Chapter -3

Serial Interfacing with Microprocessor Based System

Within a microcomputer data is transferred in parallel, because that is the fastest way to do it. For transferring data over long distances, however, parallel data transmission requires too many wires. Therefore, data to be sent long distances is usually converted from parallel form to serial form so that it can be sent on a single wire or pair of wires. Serial data received from a distant source is converted to parallel form so that it can easily be transferred on the microcomputer buses.

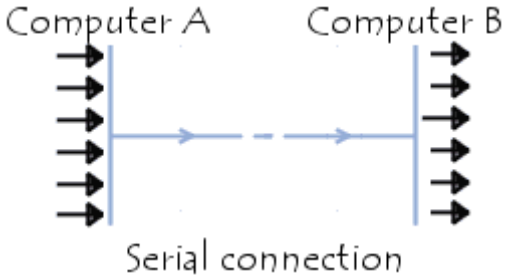
Advantages of Serial Data Transfer Over Parallel

- Longer data transmission in serial mode
  - Serial; 1 → -3V to -25V
    0 → +3V to +25V
  - Parallel; 1 → +5V
    0 → 0V
  - Voltage loss is not much a problem in serial communication.
- Serial transmission requires less number of wires than parallel and so cheaper to transmit data.
- Crosstalk is less of an issue because there are fewer conductors' compared to that of parallel cables.
- Many IC and peripherals have serial interface
- Clock skew between different cables is not an issue
- Serials can be clocked at higher data rate
- Serial cable can be longer than parallel
- Cheaper to implement

- But in serial mode of transfer, only one bit of a word is transferred at a time so that data transfer rate is very slow; it is the one of the demerit over parallel data transfer.

Serial Data Transmission

In a serial data transmission, the data are sent one bit at a time over the transmission channel. However, since most processors process data in parallel, the transmitter needs to transform incoming parallel data into serial data and the receiver needs to do the opposite.



In case of serial transmission data is sent in a serial form i.e. bit by bit on a single line. Also, the cost of communication hardware is considerable reduced since only a single wire or channel is require for the serial bit transmission. Serial data transmission is slow as compared to parallel transmission. Serial data can be sent synchronously or asynchronously.

Serial Synchronous Data Transmission

In serial synchronous data transmission, data is transmitted or received based on a clock signal. At a specific rate of data transmission, the transmitting device sends a data bit at each clock pulse. In order to interpret the data correctly, the receiving device must know the start and end of each data unit. The transmitter must know the number of data units to be transferred and the receiver must be synchronized with the data boundaries. Therefore, there must be synchronization between the transmitter and receiver. Usually one or more SYNC characters are used to indicate the start of each synchronous data stream or frame of data.

Transmitter sends a large block of data characters one after the other with no time between characters. Transmitting device sends data continuously to the receiving device. If the data is not ready to be transmitted, the line is held in marking condition. To indicate the start of transmission, the transmitter sends out one or more SYNC characters or a unique bit pattern called a flag, depending on the system being used. The receiving device waits for data, when it finds the SYNC characters or the flag then starts interpreting the data which shifts the data following the SYNC characters and converts them to parallel form so they can be read in by a computer.

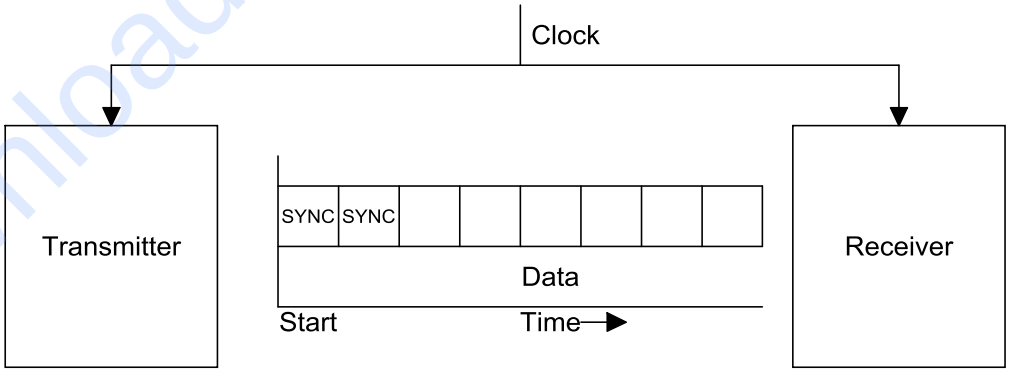


Fig: Synchronous Serial Transmission Format

Synchronous transmission has the advantage that the timing information is accurately aligned to the received data, allowing operation at much higher data rates. It also has the advantage that the receiver tracks any clock drift which may arise (for instance due to temperature variation). The penalty is however a more complex interfaces design, and potentially a more difficult interface to configure (since there are many more interface options).
Data transmission takes place without any gap between two adjacent characters. However data is send block by block. A block is a continuous steam of characters or data bit pattern coming at a fixed speed. You will find a SYNC bit pattern between any two blocks of data and hence the data transmission is synchronized. Synchronous communication is used generally when two computers are communicating to each other at a high speed or a buffered terminal is communicating to the computer.

Advantages and Disadvantages of Synchronous Communication
Main advantage of Synchronous data communication is the high speed. The synchronous communications require high-speed peripherals/devices and a good-quality, high bandwidth communication channel.
The disadvantage includes the possible in accuracy. Because when a receiver goes out of Synchronization, loosing tracks of where individual characters begin and end. Correction of errors takes additional time.

Serial Asynchronous Data Transmission
The receiving device does not need to be synchronized with the transmitting device. The transmitting device can send one or more data units when it is ready to send data. Each data unit must be formatted i.e. must contain start and stop bits for indicating beginning and the end of data unit. It also includes one parity bit to identify odd or even parity of data. To send ASCII character, the framing of data should contain:

- 1 start bit: Beginning of data
- 8 bit character: Actual data transferred
- 1 or 2 stop bits: End of data

When no data is being sent, the signal line is in a constant high or marking state. The beginning of the data character is indicated by the line going low for 1 bit time and this bit is called a start bit. The data bits are then sent out on the line one after the other where the least significant bit is sent out first. Parity bit should contain to check for errors in received data. After the data bit and a parity bit, the signal line is returned high for at least 1 bit time to identify the end of the character, this always high bit is referred to as a stop bit. Some older systems use 2 stop bits.

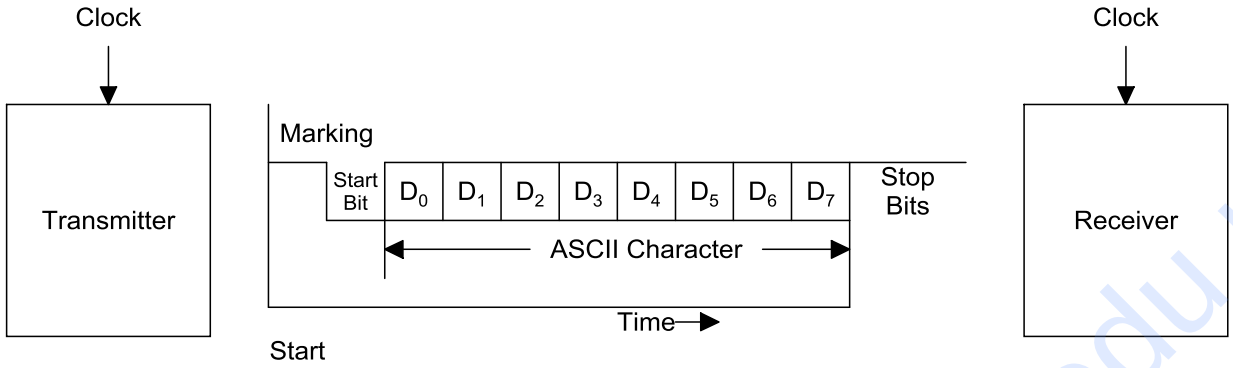


Fig: Asynchronous Serial Transmission Format

In asynchronous transmission each character is transmitted separately, that is one character at a time. The character (8-bits) is preceded by a start bit (1-bit), which tells the receiving end where the character coding begins, and is followed by a stop bit (1 or 2-bits), which tells the receiver where the character coding ends. There will be intervals of ideal time on the channel shown as gaps. Thus there can be gaps between two adjacent characters in the asynchronous communication scheme. In this scheme, the bits within the character frame (including start, parity and stop bits) are sent at the baud rate.

The START bit and STOP bit including gaps allow the receiving and sending computers to synchronize the data transmission. Asynchronous communication is used when slow speed peripherals communicate with the computer. The main disadvantage of asynchronous communication is slow speed transmission. Asynchronous communication however, does not require the complex and costly hardware equipments as is required for synchronous transmission.

