M.Tech. Credit seminar report, Electronic Systems Group, EE Dept, IIT Bombay, submitted Nov 2003

USB: Features and Circuits

Manoj Purohit (03307410)
Supervisor: Prof P.C.Pandey

Abstract

Universal Serial Bus is a built in feature of most Personal Computer chip sets, as well as operating system and other system software. It eliminates need of add-in cards and separate power supplies and its hot swapping capability allows users to easily attach and detach peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. USB communications, allows the use of only four wires for connection and up to 127 devices can be connected. Universal Serial Bus with the inclusion of High-speed mode, made available in version 2.0 has achieved equivalence with IEEE 1394 and features a wide range of chips to make existing products USB compliant. The USB device has USB trans-receiver along with serial interface engine and a processor, which serves as controller chip, which can be programmed like micro controller, thus making the development process easier and faster. This report describes the salient features of USB and various controller chips are compared to develop a USB interface for an electronic system.

Contents

1. Introduction

There are number of serial and parallel interfaces used by peripherals depending on the data transfer rate. The development of USB took place with objective of covering wide category of peripherals with a single interface. The copyright of USB 1.1 specifications was initially assigned to four corporations, all popular in PC hardware and software: Compaq, Intel, Microsoft and NEC. As it gained widespread acceptance the three giants, namely Hewlett Packard, Philips and Lucent Technologies also joined, and latest version USB 2.0 copyright is jointly owned by them and they have formed an organization called Universal Serial Bus-Implementers Forum (USB-IF). The specifications are available from USB-IF and is the authority to decide upon USB compliance. To get any new USB product in the market a nominal fee is charged to get vendor ID by USB-IF. USB is a serial bus for connecting peripherals. It is controlled by host and there can be only one host per bus [1]. It uses 4-shielded wires of which two are power (+5*V* & GND). The remaining two are twisted pair differential data signals. It uses a NRZI (Non Return to Zero Invert) encoding scheme to send data with a sync field to synchronize the host and receiver clocks [2]. The USB specification does not support any form of multimaster arrangement. However, the On-The-Go (OTG) specification has introduced a Host Negotiation Protocol, which allows two devices negotiate for the role of host. This is aimed at and limited to single point-to-point connections such as a mobile phone and personal organizer and not multiple hub, multiple device desktop configurations. The USB host is responsible for undertaking all transactions and scheduling bandwidth. Data can be sent by various transaction methods using a token-based protocol. The USB host controllers have their own specifications. With USB 1.1, there were two Host Controller Interface Specifications, UHCI (Universal Host Controller Interface) developed by Intel which puts more of the burden on software (Microsoft) and allowing for cheaper hardware and the OHCI (Open Host Controller Interface) developed by Compaq, Microsoft and National Semiconductor which places more of the burden on hardware (Intel) and makes for simpler software. With the introduction of USB 2.0 a new Host Controller Interface Specification the EHCI (Enhanced Host Controller Interface) is there whose significant Contributors include Intel, Compaq, NEC, Lucent and Microsoft so now it is one interface standard and thus only one new driver to implement in operating systems [3]. An important feature of USB is its transfer modes. In that, Isochronous transfer mode allows a device to reserve a defined about of bandwidth with guaranteed latency. This is ideal in Audio or Video applications where congestion may cause loss of data or frames to drop. Each transfer mode provides the designer trade-offs in areas such as error detection and recovery, guaranteed latency and bandwidth [4].

2. Salient features

USB supports Control, Interrupt, Bulk and Isochronous transfers. It supports plug'n'plug with dynamically loadable and unloadable drivers and uses a tiered star topology, similar to that of 10BaseT Ethernet. It uses 4-shielded wires of which two are power remaining two are twisted pair differential data signals and maximum 127 devices can be connected to any one USB bus at any one given time, but the bandwidth is shared accordingly [4].

