2.4 COMMUNICATION INTERFACE

Communication interface is essential for communicating with various subsystems of the embedded system and with the external world. For an embedded product, the communication interface can be viewed in two different perspectives; namely; Device/board level communication interface (Onboard Communication Interface) and Product level communication interface (External Communication Interface). Embedded product is a combination of different types of components (chips/devices) arranged on a printed circuit board (PCB). The communication channel which interconnects the various components within an embedded product is referred as device/board level communication interface (onboard communication interface). Serial interfaces like I2C, SPI, UART, 1-Wire, etc and parallel bus interface are examples of 'Onboard Communication Interface'.

Some embedded systems are self-contained units and they don't require any interaction and data transfer with other sub-systems or external world. On the other hand, certain embedded systems may be a part of a large distributed system and they require interaction and data transfer between various devices and sub-modules. The 'Product level communication interface' (External Communication Interface) is responsible for data transfer between the embedded system and other devices or modules. The external communication interface can be either a wired media or a wireless media and it can be a serial or a parallel interface. Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS, etc. are examples for wireless communication interface. RS-232C/RS-422/RS-485, USB, Ethernet IEEE 1394 port, Parallel port, CF-II interface, SDIO, PCMCIA, etc. are examples for wired interfaces. It is not mandatory that an embedded system should contain an external communication interface. Mobile communication equipment is an example for embedded system with external communication interface.

The following section gives you an overview of the various 'Onboard' and 'External' communication interfaces for an embedded product. We will discuss about the various physical interface, firmware requirements and initialisation and communication sequence for these interfaces in a dedicated book titled '*Device Interfacing*', which is planned under this series.

2.4.1 Onboard Communication Interfaces

Onboard Communication Interface refers to the different communication channels/buses for interconnecting the various integrated circuits and other peripherals within the embedded system. The following section gives an overview of the various interfaces for onboard communication.

2.4.1.1 Inter Integrated Circuit (I2C) Bus The Inter Integrated Circuit Bus (I2C–Pronounced 'I square C') is a synchronous bi-directional half duplex (one-directional communication at a given point of time) two wire serial interface bus. The concept of I2C bus was developed by 'Philips semiconductors' in the early 1980s. The original intention of I2C was to provide an easy way of connection between a microprocessor/microcontroller system and the peripheral chips in television sets. The I2C bus comprise of two bus lines, namely; Serial Clock–SCL and Serial Data–SDA. SCL line is responsible for generating synchronisation clock pulses and SDA is responsible for transmitting the serial data across devices. I2C bus is a shared bus system to which many number of I2C devices can be connected. Devices connected to the I2C bus can act as either 'Master' device or 'Slave' device. The 'Master' device is responsible for controlling the communication by initiating/terminating data transfer, sending data and generating necessary synchronisation clock pulses. 'Slave' devices wait for the commands from the master and respond upon receiving the commands. 'Master' and 'Slave' devices can act as either transmitter or receiver. Regardless whether a master is acting as transmitter or receiver, the synchronisation clock signal is generated by the 'Master' device only. I2C supports multi masters on the same bus. The following bus interface diagram shown in Fig. 2.26 illustrates the connection of master and slave devices on the I2C bus.

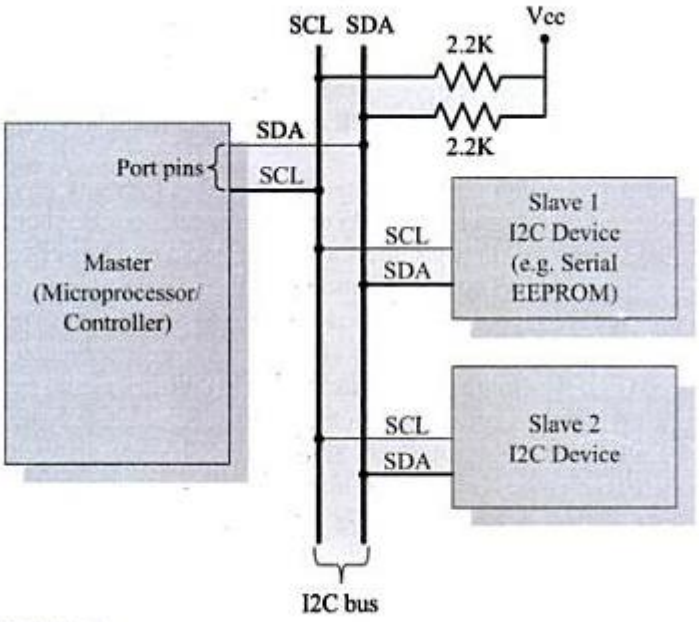


Fig. 2.26 **I2C Bus Interfacing**

The I2C bus interface is built around an input buffer and an open drain or collector transistor. When the bus is in the idle state, the open drain/collector transistor will be in the floating state and the output lines (SDA and SCL) switch to the 'High Impedance' state. For proper operation of the bus, both the bus lines should be pulled to the supply voltage (+5V for TTL family and +3.3V for CMOS family devices) using pull-up resistors. The typical value of resistors used in pull-up is 2.2K. With pull-up resistors, the output lines of the bus in the idle state will be 'HIGH'.

The address of a I2C device is assigned by hardwiring the address lines of the device to the desired logic level. The address to various I2C devices in an embedded device is assigned and hardwired at the time of designing the embedded hardware. The sequence of operations for communicating with an I2C slave device is listed below:

1. The master device pulls the clock line (SCL) of the bus to 'HIGH'
2. The master device pulls the data line (SDA) 'LOW', when the SCL line is at logic 'HIGH' (This is the 'Start' condition for data transfer)
3. The master device sends the address (7 bit or 10 bit wide) of the 'slave' device to which it wants to communicate, over the SDA line. Clock pulses are generated at the SCL line for synchronising the bit reception by the slave device. The MSB of the data is always transmitted first. The data in the bus is valid during the 'HIGH' period of the clock signal

4. The master device sends the Read or Write bit (Bit value = 1 Read operation; Bit value = 0 Write operation) according to the requirement
5. The master device waits for the acknowledgement bit from the slave device whose address is sent on the bus along with the Read/Write operation command. Slave devices connected to the bus compares the address received with the address assigned to them
6. The slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value = 1) over the SDA line
7. Upon receiving the acknowledge bit, the Master device sends the 8bit data to the slave device over SDA line, if the requested operation is 'Write to device'. If the requested operation is 'Read from device', the slave device sends data to the master over the SDA line
8. The master device waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the Slave device for a read operation
9. The master device terminates the transfer by pulling the SDA line 'HIGH' when the clock line SCL is at logic 'HIGH' (Indicating the 'STOP' condition)

I2C bus supports three different data rates. They are: Standard mode (Data rate up to 100kbits/sec (100 kbps)), Fast mode (Data rate up to 400kbits/sec (400 kbps)) and High speed mode (Data rate up to 3.4Mbits/sec (3.4 Mbps)). The first generation I2C devices were designed to support data rates only up to 100kbps. The new generation I2C devices are designed to operate at data rates up to 3.4Mbits/sec.

2.4.1.2 Serial Peripheral Interface (SPI) Bus The Serial Peripheral Interface Bus (SPI) is a synchronous bi-directional full duplex four-wire serial interface bus. The concept of SPI was introduced by Motorola. SPI is a single master multi-slave system. It is possible to have a system where more than one SPI device can be master, provided the condition only one master device is active at any given point of time, is satisfied. SPI requires four signal lines for communication. They are:

Master Out Slave In (MOSI):	Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI)
Master In Slave Out (MISO):	Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/SDO)
Serial Clock (SCLK):	Signal line carrying the clock signals
Slave Select (SS):	Signal line for slave device select. It is an active low signal

The bus interface diagram shown in Fig. 2.27 illustrates the connection of master and slave devices on the SPI bus.

The master device is responsible for generating the clock signal. It selects the required slave device by asserting the corresponding slave device's slave select signal 'LOW'. The data out line (MISO) of all the slave devices when not selected floats at high impedance state.