Synchronous versus Asynchronous serial data transmission

S.N.	Parameter	Asynchronous	Synchronous
1.	Fundamental	Transmission does not based on clock signal	Transmission based on clock signal
2.	Data Format	One character at a time	Group of characters i.e. a block of characters
3.	Speed	Low (< 20 kbps)	High (> 20 kbps)
4.	Framing Information	Start and stop bits are sent with each character.	SYNC characters are sent with each character.
5.	Implementation	Hardware / Software	Hardware

Serial Data Unit (SDU) & Serialization

- SDU is a unit with 1 start bit, 8 data bits, 1 parity bit and 1 or 2 stop bits.
- Start bit always has a value of 0 & stop bits always have a value of 1.
- Following figure shows a SDU format; for asynchronous data transmission, sender and receiver must be set up to the same format.

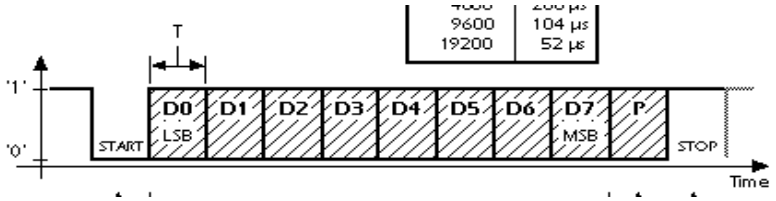


Fig: SDU or frame format

Transmitting SDU

- The interface chip has a transmitter hold register for transmitting data which first fetches the data bytes from CPU.
- According to the selected data format, the SDU logic puts the start bit in front of data bits; it then calculates the parity bit and appends it together with the stop bits to the data bits.
- Thus formed SDU is transferred into the transmitter shift register, which is operated by a clock source determined by baud rate and thus provides the individual bits at the serial output (LSB first). If no data, then the chip possesses a logical high level.

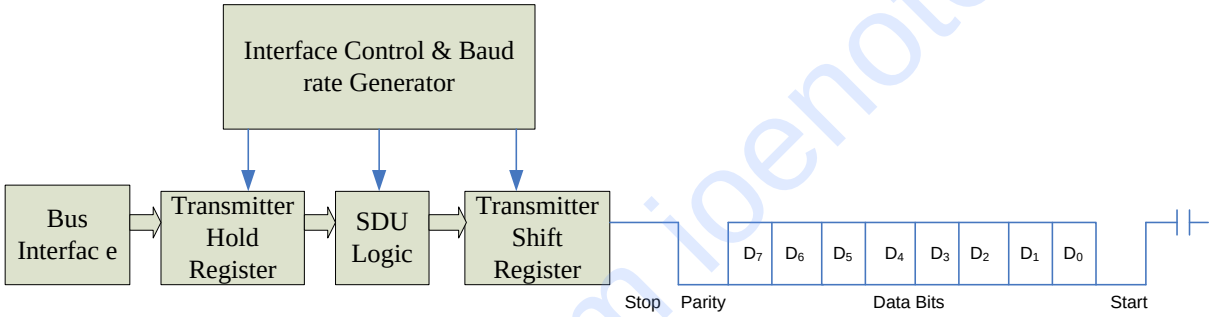


Fig: SDU at transmitter side

Receiving SDU

- Inverse reception process
- Start bit acts as trigger pulse & starts the receiver in the serial input chip.
- The SDU bits are loaded into the receiver shift register according to the phase of the setup baud rate.
- The receiver SDU logic then separates the start, parity, stop bits from the received SDU bits, calculates the parity of the data bits & compares it with the setup parity.
- Afterwards, the extracted data bits are transferred into the receiver buffer register from which they may be read out as the received data byte by the CPU.

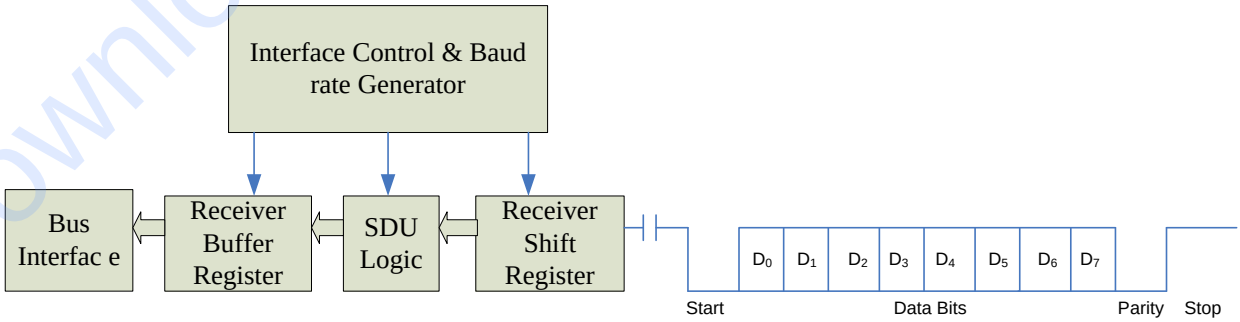


Fig: SDU at receiver Side

Errors in Serial Data Transfer

From the description of the transmission and reception process, it can be readily seen that transmitter and receiver must be set to the same baud rate. Additionally, the set data formats (i.e. number of data bits, parity, start and stop bits) must also coincide; otherwise the receiver may resemble possibly a different byte from that which the transmitter was passed for transmitting. Upon reception of an SDU, various errors may occur.

1. **Framing Error**
   Data does not fit in frame that data format and baud rate defined i.e. non-synchronized start / stop bit. Eg:- Change in no. of bits in receiving and transmitting end.
2. **Break Error**
   If the reception line is at logic low level for longer time than the SDU usually lasts, then the receiver assumes that the connection to the transmitter has broken. Unless the transmitter drives the line to a logical high level, no data is transferred.
3. **Overrun Error**
   If data arriving at the receiver is much faster than it can be read from the receiver buffer; the latter received byte overwrites the older data in the buffer.
4. **Parity Error**
   The calculated parity does not coincide with the set one. It may be due to the noise or a different set for parity at transmitter and receiver sides.
   - No parity
   - Even parity
   - Odd parity
   - Mark parity
   - Space parity

Error Checks in Data Communication

During transmission, various types of errors can occur such as data bits may change because of noise or can be misunderstood by the receiver due to different clocks between transmitter and receiver. These errors need to be checked; therefore, additional information for error checking is sent during the transmission. The receiver can check the received data against the error check information, and if an error is detected, the receiver can request the retransmission of that data segment or it can correct by proper coding techniques. Three methods are generally in common practice; they are parity check, checksum and cyclic redundancy check.

Parity Check

This is the simplest method of error checking which checks the characters by counting the number of 1s. In this method, D_7 of each ASCII code is used to transmit parity check information. Parity may be the even parity (having even number of 1s in a character) or the odd parity (having odd number of 1s in a character).

In an even parity system, when a character has an odd number of 1s, the bit D_7 is set to 1 and an even number of 1s is transmitted. On the other hand, in an odd parity system, when a character has an even number of 1s, the bit D_7 is set to 1 and an odd number of 1s is transmitted.

For an example, character to be sent is 'A' whose ASCII code is 41H (0100 0001) with two 1s. If the character is transmitted in an odd parity system, the bit D7 is set to 1 and if it is transmitted in an even parity system, the bit D7 is set to 0. Most of microprocessors are designed to detect

parity using the parity flag. However, the parity check cannot detect multiple errors in any given character.

Checksum

The checksum technique is used when blocks of data are transmitted. It involves adding all the bytes in a block without carriers. Then, the 2's complement of the sum (negative of the sum) is transmitted as the last byte. The receiver adds all the bytes, including the 2's complement of the sum; thus, the result should be zero if there is no error in the block.

Cyclic Redundancy Check (CRC)

This technique is based on mathematical relationships of polynomials. A stream of data can be represented as a polynomial that is divided by a constant polynomial, and the remainder, unique to that set of bits, is generated. The remainder is sent out as a check for errors. The receiver checks the remainder to detect an error in the transmission. This is a somewhat complex technique for error checking.

Baud Rate / Bit Rate

The difference between Bit and Baud rate is complicated and intertwining. Both are dependent and inter-related.

Bit Rate is how many data bits are transmitted per second.

A baud Rate is the number of times per second a signal in a communications channel changes. Bit rates measure the number of data bits (that is 0′s and 1′s) transmitted in one second in a communication channel. A figure of 2400 bits per second means 2400 zeros or ones can be transmitted in one second, hence the abbreviation "bps." Individual characters (for example letters or numbers) that are also referred to as bytes are composed of several bits.

A baud rate is the number of times a signal in a communications channel changes state or varies. For example, a 2400 baud rate means that the channel can change states up to 2400 times per second. The term "change state" means that it can change from 0 to 1 or from 1 to 0 up to X (in this case, 2400) times per second. It also refers to the actual state of the connection, such as voltage, frequency, or phase level).

The main difference between the two is that one change of state can transmit one bit, or slightly more or less than one bit, that depends on the modulation technique used. So the bit rate (bps) and baud rate (baud per second) have this connection:

If signal is changing every 10/3 ns then,

Baud rate = 1/10/3ns $= 3/10*10^9$ $= 3*10^8$

$= 300$ mbd

Note:
If 1 frame of data is coded with 1 bit then band rate and bit rate are same. Sometimes frame of data are coded with two or more bits then baud rate and bit rate are not same.

