction is used to start it and it is stopped by the **CLR** instruction.
- These instructions start and stop the timers as long as GATE = 0 in the TMOD register.
- Timers can be started and stopped by an external source by making GATE = 1 in the TMOD register.

C/T (CLOCK / TIMER)

- This bit in the TMOD register is used to decide whether a timer is used as a **delay generator** or an **event manager**.
- If C/T = 0, it is used as a timer for timer delay generation.
- The clock source to create the time delay is the crystal frequency of the 8051.
- If C/T = 0, the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks at a regular interval.
- Timer frequency is always 1/12th of the frequency of the crystal attached to the 8051.
- Although various 8051 based systems have an XTAL frequency of 10 MHz to 40 MHz, we normally work with the XTAL frequency of 11.0592 MHz.
- It is because the baud rate for serial communication of the 8051.
- XTAL = 11.0592 allows the 8051 system to communicate with the PC with no errors.

M1	M2	Mode
0	0	Mode 0:13-bit timer mode.
0	1	Mode 1:16-bit timer mode.
1	0	Mode 2: 8-bit auto reload mode.
1	1	Mode 3: Spilt mode.

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

M1	M0	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overfolws
1	1	3	**Split timer mode**

Gating control when set. Timer/counter is enable only while the INTx pin is high and the TRx control pin is set
When cleared, the timer is enabled whenever the TRx control bit is set

Timer or counter selected
Cleared for timer operation (input from internal system clock)
Set for counter operation (input from Tx input pin)

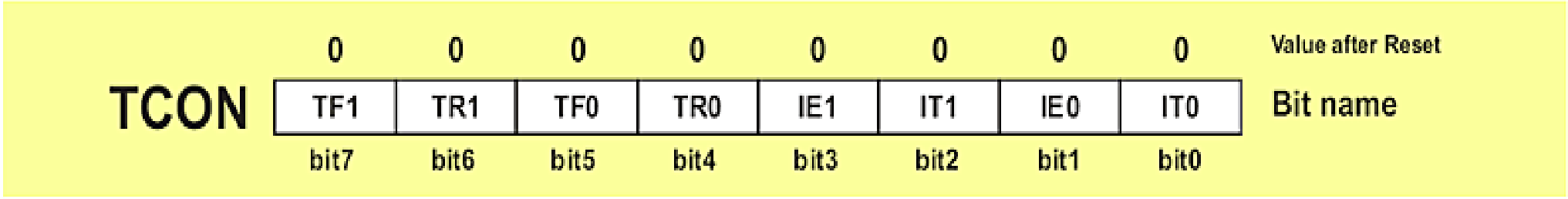
Bits of Registers

- **GATE0** enables and disables Timer 0 by means of a signal brought to the INT0 pin (P3.3):
  - **1** – Timer 0 operates only if the INT0 bit is set.
  - **0** – Timer 0 operates regardless of the logic state of the INT0 bit.
- **C/T1** selects pulses to be counted up by the timer/counter 1:
  - **1** – Timer counts pulses brought to the T0 pin (P3.5).
  - **0** – Timer counts pulses from internal oscillator.
- **T0M1,T0M0** These two bits select the operational mode of the Timer 0.

Timer Control (TCON) Register

- **TF1** bit is automatically set on the Timer 1 overflow.
- **TR1** bit enables the Timer 1.
  - **1** – Timer 1 is enabled.
  - **0** – Timer 1 is disabled.

- **TF0** bit is automatically set on the Timer 0 overflow.
- **TR0** bit enables the timer 0.
  - **1** – Timer 0 is enabled.
  - **0** – Timer 0 is disabled.



(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 Overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
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 on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.
IE1	TCON.3	Interrupt 1 Edge Flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge Flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

Different Modes of Timers

Mode 0 (13-Bit Timer Mode)

- Both Timer 1 and Timer 0 in Mode 0 operate as 8-bit counters (with a divide-by-32 prescaler).
- Timer register is configured as a 13-bit register consisting of all the 8 bits of TH1 and the lower 5 bits of TL1.
- The timer interrupt flag TF1 is set when the count rolls over from all 1s to all 0s. Mode 0 operation is the same for Timer 0 as it is for Timer 1.
- When C/T=0 timer will work as timer to provide time delay
- When C/T=1 timer will work as counter

MODE 0

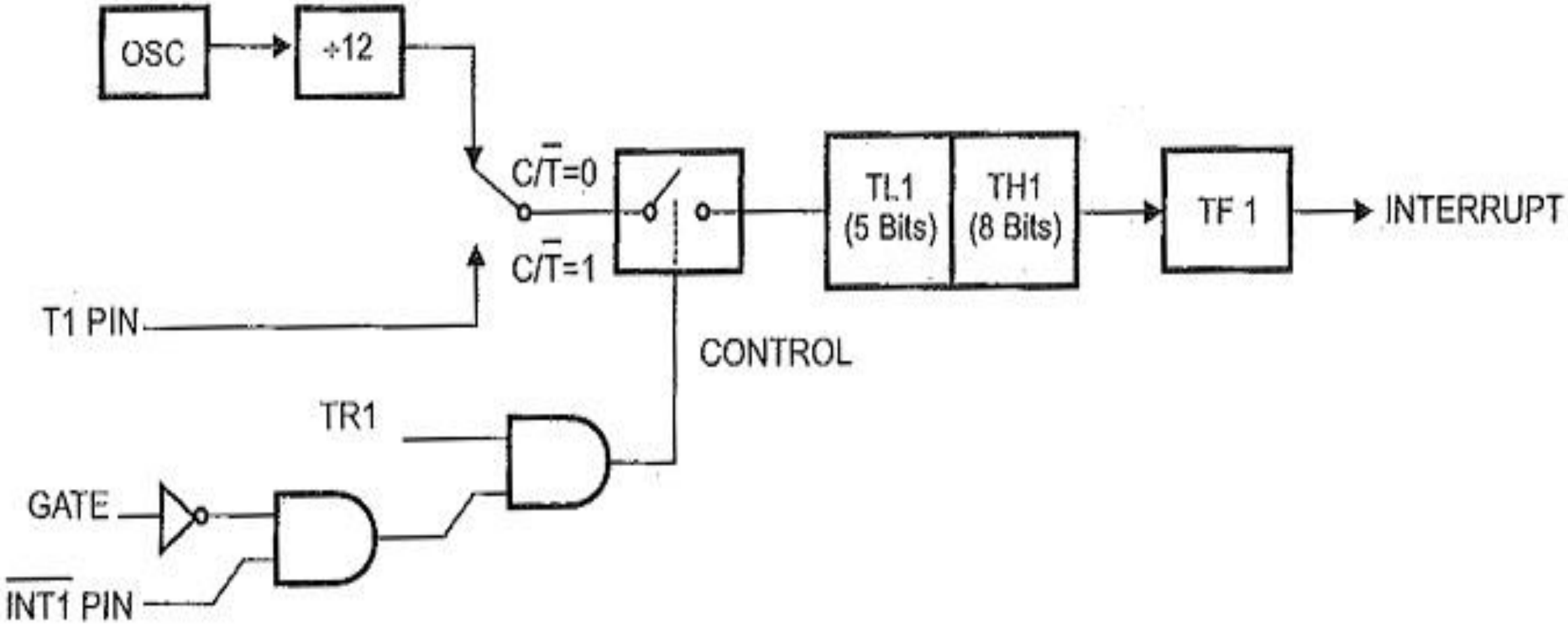


Fig. 12.16 Timer/counter 1 mode 0 : 13-bit counter

Mode 1 (16-Bit Timer Mode)

- Timer mode "1" is a 16-bit timer and is a commonly used mode.
- It functions in the same way as 13-bit mode except that all 16 bits are used.
- TLx is incremented starting from 0 to a maximum 255. Once the value 255 is reached, TLx resets to 0 and then THx is incremented by 1.
- As being a full 16-bit timer, the timer may contain up to 65536 distinct values and it will overflow back to 0 after 65,536 machine cycles.

Mode 1 Programming:

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register 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 SETB TR0 for timer 0 and SETB TR1 for timer 1.
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). Each timer has its own timer flag: TF0 for timer 0 and TF1 for timer 1. This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions CLR TR0 or CLR TR1, for timer 0 and timer 1, respectively.
4. After the timer reaches its limit and rolls over, in order to repeat the process. TH and TL must be reloaded with the original value, and TF must be reloaded to 0.

Steps to program in mode 1:

To generate a time delay, using timer in mode 1, following are the steps:

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 value.
3. Start the timer.
4. Keep monitoring the timer overflow flag (TF) with the JNB TFx, target instruction 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.

Example 1

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. Also calculate the delay generated. Assume XTAL=11.0592MHz.

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 (CPL instruction logically complements the value of the specified destination operand and stores the result back in the destination operand)
ACALL DELAY
SJMP HERE ;Short jump
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 ; Return from Subroutine
```

(a)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

Example 2: Assume that XTAL = 11.0592 MHz, write a program to generate a square wave of 2 kHz frequency on pin P1.5.

Solution:

Look at the following steps.

XTAL(Crystal Frequency)=11.0592 We divide it by 12 we get XTAL=0.9216 MHZ

We know, T=1/F=1/0.9216=1.085

(a) T = 1 / f = 1 / 2 kHz = 500 us the period of square wave.

(b) Divide 500 us in two equal parts i.e.1 / 2 of it for the high and low portion of the pulse is 250 us.

(c) 250 us / 1.085 us = 230 and 65536 – 230 = 65306 which in hex is FF1AH.

(d) TL = 1A and TH = FF, all in hex. The program is as follow.