USB speeds

USB version 1.1 supported two speeds, a full speed mode of 12Mbits/s and a low speed mode of 1.5Mbits/s. The 1.5Mbits/s mode is slower and less susceptible to EMI, thus reducing the cost of ferrite beads and quality components [4].

1) High Speed – 480Mbits/s
2) Full Speed – 12Mbits/s
3) Low Speed – 1.5Mbits/s

Data signaling rate

It specifies the tolerance of the USB clocks in the USB specification [3]

1) High-speed data is clocked at 480.00 Mb/s with a data signaling tolerance of ± 500ppm.
2) Full speed data is clocked at 12.000 Mb/s with a data signaling tolerance of ± 0.25% or 2,500ppm.
3) Low speed data is clocked at 1.50 Mb/s with a data signaling tolerance of ± 1.5% or 15,000ppm.

This allows resonators to be used for low cost low speed devices, but rules them out for full or high-speed devices [3].

3. Mechanical interface

3.1 USB connectors

All devices have an upstream connection to the host and all hosts have a downstream connection to the device. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loop back connections at hubs such as a downstream port connected to a downstream port. There are commonly two types of connectors, called type A and type B [3].

Type A plugs (Figure 1) always face upstream. Type A sockets will typically find themselves on hosts and hubs. For example type A sockets are common on computer main boards and hubs.

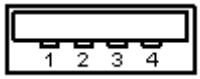

FIGURE 1: TYPE A CONNECTOR [4]

Type B plugs (Figure 2) are always connected downstream and consequently type B sockets are found on devices.

FIGURE 2: TYPE B CONNECTOR [4]

USB 2.0 included errata, which introduces mini-USB B connectors for the range of miniature electronic devices such as mobile phones and organizers, as the current type B connector is too large to be easily integrated into these devices. Recently released has been the On-The-Go specification, which adds peer-to-peer functionality to USB. This introduces USB hosts into mobile phone and electronic organizers, and thus has included a specification for mini-A plugs, mini-A receptacles, and mini-AB receptacles [4].

3.2 Cable assembly and length

A full speed segment could be up to 5 meters and a low speed segment could be up to 3 meters. The USB specification prohibits use of extension cable, except an active extension cable, which consists of a hub, a down-stream port and a cable. The cable consists of four wires (Table 1).

TABLE 1: USB PIN FUNCTIONS [4]

Pin Number	Cable Color	Function
1	Red	V_{BUS} (5 V)
2	White	D -
3	Green	D +
4	Black	Ground

4. USB architecture

The USB supports USB devices attaching to and detaching from USB at any time. A USB system is described by interconnection, devices and host. The device can be directly connected to Host or through a Hub (Figure 3). The USB provides a shared interconnect and access is scheduled to provide various data transfer types with elimination of arbitration overhead.