Simplex, Half Duplex and Full Duplex Data Communication

Simplex Mode

Simplex transmission allows data to travel only in a single, pre specified direction. An example from everyday life is doorbell the signal can go only from the button to the chime. Two other examples are television and radio broadcasting. The simplex standard is relatively uncommon for most types of computer-based telecommunications applications; even devices that are designed primarily to receive information, such as printers must be able to communicate acknowledgement signals back to the sender devices.

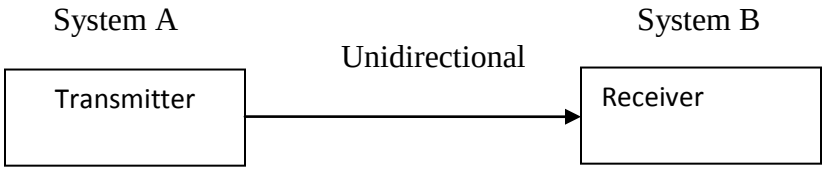


Fig: Simplex mode

Half Duplex Mode

It is a two way communication between two ports provided that only party can communicate at a time. In half duplex transmission messages can move in either direction, but only one way at a time. The press to talk radio phones used in police cars employs the half-duplex standard; only one person can talk at a time. Often the line between a desktop workstation and a remote CPU conforms to the half duplex patterns as well. If another computer is transmitting to a workstation, the operator cannot send new messages until the other computer finishes its message to acknowledge an interruption.

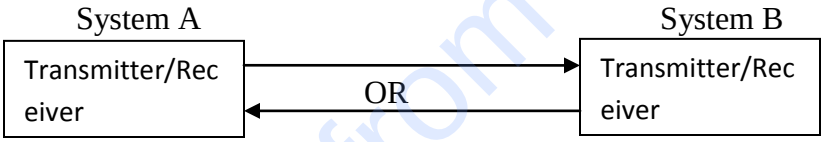


Fig:Half Duplex mode

Full Duplex Mode

It provides simultaneous two way transmission without the intervening stop-and-wait aspect of half duplex. Full duplex is widely used in applications requiring continuous channels usage. Full duplex transmission works like traffic on a busy two way street the flow moves in two directions at the same time. Full-duplexing is ideal for hardware units that need to pass large amounts of data between each other as in mainframe-to-mainframe communications.

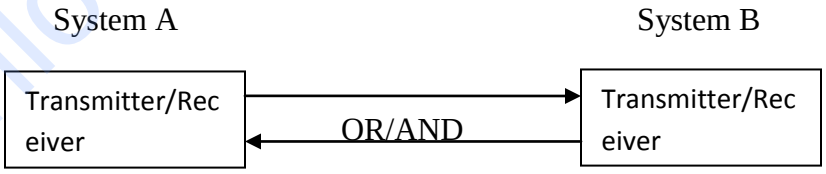


Fig: Full Duplex mode

Standards in Serial I/O

The serial I/O technique is commonly used to interface different peripheral terminals such as printers, modems with microcomputers which are designed and manufactured by various manufacturers. Therefore, a common understanding must exist, among various manufacturing and user groups that can ensure compatibility among different equipment. The standard is defined as the understanding which is accepted in industry and by users. A standard is normally defined by a professional organizations such as IEEE (Institute of Electrical and Electronics Engineers), EIA (Electronic Industries Association) as a de jure standard. However, a widespread practice can become a de facto standard.

In serial I/O, data can be transmitted as either current or voltage. When data are transmitted with current signal such for teletype equipment, 20 mA (or 60 mA) current loops are used. When a teletype is marking or at logic 1, current flows; when it is at logic 0 (space), the current flow is interrupted. The advantage of the current loop method is that signals are relatively noise-free and are suitable for transmission over a distance.

When data are transmitted with voltage signal, there are various standards which are explained in this section.

RS-232C

Serial transmission of data is used as an efficient means for transmitting digital information across long distances, the existing communication lines usually the telephone lines can be used to transfer information which saves a lot of hardware. RS-232C is an interface developed to standardize the interface between data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data exchange. Modem and other devices used to send serial data are called data communication equipment (DCE). The computers or terminals that are sending or receiving the data are called data terminal equipment (DTE).

RS- 232C is the interface standard developed by electronic industries Association (EIA) in response to the need for the signal and handshake standards between the DTE and DCE. RS-232C has following standardize features.

- It uses 25 pins (DB – 25P) or 9 Pins (DE – 9P) standard where 9 pins standard does not use all signals i.e. data, control, timing and ground.
- It describes the voltage levels, impendence levels, rise and fall times, maximum bit rate and maximum capacitance for all signal lines.
- It specifies that DTE connector should be male and DCE connector should be female.
- It can send 20kBd for a distance of 50 ft.
- The voltage level for RS-232 are:
    - A logic high or 1 or mark, -3V to -15V
    - A logic low or 0 or space, +3v to +15v
- Normally ±12V voltage levels are used

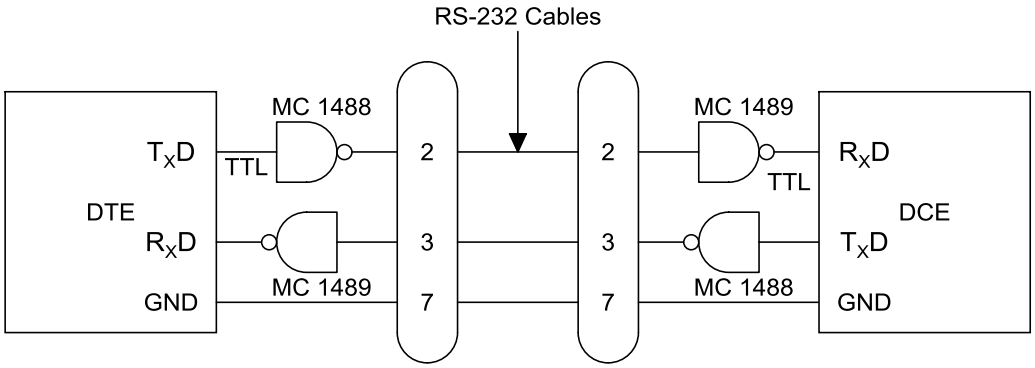


Fig: Connection of DTE and DCE through RS-232C Interface

- Mc1488 line driver converts logic 1 to -9V
  Logic 0 to +9v
- Mc1489 line receiver converts RS – 232 to TTL
- Signal levels of RS-232 are not compatible with that of the DTE and DCE which are TTL signals for that line driver such as M 1488 and line receiver MC1489 are used.

RS- 232 signals used in handshaking:

Signal Flow	DE-9P	DB-25P	Signal	Description
-	1	-	Protective Ground	-
DTE to DCE	3	2	TxD	Transmitted Data
DCE to DTE	2	3	RxD	Received Data
DTE to DCE	7	4	$\overline{RTS}$	Request To Send
DCE to DTE	8	5	$\overline{CTS}$	Clear To Send
DCE to DTE	6	6	$\overline{DSR}$	Data Set Ready
Common Ref	5	7	GND	Signal Ground
DCE to DTE	1	8	$\overline{DCD}$	Data Carrier Detect
DTE to DCE	4	20	$\overline{DTR}$	Data Terminal Ready
DCE to DTE	9	22	RI	Ring Indicator
DCE to DTE	-	23	DSRD	Data Signal Rate Detector

Data Terminal Ready (DTR):
After the terminal power is turned on and terminal runs any self checks, it asserts data terminal ready (DTR') signal to tell the modem that it is ready.

Data Set Ready (DSR):
When the MODEM is powered up and ready to transmit or receive data, it will assert data set ready (DSR') to the terminal. Under manual control or terminal control, modem then dials up the computer. If the computer is available, it will send back a specified tone.

Request to send (RTS):
When a terminal has a character ready to send, it will assert a request-to-send (RTS') signal to the modem.

Data Carrier Detect (DCD):
The modem will then assert its data-carrier-detect (DCD') signal to the terminal to indicate that it has established connection with the computer.

Clear to send (CTS):
When the modem is fully ready to receive data, it asserts the clear-to-send (CTS') signal back to the terminal.

Ring indicator (RI):
It indicates that a ring has occurred at modem. Deactivating DTR or DSR breaks the connection but RI works independently of DTR i.e. a modem may activate RI signal even if DTR is not active.

Transmitted Data (TxD):
The terminal then sends serial data characters to the modem.

Received Data (RxD):
Modem will receive data from terminal through this line.

Data Signal Rate Detect (DSRD):
It is used for switching different baud rate.

Digital Data Transmission Using Modem and standard Phone Lines
Standard telephone system can be used for sending serial data over long distances. However, telephone lines are designed to handle voice, bandwidth of telephone lines ranges from 300 HZ to 3400 HZ. Digital signal requires a bandwidth of several megahertz. Therefore, data bits should be converted into audio tones, this is accomplished through modems.