The serial data transmission through SPI bus is fully configurable. SPI devices contain a certain set of registers for holding these configurations. The serial peripheral control register holds the various configuration parameters like master/slave selection for the device, baudrate selection for communication, clock signal control, etc. The status register holds the status of various conditions for transmission and reception.

SPI works on the principle of 'Shift Register'. The master and slave devices contain a special shift register for the data to transmit or receive. The size of the shift register is device dependent. Normally it is a multiple of 8. During transmission from the master to slave, the data in the master's shift register is shifted out to the MOSI pin and it enters the shift register of the slave device through the MOSI pin of the slave device. At the same time the shifted out data bit from the slave device's shift register enters the shift register of the master device through MISO pin. In summary, the shift registers of 'master' and 'slave' devices form a circular buffer. For some devices, the decision on whether the LS/MS bit of data needs to be sent out first is configurable through configuration register (e.g. LSBF bit of the SPI control register for Motorola's 68HC12 controller).

When compared to I2C, SPI bus is most suitable for applications requiring transfer of data in 'streams'. The only limitation is SPI doesn't support an acknowledgement mechanism.

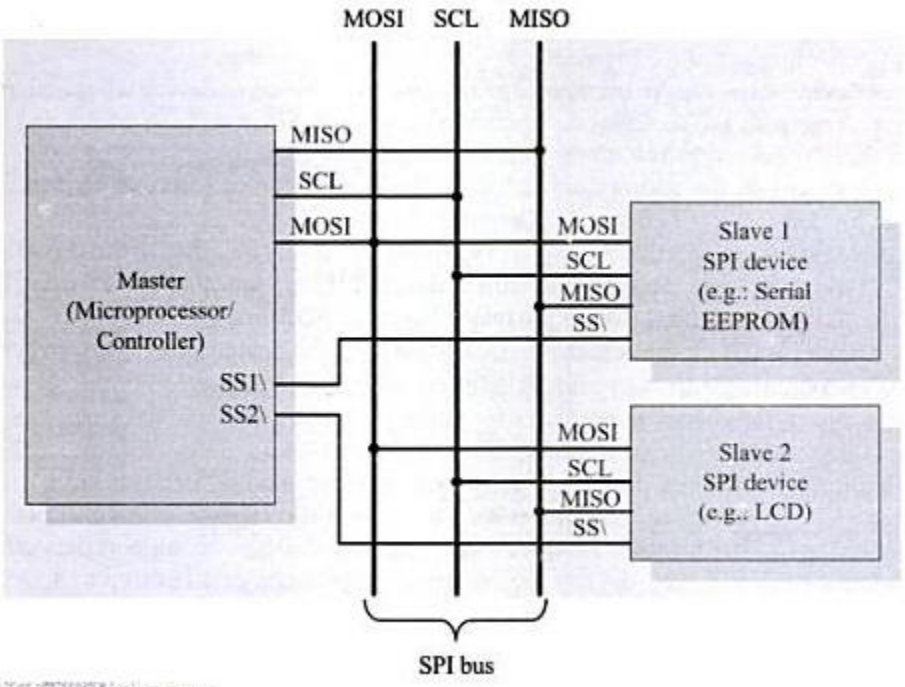


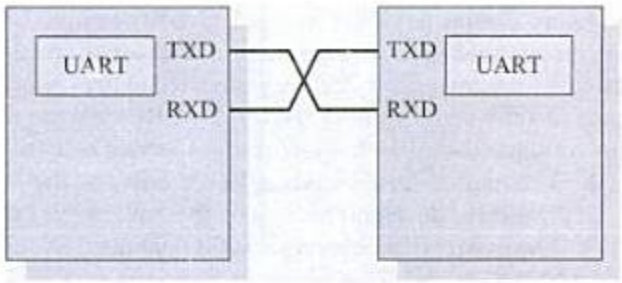
Fig. 2.27 **SPI bus interfacing**

2.4.1.3 Universal Asynchronous Receiver Transmitter (UART) Universal Asynchronous Receiver Transmitter (UART) based data transmission is an asynchronous form of serial data transmission. UART based serial data transmission doesn't require a clock signal to synchronise the transmitting end and receiving end for transmission. Instead it relies upon the pre-defined agreement between the transmitting device and receiving device. The serial communication settings (Baudrate, number of bits per byte, parity, number of start bits and stop bit and flow control) for both transmitter and receiver should be set as identical. The start and stop of communication is indicated through inserting special bits in the data stream. While sending a byte of data, a start bit is added first and a stop bit is added at the end of the bit stream. The least significant bit of the data byte follows the 'start' bit.

The 'start' bit informs the receiver that a data byte is about to arrive. The receiver device starts polling its 'receive line' as per the baudrate settings. If the baudrate is 'x' bits per second, the time slot available for one bit is $1/x$ seconds. The receiver unit polls the receiver line at exactly half of the time slot available for the bit. If parity is enabled for communication, the UART of the transmitting device adds a parity bit (bit value is 1 for odd number of 1s in the transmitted bit stream and 0 for even number of 1s). The UART of the receiving device calculates the parity of the bits received and compares it with the received parity bit for error checking. The UART of the receiving device discards the 'Start', 'Stop' and 'Parity'

bit from the received bit stream and converts the received serial bit data to a word (In the case of 8 bits/byte, the byte is formed with the received 8 bits with the first received bit as the LSB and last received data bit as MSB).

For proper communication, the 'Transmit line' of the sending device should be connected to the 'Receive line' of the receiving device. Figure 2.28 illustrates the same.



TXD: Transmitter line
RXD: Receiver line

Fig. 2.28 **UART Interfacing**

In addition to the serial data transmission function, UART provides hardware handshaking signal support for controlling the serial data flow. UART chips are available from different semiconductor manufacturers. National Semiconductor's 8250 UART chip is considered as the standard setting UART. It was used in the original IBM PC.

Nowadays most of the microprocessors/controllers are available with integrated UART functionality and they provide built-in instruction support for serial data transmission and reception.

2.4.1.4 1-Wire Interface 1-wire interface is an asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor (http://www.maxim-ic.com). It is also known as **Dallas 1-Wire® protocol**. It makes use of only a single signal line (wire) called DQ for communication and follows the master-slave communication model. One of the key feature of 1-wire bus is that it allows power to be sent along the signal wire as well. The 12C slave devices incorporate internal capacitor (typically of the order of 800 pF) to power the device from the signal line. The 1-wire interface supports a single master and one or more slave devices on the bus. The bus interface diagram shown in Fig. 2.29 illustrates the connection of master and slave devices on the 1-wire bus.

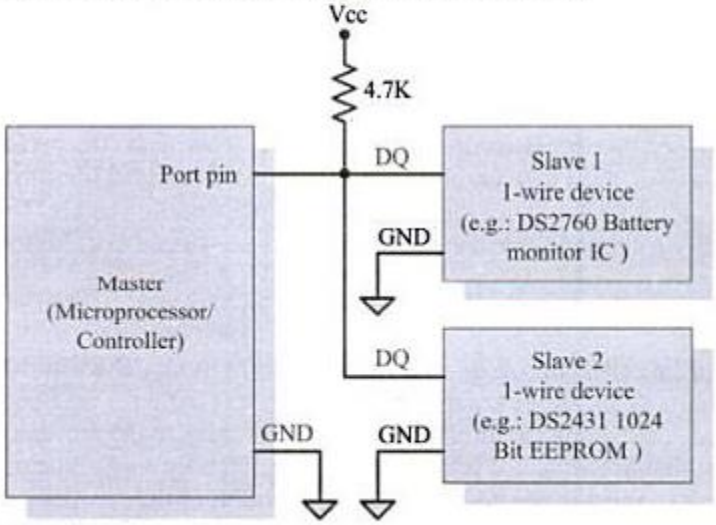


Fig. 2.29 **1-Wire Interface bus**

Every 1-wire device contains a globally unique 64bit identification number stored within it. This unique identification number can be used for addressing individual devices present on the bus in case there are multiple slave devices connected to the 1-wire bus. The identifier has three parts: an 8bit family code, a 48bit serial number and an 8bit CRC computed from the first 56 bits. The sequence of operation for communicating with a 1-wire slave device is listed below.