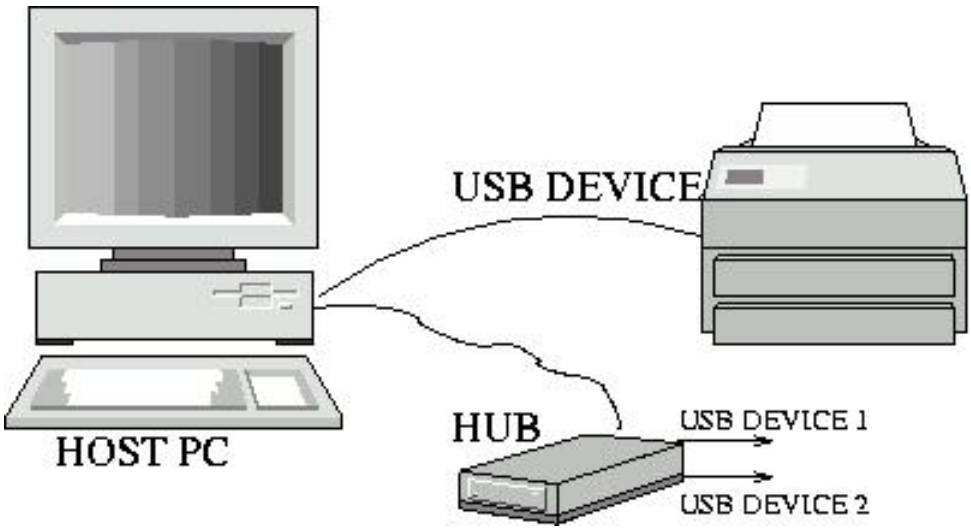

FIGURE 3: USB DEVICE CAN BE CONNECTED TO HOST DIRECTLY OR THROUGH A HUB

4.1 Bus topology

This defines the connection model between USB devices and host, the physical interconnect is a seven tiered star topology with host at its apex. The timing constraints posed by cable and hub limits tiers up to seven. A compound device can be connected only up to fifth tier because it occupies two tiers and consists of a hub inside a peripheral.

4.2 Interlayer relationship

There are different layers namely functional layer, device layer and USB interface layer, beneath a simple connection between host and peripheral. Even though physical and logical topology of USB reflects shared nature of bus, the interlayer relationship at both host and device end presents a view that client software manipulating a USB function deals only with interface of interest.

4.3 System configuration [3]

USB supports device attachment and removal at any time, so the system software must accommodate dynamic changes in the physical bus topology. Signal integrity using differential drivers, receivers, and shielding are attributes contributing to robustness of USB.

4.3.1 Device attachment

When a device is attached, the host enables the port and addresses of the USB device through the control pipe of device at the default address. The host assigns a unique USB address to device and

then determines if the newly attached device is a Hub or a function. The host establishes its end of the control pipe for the USB device using the assigned USB address and endpoint number zero. If the attached device is a hub and USB devices are attached to its ports then same procedure is carried for each device.

4.3.2 Device removal

When a User removes a device from a bus, the hub disables the device's port. The host learns that the removal occurred after polling the hub, learning that an event has occurred, and sending port status request to find out nature of event, operating system removes device from device manager's display and device's address becomes available to another newly attached device.

4.3.3 Bus Enumeration

Bus enumeration is the activity that identifies and assigns unique addresses to devices attached to a bus. Because devices can be attached or detached at any given time, bus enumeration is an on-going activity for USB system software. Additionally, bus enumeration also includes the detection and processing of removals.

4.4 USB devices and characterization

USB devices are divided into device classes such as hub, human interface, printer, imaging, or mass storage device. The hub device class indicates a specially designated USB device that provides additional USB attachment points. USB devices are required to carry information for self-identification and generic configuration. USB devices are accessed by a USB address that is assigned when the device is attached and enumerated. Each device in addition to pipe at endpoint zero, can support additional pipes through which host may communicate with the device. The information required to completely describe the USB device falls in category of Standard, Class, USB vendor, control and status [3].

4.5 USB data flow model [4]

USB bus interface layer provides physical/signaling/packet connectivity between host and a device.

4.5.1 USB Functions

USB functions are the USB devices, which provide a capability, or function such as a Printer, Zip Drive, and Scanner, Modem or other peripheral. Most functions will have a series of buffers, typically 8 bytes long. Each buffer will belong to an endpoint - EP0 IN, EP0 OUT etc. A USB device means a USB transceiver device used at the host or peripheral, a USB Hub or Host Controller IC device, or a USB peripheral device.

4.5.2 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. Endpoints act 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.

4.5.3 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. Default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type. USB bandwidth is allocated among pipes. USB devices are required to provide buffering of data so devices requiring more bandwidth provide large buffers.

USB defines two types of pipes: -

1. Stream Pipes have no defined USB format, any type of data can be sent down a stream pipe and it can retrieve the data out the other end. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes support bulk, isochronous and interrupt transfer types. Stream pipes can be controlled by either the host or device.
2. 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. Message pipes allow data to flow in both directions but supports only control transfers.