```
MOV TMOD,#01 ;Timer 0, 16-bitmode
AGAIN: MOV TL0,#1AH ;TL1=1A, low byte of timer
MOV TH0,#0FFH ;TH1=FF, the high byte
SETB TR0 ;Start timer 1
BACK: JNB TF0,BACK ;until timer rolls over
CLR TR0 ;Stop the timer 1
CLR P1.5 ;Clear timer flag 1
CLR TF0 ;Clear timer 1 flag
SJMP AGAIN ;Reload timer
```

Mode 2 (8 Bit Auto Reload)

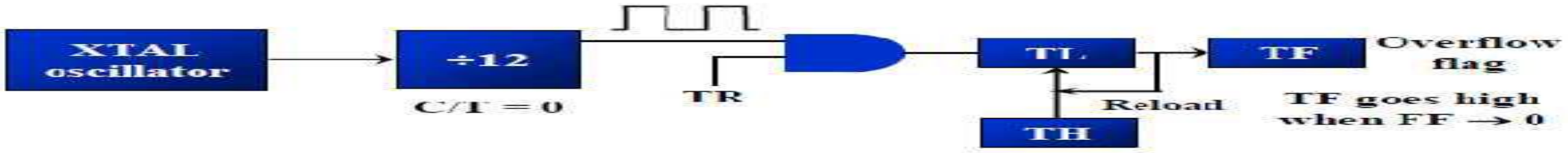
- Both the timer registers are configured as 8-bit counters (TL1 and TL0) with automatic reload. Overflow from TL1 (TL0) sets TF1 (TF0) and also reloads TL1 (TL0) with the contents of Th1 (TH0), which is preset by software. The reload leaves TH1 (TH0) unchanged.
- The benefit of auto-reload mode is that you can have the timer to always contain a value from 200 to 255.
- **If you use mode 0 or 1**, you would have to check in the code to see the overflow and, in that case, reset the timer to 200. In this case, precious instructions check the value and/or get reloaded.
- In mode 2, the microcontroller takes care of this. Once you have configured a timer in mode 2, you don't have to worry about checking to see if the timer has overflowed, nor do you have to worry about resetting the value because the microcontroller hardware will do it all for you.
- The auto-reload mode is used for establishing a common baud rate.

*The baud rate is **the rate at which information is transferred in a communication channel**. Baud rate is commonly used when discussing electronics that use serial communication. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.*

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
   - This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1
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)
4. When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register
   - To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value
   - This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL



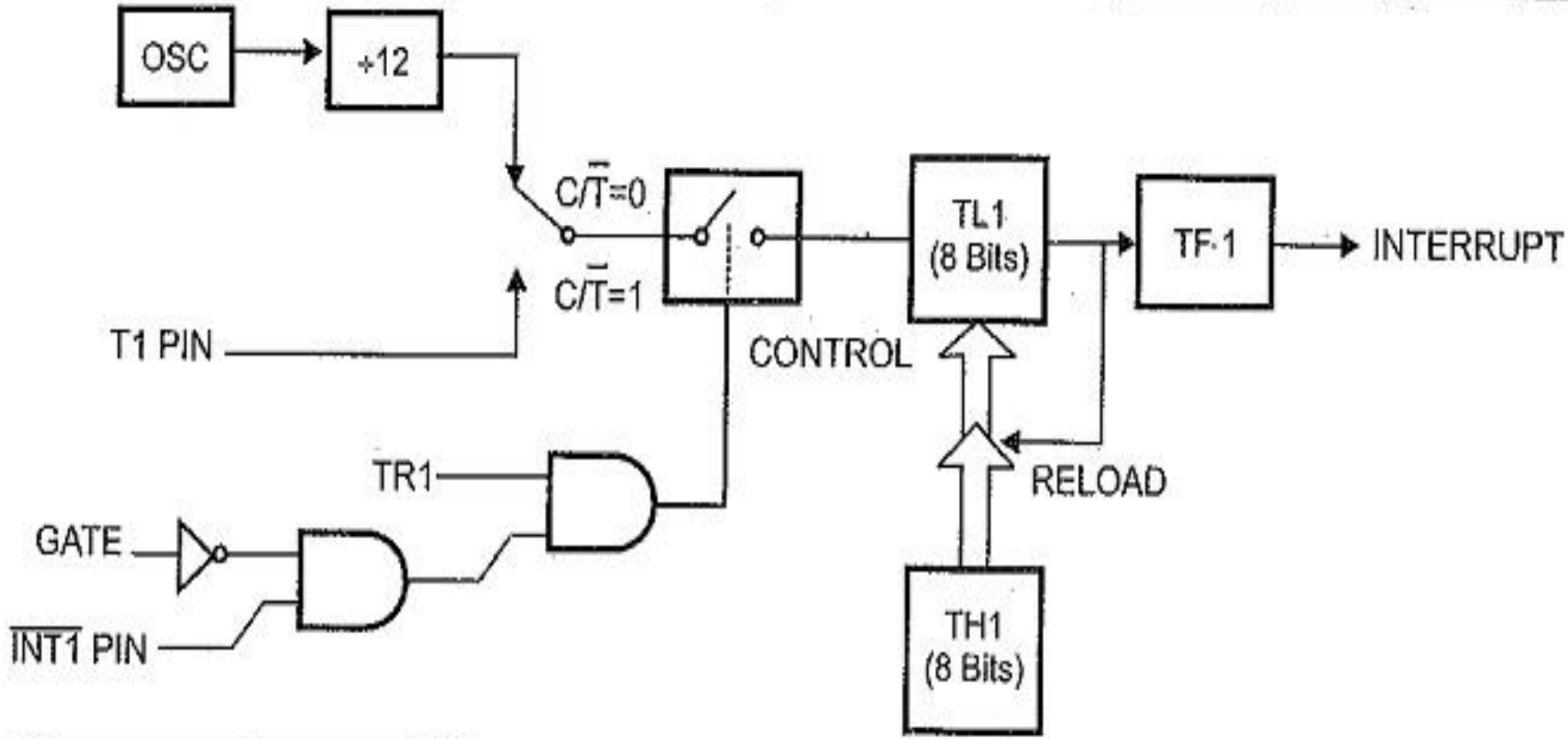


Fig. 12.18 Timer/counter 1 mode 2 : 8-bit auto reload

Steps to program in mode 2:

To generate a time delay

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used, and the timer mode (mode 2) is selected.
2. Load the TH registers with the initial count value.
3. Start timer.
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag and
6. Go back to Step 4, since mode 2 is auto reload.

Example: Subroutine program to generate delay of 0.1ms continuously. Use timer 1 in mode 2.

Solution:

Delay = (FFh - Count)*time for one clock cycle ----------for hexadecimal

Delay = (256-count)*time for one clock cycle----------for decimal

Count to generate delay of 0.1 ms

0.1 ms = (256 - count)*1.085

[Since, XTAL(Crystal Frequency)=11.0592 We divide it by 12 we get XTAL=0.9216 MHZ We know, T=1/F=1/0.9216=1.085 Thus time for 1 clock cycle is 1.085 ms]

Thus, count = 164 in decimal and A4H in hexadecimal

```
MOV TMOD, #20 H
MOV TH1, #A4H
SETB TCON.6
L1: SJMP L1
CLR TCON.6
```

Example : Assume XTAL = 11.0592 MHz, find the frequency of the square wave generated on pin P1.0 in the following program(assume count=5)

Program:

```
MOV TMOD, #20H ; T1/8-bit/auto reload
MOV TH1, #5 ; TH1 = 5
SETB TR1 ; start the timer 1
BACK: JNB TF1, BACK ; till timer rolls over
CPL P1.0 ; P1.0 to high, low
CLR TF1 ; clear Timer 1 flag
SJMP BACK ; mode 2 is auto-reload
```

Solution:

First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload. Now (256 - 05) × 1.085 us = 251 × 1.085 us = 272.33 us is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 272.33 us = 544.67 us and the frequency = 1.83597 kHz

Example : Write an ALP to generate a square wave of frequency 72Hz on pin P1.0.

Solution: Assume XTAL=11.0592MHz. With TH=00, the delay generated is 256 x 1.085 µs = 277.76 µs. therefore to generate a delay of (1 / 72) = 138.88ms, the count to be loaded is 250 x 2=500. That is T = 2 (250 × 256 × 1.085 µs) = 138.88ms, and frequency = 72 Hz

Program:

```
MOV TMOD, #2H ; Timer 0, mod 2;(8-bit, auto reload)
MOV TH0, #0
AGAIN: MOV R5, #250 ; multiple delay count
ACALL DELAY
CPL P1.0
SJMP AGAIN
DELAY: SETB TR0 ; start the timer 0
BACK: JNB TF0,BACK ; stay timer rolls over
CLR TR0 ; stop timer
CLR TF0 ; clear TF for next round
DJNZ R5,DELAY
RET
```

Mode 3 (Split Timer Mode)

- Timer mode "3" is known as **split-timer mode**. When Timer 0 is placed in mode 3, it becomes two separate 8-bit timers.
- Timer 0 is TL0 and Timer 1 is TH0. Both the timers count from 0 to 255 and in case of overflow, reset back to 0. All the bits that are of Timer 1 will now be tied to TH0.
- When Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be set in modes 0, 1 or 2, but it cannot be started/stopped as the bits that do that are now linked to TH0. The real timer 1 will be incremented with every machine cycle.

Initializing a Timer

Decide the timer mode. Consider a 16-bit timer that runs continuously, and is independent of any external pins.

Initialize the TMOD SFR. Use the lowest 4 bits of TMOD and consider Timer 0. Keep the two bits, GATE 0 and C/T 0, as 0, since we want the timer to be independent of the external pins. As 16-bit mode is timer mode 1, clear T0M1 and set T0M0. Effectively, the only bit to turn on is bit 0 of TMOD. Now execute the following instruction −

MOVNow, Timer 0 is in 16-bit timer mode, but the timer is not running. To start the timer in running mode, set the TR0 bit by executing the following instruction −

SETB TR0

Now, Timer 0 will immediately start counting, being incremented once every machine cycle.

TMOD,#01h

Reading a Timer

A 16-bit timer can be read in two ways. Either read the actual value of the timer as a 16-bit number, or you detect when the timer has overflowed.

Detecting Timer Overflow

When a timer overflows from its highest value to 0, the microcontroller automatically sets the TFx bit in the TCON register. So instead of checking the exact value of the timer, the TFx bit can be checked. If TF0 is set, then Timer 0 has overflowed; if TF1 is set, then Timer 1 has overflowed.

Serial Communication- Basics of Serial Data Communication

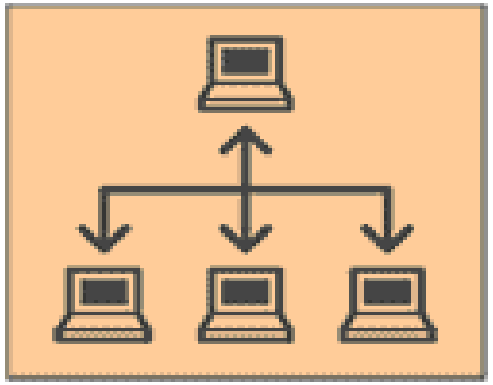
- **Serial communication** is the most widely used approach to transfer information between data processing equipment and peripherals.
- In general, communication means interchange of information between individuals through written documents, verbal words, audio and video lessons.
- Every device might it be your Personal computer or mobile runs on serial protocol.
- The protocol is the secure and reliable form of communication having a set of rules addressed by the source host (*sender*) and destination host (*receiver*).

- In embedded system, Serial communication is the way of exchanging data using different methods in the form of serial digital binary.
- Some of the well-known interfaces used for the data exchange are RS-232, RS-485, I2C, SPI etc.

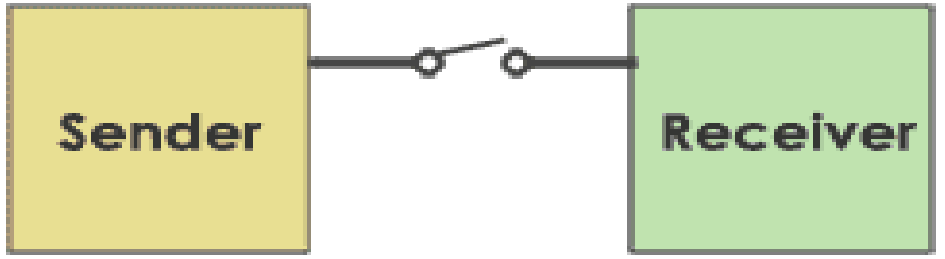
- **In serial communication**, data is in the form of binary pulses.
- In other words, we can say Binary One represents a logic HIGH or 5 Volts, and zero represents a logic LOW or 0 Volts.
- Serial communication can take many forms depending on the type of transmission mode and data transfer.
- The **transmission modes** are classified as Simplex, Half Duplex, and Full Duplex.
- There will be a source (also known as a *sender)* and destination (also called *a receiver*) for each transmission mode.

Transmission Modes in Serial Communication

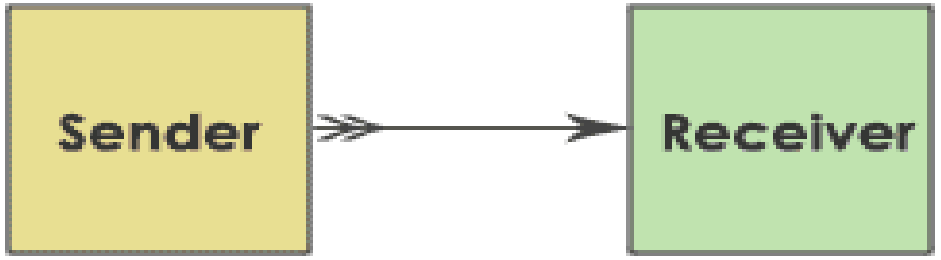
Network



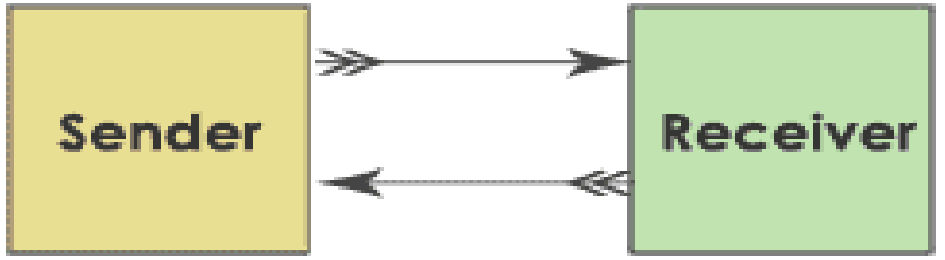
Half Duplex



Simplex



Full Duplex



Codrey Electronics

The **Simplex method** is a one-way communication technique. Only one client (either the sender or receiver is active at a time). If a sender transmits, the receiver can only accept. Radio and Television transmission are the examples of simplex mode.

In **Half Duplex mode**, both sender and receiver are active but not at a time, i.e. if a sender transmits, the receiver can accept but cannot send and vice versa. A good example is an internet. If a client (laptop) sends a request for a web page, the web server processes the application and sends back the information.

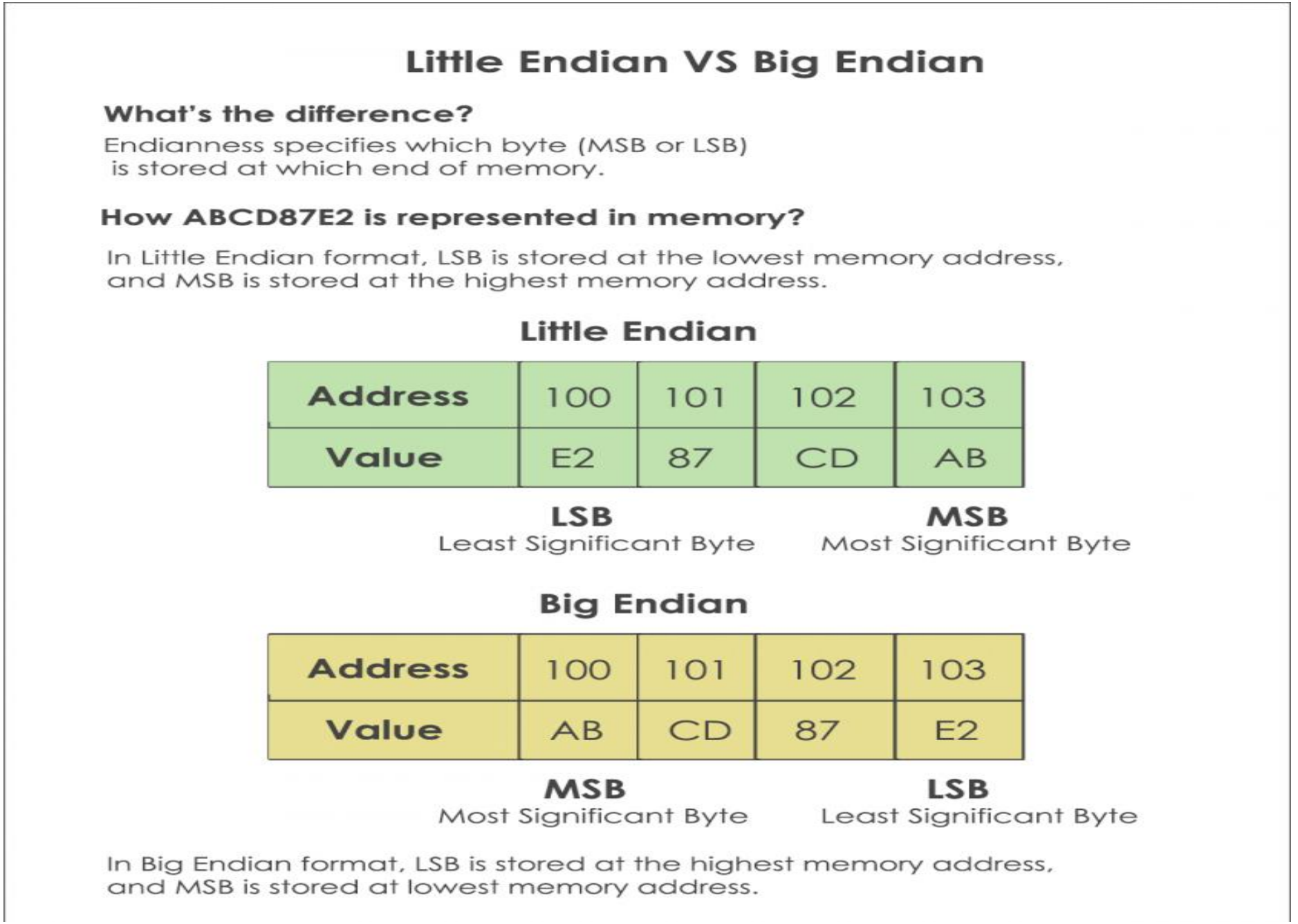
The **Full Duplex mode** is widely used communication in the world. Here both sender and receiver can transmit and receive at the same time. An example is your smartphone.

Beyond the transmission modes, we have to consider the endianness and protocol design of the host computer (sender or receiver).

Endianness is the way of storing the data at a particular memory address. Depending on the data alignment endian is classified as

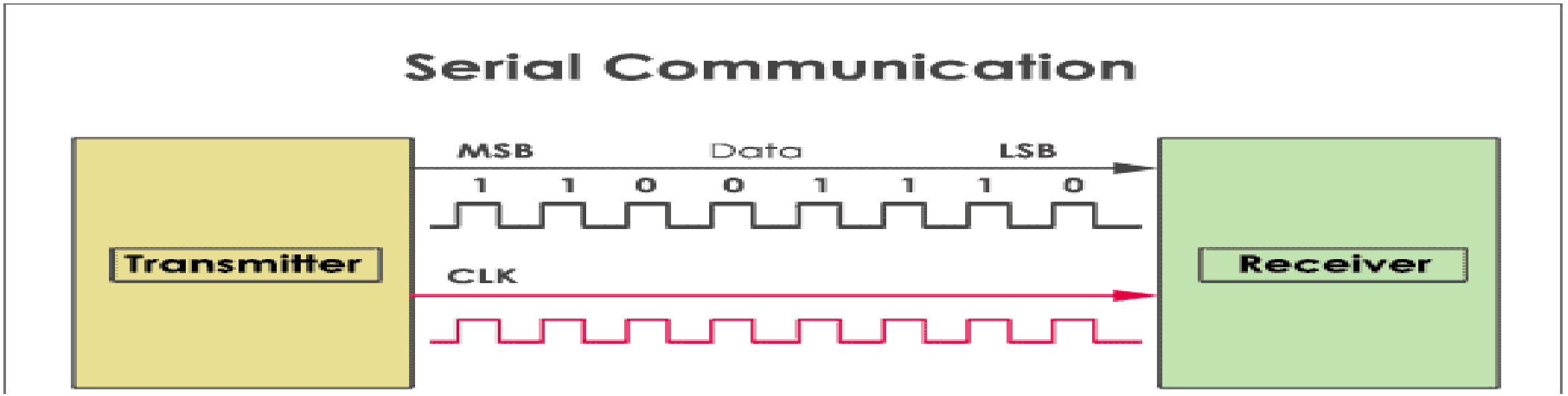
- Little Endian and
- Big Endian.

Take this example to understand the concept of endianness. Suppose, we have a 32-bit hexadecimal data **ABCD87E2**. How is this data stored in memory?



Data transfer can happen in two ways**. They are serial communication and parallel communication.** **Serial communication** is a technique used to send data bit by bit using a two-wires i.e. transmitter (sender) and receiver.

For example, to send an 8-bit binary data **11001110** from the transmitter to the receiver. But, which bit goes out first? Most Significant Bit – MSB (7^{th} bit) or Least Significant Bit- LSB (0^{th} Bit). We cannot say. Here I am considering LSB is moving first (for little Endian).

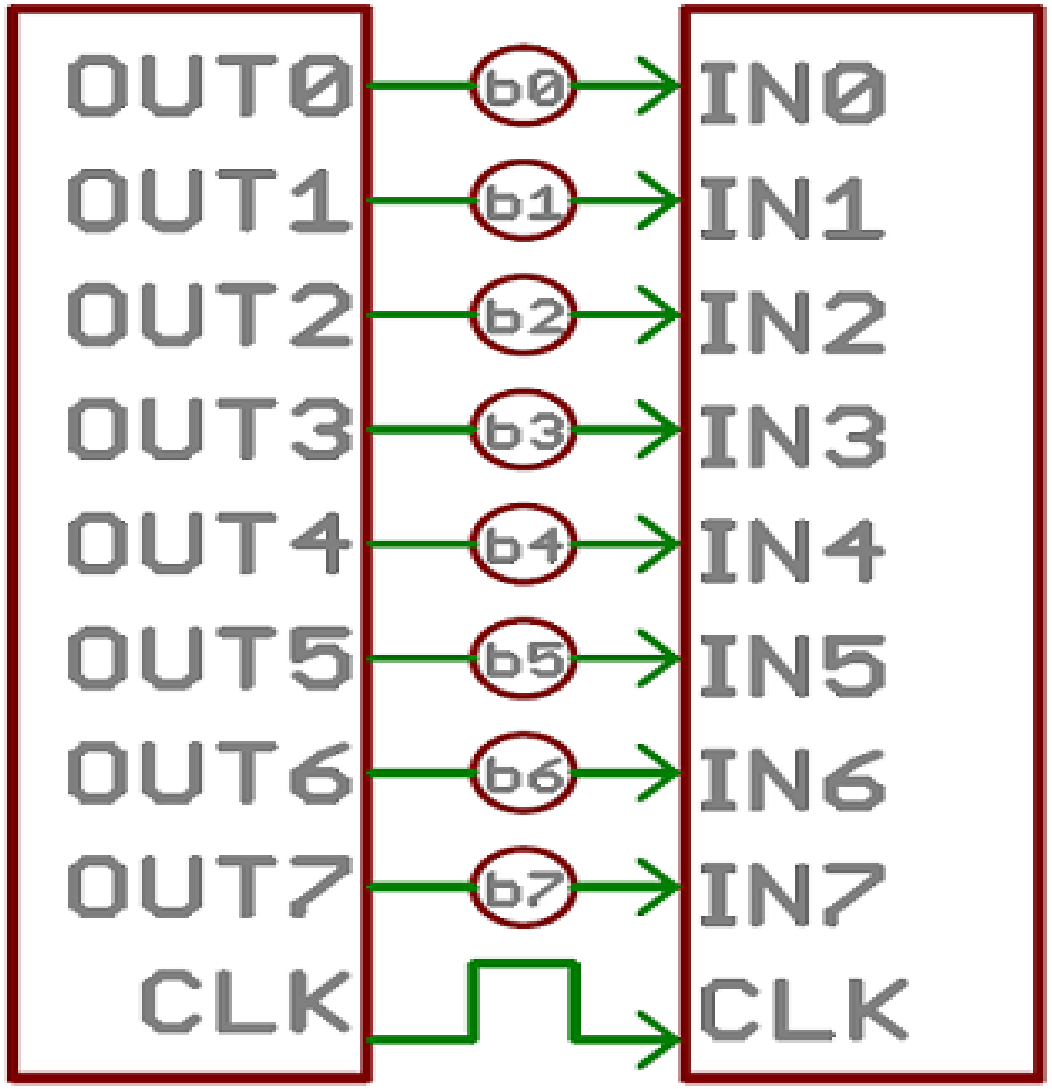


Difference between Serial and Parallel communication

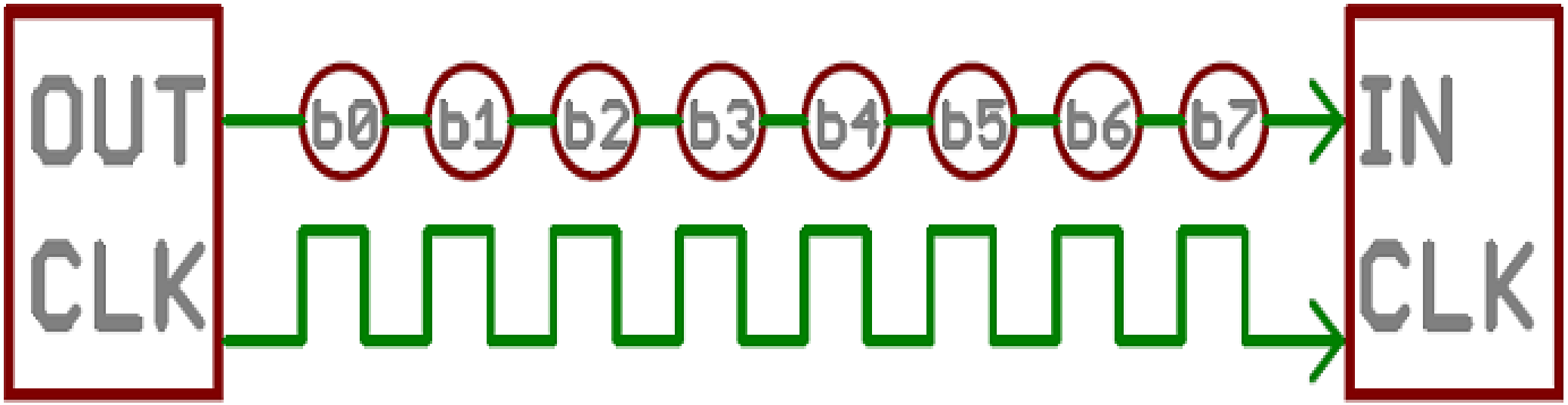
- Serial communication sends only one bit at a time.
- So, these require fewer I/O (input-output) lines.