1. The master device sends a 'Reset' pulse on the 1-wire bus.
2. The slave device(s) present on the bus respond with a 'Presence' pulse.
3. The master device sends a ROM command (Net Address Command followed by the 64bit address of the device). This addresses the slave device(s) to which it wants to initiate a communication.
4. The master device sends a read/write function command to read/write the internal memory or register of the slave device.
5. The master initiates a Read data/Write data from the device or to the device

All communication over the 1-wire bus is master initiated. The communication over the 1-wire bus is divided into timeslots of 60 microseconds. The 'Reset' pulse occupies 8 time slots. For starting a communication, the master asserts the reset pulse by pulling the 1-wire bus 'LOW' for at least 8 time slots (480µs). If a 'slave' device is present on the bus and is ready for communication it should respond to the master with a 'Presence' pulse, within 60µs of the release of the 'Reset' pulse by the master. The slave device(s) responds with a 'Presence' pulse by pulling the 1-wire bus 'LOW' for a minimum of 1 time slot (60µs). For writing a bit value of 1 on the 1-wire bus, the bus master pulls the bus for 1 to 15µs and then releases the bus for the rest of the time slot. A bit value of '0' is written on the bus by master pulling the bus for a minimum of 1 time slot (60µs) and a maximum of 2 time slots (120µs). To Read a bit from the slave device, the master pulls the bus 'LOW' for 1 to 15µs. If the slave wants to send a bit value '1' in response to the read request from the master, it simply releases the bus for the rest of the time slot. If the slave wants to send a bit value '0', it pulls the bus 'LOW' for the rest of the time slot.

2.4.1.5 Parallel Interface The on-board parallel interface is normally used for communicating with peripheral devices which are memory mapped to the host of the system. The host processor/controller of the embedded system contains a parallel bus and the device which supports parallel bus can directly connect to this bus system. The communication through the parallel bus is controlled by the control signal interface between the device and the host. The 'Control Signals' for communication includes 'Read/Write' signal and device select signal. The device normally contains a device select line and the device becomes active only when this line is asserted by the host processor. The direction of data transfer (Host to Device or Device to Host) can be controlled through the control signal lines for 'Read' and 'Write'. Only the host processor has control over the 'Read' and 'Write' control signals. The device is normally memory mapped to the host processor and a range of address is assigned to it. An address decoder circuit is used for generating the chip select signal for the device. When the address selected by the processor is within the range assigned for the device, the decoder circuit activates the chip select line and thereby the device becomes active. The processor then can read or write from or to the device by asserting the corresponding control line (RD\ and WR\ respectively). Strict timing characteristics are followed for parallel communication. As mentioned earlier, parallel communication is host processor initiated. If a device wants to initiate the communication, it can inform the same to the processor through interrupts. For this, the interrupt line of the device is connected to the interrupt line of the processor and the corresponding interrupt is enabled in the host processor. The width of the parallel interface is determined by the data bus width of the host processor. It can be 4bit, 8bit, 16bit, 32bit or 64bit etc. The bus width supported by the device should be same as that of the host processor. The bus interface diagram shown in Fig. 2.30 illustrates the interfacing of devices through parallel interface.

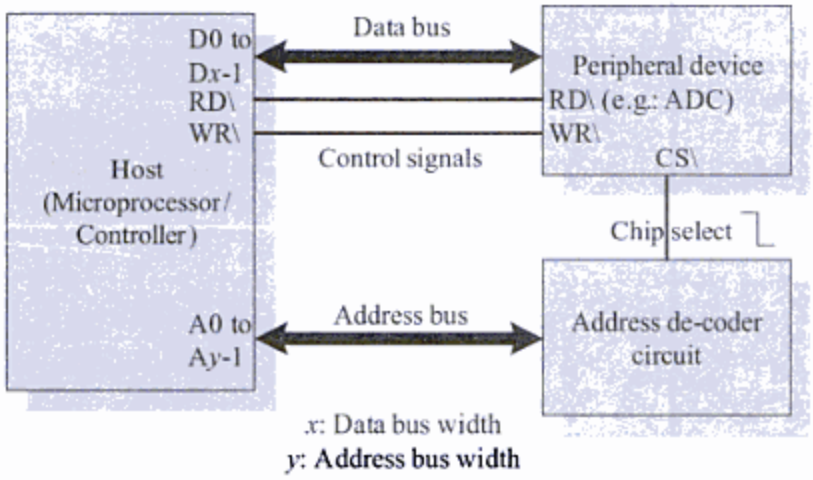


Fig. 2.30 **Parallel Interface Bus**

Parallel data communication offers the highest speed for data transfer.

2.4.2 External Communication Interfaces

The External Communication Interface refers to the different communication channels/buses used by the embedded system to communicate with the external world. The following section gives an overview of the various interfaces for external communication.

2.4.2.1 RS-232 C & RS-485 RS-232 C (Recommended Standard number 232, revision C from the Electronic Industry Association) is a legacy, full duplex, wired, asynchronous serial communication interface. The RS-232 interface is developed by the Electronics Industries Association (EIA) during the early 1960s. RS-232 extends the UART communication signals for external data communication.

UART uses the standard TTL/CMOS logic (Logic 'High' corresponds to bit value 1 and Logic 'Low' corresponds to bit value 0) for bit transmission whereas RS-232 follows the EIA standard for bit transmission. As per the EIA standard, a logic '0' is represented with voltage between +3 and +25V and a logic '1' is represented with voltage between -3 and -25V. In EIA standard, logic '0' is known as 'Space' and logic '1' as 'Mark'. The RS-232 interface defines various handshaking and control signals for communication apart from the 'Transmit' and 'Receive' signal lines for data communication. RS-232 supports two different types of connectors, namely; DB-9: 9-Pin connector and DB-25: 25-Pin connector. Figure 2.31 illustrates the connector details for DB-9 and DB-25.

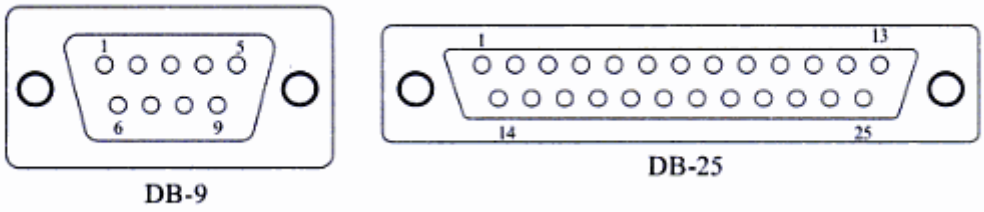


Fig. 2.31 DB-9 and DB-25 RS-232 Connector Interface

The pin details for the two connectors are explained in the following table:

Pin Name	Pin no: (For DB-9 Connector)	Pin no: (For DB-25 Connector)	Description
TXD	3	2	Transmit Pin for Transmitting Serial Data
RXD	2	3	Receive Pin for Receiving Serial Data
RTS	7	4	Request to send.
CTS	8	5	Clear To Send
DSR	6	6	Data Set Ready
GND	5	7	Signal Ground
DCD	1	8	Data Carrier Detect
DTR	4	20	Data Terminal Ready
RI	9	22	Ring Indicator
FG		1	Frame Ground
SDCD		12	Secondary DCD
SCTS		13	Secondary CTS
STXD		14	Secondary TXD
TC		15	Transmission Signal Element Timing
SRXD		16	Secondary RXD
RC		17	Receiver Signal Element Timing
SRTS		19	Secondary RTS
SQ		21	Signal Quality detector
NC		9	No Connection
NC		10	No Connection
NC		11	No Connection
NC		18	No Connection
NC		23	No Connection
NC		24	No Connection
NC		25	No Connection

RS-232 is a point-to-point communication interface and the devices involved in RS-232 communication are called 'Data Terminal Equipment (DTE)' and 'Data Communication Equipment (DCE)'. If no data flow control is required, only TXD and RXD signal lines and ground line (GND) are required for data transmission and reception. The RXD pin of DCE should be connected to the TXD pin of DTE and vice versa for proper data transmission.