4.5.4 Data flow types

The Universal Serial Bus specification defines four transfer/endpoint types depending upon the kind of application needed: -

1) *Control Transfers*: Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed using control transfers. They are typically burst, random packets that are initiated by the host and use best effort delivery. The packet length of control transfers in low speed devices must be 8 bytes; high-speed devices allow a packet size of 8, 16, 32 or 64 bytes and full speed devices must have a packet size of 64 bytes.
2) *Interrupt Transfers*: In USB if a device requires the attention of the host, it must wait until the host polls it before it can report that it needs urgent attention. Interrupt transfers are typically non-periodic, small device "initiated" communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data.
3) *Isochronous Transfers*: Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. If there were a delay or retry of data in an audio stream, then one would expect some erratic audio containing glitches. The beat may no longer be in sync. However if a packet or frame was dropped every now and again, it is less likely to be noticed by the listener.
4) *Bulk Transfers*: Bulk transfers can be used for large burst data. Such examples could include a print-job sent to a printer or an image generated from a scanner. Bulk transfers provide error correction in the form of a CRC16 field on the data payload and error detection/re-transmission mechanisms ensuring data is transmitted and received without error.

5. USB communications

USB communications can be divided into two types, depending on whether they are used in initial configuration or in applications. In configuration communications, the host learns about the device and prepares it for exchanging data. Most of these communications take place when the host enumerates the device on power up or attachment. Application communications occur when application on host exchange data with an enumerated device. These are communication that carry out device purpose, e.g. for a keyboard, the application communications are the sending of key press data to the host, to tell an application to display a character or perform other actions [2].

5.1 Configuration communications

During enumeration, the device firmware responds to a series of standard requests from the host. The device must identify each request, return the requested information, and take other actions specified by the requests, on PCs, Windows performs the enumeration, so there is no user's programming involved. However to complete the enumeration, windows must have to files available, an INF file that identifies the file name and location of the device's driver, and the device driver itself. Depending on the device and how it will be used, the device driver may be one that is included with Windows or provided by chip or peripheral vendor, the INF file is basically a text file.

5.2 Application communications

After Host has exchanged enumeration information with the device and a device driver has been assigned and loaded, applications at the host can use standard Windows API functions to read and write to the device. At the device, transferring data typically requires placing data to send in the USB controller's transmit buffer, reading received data from the receive buffer when a hardware interrupt signals that data has arrived, and on completing a transfer, ensuring the device is ready for the next transfer. Most devices also require some additional support for handling errors and other events.

6. USB protocol layers

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. USB controller ICs takes care of the lower layer, thus making

it almost invisible to the end designer. Each USB transaction consists number of packets which are classified as: -

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

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 a handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data [4].

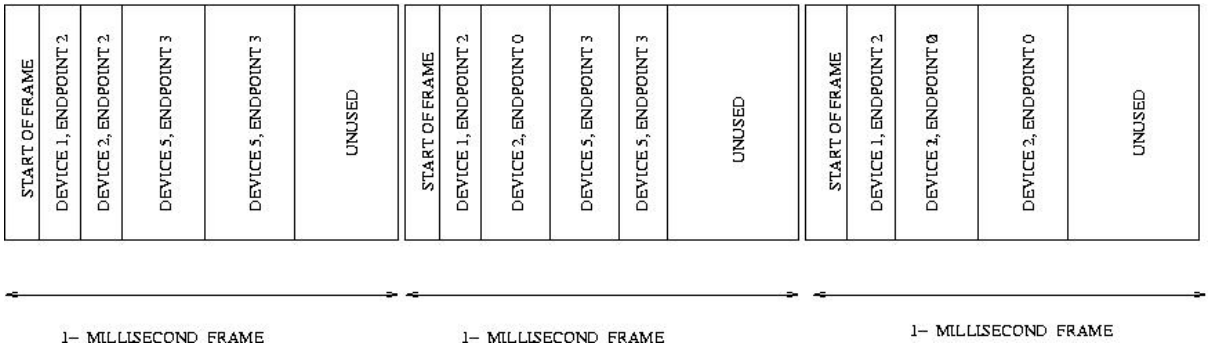

FIGURE 4: HOST SCHEDULES TRANSACTIONS WITHIN 1-MILLISECOND FRAME [2]

The transfer is in frames of 1ms (figure 4). Each frame begins with a Start-of-frame packet, followed by transactions that transfer data to or from device endpoints. The host can schedule transactions anywhere within a frame.