- Hence, occupying less space and more resistant to cross-talk.
- The main advantage of serial communication is, the cost of the entire embedded system becomes cheap and transmits the information over a long distance.
- Serial transfer is used in DCE (Data communication Equipment) devices like a modem.

- In parallel communication, a chunk of data (8,16 or 32 bit) is sent at a time.
- So, each bit of data requires a separate physical I/O line.
- The advantage of parallel communication is it is fast but its drawback is it use more number of I/O (input-output) lines.
- Parallel transfer is used in PC (personal computer) for interconnecting CPU (central processing unit), RAM (random access memory), modems, audio, video and network hardware.

Serial Communication	Parallel Communication
Sends data bit by bit at one clock pulse	Transfers a chunk of data at a time
Requires one wire to transmit the data	Requires ‘n’ number of lines for transmitting ‘n’ bits
Communication speed is slow	Communication speed is fast
Installation cost is low	Installation cost is high
Preferred for long distance communication	Used for short distance communication
Example: Computer to Computer	Computer to multi function printer



An 8-bit data bus, controlled by a clock, transmitting a byte every clock pulse. 9 wires are used.



Example of a serial interface, transmitting one bit every clock pulse. Just 2 wires required!

Clock Synchronization

For efficient working of serial devices, the clock is the primary source. Malfunction of the clock may lead to unexpected results. The clock signal is different for each serial device, and it is categorized as synchronous protocol and asynchronous protocol.

Synchronous serial interface

All the devices on *Synchronous* serial interface use the single CPU bus to share both clock and data. Due to this fact, data transfer is faster. The advantage is there will be no mismatch in baud rate. Moreover, fewer I/O (input-output) lines are required to interface components. Examples are I2C, SPI etc.

Asynchronous serial interface

The *asynchronous* interface does not have an external clock signal, and it relies on four parameters namely

1.Baud rate control

2.Data flow control

3.Transmission and reception control

4.Error control.

Asynchronous protocols are suitable for stable communication. These are used for long distance applications. Examples of asynchronous protocols are RS-232, RS-422, and RS-485.

Baud rate is the speed of transferring data from the transmitter to a receiver in the form of bits per second.

Serial communication Working

- Advanced CPU such as microcontroller and Microprocessor make use of serial communication to communicate with the external world as well as on the chip peripherals.
- To get familiar, let us take a simple example.
- Suppose, you want to send a file present in your laptop to smartphone. How would you send? Probably using Bluetooth or WiFi protocol, Right.

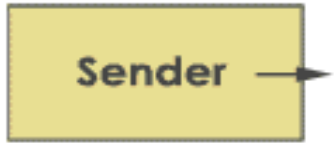
So, here are the steps to establish the serial communication

1.Add the connection.

In the first step, your laptop will search for devices nearby 100m and will list out the devices found. This process is often called roaming.

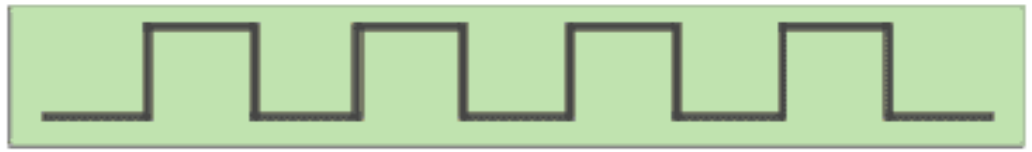
2.Select the device you want to communicate.

To connect to your mobile, the pairing has to be done. The default configuration is already present in the software. So no need to configure the baud rate manually. Beyond this, there are four unknown rules. They are baud rate, data bit selection (framing), start-stop bit, and parity.



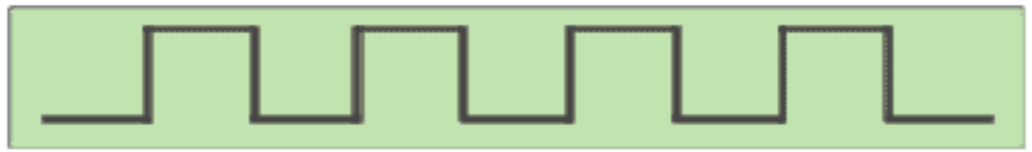
Rule 1: Baud Rate

No.of bits transmitted per second from sender to receiver.

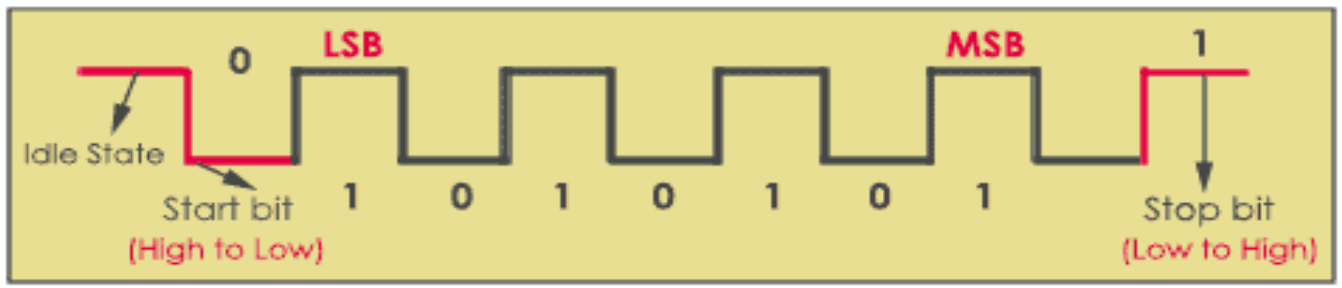


Rule 2: Data length selection

7 bit data sent from sender to receiver.



Rule 3: Synchronization



Start bit - Indicated by ZERO **Stop bit - Indicated by ONE**

Rule 4: Error Checking

Parity bit is '1' for even number of binary ones and '0' for odd number of binary ones. According to rule 3 it is set to 1.

Baud rate:-

- **Baud rate** is the speed of transferring data from the transmitter to a receiver in the form of bits per second. Some of the standard baud rates are 1200, 2400, 4800, 9600, 57600.
- You have to set the same baud rate on both sides (Mobile and Laptop).

Note: The Higher a baud rate, more data can be transferred in less amount of time.

Framing

Framing shows how many data bits you want to send from the host device (Laptop) to mobile (receiver). Is it 5, 6, 7, or 8 bits? Mostly many devices, 8 bits are preferred. After selecting the8-bit data chunk, endianness has to be agreed by the sender and receiver.

Synchronization

Transmitter appends **synchronization bits (**1 ***Start*** bit and 1 or 2 ***Stop*** bit) to the original data frame. Synchronization bits help the receiver to identify the start and end of the data transfer. This process is known as ***asynchronous data transfer***.

Error Control

Data corruption may happen due to external noise at the receiver end. The only solution to get the stable output is to check the **Parity**.

If the binary data contains an even number of **1's** it is known as ***even parity*** and the Parity bit is set to '**1**'. If the binary data include an odd number of **1's**, it is called ***odd parity***, and now parity bit is set to '**0**'.

Asynchronous Serial Protocols

The most common question that will come to mind when you start working on the embedded system is why to use Asynchronous protocols?

•To move around the information at a longer distance and

•For more reliable data transfer.

Some of the asynchronous communication protocols are:

RS-232 protocol