If hardware data flow control is required for serial transmission, various control signal lines of the RS-232 connection are used appropriately. The control signals are implemented mainly for modem communication and some of them may not be irrelevant for other type of devices. The Request To Send (RTS) and Clear To Send (CTS) signals co-ordinate the communication between DTE and DCE. Whenever the DTE has a data to send, it activates the RTS line and if the DCE is ready to accept the data, it activates the CTS line.

The Data Terminal Ready (DTR) signal is activated by DTE when it is ready to accept data. The Data Set Ready (DSR) is activated by DCE when it is ready for establishing a communication link. DTR should be in the activated state before the activation of DSR.

The Data Carrier Detect (DCD) control signal is used by the DCE to indicate the DTE that a good signal is being received.

Ring Indicator (RI) is a modem specific signal line for indicating an incoming call on the telephone line.

The 25 pin DB connector contains two sets of signal lines for transmit, receive and control lines. Nowadays DB-25 connector is obsolete and most of the desktop systems are available with DB-9 connectors only.

As per the EIA standard RS-232 C supports baudrates up to 20Kbps (Upper limit 19.2 Kbps) The commonly used baudrates by devices are 300bps, 1200bps, 2400bps, 9600bps, 11.52Kbps and 19.2Kbps. 9600 is the popular baudrate setting used for PC communication. The maximum operating distance supported by RS-232 is 50 feet at the highest supported baudrate.

Embedded devices contain a UART for serial communication and they generate signal levels conforming to TTL/CMOS logic. A level translator IC like MAX 232 from Maxim Dallas semiconductor is used for converting the signal lines from the UART to RS-232 signal lines for communication. On the receiving side the received data is converted back to digital logic level by a converter IC. Converter chips contain converters for both transmitter and receiver.

Though RS-232 was the most popular communication interface during the olden days, the advent of other communication techniques like Bluetooth, USB, Firewire, etc are pushing down RS-232 from the scenes. Still RS-232 is popular in certain legacy industrial applications.

RS-232 supports only point-to-point communication and not suitable for multi-drop communication. It uses single ended data transfer technique for signal transmission and thereby more susceptible to noise and it greatly reduces the operating distance.

RS-422 is another serial interface standard from EIA for differential data communication. It supports data rates up to 100Kbps and distance up to 400 ft. The same RS-232 connector is used at the device end and an RS-232 to RS-422 converter is plugged in the transmission line. At the receiver end the conversion from RS-422 to RS-232 is performed. RS-422 supports multi-drop communication with one transmitter device and receiver devices up to 10.

RS-485 is the enhanced version of RS-422 and it supports multi-drop communication with up to 32 transmitting devices (drivers) and 32 receiving devices on the bus. The communication between devices in the bus uses the 'addressing' mechanism to identify slave devices.

2.4.2.2 Universal Serial Bus (USB) Universal Serial Bus (USB) is a wired high speed serial bus for data communication. The first version of USB (USB1.0) was released in 1995 and was created by the USB core group members consisting of Intel, Microsoft, IBM, Compaq, Digital and Northern Telecom. The USB communication system follows a star topology with a USB host at the centre and one or more USB peripheral devices/USB hosts connected to it. A USB host can support connections up to 127, including slave peripheral devices and other USB hosts. Figure 2.32 illustrates the star topology for USB device connection.

USB transmits data in packet format. Each data packet has a standard format. The USB communication is a host initiated one. The USB host contains a host controller which is responsible for controlling the data communication, including establishing connectivity with USB slave devices, packetizing and formatting the data. There are different standards for implementing the USB Host Control interface; namely Open Host Control Interface (OHCI) and Universal Host Control Interface (UHCI).

The physical connection between a USB peripheral device and master device is established with a USB cable. The USB cable supports communication distance of up to 5 metres. The USB standard uses two different types of connector at the ends of the USB cable for connecting the USB peripheral device and host device. 'Type A' connector is used for upstream connection (connection with host) and Type B connector is used for downstream connection (connection with slave device). The USB connector present in desktop PCs or laptops are examples for 'Type A' USB connector. Both Type A and Type B connectors contain 4 pins for communication. The Pin details for the connectors are listed in the table given below.

Pin no:	Pin name	Description
1	V_{BUS}	Carries power (5V)
2	D–	Differential data carrier line
3	D+	Differential data carrier line
4	GND	Ground signal line

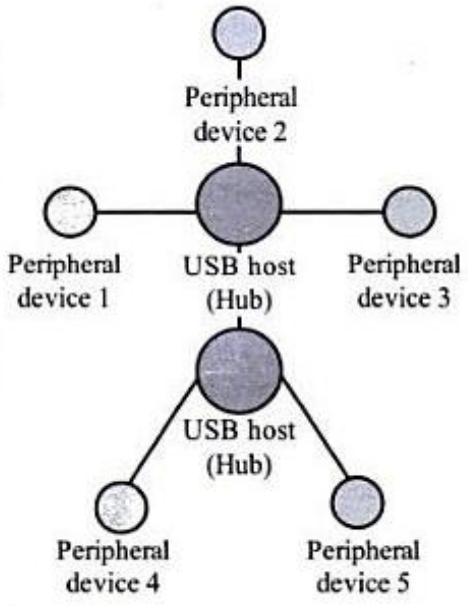


Fig. 2.32 **USB Device Connection topology**

USB uses differential signals for data transmission. It improves the noise immunity. USB interface has the ability to supply power to the connecting devices. Two connection lines (Ground and Power) of the USB interface are dedicated for carrying power. It can supply power up to 500 mA at 5 V. It is sufficient to operate low power devices. Mini and Micro USB connectors are available for small form factor devices like portable media players.

Each USB device contains a Product ID (PID) and a Vendor ID (VID). The PID and VID are embedded into the USB chip by the USB device manufacturer. The VID for a device is supplied by the USB standards forum. PID and VID are essential for loading the drivers corresponding to a USB device for communication.

USB supports four different types of data transfers, namely; Control, Bulk, Isochronous and Interrupt. **Control transfer** is used by USB system software to query, configure and issue commands to the USB device. **Bulk transfer** is used for sending a block of data to a device. Bulk transfer supports error checking and correction. Transferring data to a printer is an example for bulk transfer. **Isochronous data** transfer is used for real-time data communication. In Isochronous transfer, data is transmitted as streams in real-time. Isochronous transfer doesn't support error checking and re-transmission of data in case of any transmission loss. All streaming devices like audio devices and medical equipment for data collection make use of the isochronous transfer. **Interrupt transfer** is used for transferring small amount of data. Interrupt transfer mechanism makes use of polling technique to see whether the USB device has any data to send. The frequency of polling is determined by the USB device and it varies from 1 to 255 milliseconds. Devices like Mouse and Keyboard, which transmits fewer amounts of data, uses Interrupt transfer.

USB.ORG (www.usb.org) is the standards body for defining and controlling the standards for USB communication. Presently USB supports four different data rates namely; Low Speed (1.5Mbps), Full Speed (12Mbps), High Speed (480Mbps) and Super Speed (4.8Gbps). The Low Speed and Full Speed specifications are defined by USB 1.0 and the High Speed specification is defined by USB 2.0. USB 3.0 defines the specifications for Super Speed. USB 3.0 is expected to be in action by year 2009. There is a move happening towards wireless USB for data transmission using Ultra Wide Band (UWB) technology. Some laptops are already available in the market with wireless USB support.

3.10.3 USB Bus

Universal Serial Bus (USB) is a bus between host system and number of interconnected peripheral devices. A maximum 127 devices can connect to a host. It provides a fast (up to 12 Mbps) and as well as a low speed (up to 1.5 Mbps) serial transmission and reception between host and serial devices. A USB host, which includes controller for function as bus master can connect flash memory cards, pen-like memory devices, digital camera, printer, mice, PocketPC and video games. Thre are three standards: USB 1.1(a low speed 1.5 Mbps 3 m channel along with a high speed 12 Mbps, 25 m channel); USB 2.0 (high speed 480 Mbps 25 meter channel), and wireless USB (high speed 480 Mbps 3 m).