7. Signals and encoding

.

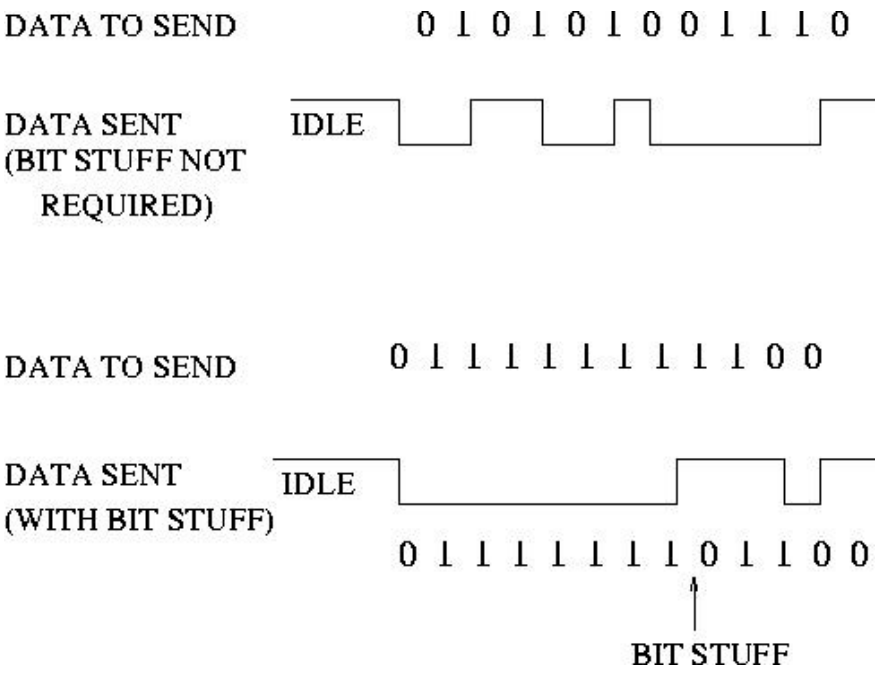

FIGURE 5: NRZI ENCODING, 0 - CHANGE IN LEVEL AND 1-NO CHANGE, BIT STUFFING ADDS A ZERO AFTER SIX CONSECUTIVE 1s [2]

All data on USB is encoded. The encoding format, called Non-Return to Zero Inverted (NRZI) with bit stuffing, ensures that receiver remains synchronized with the transmitter without the overhead of sending a separate clock signal or Start and Stop bits with each byte. Instead of defining logic 0 and 1 as voltages, the USB defines logic 0 as a voltage, and logic 1 as a voltage that remains the same. The bits transmit least significant bit (LSB) first. USB uses two techniques to remains synchronized: bit stuffing and Sync field [2].

7.1 Bit stuffing

If the data is all 0s, there are plenty of transitions, but if, the data contains a long string of 1s, the lack of transition could cause the receiver to get out of sync. To solve this, if data has six consecutive 1s, the transmitter stuffs, or inserts, a 0 (represented by a transition) after the sixth 1 (figure 5). This ensures at least one transition for every seven-bit width. The receiver detects and discards any bit that follows six consecutive 1s.