•RS232 is the first serial protocol used for connecting modems for telephony. RS stands for *Recommended Standard*, and now it has changed to EIA (*Electronic Industries Alliance*) / TIA (*Telecommunication Industry Association).*

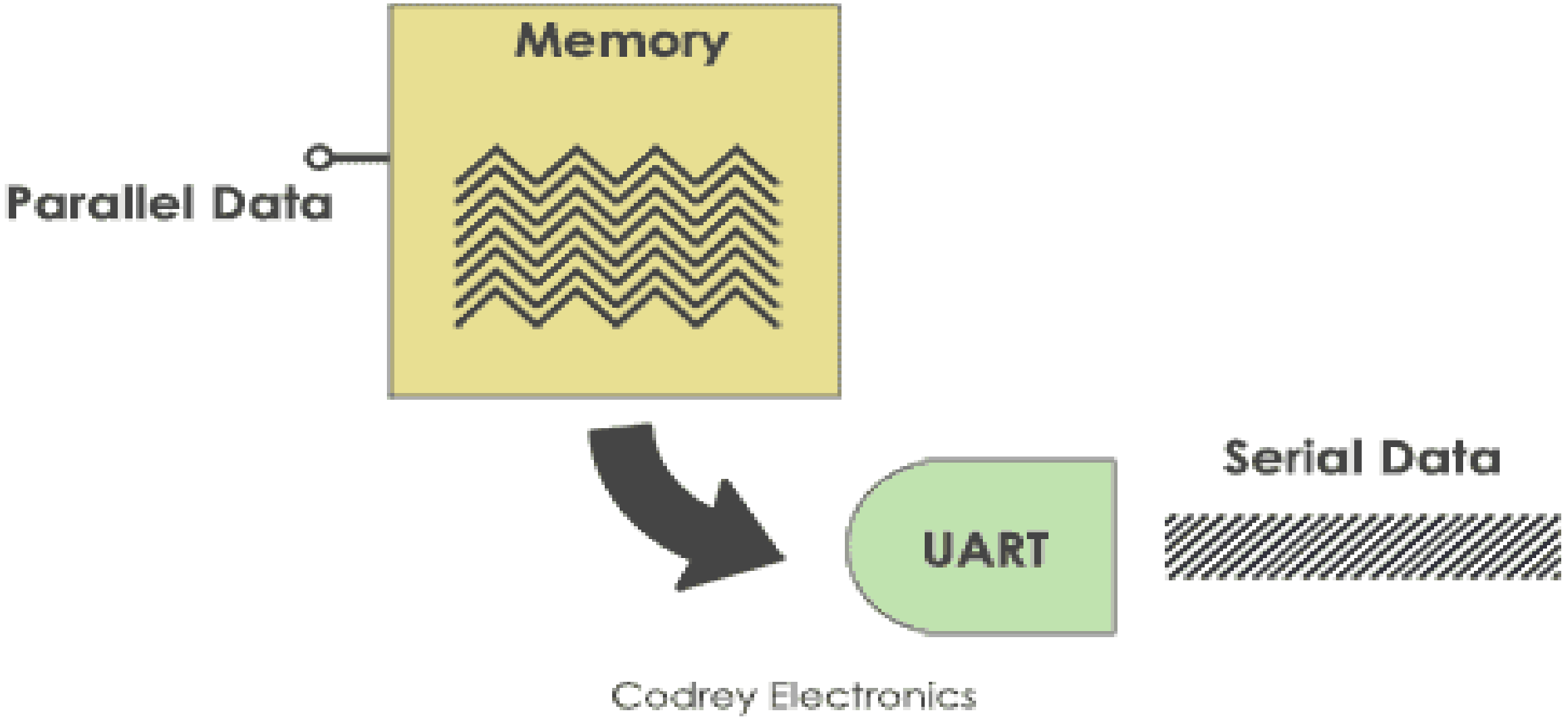
•It is also used in modem, mouse, and CNC (computed numerical computing) machines. You can connect only a single transmitter to a single receiver.

•It supports full duplex communication and allows baud rate up to 1Mbps.

•Cable length is limited to 50 feet.

As you know, the data stored in the memory are in the form of bytes. You may have a doubt How is the byte-wise data converted to binary bits? The answer is a Serial port. The serial port has an internal chip called **UART**. **UART is an acronym for Universal Asynchronous Receiver Transmitter** which converts the parallel data (byte) into the bitwise serial form.

RS232 Serial Port

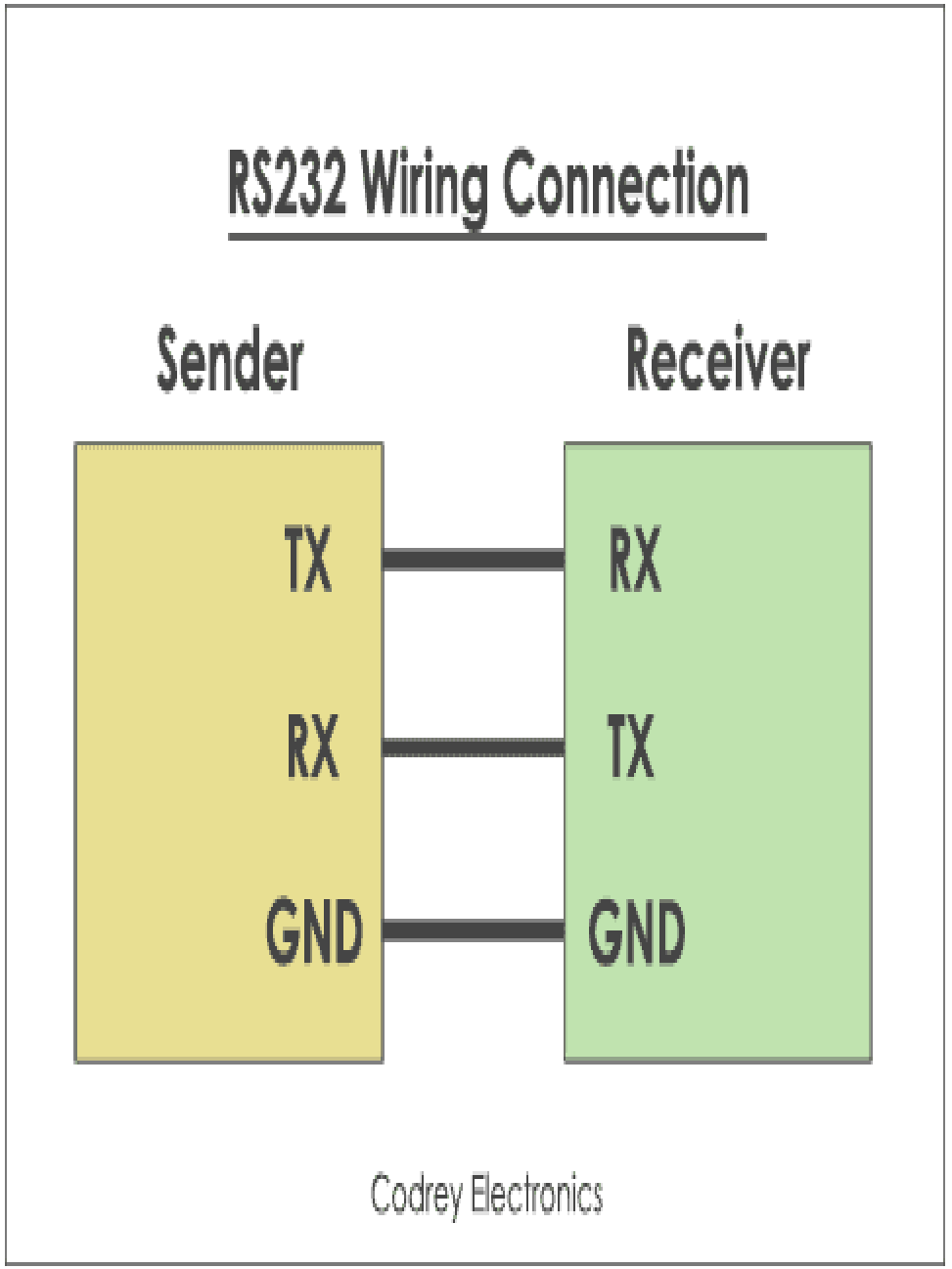


Codrey Electronics

RS-232 Wiring Connection

The RS232 serial port has nine pins, male or female type models. *RS 232C serial communication interface* is the later version of RS232.

All the features present in RS232 is present in the RS232C model except it has 25 pins. Out of 25 or 9 pins, we use only three pins for the connection of terminal devices.



- RS-232 is a standard communication protocol for connecting computers and their peripheral devices to enable serial data exchange.
- In simple terms, RS232 represents the voltage for the path used for data exchange between the devices. It determines the common voltage and signal level, common pin wire configuration and minimum, amount of control signals.
- RS232 represents the signals connecting between DTE and DCE. Therefore, **DTE represents Data Terminal Equipment** and an example for DTE is a computer.
- **DCE represents Data Communication Equipment** or **Data Circuit Terminating Equipmen**t and an example for DCE is a modem.
- RS232 was introduced in the 1960s and was initially referred to as EIA Recommended Standard 232.
- RS232 is one of the first serial communication standards with provided simple connectivity and compatibility across multiple manufacturers.
- The DTEs in RS32 are electromechanical typewriters and DCEs are modems.

- The RS-232 interface works in a mixture with UART (universal asynchronous receiver/transmitter). It is a part of an integrated circuit integrated within the processor or controller. It creates bytes and sequentially sends the single bits in a frame.
- A frame is a defined structure, carrying a meaningful sequence of bits or bytes of data.
- It has a start bit followed by 8 data bits, a parity bit, and a stop bit. Once data is modified into bits separate line drivers are used to changing the logic level of UART to RS-232 logic.
- Finally, the signals are shared along the interface cable at the particular voltage level of RS-232. The data is transmitted serially on RS232.
- Each bit is transmitted one after the other. This mode of transmission needed that the receiver is aware of when the actual data bits are appearing to synchronize itself with arriving data. Therefore, logic 0 is sent as a start bit.

- The start bit in the frame signals the receiver that a new character is arriving. Once the receiver acknowledges the next five to eight bits are sent which defines the character.
- This is followed by the parity bit used for error detection. The parity bit can determine an even or an odd number of ones in the set of bits.
- For error detection, it can add a more bit to the data word.
- The transmitter evaluates the value of the bit based on the data sent and the receiver also implements the same computation.
- It tests the parity value to the computed value. The stop bit supports the receiver to recognize the end of the message. The start bit continually has space value and the stop bit has mark value.
- This generates a framing error condition in the receiving UART. The device then attempts to resynchronize on more incoming bits.
- At the other end again the line driver interface changes it into UART compatible logic levels. At the destination, a second UART re-assembles the bits into bytes.
- This is how RS232 creates the data exchange compatible and reliable.

Electrical Specifications

Let us discuss the electrical specifications of RS232 given below:

•**Voltage Levels:** RS232 also used as ground & 5V level. Binary 0 works with voltages up to +5V to +15Vdc. It is called as 'ON' or spacing (high voltage level) whereas Binary 1 works with voltages up to -5V to -15Vdc. It is called as 'OFF' or marking (low voltage level).

•**Received signal voltage level:** Binary 0 works on the received signal voltages up to +3V to +13 Vdc & Binary 1 works with voltages up to -3V to -13 Vdc.

•**Line Impedances:** The impedance of wires is up to 3 ohms to 7 ohms & the maximum cable length are 15 meters, but new maximum length in terms of capacitance per unit length.

•**Operation Voltage:** The operation voltage will be 250v AC max.

•**Current Rating:** The current rating will be 3 Amps max.

•**Dielectric withstanding voltage:** 1000 VAC min.

•**Slew Rate:** The rate of change of signal levels is termed as Slew Rate. With its slew rate is up to 30 V/microsecond and the maximum bitrate will be 20 kbps.

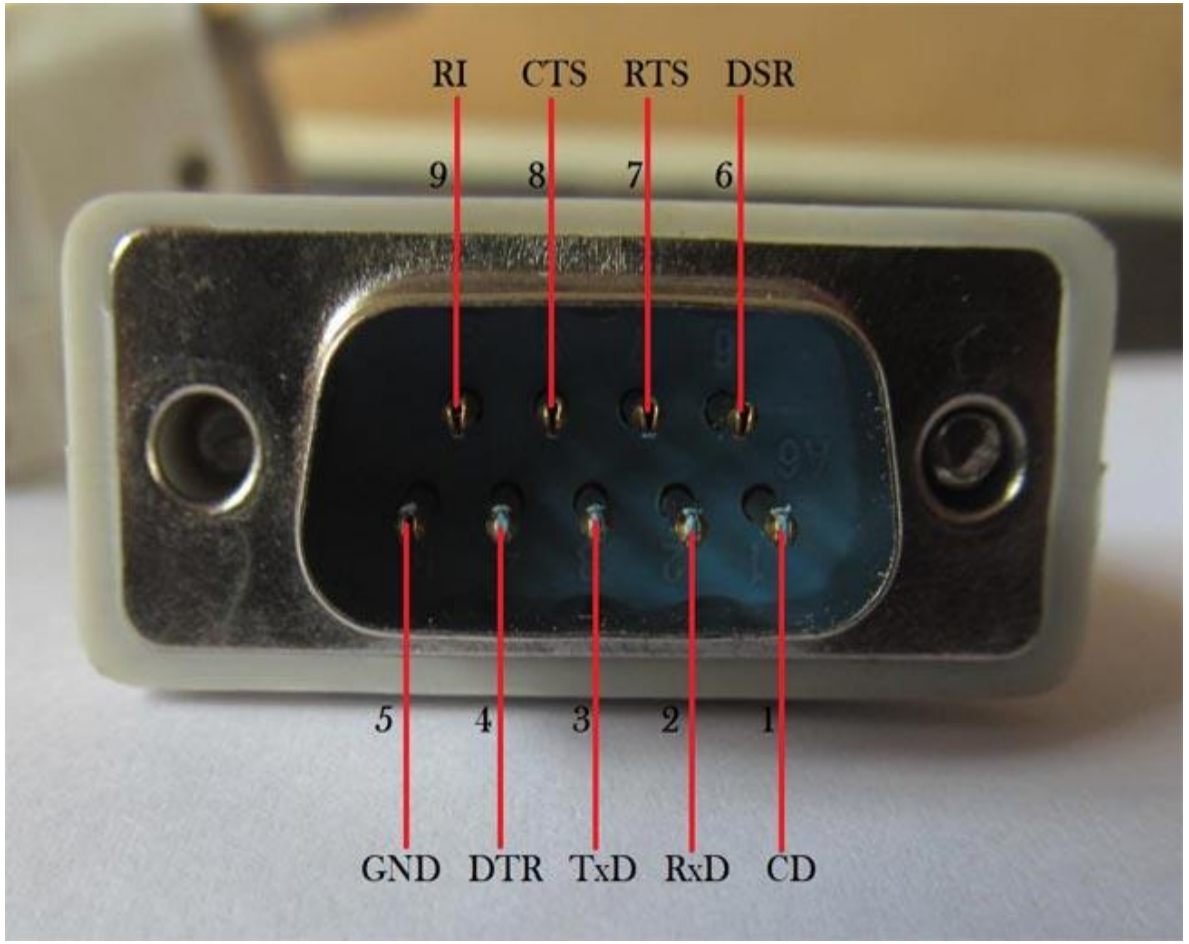
Mechanical Specification

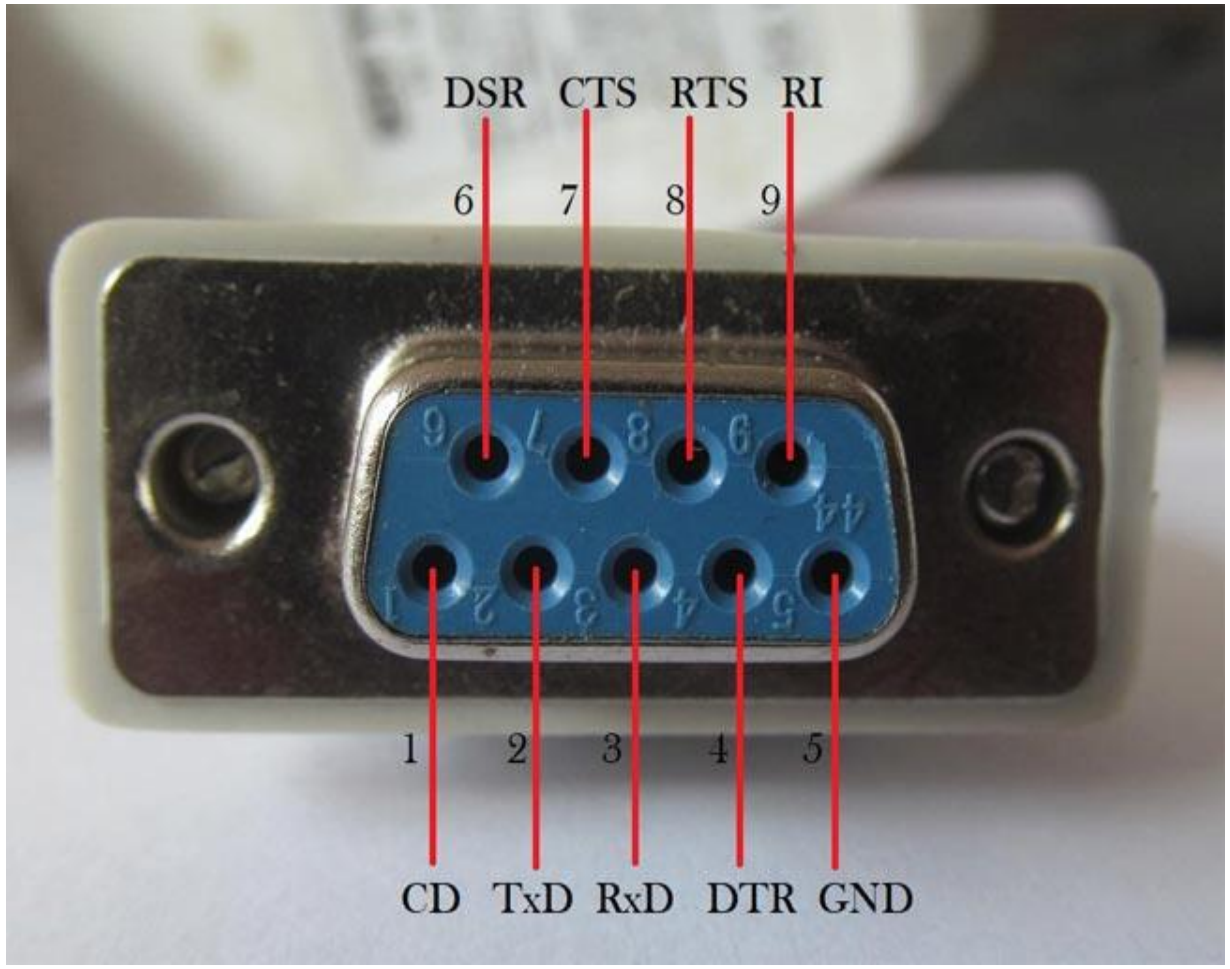
- For mechanical specifications, we have to study about two types of connectors that is **DB-25** and **DB-9**.
- In DB-25, there are 25 pins available which are used for many of the applications, but some of the applications didn't use the whole 25 pins. So, the 9 pin connector is made for the convenience of the devices and equipment.