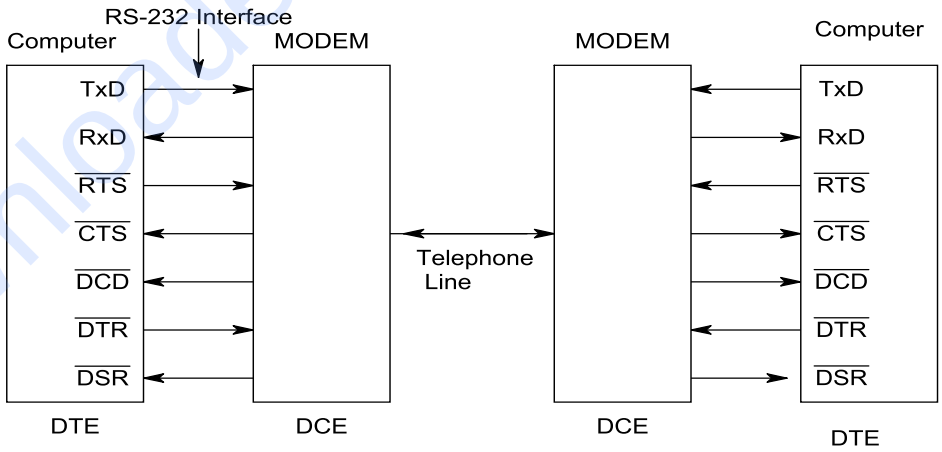


Fig: Digital Data transmission using MODEM and Telephone Line

- DTE asserts $\overline{DTR}$ to tell the modem it is ready.
- Then DCE asserts $\overline{DSR}$ signal to the terminal and dials up.
- DTE asserts $\overline{RTS}$ signal to the modem.
- Modem then asserts $\overline{DCD}$ signal to indicate that it has established connection with the computer.
- DCE asserts $\overline{CTS}$ signals, then DTE sends serial data.
- When sending completed, DTE asserts $\overline{RTS}$ high, this causes modem to un assert its $\overline{CTS}$ signal and stop transmitting similar handshake taken between DCE and DTE other side.
- To communicate from serial port of a computer to serial port of another computer without modem, null-modem is used.

Simplex, Half Duplex and Full Duplex Operation Using RS-232 port

Simplex Connection for RS-232C
There are two possibilities; data transfer from DTE to DCE or vice versa.

From DTE to DCE
The DTE transfers data to the DCE via the TxD line. The RxD line is not connected. The DCE does not use RTS or DTE holds RTS signal active all the time. The DCE always outputs an inactive DCD signal as it can receive data from DTE and transfer it to destination. By means of DTR signal, DTE can indicate DCE that it is ready for operation as usual and may activate or disable DCE. RI signal has no meaning because normally transmitter calls receiver.

Form DCE to DTE
In this case, only the DCE transfers data to the DTE via RxD line. The TxD line is not connected. The DCE does not either use RTS or CTS signal or holds them constantly at an active level. The DCE may output an active DCD signal as it can detect a carrier signal from an external device and transfer data to DTE. By means of DTR, the DTE can indicate that it is ready for operation and it can activate or disable the DCE as usual. The RI signal has a meaning as external device may call DTE via DCE.

Half Duplex Connection for RS-232C
On a half duplex connection, both the DTE and DCE can operate as receiver and transmitter, but only one data line is available which is alternatively used by the DTE and DCE. The TxD and RxD lines output and receive data respectively in a strictly ordered manner for assigning the roles as receiver and transmitter between DTE and DCE; the handshake control signals RTS and CTS are used. If a DTE device wants to act as a transmitter, then it activates the RTS signal and waits for an acknowledgement of other DCE device by means of CTS signal. Now, data can be exchanged while DTE acting as transmitter and DCE as receiver otherwise DCE may operate as transmitter and DTE as receiver.

Full Duplex Connection for RS-232C
Most microcomputer modems are full duplex, and transfer data in both directions simultaneously; thus DTE and DCE act simultaneously as receiver and transmitter. The RTS and CTS signals are meaningless and are usually not used or are always active. Further, the DSR

signal is also enabled all the time on most modems but on some DCEs, DSR may be active only if preparations for calling destination device are completed. The signal is normally activated by DCE only if it has detected a carrier signal from the destination device. Also, in this connection, DTR signal acts as a main switch and RI indicates that an external device wants to establish a connection with DTE via DCE.
A full duplex connection is very comfortable, as we need not pay attention to the roles of receiver and transmitter i.e. we may keep RTS signal active all time ignoring CTS and DSR signals.

Null (Zero) Modem Connection

A zero modem serves for data exchange between DTEs. Since both the computers are configured as DTEs, directly connecting them by means of the conventional serial interface cable is impossible; not even the plug fits into the jack of the second terminal. Also the TxD meets TxD and RxD meets RxD, DTR meets DTR and DSR meets DSR etc. This means that outputs are connected to outputs and inputs are connected to inputs. With this convention, no data transfer is possible.

For the transmission of data, it is required to twist the TxD and RxD lines. In this way, the transmitted data of one terminal (PC) becomes received data of other and vice versa. As shown in figure, activation of RTS to begin a data transfer gives rise to an activation of CTS on same DTE and to an activation of DCD on other DTE. Further, an activation of DTR leads to rise of DSR and RI on other DTE. Hence for every DTE, it is simulated that a DCE is on the end of line, although a connection between two DTEs is actually present. Zero modem can be operated with standard BIOS and DOS functions.

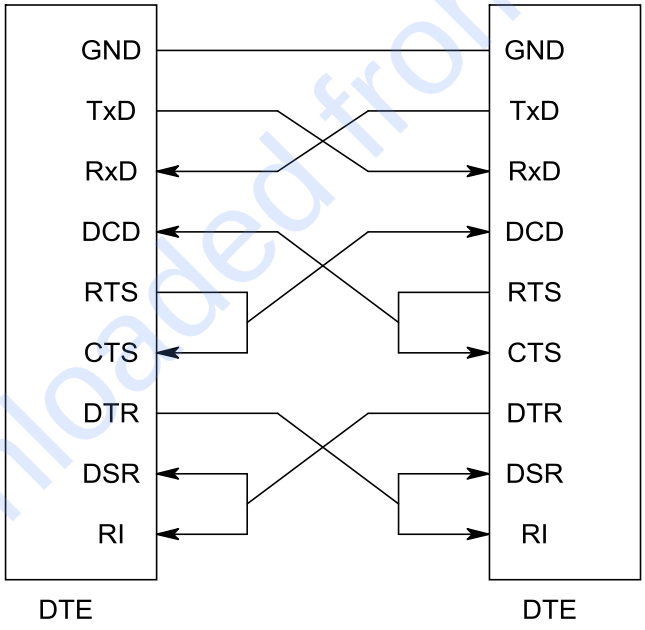


Fig: Null MODEM Connection for RS-232 Terminals

Connection to Printers

As a printer is not DCE, various control and status lines have to be connected or interchanged to emulate behavior of a DCE. TxD data of PC becomes received data of printer. DCD and RI signals on PC are meaningless. On PC, RTS and CTS are connected to each other so that a transmission request from PC immediately enables the transmission. Since, printer as DTE refers to print anything as long as no active signal is present at inputs CTS, DSR and DCD. This problem is solved by connecting RTS with CTS and DTR with DCD and DSR. Thus, activating RTS gives rise to an activation of CTS and that of DTR to an activation of DCD and DSD.

Overrun error arises in serial interface as PC can transmit data much faster than printer can print it so internal printer buffer gets full. On parallel interface, this problem is solved as printer activates BUSY signal informing PC that it cannot accept data temporarily. In serial interface, pin 19 of printer is used to output a <<Buffer Full Signal>>. On DTE, DSR provide an input for this signal. If printer buffer is full, printer simply disables handshake signal at pin 19 and DTE knows that temporarily no additional data can be transferred. If enough room is available in buffer again, printer enables signal once more; PC may transfer data to printer. Not all printers with serial interface provide such a buffer full signal at pin 19.

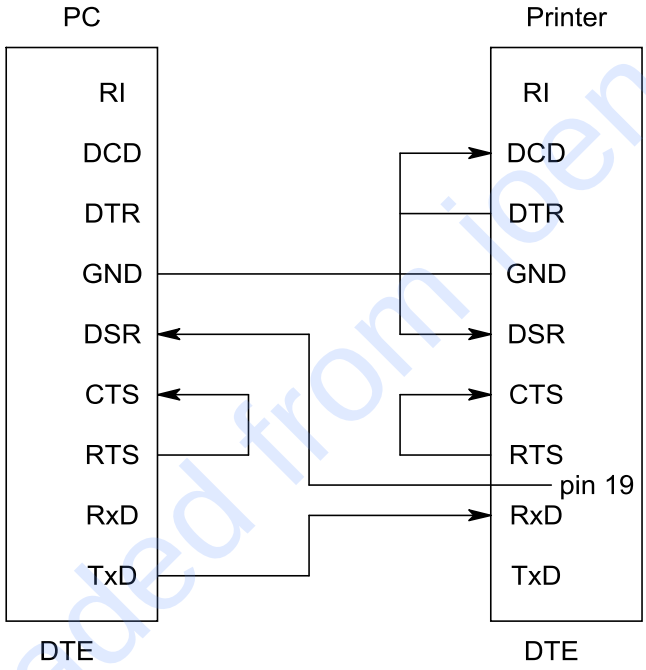


Fig: Connection to Printer

RS-423A

A major problem with RS-232C is that it can only transmit data reliably for about 50 ft at its maximum rate of 20Kbd. If longer lines are used the transmission rate has to be drastically reduced due to open signal lines with a common signal ground. Another EIA standard which is improvement over RS-232C is RS-423A. The standardize features of RS-423 are:

- This standard specifies a low impendence single ended signal which can be sent over 50Ω coaxial cable and partially terminated at the receiving end to prevent reflection.
- Voltage levels
    - Logic High 4V - 6V negative
    - Logic Low 4V - 6V positive

- It allows a maximum data rate of 100 Kbd over 40 ft line or a maximum baud rate of 1 Kbd over 4000 ft line.