7.2 Sync field

Bit stuffing is not alone to ensure that the transmitting and receiving clocks in a transfer are synchronized. Sync field is there in starting of each packet, to enable the receiving device to align, or synchronize, its clock to the transmitted data. The sync field is eight bits: KJKJKJKK. The transition from idle to first K serves as a sort of Start bit that indicates the arrival of a new packet, but there is just one sync field per packet, rather than a Start bit for each byte [2]. The alternating Ks and Js provide transition for synchronizing, and the final two Ks mark the end of the field. By the end of the sync field, the receiving device knows precisely when, each bit in the packet arrives

8. USB device framework

8.1 Speed identification

A USB device must indicate its speed by pulling either the D- (figure 6) or D+ (figure 7) line high to 3.3 volts. A full speed device will use a pull up resistor attached to D+ to specify itself as a full speed device. The host or hub to detect, the presence of a device connected to its port, also uses these pull up resistors at the device end. Without a pull up resistor, USB assumes there is nothing connected to the bus. (Some devices have this resistor built into its silicon, which can be turned on and off under firmware control, others require an external resistor).

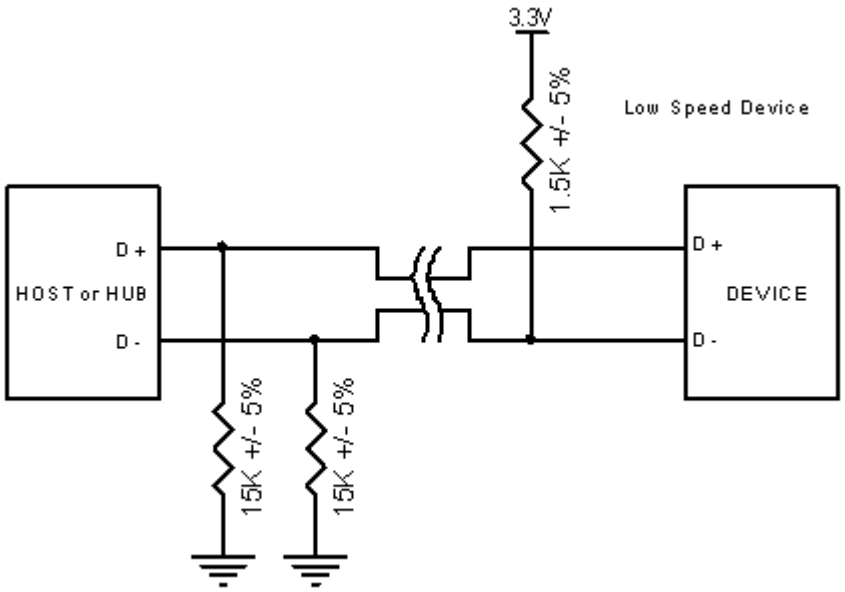

FIGURE 6: LOW SPEED DEVICE WITH PULL UP RESISTER CONNECTED TO D- [4]

High-speed devices will start by connecting as a full speed device (1.5k to 3.3V). Once device is attached, it does a high-speed chirp during reset and establish a high speed connection if the hub supports it. If the device operates in high-speed mode, then the pull up resistor is removed to balance the line.