Now, here we are discussing the **DB-9** pin connector which is used for connection between microcontrollers and connector.

- These are of two types: **Male Connector (DTE)** & **Female Connector (DCE)**.
- There are 5 pins on the top row and 4 pins in the bottom row.
- It is often called **DE-9 or D-type connector.**

Pin Structure of DB-9 Connector:



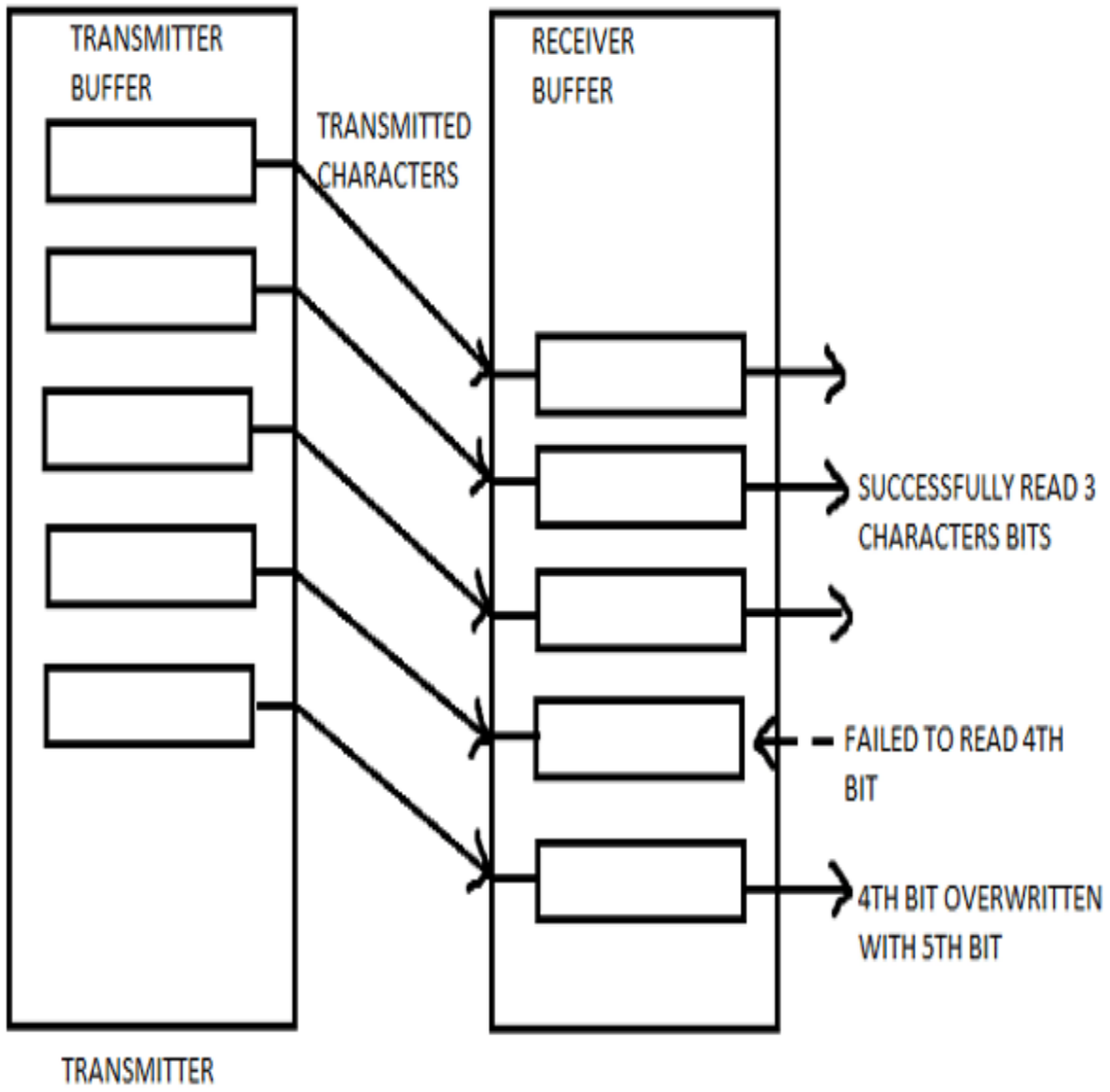


PIN No.	Pin Name	Pin Description
1	CD (Carrier Detect)	Incoming signal from DCE
2	RD (Receive Data)	Receives incoming data from DTE
3	TD (Transmit Data)	Send outgoing data to DCE
4	DTR (Data Terminal Ready)	Outgoing handshaking signal
5	GND (Signal ground)	Common reference voltage
6	DSR (Data Set Ready)	Incoming handshaking signal
7	RTS (Request to Send)	Outgoing signal for controlling flow
8	CTS (Clear to Send)	Incoming signal for controlling flow
9	RI (Ring Indicator)	Incoming signal from DCE

- **What is Handshaking?**
- *How can a transmitter, transmits and the receiver receives data successfully. So, the Handshaking defines, for this reason.*
- Handshaking is the process which is used to transfer the signal from DTE to DCE to make the connection before the actual transfer of data. The messaging between transmitter & receiver can be done by handshaking.
- There are **3 types of handshaking processes** named as:-

No Handshaking:

- If there is no handshaking, then DCE reads the already received data while DTE transmits the next data.
- All the received data stored in a memory location known as receiver's buffer. This buffer can only store one bit so receiver must read the memory buffer before the next bit arrives.
- If the receiver is not able to read the stored bit in the buffer and next bit arrives then the stored bit will be lost.
- As shown in below diagram, a receiver was unable to read the 4^{th} bit till the 5^{th} bit arrival and this result overriding of 4^{th} bit by 5^{th} bit and 4^{th} bit is lost.



Hardware Handshaking:

•It uses specific serial ports, i.e., RTS & CTS to control data flow.

•In this process, transmitter asks the receiver that it is ready to receive data then receiver checks the buffer that it is empty, if it is empty then it will give signal to the transmitter that I am ready to receive data.

•The receiver gives the signal to transmitter not to send any data while already received data cannot be read.

•Its working process is same as above described in handshaking.

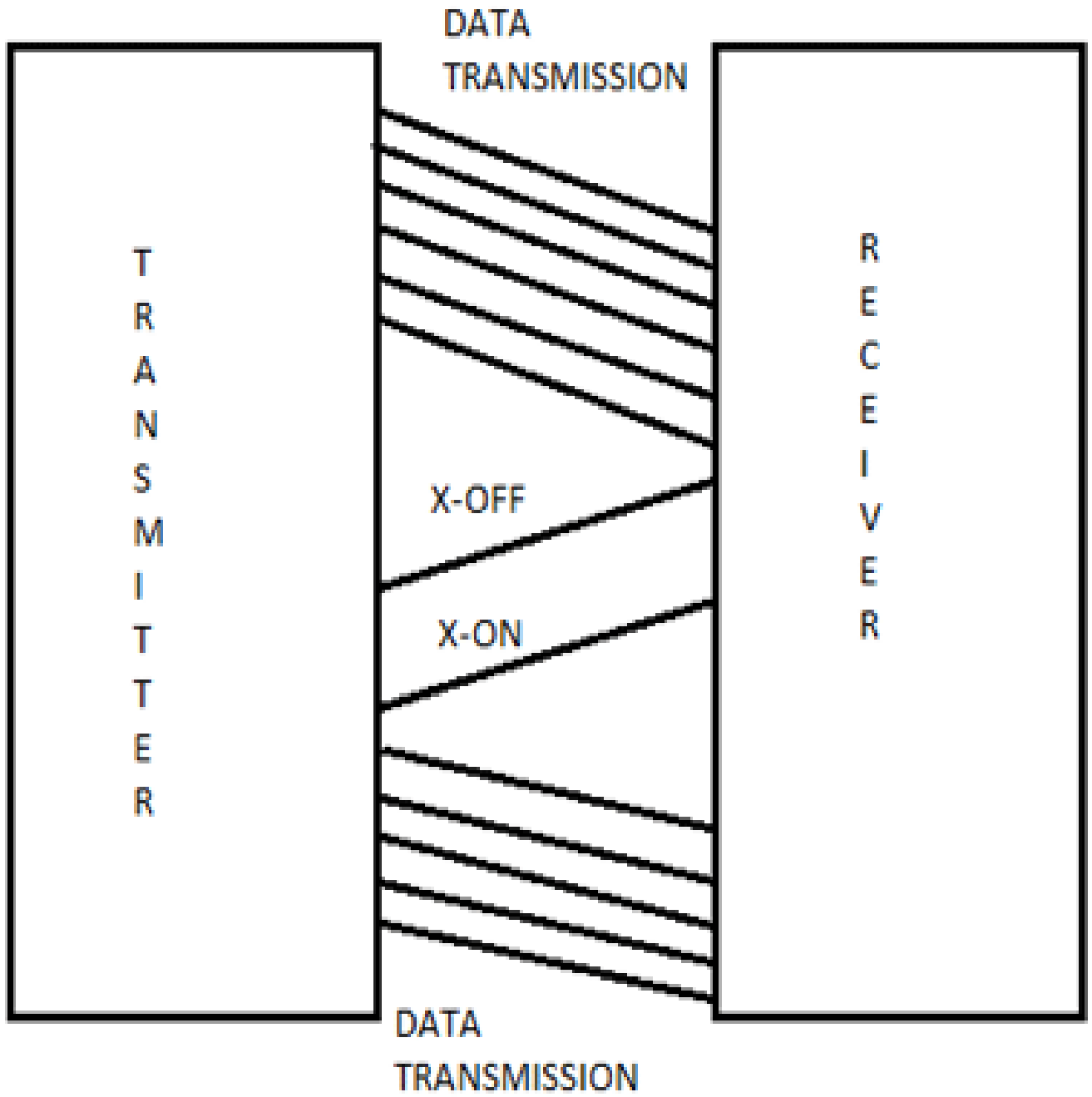
Software Handshaking:

•In this process, there are two forms, i.e., X-ON & X-OFF. Here, 'X' is the transmitter.

•X-ON is the part in which it resumes the data transmission.

•X-OFF is the part in which it pauses the data transmission.

•It is used to control the data flow and prevent loss during transmission.


DATA
TRANSMISSION
TRANSMITTER
RECEIVER
X-OFF
X-ON
DATA
TRANSMISSION

Applications of RS232 Communication

- RS232 serial communication is used in old generation PCs for connecting the peripheral devices like mouse, printers, modem etc.
- Nowadays, RS232 is replaced by advanced USB.
- It is also used in PLC machines, CNC machines, and servo controllers because it is far cheaper.
- It is still used by some microcontroller boards, receipt printers, point of sale system (PoS), etc.

Serial Communication Registers

SBUF Register:

- It is an 8-bit register used for serial communication. To transmit a byte of data via the TxD line, it must be placed in the SBUF register.
- When a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 RxD line.

SCON (serial control) register:

- It is an 8-bit register used to program start bit, stop bit, and data bits of data for framing, among other things. In this SM0 and SM1 are used to determine the mode.
- If (SM0,SM1) is (0,0) then it is mode 0, in this mode the serial port function as half duplex serial port with fixed baud rate.
- If (SM0,SM1) is (0,1) it is mode1,in this mode the serial port function as full duplex serial port with variable baud rate.
- If (SM0,SM1) is (1,0) then it is mode 2, in this mode the serial port function as full duplex serial port with a baud rate of either 1/32 or 1/64 of the oscillator frequency.
- If (SM0,SM1) is (1,1) then it is mode 3.

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. (Make it 0.)
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 hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Use of Baud rate in 8051

- Baud rate in 8051 is programmable.
- It is done with the help of Timer 1.
- Timer 1 must be programmed in mode 2, that is, 8-bit, auto-reload.
- We must make sure that the baud rate of the 8051 system matches the baud rate of the system used for communication.
- Relationship between the crystal frequency and the baud rate in the 8051 is as follows:
    - 8051 divides the crystal frequency by 12 to get the machine cycle frequency.
    - Load FD/FA/F4/E8 in TH1 to get 9600/4800/2400/1200 baud rate.

Table 10-4: Timer 1 TH1 Register Values for Various Baud Rates

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

Steps to send data serially in serial port communication

The following are the steps to send the data serially in serial port communication:

1. Set baud rate by loading TMOD register with the value 20H; this indicates timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with proper values to set baud rate for serial data transfer.
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. Transmit Interrupt(TI) is cleared by CLR TI instruction.
6. The character byte to be transferred serially is written into SBUF register.
7. The TI flag bit is monitored to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5.

#WAP to transfer letter “A” serially with baud rate of 4800.