USB protocol has this feature—a USB device can be hot plugged (attached), configured and used, reset, reconfigured and used; it can share bandwidth with other devices, detached (while others are in operation)

and reattached. Attaching and detaching can be done without rebooting. The host schedules sharing of bandwidth among the attached devices. A USB device can either be bus-powered or self-powered. In addition, there is a power management by software at host for USB ports.

USB host connects to devices or nodes using USB port-drving software and the host controller connected to a root hub. A hub is one that connects to other nodes or hubs. A tree-like topology forms as follows. The root hub connects to the hub and node at level 1. A hub at level 1 connects to the hub and node at level 2 and so on. Only the nodes are present at the last level. The root hub and each hub at a level connect in a star topology with the next level. The USB device descriptor data structure has a hierarchy, which is as follows: It has device descriptor at the root that has number of configuration descriptors and each configuration descriptor has number of interface descriptors and which has number of end point descriptors.

USB bus cable has four wires, one for +5 V, two for twisted pairs and one for ground. There are termination impedances at each end that are as per the device speed. Electromagnetic Interference (EMI)-shielded cable is used for 15 Mbps USB devices.

Serial signals are Non Return to Zero ((NRZI) and the clock is encoded by inserting a synchronous code (SYNC) field before each packet. [Refer to Table 3.2]. The receiver synchronizes its bit recovery clock continuously. The data transfer is of four types: (a) Controlled data transfer (b) Bulk data transfer (c) Interrupt driven data transfer (d) Isosynchronous transfer.

USB is a polled bus. The host controller circuit regularly polls the presence of a device as scheduled by the software. It sends a token packet. The token consists of fields for type, direction, USB device address and device end-point number. The device does the handshaking through a handshake packet, indicating successful or unsuccessful transmission. A CRC field in a data packet enables transmission error detection at the receiver.

USB supports three types of pipes—(a) 'Stream' with no USB-defined protocol. It is used when the connection is already established and the data flow starts. (b) 'Default Control' for providing access. (c) 'Message' for the control functions of the device. The host configures each pipe for the followings: (a) data bandwidth to be used, (b) transfer service type and (c) buffer sizes.

Wireless USB is wireless extension of USB 2.0 and it operates at UWB (ultra wide band) 3.1 to 10.6 GHz frequencies. It is used for short-range personal area network (high speed 480 Mbps, 3 m or 110 Mbps, 10 m channel). FCC has recommended a host wire adapter (HWA) and a device wire adapter (DWA), which provide wireless USB solutions. Wireless USB also supports dual-role devices (DRDs). A device can be a USB device as well as a limited capability host. For example, a wireless USB digital camera uses a USB host when connected to a printer and a USB device when connected to a personal computer. A wireless USB device is used to provide Internet connectivity between laptop or computer and mobile service provider network.

USB is a serial bus that interconnects a system. It attaches and detaches a device from the network. It uses a root hub. Nodes containing the devices can be organized like a tree structure. It is mostly used in networking the IO devices like scanner in a computer system. Wireless USB is used for remote connections without wires.

5.6 Universal Serial Bus (USB)

One basic problem with RS-232C and similar such old standards of communication is that of difficulty in making two devices talk to each other. They need right type of cable and connectors, manual intervention to coordinate the parameters like data rate, parity, handshaking, etc. *Universal Serial Bus* (USB) has emerged as a solution to interconnect peripherals and computers in a standard way. It uses a single, standardized interface socket and provides improved plug-and-play capabilities allowing devices to be connected and disconnected without rebooting the computer (hot swapping). It also provides power to low consumption devices without the need for an external power supply and allowing many devices to be used without requiring manufacturer specific, individual device drivers to be installed.

USB 1.0 specification was introduced in 1995, promoted by Intel, Microsoft, Philips etc. The revised version *USB 1.1* came out in 1998. USB 1.1 supports data rates of 12 Mbps (*Full Speed*) and 1.5 Mbps (*Low Speed*). *Full Speed* was intended for high speed devices, such as disk drives, whereas, *Low Speed* for slower devices, such as joysticks. USB 2.0 was released in 2000 and ratified in 2001. It provides higher data transfer rate of 480 Mbps, while being fully compatible with USB 1.1. USB 3.0 has been published in 2008 and devices having this interface started reaching the market by 2010. Apart from being backward compatible with USB 2.0, USB 3.0 includes a new higher speed bus, called *SuperSpeed* in parallel with USB 2.0 bus. It has data transfer rate of upto 5 Gbps. Two-way communication is possible in USB 3.0. Using the *SuperSpeed* transfer, full duplex communication can be achieved, while the earlier USB 1.1 and USB 2.0 are only half duplex.

USB has an asymmetric design, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure, subject to a limit of five levels of tiers. One such structure has been shown in Fig. 5.8. A USB host may have multiple host controllers. Each host controller may provide one or more USB ports. Up to 127 devices

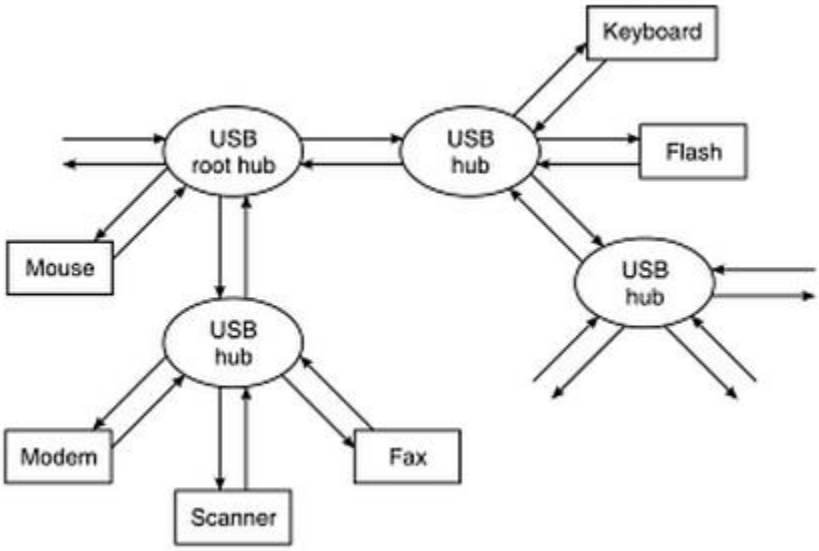


Fig. 5.8 Tiered USB tree structure.

including hub devices can be connected to a single host controller. Each host controller has a *root hub*. *Sharing hubs* are also possible, allowing multiple computers to access the same peripheral device(s). A single USB device may consist of several logical sub-devices, referred to as *device functions*. For example, a webcam (video device function) with a built-in microphone (audio device function).

The host regularly polls hubs for their status. When a new device is plugged into a hub, the hub advises the host of its change of state. The host in turn issues a command to enable and reset the port. The device responds and the host collects information about the device. Based on this retrieved information, the host operating system determines the software driver to be used for the device. A unique address is assigned to the device. On the other hand, when a device is unplugged, the hub advises the host about the change in state, when it is polled by the host. The host, in turn, removes the device from its list of available resources. This process of detection and identification of USB devices by a host is called *bus enumeration*.

USB device communication is based on *pipes* (logical channels). Pipes are connections from the host controller to a logical entity on the device, known as *endpoint*. A USB device can have up to 32 active pipes—16 into the host controller, 16 out of the host controller. Each pipe is unidirectional—an endpoint can transfer data in one direction only. Endpoints are grouped into *interfaces*, each interface is associated with a single device function. Endpoint zero is used for device configuration, and thus not associated with any interface. Figure 5.9 shows some USB endpoints.

USB supports various *device classes*. Devices attached to the bus can be full-custom devices, requiring a full-custom device driver, or may belong to a device class. The classes define the expected behaviour of the device and interface descriptors, so that the same device driver can be used for all such devices belonging to a class. An operating system is expected to implement all device classes to provide generic drivers for any USB device. Some of the common such device classes are noted in Table 5.3.