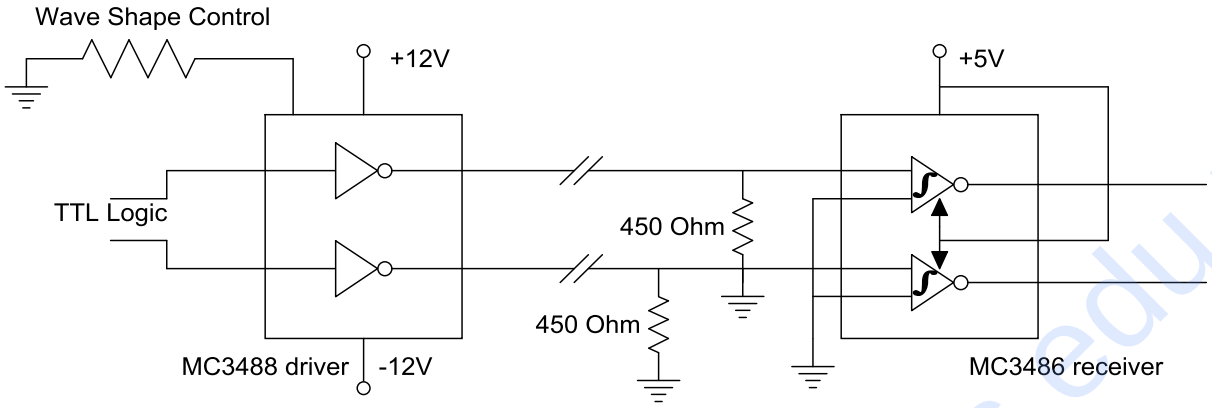


Fig: MC3488 driver and MC3489 receiver used for RS-423A Interface

RS-422A

It is a newer standard for serial data transfer. It specifies that each signal will be sent differentially over two adjacent wires in a ribbon cable or a twisted pair of wires uses differential amplifier to reject noise. The term differential in this standard means that the signal voltage is developed between two signal lines rather than between signal line and ground as in RS-232C and RS-423A. Any electrical noise induced in one signal line will be induced equally in the other signal line. A differential line receiver MC3486 responds only to the voltage difference between its two inputs so any noise voltage that is induced equally on two inputs will not have any effect on the output of the differential receiver.

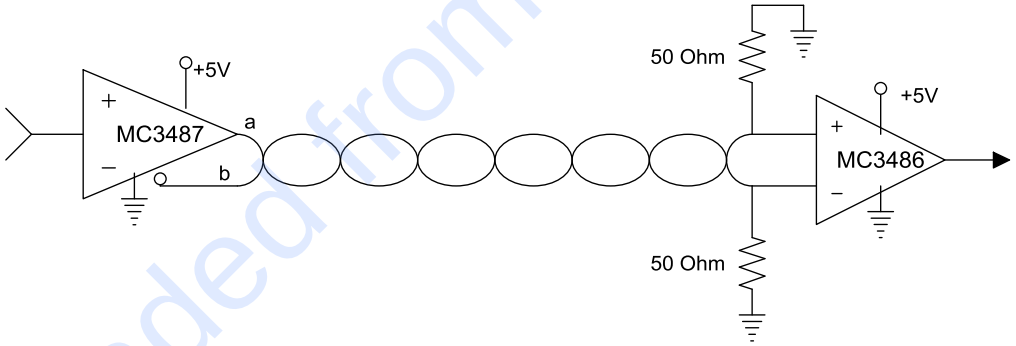


Fig: MC3487 driver and MC3486 receiver used for RS-422A Interface

RS-422A has following standardized features:

- Logic high is transmitted by making 'b' line more positive than 'a' line.
- Logic low is transmitted by making 'a' line more positive than 'b' line.
- The voltage difference between the two lines must be greater than 0.4V but less than 12V.
- The MC3487 driver provides a differential voltage of about 2V.
- The center or common mode voltage on the lines must be between -7v and +7v
- Transmission rate is 10 MBd for 40 ft and 100 KBd for 4000 ft.
- The high data transfer is because of differential line functions as a fully terminated transmission line.
- Mc 3486 receiver only responds to the differential voltage eliminating noise.

Comparison of Serial I/O Standards

S.N.	Specifications	RS-232C	RS-423A	RS-422A
1.	Speed	20 Kbaud	100 Kbaud at 40 ft 1 kbaud at 4000 ft	10 Mbaud at 40 ft 100 kbaud at 4000 ft
2.	Distance	50 ft	4000 ft	4000 ft
3.	Logic 0	+3 V to +25 V	+4 V to +6 V	B line > A line
4.	Logic 1	-3 V to -25 V	-4 V to –6 V	A line > B line
5.	Receiver Input Voltage	±15V	±12V	±7V
6.	Mode of Operation	Single ended input and output	Differential input and single ended output	Differential input and output
7.	Noise Immunity	2.0 V	3.4 V	1.8 V
8.	Input Impedance	3-7 KOhm and 2500 pf	>4 KOhm	>4 KOhm
9.	Short circuit current	500 mA	150 mA	150 mA

Universal Serial Bus (USB)

- In the past, connecting multiple peripheral devices to computer has been a real problem. There were too many different port types (serial port, parallel port, PS/2 etc.) and their use imposes limitations such as no hot-plug ability and automatic configuration.
- USB is designed to allow many peripherals to be connected using a single standardized interface. It provides an expandable, fast, bi-directional, low-cost, hot-pluggable Plug and Play serial hardware interface that makes the life of the computer users easier allowing them to plug different peripheral devices into a USB port and have them automatically configured and ready to use. Using a single connector type, USB allows the user to connect a wide range of peripheral devices, such as keyboards, mice, printers, scanners, mass storage devices, telephones, modems, digital still-image cameras, video cameras, audio devices to a computer. USB devices do not directly consume system resources.
- USB is an industry standard developed in the mid-1990s that defines the cables, connectors and protocols used for connection, communication and power supply between computers and electronic devices.
- . It has become commonplace on other devices, such as smart phones, PDAs and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

Features of USB

- *Single connector type*: USB replaces all the different legacy connectors with one well-defined, standardized USB connector for all USB peripheral devices, eliminating the need for different cables and connectors and thus simplifying the design of the USB devices. So all USB devices can be connected directly to a standard USB port on a computer.

- *Hot-swappable*: USB devices can be safely plugged and unplugged as needed while the computer is running. So there is no need to reboot.
- *Plug and Play*: Operating system software automatically identifies, configures, and loads the appropriate device driver when a user connects a USB device.
- *High performance*: USB offers low speed (1.5 Mbit/s), full speed (12 Mbit/s) and high speed (up to 480 Mbit/s) transfer rates that can support a variety of USB peripherals. USB 3.0 (SuperSpeed USB) achieves the throughput up to 5.0 Gbit/s.
- *Expandability*: Up to 127 different peripheral devices may theoretically be connected to a single bus at one time.
- *Power supplied from the bus*: USB distributes the power to all connected devices eliminating the need for external power source for low-power devices. High-power devices can still require their own local power supply. USB also supports power saving suspend/resume modes.
- *Easy to use for end user*: A single standard connector type for all USB devices simplifies the end user's task at figuring out which plugs go into which sockets. The operating system automatically recognizes the USB device attachment and loads appropriate device drivers.
- *Low-cost implementation*: Most of the complexity of the USB protocol is handled by the host, which along with low-cost connection for peripherals makes the design simple and low cost.
- *Wide range of workloads and applications:*
  - Suitable for device bandwidths ranging from a few kb/s to several Mb/s
  - Supports isochronous as well as asynchronous transfer types over the same set of wires
  - Supports concurrent operation of many devices (multiple connections)
  - Supports up to 127 physical devices
  - Supports transfer of multiple data and message streams between the host and devices
  - Allows compound devices (i.e., peripherals composed of many functions)
  - Lower protocol overhead, resulting in high bus utilization
- *Isochronous bandwidth*
  - Guaranteed bandwidth and low latencies appropriate for telephony, audio, etc.
  - Isochronous workload may use entire bus bandwidth
- *Robustness*
  - Error handling/fault recovery mechanism is built into the protocol
  - Dynamic insertion and removal of devices is identified in user-perceived real-time
  - Supports identification of faulty devices

USB Standards

USB 1.0

- *USB 1.0*: Released in January 15, 1996.
  Specified data rates of *1.5 Mbit/s* (*Low-Bandwidth*) and *12 Mbit/s* (*Full-Bandwidth*). Does not allow for extension cables or pass-through monitors (due to timing and power limitations). Few such devices actually made it to market.
- *USB 1.1*: Released in September 23, 1998.
  Introduced the improved specification and was the first widely used version of USB.