```
        ORG 0000H
        SJMP MAIN
MAIN: MOV TMOD, #20H ; Configure TIMER-1 in MODE 2
        MOV TH1, #OFAH  ; load baud rate value in TH1
        MOV SCON, #50 H; configure MODE1 for serial communication 8-bit
        SETB TR1 ; start timer 1
AGAIN: MOV SBUF, #’A’ ; send the data byte to SBUF register
HERE : JNB TI, HERE ; wait for TI flag to get set
        CLR TI
        CLR TR1  ; Stop the TIMER
        SJMP AGAIN
        END
```

#WAP to transfer letter “A” serially with baud rate of 4800.

```
#include <reg51.h>
void main (void)
{
TMOD=0x20; // timer 1 mode 2
TH1 = 0xFA; // baud rate 4800
SCON = 0x50; //8 bit data, 1 stop bit , receiver enable
TR1=1 ; //start timer 1
        while (1)   // repeat the transmission
        {
        SBUF = A  // place the value in buffer
        while (TI == 0)
        TI = 0; // clear for next operation
        }
```

WAP to transfer “YES” serially at 9600 baud rate, 8 bit data, 1 stop bit, do this continuously.

```
SOL:
        MOV TMOD, #20H ; timer1, mode 2(auto reload)
        MOV TH1, #-3       ; buadrate 9600
        MOV SCON,#50h ; 8 bit, 1 stop, REN enable
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
TRANS : MOV SBUF, A  ; load SBUF
HERE : JNB TI, HERE ;wait for the last bit
        CLR TI
        RET
```

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 1,8-BIT auto-reload
      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;
   }
```