Four different types of data transfer can take place in USB. These are:

- Control transfer—to configure the bus and return status information.
- Bulk transfer—to move data asynchronously. It is bidirectional.
- Isochronous transfer—to move time critical data to an output device. This is unidirectional transfer without any error check.
- Interrupt transfer—to receive/transmit data at regular intervals, ranging from 1 to 255 milli seconds.

There are two types of pipes in USB communication—stream pipes and message pipes. A stream pipe is a unidirectional pipe connected to a unidirectional endpoint. Such a pipe is used in isochronous, interrupt, and bulk transfers. On the other hand, a message pipe is bidirectional, connected to a bidirectional endpoint and used exclusively for control transfers. To start a data transfer session, the host sends a TOKEN packet to the control endpoint of the device. The TOKEN packet can be an IN packet or an OUT packet. If the direction of data transfer is from the host to the endpoint, an OUT packet having the desired device address and endpoint number is sent by the host. For a data transfer from a device to the host, the host sends an IN packet. Once the TOKEN packet is accepted by the device, data transfer can start. This constitutes a control transfer, it uses message pipes.

5.1 Serial Peripheral Interface (SPI)

Serial Peripheral Interface has been developed by *Motorola* to provide a simple, low-cost interface between microcontrollers and peripheral chips. The interface is also known as a *four-wire interface* as it uses four main signals to enable the data transfer. These signals, as shown in Fig. 5.1, are:

1. *MOSI:* Master Out Slave In
2. *MISO:* Master In Slave Out
3. *SCLK:* Serial Clock
4. *CS:* Chip Select

The *SPI* interface can be used for interfacing memory, ADC, DAC, real-time clock, LCD drivers, sensors, audio chips, and even other processors. The following are a few examples of such devices.

- *Temperature sensor:* LM74 having 12-bit plus sign temperature resolution (0.0625°C per LSB) with a temperature range of −55°C to +150°C.
- *Pressure sensor:* SCP1000 with 17-bit resolution. Under ideal conditions it can detect the pressure difference within a 9 cm column of air.
- *Analog-to-Digital converter:* LTC2452 is an ultra-tiny, differential, 16-bit delta-sigma ADC.

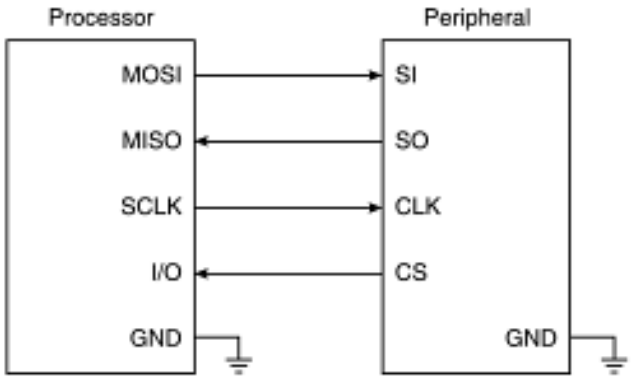


Fig. 5.1 The SPI interface.

- *Touch screen:* SX8652 is a very low power, high reliability controller for resistive touch screens. Supports wide input supply range of 1.65 V to 3.7 V.
- *EEPROM:* Microchip 25XXX serial EEPROM, with densities from 128 bits to 512 Kbits, 3-20MHz bus speed, low power operation, built-in write protection.
- *Real time clock:* DS3234 is a low-cost extremely accurate real-time clock with integrated temperature-compensated crystal oscillator and crystal. It counts seconds, minutes, hours, day, date, month, year with leap year compensation, valid upto 2099.
- *Memory card:* MMC and SD cards.

The most important feature of *SPI*, as compared to a standard serial port is that *SPI* is a synchronous protocol in which all transmissions are referenced to a common clock, generated by the master. The receiving peripheral (slave) utilizes the clock to synchronize the transmission.

Both the master and slave contain a serial shift register. The contents of these shift registers are exchanged to facilitate data transfer between the master and slave. The processor (master) initiates the transfer by writing a byte to its SPI shift register. As the register transmits the byte to the slave on the *MOSI* signal line, the slave transfers the content of its shift register back to the master on the *MISO* signal line. Thus, the contents of the two registers are exchanged (Fig. 5.2). Both a write and a read operation are performed with the slave simultaneously. If only the write operation is desired by the master, it simply ignores the byte it receives from the slave. On the other hand, if only a read operation is required by

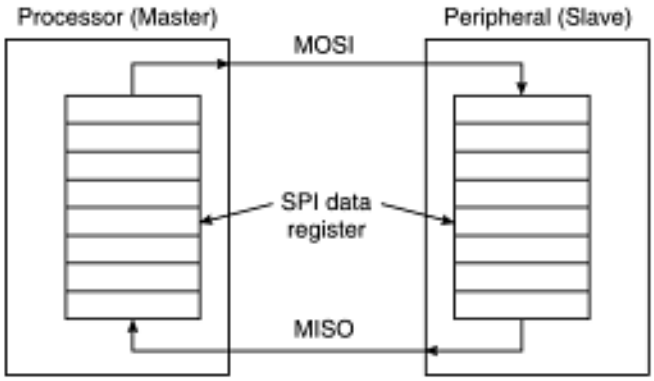


Fig. 5.2 Data transfer through SPI interface.

the master, it must transfer a dummy byte to the slave in order to initiate a slave transmission. Transmission often consists of 8-bit words, however, other word sizes are also common (for example, 16-bit words for touch-screen controllers and audio codecs, 12-bit words for many DACs and ADCs). Multibyte transmission is also possible. More than one peripheral chip can be connected to the same master through *SPI* interface. The slaves are selected by the master by asserting the corresponding chip-select input. A multi-peripheral interface has been shown in Fig. 5.3.

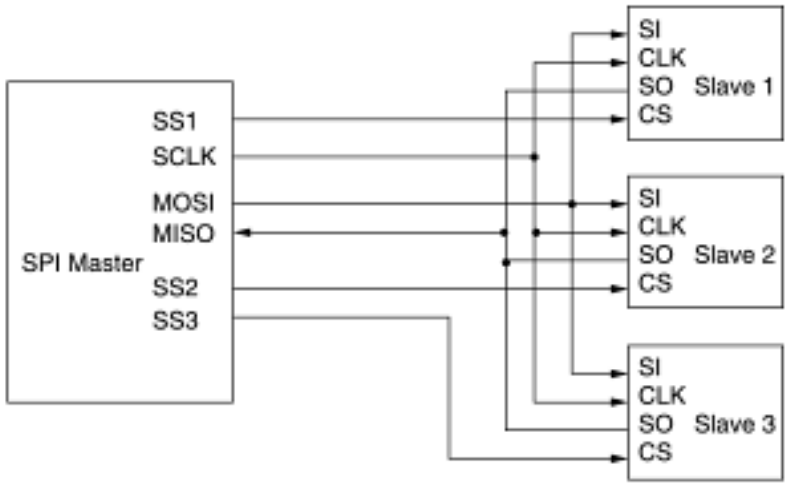


Fig. 5.3 Connecting multiple slave devices to a master.

Conventionally, apart from setting the clock frequency, the master also configures the polarity and phase of the clock signal with respect to data. This is done by setting the *CPOL* and *CPHA*. These often constitute two bits of the SPI control register. With $CPOL = 0$, the base value of the clock is taken as zero. Now, if $CPHA = 0$, data are captured on the rising edge of the clock and transmitted at the falling edge. With $CPHA = 1$, data are captured on the falling edge of the clock signal and transmitted on the rising edge. With $CPOL = 1$, the base value of clock signal is taken as 1. The situation has been shown in Fig. 5.4.