Fixed problems identified in 1.0, mostly relating to hubs. Earliest revision to be widely adopted.

USB 2.0

- The USB 2.0 specification was released in April 27, 2000 and was ratified by the USB Implementers Forum (USB-IF) at the end of 2001.
- The major feature of revision 2.0 was the addition of a high-speed transfer rate of 480 Mbit/s. USB 2.0 supports three speeds namely High Speed - 480Mbits/s, Full Speed - 12Mbits/s and Low Speed - 1.5Mbits/s with one host per bus (at a time).

USB 3.0

- The USB 3.0 specification was published on 12 November 2008.
- Brings significant performance enhancements to the USB standard while offering backward compatibility with the peripheral devices currently in use. Legacy USB 1.1/2.0 devices continue to work while plugged into new USB 3.0 host and new USB 3.0 devices work at USB 2.0 speed while plugged into USB 2.0 host.
- Delivering data transfer rates up to ten times faster (the raw throughput is up to 5.0 Gbit/s) than Hi-Speed USB (USB 2.0), SuperSpeed USB is the next step in the continued evolution of USB technology.
- Its main goals were to increase the data transfer rate (up to 5 Gbit/s), to decrease power consumption, to increase power output, and to be backwards-compatible with USB 2.0. USB 3.0 includes a new, higher speed bus called SuperSpeed in parallel with the USB 2.0 bus. For The first USB 3.0 equipped devices were presented in January 2010
- Transfer of 25 GB file in approx 70 seconds
- Extensible – Designed to scale > 25Gbps
- Optimized power efficiency
    - No device polling (asynchronous notifications)
    - Lower active and idle power requirements
- Backward compatible with USB 2.0
    - USB 2.0 device will work with USB 3.0 host
    - USB 3.0 device will work with USB 2.0 host

Wireless USB

- Released in May 12, 2005 which uses UWB (Ultra Wide Band) as the radio technology.
- 480 M bits/sec up to 3m
- 110 m bits/sec up to 10m

Signals, Throughput & Protocol

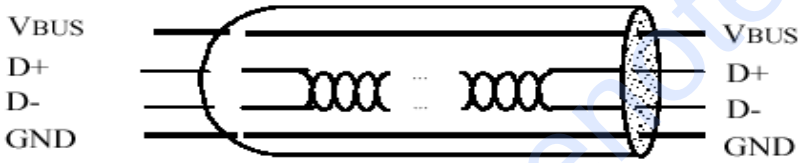
USB Interconnect

- Bus Topology: Connection model between USB devices and the host.
- Inter-layer Relationships: In terms of a capability stack, the USB tasks that are performed at each layer in the system.
- Data Flow Models: The manner in which data moves in the system over the USB between producers and consumers.

- USB Schedule: The USB provides a shared interconnect. Access to the interconnect is scheduled in order to support isochronous data transfers and to eliminate arbitration overhead.

Fig: 'A' Plug, 'B' Plug and 'Mini-B' Plug

Signals



Pin	Color	Name	Description
1	Red	Vcc	+5V dc
2	White	D-	Data-
3	Green	D+	Data+
4	Black	GND	Ground

Fig: USB electrical signals

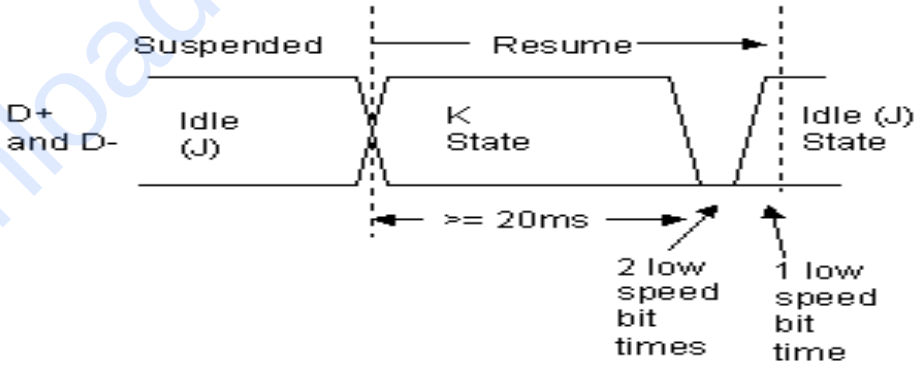


Fig: USB signals and states

Bus State	Levels
Differential '1'	D+ high, D- low
Differential '0'	D- high, D+ low
Single Ended Zero (SE0)	D+ and D- low
Single Ended One (SE1)	D+ and D- high
Data J State: Low-speed Full-speed	 Differential '0' Differential '1'
Data K State: Low-speed Full-speed	 Differential '1' Differential '0'
Idle State: Low-speed Full-speed	 D- high, D+- low D+ high, D- low
Resume State	Data K state
Start of Packet (SOP)	Data lines switch from idle to K state
End of Packet (EOP)	SE0 for 2 bit times followed by J state for 1 bit time
Disconnect	SE0 for >= 2us
Connect	Idle for 2.5us
Reset	SE0 for >= 2.5 us

J, K and SEO States

To make it easier to talk about the states of the data lines, some special terminology is used. The 'J State' is the same polarity as the idle state (the line with the pull-up resistor is high, and the other line is low), but is being driven to that state by either host or device.

The K state is just the opposite polarity to the J state.

The Single Ended Zero (SE0) is when both lines are being pulled low.

The J and K terms are used because for Full Speed and Low Speed links they are actually of opposite polarity.

Single Ended One (SE1)

This is the **illegal** condition where both lines are high. It should never occur on a properly functioning link.

Reset

When the host wants to start communicating with a device it will start by applying a 'Reset' condition which sets the device to its default unconfigured state.

The Reset condition involves the host pulling down both data lines to low levels (SE0) for at least 10 ms. The device may recognize the reset condition after 2.5 us.

This 'Reset' should not be confused with a micro-controller power-on type reset. It is a USB protocol reset to ensure that the device USB signaling starts from a known state.

EOP signal
The End of Packet (EOP) is an SE0 state for 2 bit times, followed by a J state for 1 bit time.

Suspend
One of the features of USB which is an essential part of today's emphasis of 'green' products is its ability to power down an unused device. It does this by suspending the device, which is achieved by not sending anything to the device for 3 ms.
Normally a SOF packet (at full speed) or a Keep Alive signal (at low speed) is sent by the host every 1 ms, and this is what keeps the device awake.
A suspended device may draw no more than 0.5 mA from Vbus.
A suspended device must recognise the resume signal, and also the reset signal.

Resume
When the host wants to wake the device up after a suspend, it does so by reversing the polarity of the signal on the data lines for at least 20ms. The signal is completed with a low speed end of packet signal.
It is also possible for a device with its remote wakeup feature set, to initiate a resume itself. It must have been in the idle state for at least 5ms, and must apply the wakeup K condition for between 1 and 15 ms. The host takes over the driving of the resume signal within 1 ms.

Keep Alive Signal
This is represented by a Low speed EOP. It is sent at least once every millisecond on a low speed link, in order to keep the device from suspending.

Throughput

- Throughput is the actual output of any device, USB's actual throughput is a function of many variables:
  - Target device's ability to source or sink data
  - Bandwidth consumption by other devices in the bus
  - Efficiency of host's USB ports
  - Types of data

Speed

A USB device must indicate its speed by pulling either the D+ or D- line high to 3.3 volts. A full speed device, pictured below will use a pull up resistor attached to D+ to specify itself as a full speed device. These pull up resistors at the device end will also be used by the host or hub to detect the presence of a device connected to its port. Without a pull up resistor, USB assumes there is nothing connected to the bus.