Importance of the TI flag:

The following steps are the importance of TI flag in 8051 serial port communication

- Check the TI flag bit, we know whether or not 8051 is ready to transfer another byte.
- TI flag bit is raised by the 8051 after transfer of data.
- TI flag is cleared by the programmer by instructions like “CLR TI”.
- Writing a byte into SBUF before the TI flag bit is raised, may lead to loss of a portion of the byte being transferred.

Steps to receive data serially in serial port communication

The following are the steps to receive the data serially in the serial port communication:

1. Set baud rate by loading TMOD register with the value 20H; this indicates setting of
timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with proper values to set baud rate.
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. Receive Interrupt(RI) is cleared by CLR RI instruction.
6. The RI flag bit is monitored 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 next character, go to step 5.

```
#WAP to receive bytes of data serially with baud rate of 4800.
        ORG 0000H
        SJMP MAIN
MAIN: MOV TMOD, #20H ; Configure TIMER-1 in MODE 2
        MOV TH1, #OFAH  ; load baud rate value in TH1
        MOV SCON, #50 H; configure MODE1 for serial
communication 8-bit
        SET TR1 ; start timer 1
HERE : JNB RI, HERE ; wait for character to come in
        MOV A, SBUF ; save incoming byte in A
        MOV F1, A ; send to port 1
        CLR RI              ; get ready to receive next byte
        SJMP HERE       ; keep getting data
        END
```

Importance of the RI flag:
The following point to the importance of RI flag in 8051 serial port communication:
1. It receives the start bit, next bit is the first bit of the character about to be received.
2. When the last bit is received, a byte is formed and placed in the SBUF register.
3. When the stop bit is received, it makes RI = 1 indicating that the entire character byte
has been received and can be written before being overwritten.
4. When RI=1, received byte is in the SBUF register, copy SBUF contents to a safe
place.
5. After the SBUF contents are copied, the RI flag bit must be cleared to 0.

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.

Solution:

```
#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;
         }
   }
```