5.2 Inter-Integrated Circuit (*IIC*, I^2C)

I^2C bus is a very cheap, yet effective network used to interconnect peripheral devices within small-scale embedded systems. It uses two wires to connect multiple devices in a multidrop bus. The bus is bi-directional, low-speed, and synchronous to a common clock. Achievable data rate varies between 100–400 Kbps. The two wires are named as,

- *SDA:* Serial Data
- *SCL:* Serial Clock

Both the lines are open-drain and bi-directional. Unlike *SPI*, I^2C uses same signal line for master transmission, as well as slave response. A typical interface is shown in Fig. 5.5. The mode of operation of I^2C is also much simpler than *SPI*. In the idle condition, both *SDA* and *SCL* are high. The start condition is indicated by the sequence of *SDA* going low followed by *SCL* going low. Now *SDA* transitions for the first valid data bit. For each bit that is transmitted, it must become valid on *SDA* while *SCL* is low. The rising edge of *SCL* is used to sample the bit. The bit must remain valid till *SCL* goes low once more. Now, *SDA* transitions to the next bit, before *SCL* goes high once more. The stop condition is indicated by *SCL* returning to high followed by *SDA*.

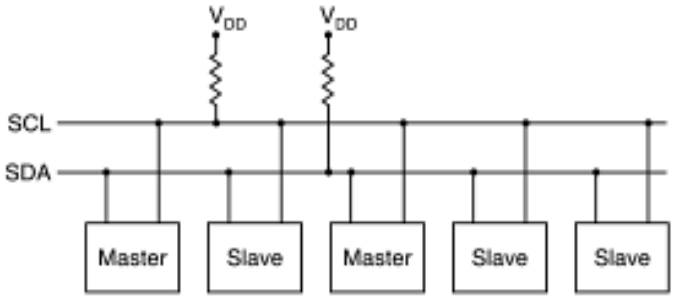


Fig. 5.5 Data transfer through I^2C interface.

Any number of bytes can be transmitted in an I^2C packet. If the receiver is unable to accept any more bytes, it can abort transmission by holding *SCL* low. In that case, transmitter waits till *SCL* is released by the receiver.

To facilitate acknowledgement, after transmitting the 8-th bit, master releases the *SDA* and gives an additional clock pulse on *SCL*. This triggers the receiver to acknowledge the byte by pulling *SDA* low.

Since there can be multiple devices on the same bus, each device has a unique 7-bit address. The first byte transmitted is the address byte along with a direction bit. The direction bit '0' indicates a *write* operation in which the slave will receive data, whereas the direction bit '1' indicates a *read* operation in which the slave will send data. Figure 5.6 shows the timing diagram of I^2C transfer.

To compare between SPI and I^2C, SPI supports full duplex communication with higher throughput than I^2C. SPI communication is not limited to 8-bit words. As discussed earlier, it can utilize higher word sizes. Hence, we can send any message sizes with arbitrary content and purpose. The SPI interface does not require pull-up resistors. This results in lower power consumption. However, I^2C is simpler by having fewer lines which means fewer pins are required to interface to an IC. When communicating with more than one slave, I^2C has the advantage of in-band addressing as opposed to have a chip select line for each slave. I^2C also supports slave acknowledgement. This ensures the existence of the receiving device to the transmitter. This is not possible with SPI. In general SPI is better suited for applications that deal with longer data streams and not just words like address locations. Mostly longer data streams exist in applications working with a digital signal processor or analog-to digital converter. For example, SPI would be perfect for playing back some audio stored in an EEPROM and played through a digital to analog converter DAC. Moreover, since SPI can support significantly higher data rates comparing to I^2C, mostly due to its duplex capability, it is far more suited for higher speed applications reaching tens of megahertz. Since there is no device addressing involved in SPI, the protocol is a lot harder to use in multiple slave systems. This means when dealing with more than one node, generally I^2C is the way to go. The choice of SPI or I^2C depends on the following criteria.

- 2 or 3 wires are available
  - 2 I^2C
  - 3/4 SPI
- Speed requirement
  - 100–400 Kbits I^2C
  - 1 to 4 Mbit SPI

The CAN bus

Each node in a CAN consists of the following:

- *A host processor:* It determines the type of the messages received, their meaning, and also the messages the node wants to transmit.
- *A CAN controller:* It is a hardware unit working on a synchronous clock signal. It performs the operation of receiving and sending messages. On the receiver side, it stores the received bits from the bus until the entire message is available. The CAN controller then sends an interrupt to the host processor to inform it to collect the message. For sending messages, host controller stores the message to be transmitted into the CAN controller which then transmits bits serially through the bus.
- *A transceiver:* The transceiver may be integrated with the CAN controller. On the receiving side, it adapts signal levels from the bus to the levels that the CAN controller expects. It also possesses protective circuitry that protects the CAN controller. On the sending side, it converts the transmit bit signal received from CAN controller into a signal that can be sent over the CAN bus.

The structure of a CAN-bus has been shown in Fig 5.19. The High-Speed ISO 11898 Standard specification suggests both ends of the bus to be terminated by a 120 Ω resistor to avoid reflection. CAN-bus can transmit data at 1 Mbps at network lengths below 40 m with a maximum of 30 nodes. At lower data rates (for example, 125 Kbits/s), distances can be increased. For exmaple, at 125 Kbits/s, 500 m distance can be covered. The two signal lines in the CAN bus, CANH and CANL are passively biased at a quiscent voltage when no signal is being transmitted. In the dominant state, CANH assumes a positive voltage level compared to the quiscent level, while CANL assumes a negative value, creating a differential signal over the two lines.

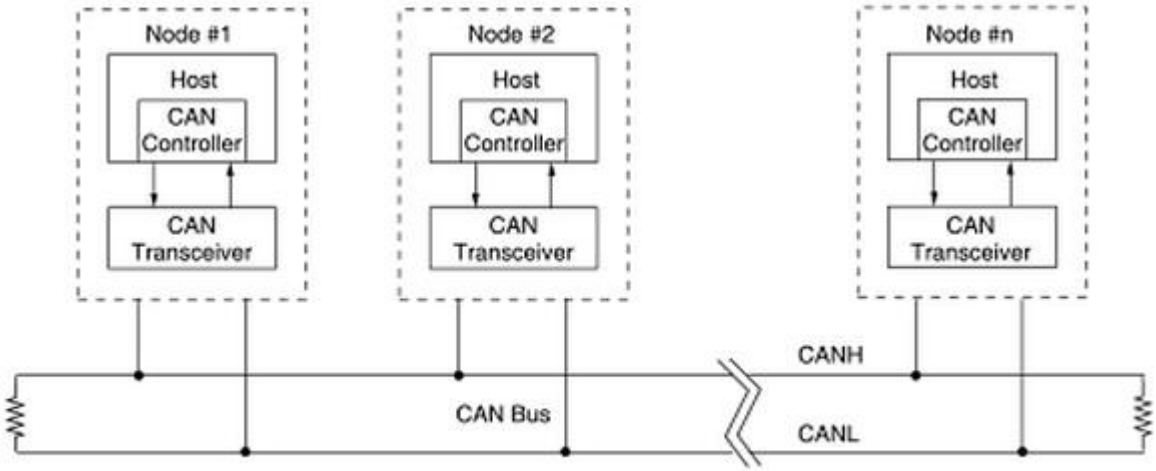


Fig. 5.19 CAN bus structure.

The CAN protocol consists of the following layers:

- *Physical layer:* It defines how the signals are actually transmitted. It includes signal level and bit representations with conversion, and maintenance of transmission medium.
- *Transfer layer:* It is the kernel of the CAN protocol responsible for bit timing and synchronization, message framing, arbitration, acknowledgement, error detection and signalling, fault confinement, etc.
- *Object layer:* It is responsible for message filtering, message and status handling.
- *Application layer:* This is responsible for differnt applications built on the underlying CAN protocol layers.