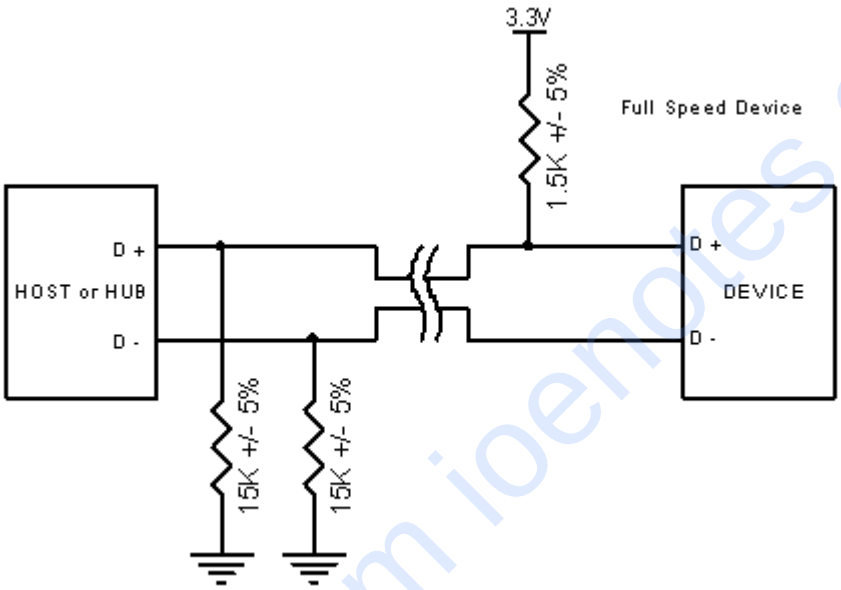


Figure : Full Speed Device with pull up resistor connected to D+

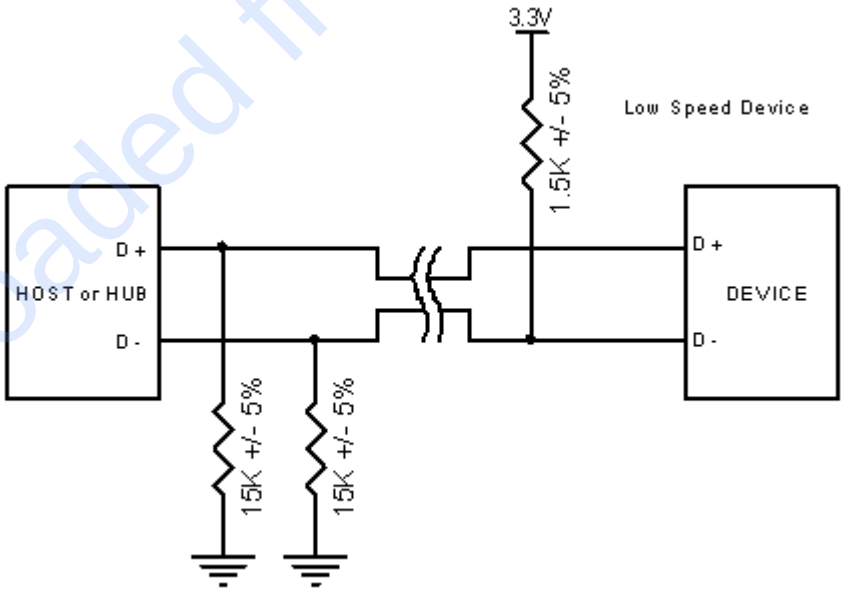


Figure : Low Speed Device with pull up resistor connected to D-

USB Protocols

Unlike RS-232 and similar serial interfaces where the format of data being sent is not defined, USB is made up of several layers of protocols. While this sounds complicated, don't give up now. Once you understand what is going on, you really only have to worry about the higher level layers. In fact most USB controller I.C.s will take care of the lower layer, thus making it almost invisible to the end designer.

Each USB transaction consists of a

- Token Packet (Header defining what it expects to follow), an
- Optional Data Packet, (Containing the payload) and a
- Status Packet (Used to acknowledge transactions and to provide a means of error correction)

As we have already discussed, USB is a host centric bus. The host initiates all transactions. The first packet, also called a token is generated by the host to describe what is to follow and whether the data transaction will be a read or write and what the device's address and designated endpoint is. The next packet is generally a data packet carrying the payload and is followed by an handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data.

Common USB Packet Fields

Data on the USB bus is transmitted LSB first. USB packets consist of the following fields,

- **Sync:** All packets must start with a sync field. The sync field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronize the clock of the receiver with that of the transmitter. The last two bits indicate where the PID fields starts.
- **PID:** PID stands for Packet ID. This field is used to identify the type of packet that is being sent.

There are 4 bits to the PID, however to insure it is received correctly, the 4 bits are complemented and repeated, making an 8 bit PID in total. The resulting format is shown below.

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

- **ADDR:** The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. Address 0 is not valid, as any device which is not yet assigned an address must respond to packets sent to address zero.
- **ENDP:** The endpoint field is made up of 4 bits, allowing 16 possible endpoints. Low speed devices, however can only have 2 additional endpoints on top of the default pipe. (4 endpoints max)
- **CRC:** Cyclic Redundancy Checks are performed on the data within the packet payload. All token packets have a 5 bit CRC while data packets have a 16 bit CRC.
- **EOP:** End of packet. Signalled by a Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time.

USB Packet Types

USB has four different packet types. Token packets indicate the type of transaction to follow, data packets contain the payload, handshake packets are used for acknowledging data or reporting errors and start of frame packets indicate the start of a new frame.

- **Token Packets:** There are three types of token packets,
  - **In** - Informs the USB device that the host wishes to read information.
  - **Out** - Informs the USB device that the host wishes to send information.
  - **Setup** - Used to begin control transfers.

  Token Packets must conform to the following format,

Sync	PID	ADDR	ENDP	CRC5	EOP

- **Data Packets:** There are two types of data packets each capable of transmitting up to 1024 bytes of data.
  - Data0
  - Data1

    High Speed mode defines another two data PIDs, DATA2 and MDATA.

    Data packets have the following format,

Sync	PID	Data	CRC16	EOP

  - Maximum data payload size for low-speed devices is 8 bytes.
  - Maximum data payload size for full-speed devices is 1023 bytes.
  - Maximum data payload size for high-speed devices is 1024 bytes.
  - Data must be sent in multiples of bytes.

- **Status / Handshake Packets:** There are three type of handshake packets which consist simply of the PID
  - **ACK** - Acknowledgment that the packet has been successfully received.
  - **NAK** - Reports that the device temporary cannot send or received data. Also used during interrupt transactions to inform the host there is no data to send.
  - **STALL** - The device finds its in a state that it requires intervention from the host.

    Handshake Packets have the following format,

Sync	PID	EOP

- **Start of Frame Packets**

  The SOF packet consisting of an 11-bit frame number is sent by the host every 1ms ± 500ns on a full speed bus or every 125 µs ± 0.0625 µs on a high speed bus.

Sync	PID	Frame Number	CRC5	EOP

Transfer Model

Endpoints

Endpoints can be described as sources or sinks of data. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. At the software layer, your device driver may send a packet to your devices EP1 for example. As the data is flowing out from the host, it will end up in the EP1 OUT buffer. Your firmware will then at its leisure read this data. If it wants to return data, the function cannot simply write to the bus as the bus is controlled by the host. Therefore it writes data to EP1 IN which sits in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen

as the interface between the hardware of the function device and the firmware running on the function device.
All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

Pipes
While the device sends and receives data on a series of endpoints, the client software transfers data through pipes. A pipe is a logical connection between the host and endpoint(s). Pipes will also have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type (Control, Bulk, Iso or Interrupt) it uses, a direction of data flow and maximum packet/buffer sizes. For example the default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type.
USB defines two types of pipes

- **Stream Pipes** have no defined USB format, that is you can send any type of data down a stream pipe and can retrieve the data out the other end. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes will support bulk, isochronous and interrupt transfer types. Stream pipes can either be controlled by the host or device.

- **Message Pipes** have a defined USB format. They are host controlled, which are initiated by a request sent from the host. Data is then transferred in the desired direction, dictated by the request. Therefore message pipes allow data to flow in both directions but will only support control transfers.

Data Flow Types

- Control Transfers:
  - typically used for short, simple commands to the device, and a status response, used e.g. by the bus control pipe number 0
- Bulk Data Transfers:
  - Large sporadic transfers using all remaining available bandwidth (but with no guarantees on bandwidth or latency). A device like a printer, which receives data in one big packet, uses the bulk transfer mode. A block of data is sent to the printer (in 64-byte chunks) and verified to make sure it is correct.
- Interrupt Data Transfers:
  - Devices that need guaranteed quick responses (bounded latency). A device like a mouse or a keyboard, which will be sending very little data, would choose the interrupt mode.
- Isochronous Data Transfers:
  - At some guaranteed speed (often but not necessarily as fast as possible) but with possible data loss A streaming device (such as speakers) uses the isochronous mode. Data streams between the device and the host in real-time, and there is no error correction.

Devices (Nodes), Hosts and On-The-Go

- The USB is based on a so-called 'tiered star topology' in which there is a single host controller and up to 127 'slave' devices. The host controller is connected to a hub,

integrated within the PC, which allows a number of attachment points (often loosely referred to as ports). A further hub may be plugged into each of these attachment points, and so on. However there are limitations on this expansion.

- A device can be plugged into a hub, and that hub can be plugged into another hub and so on. However the maximum number of tiers permitted is six.
- All devices have an upstream connection to the host and all hosts have a downstream connection to the device.
- The length of any cable is limited to 5 metres. This limitation is expressed in the specification in terms of cable delays etc, but 5 metres can be taken as the practical consequence of the specification. This means that a device cannot be further than 30 metres from the PC, and even to achieve that will involve 5 external hubs, of which at least 2 will need to be self-powered.
- So the USB is intended as a bus for devices near to the PC. For applications requiring distance from the PC, another form of connection is needed, such as Ethernet.