3.11.2 PCI and PCI/X Buses

Recently, the most used synchronous parallel bus in the computer system for interfacing PC-based devices is PCI (Peripheral Component Interconnect). PCI provides a superior throughput than EISA. It is almost platform-independent, unlike the ISA, which depended on the IBM PC platform, interrupt vectors, IO addresses and memory allocations. Its clock rate is nearest to the submultiple of the system clock. PCI provides three types of synchronous parallel interfaces. Its versions are 32/33 MHz, 64/66 MHz, PCI-X 64/100 MHz, PCI Super V2.3 264/528 MBps 3.3 V (on a 64-bit bus), 132/264 (on a 32-bit bus) and PCI-X Super V1.01a for 800 MBps 64-bit bus 3.3 V.

PCI bus has 32-bit data bus extendible to 64 bits. In addition, it has 32-bit addresses extendible to 64 bits. Its protocol specifies the interaction between the different components of a computer. A specification is version 2.1. Its synchronous/asynchronous throughput is up to 132/ 528 MB/s [33 M x 4/ 66 M x 8 Byte/s]. It operates on 3.3 V signals. A typical application is an exemplary PCI Card has a 16 MB Flash ROM with a router gateway for a LAN.

A PCI driver can access hardware automatically as well as by addresses assigned by the programmer. The PCI feature of automatically detecting the interfacing systems and assigning new addresses is important for coding a device driver. The PCI bus therefore simplifies the addition and deletion (attachment and detachment) of the system peripherals. A manufacturer registers a global number for PCI device or card, just as, 68HC11 or 80386 are globally registered numbers. A 16-bit register in PCI device identifies this number to let that device auto-detected. Another 16-bit register is for a device ID number. These two numbers allow the device to carry out auto-detection by its host computer. Each device may use FIFO controller with FIFO buffer for maximum throughput.

A device or host identifies its address space by three identification numbers (i) IO port, (ii) memory locations and (iii) configuration registers of total 256 B with a 4-byte unique ID. Each PCI device has address space allocation of 256 bytes to access it by the host computer. The unique feature of PCI bus is its configuration address space. A uniquely assigned interrupt type (a number) handles an interrupt. For example, interrupt type 3 has the interrupt vector address 0x0000C and four bytes at the address specify the interrupt service routine address. Interrupt type can be between 0x00 and 0xFF. A configuration register number 60 stores the one byte for the interrupt type n(pci) . The PCI device or host when interrupted handles the interrupt of type n(pci). Figure 3.13 shows 64-byte standard configuration registers in a PCI device. Following are the abbreviations used in the figure.

VID: Vendor ID. *DID*: Device ID. *RID*: Revision ID. *CR*: Common Register. *CC*: Class Code. *SR*: Status Register. *CL*: Cache Line. *LT*: Latency Timer. *BIST*: Base Input Tick. *HT*: Header Type. *BA*: Base Address. *CBCISB*: Card Base CIS Pointer. *SS*: Sub System. *ExpROM*: Expansion ROM. *MIN_GNT*: Minimum Guaranteed time. *MAX_GNT*: Maximum Guaranteed Time.

VID, *DID*, *RID*, *CR*, *SR*, and *HT* are compulsorily configured. The rest are optional.

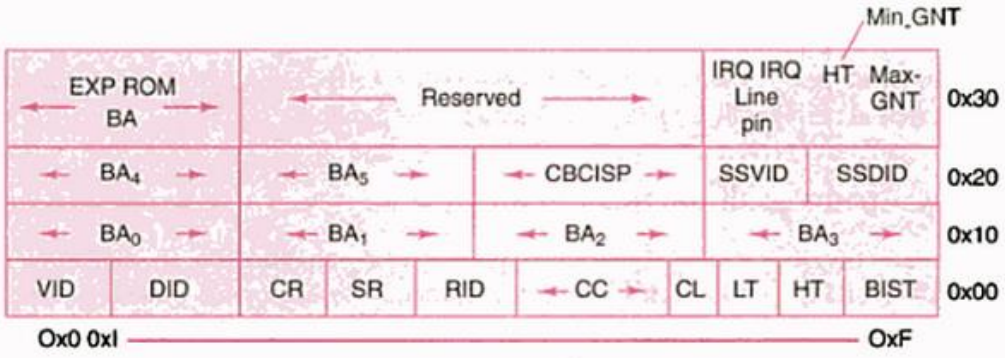


Fig. 3.13 64 bytes at standard device independent configuration registers in a PCI device or host

A PCI controller must access one device at a time. Thus, all the devices within host computer can share IO port addresses and memory locations but cannot share the configuration registers. That means that a device cannot modify other configuration registers but can access other device resources or share the work or assist the other device. If there are reasons for it doing so, a PCI driver can change the default bootup assignments on configuration transactions.

A device can initialize at booting time. This helps in avoiding any address collision. A PCI device on bootup disables its interrupt. Its address space is inaccessible and only the configuration registers space remains accessible. PCI BIOS with the device performs the configuration transactions and then memory and address spaces automatically map to the address space in host computer.

PCI parallel bus is popular in distributed embedded devices. PCI and PCI/X buses are used for parallel bus communication and these are independent from the IBM architecture. PCI/X is an extension of PCI and supports 64/100 MHz transfers. PCI bus new version support 132/528 MB/s data transfer with synchronous/ asynchronous throughputs.

3.11.1 ISA Bus

ISA bus (used in IBM Standard Architecture) connects only to an embedded device that has an 8086 or 80186 or 80286 processor, and in which the processor addressing and IBM PC architecture addressing limitations and interrupt vector address assignments are taken into account. There is no geographical addressing.

The limitation for memory access by a system using the ISA bus of the original IBM PC were as follows: ISA bus memory accesses can be in two ranges, 640 to 1 MB and 15 to 16 MB. The former range also overlaps with the range used by video boards and BIOS. [*Note*: Linux OS does not support the second range for accessing directly a device.]

The IO port address limitations for devices are as follows: The 8086 to 80286 processor has IO mapped IOs, not memory mapped IOs. Though the instruction set provides for IO instructions for 64 kB IO addresses, the IBM PC configuration ignores the address lines A_{10} to A_{15} and these are not decoded. Therefore, only 1024 IO port addresses are available. A hexadecimal addressing scheme with three nibble addressing between 000 to 3FF only can be used for a device. The A_{10} to A_{15} bits are thus immaterial. The following are the addresses allocated in IBM Standard Architecture (ISA).

1. Addresses allocated are 0x000–0x00F for DMA chip 8237. The addresses for other devices are as follows.
2. 0x020–0x021 addresses allocated are for programmable interrupt controller 8255. Hex 0x040–0x043 for timer 8253.
3. 0x060–0x063 for parallel port programmable parallel interface.
4. The hexa-decimal addresses 080-083, 0A0-0AF, 0C0-0CF, 0E0-0EF allocated are for components on the motherboard.

5. Reserved addresses from peripherals are hex 220-24F, 278-27F, 2F0-2F7, 3C0-3CF and 3E0 to 3F0.
6. The addresses allocated are hex 2F8-2FF and 3F8-3FF for IBM COM ports.
7. Addresses are hex. 320-32F and 3F0-3F7 for hard disk and floppy diskette, respectively.
8. Only 32 addresses between 0x300 to 0x31F are available for prototype card; for example, ADC card.
9. Addresses allocated are between hex 380-389 and 3A0-3A9 for synchronous communication.
10. Synchronous Data Link Control (SDLC) addresses allocated are between hex. 380-38C.
11. Display monitor ports are within 380-38F (monochrome) and 3D0-3DF for (colour and graphics).

There is a limited availability of interrupt vectors in the IBM PC 80x86 family. Only 256 vectors are available. Interrupt service functions are now shared at software level: for example, SWT interrupts. Original ISA specifications did not allow that.

EISA bus is a 32-bit data and address-lines version of ISA, and devices (system using this bus for IOs) are also supported. An EISA device driver first checks the EISA bus availability on the hosting computer or system. It supports the sharing of interrupt functions, SCI (Serial Communication Interface) controller and Ethernet devices. Unix and Linux support the EISA bus-driven cards and devices.

ISA and EISA buses are compatible with IBM architecture. They are used for connecting devices following IO addresses and interrupt vectors as per IBM PC architecture. EISA is 32-bit extension of ISA. It also supports software interrupt functions and Ethernet devices.