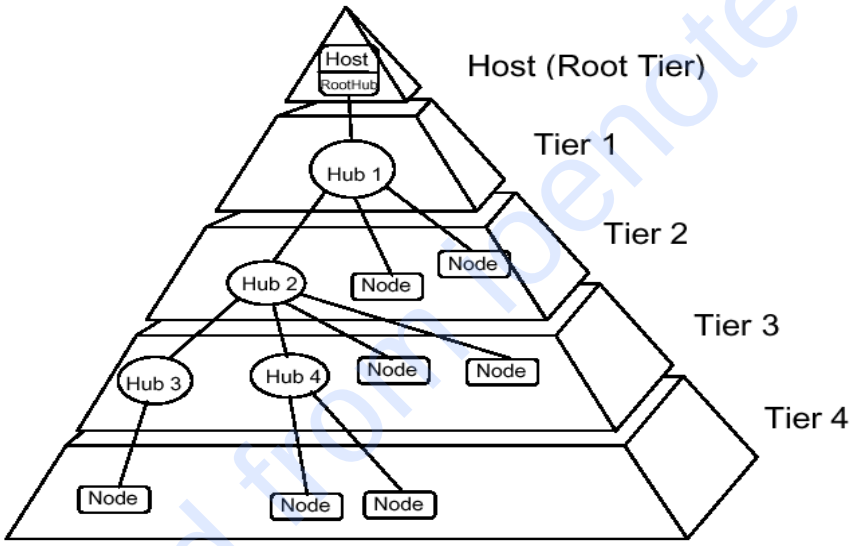


Fig: USB network protocol architecture

Hub

- Hub has two major roles: power management and signal distribution.
- Hubs can be linked, potentially giving you unlimited USB ports to your computer.
- The biggest difference between types of hubs that is important to know when dealing with USB devices is between un-powered and powered hubs.

Powered Hub

- Needed when connecting multiple unpowered devices such as mice or digital cameras.
- These low-powered devices derive their power source from the bus.
- If too many are connected through a hub, the computer may not be able to handle it.

Un-powered Hub

- Un-powered hubs can be used with any number of high-power devices such as printers and scanners that have their own power supply, thus not requiring power from the bus.
- Safe to use with low-power devices (mice, cameras, joysticks, etc.) as long as too many aren't connected as once.

USB On The Go (OTG)

USB OTG is a new supplement to the USB 2.0 specification that arguments the capability of existing mobile devices and USB peripherals by adding host functionality for connection to USB peripherals. Since USB has traditionally consisted of host-peripheral topology where the PC was the host and the peripheral was the relatively dump device, following new features were needed to upgrade USB technology:

- A new standard for small form factor USB connectors and cables
- The addition of host capability to products that have traditionally been peripherals only, to enable point-to-point connection
- The ability to be either host or peripheral (dual role devices) and to dynamically switch between the two.
- Lowest power requirements to facilitate USB on battery powered devices.

USB On-The-Go (OTG) allows two USB devices to talk to each other without requiring the services of a personal computer (PC). Although OTG appears to add peer-to-peer connections to the USB world, it does not. Instead, USB OTG retains the standard USB host/peripheral model, in which a single host talks to USB peripherals. OTG does introduce, however, the dual-role device, or simply stated a device capable of functioning as either host or peripheral. Part of the magic of OTG is that a host and peripheral can exchange roles if necessary.

Interface Chips: USB device and USB host

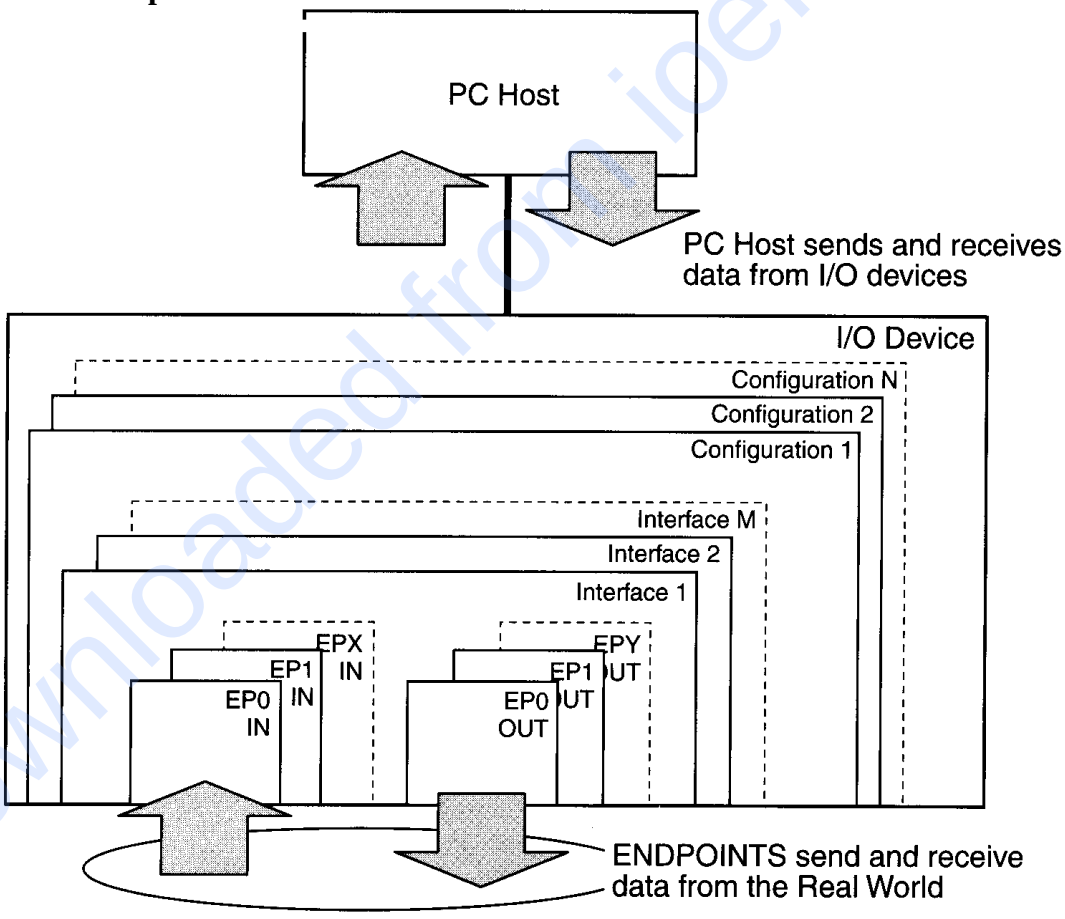


Fig: Logical view of device host interface

- Endpoint is where data enters or leaves the USB system. An IN endpoint is data creator and OUT endpoint is data consumer. For reliable data delivery scheme, need multiple IN and OUT endpoints.
- The collection of endpoints is called an interface and is directly related to the real world connection.
- An operating system will have a driver that corresponds to each interface.
- Some devices may have multiple interfaces such as a telephone has a keypad interface and audio interface. Operating system will manage two separate device drivers.
- A collection of interface is called a configuration, and only one configuration can be active at a time.
- A configuration defines the attribute and features of a specific model.

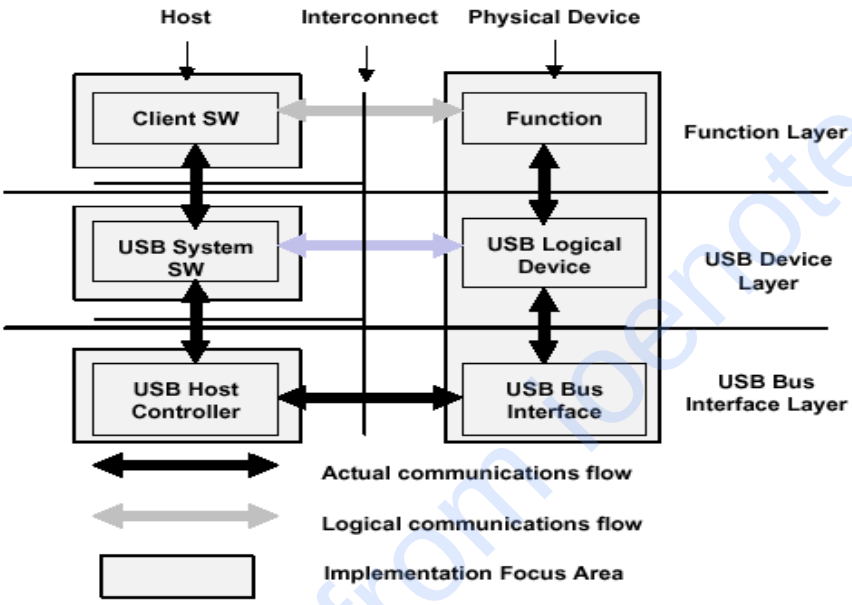


Fig: Interface between device and host