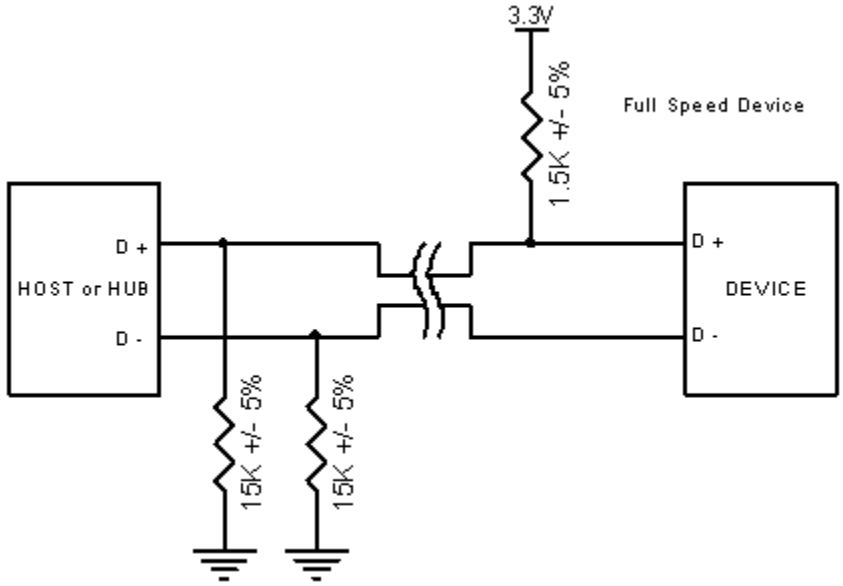

FIGURE 7: FULL SPEED DEVICE WITH PULL UP RESISTER CONNECTED TO D+ [4]

A USB 2.0 compliant device is not required to support high-speed mode. This allows cheaper devices to be produced if the speed is not critical. This is also the case for a low speed USB 1.1 device that is not required to support full speed. A high-speed device must not support low speed mode. It should only support full speed mode needed to connect first, then high-speed mode if successfully negotiated later. A USB 2.0 compliant downstream facing device (Hub or Host) must support all three modes, high speed, full speed and low speed.

8.2 Enumeration [2]

Enumeration is the process of determining what device has just been connected to the bus and what parameters it requires such as power consumption, number and type of endpoint(s), class of product etc. The host will then assign the device an address and enable a configuration allowing the device to transfer data on the bus. Windows enumeration involves the following steps: -

1) The user plugs a device into a USB port.
2) The hub detects the device.
3) The host learns of a new device.
4) The hub resets the device.
5) The hub establishes a signal path between the device and the bus.
6) The hub detects the speed of the device.
7) The host sends a get_descriptor request to learn the maximum packet size of the default pipe.
8) The host assigns an address.
9) The host learns about the device abilities.
10) The host assigns and loads a device driver.
11) The device driver of host selects a configuration.

The enumeration process is common to Windows 2000, Windows XP and Windows 98 SE. When something is wrong with a descriptor or how it is being sent, the host will attempt to read it three times with long pauses in between requests. After the third attempt, the host gives up reporting and with device.

8.3 Power (V_{BUS})

One of the benefits of USB is bus-powered devices that obtain its power from the bus and requires no external plug packs or additional cables. A USB device specifies its power consumption expressed in

2m*A* units in the configuration descriptor. A device cannot increase its power consumption, greater than what it specifies during enumeration, even if it looses external power. There are three classes of USB functions: -

1) Low-power bus powered functions
2) High-power bus powered functions
3) Self-powered functions

No USB device, whether bus powered or self-powered can drive the V_{BUS} on its upstream facing port. If V_{BUS} is lost, the device has a lengthy 10 seconds to remove power from the D+/D- pull-up resistors used for speed identification.

8.4 Suspend current

Suspend mode is mandatory on all devices. During suspend, additional constraints come into force. The maximum suspend current is proportional to the unit load. For a 1-unit load device (default) the maximum suspend current is 500μ*A*. This includes current from the pull up resistors on the bus. At the hub, both D- and D+ have pull down resistors of 15k ohms [4].

8.5 Entering suspend mode

A USB device will enter suspend when there is no activity on the bus for greater than 3 ms. It then has a further 7 ms to shutdown the device and draw no more than the designated suspend current and thus must be only drawing the rated suspend current from the bus 10 ms after bus activity stopped. In order to maintain connected to a suspended hub or host, the device must still provide power to its pull up speed selection resistors during suspend. USB has a start of frame packet or keep alive sent periodically on the bus. This prevents an idle bus from entering suspend mode in the absence of data.

A high-speed bus will have micro-frames sent every 125.0 μs ± 62.5 ns.

A full speed bus will have a frame sent down each 1.000 ms ± 500 ns.

A low speed bus will have a keep alive, which is an EOP (End of Packet) every 1 ms only in the absence of any low speed data.

The device will resume operation when it receives any non-idle signaling. If a device has remote wakeup enabled then it may signal to the host to resume from suspend.

9. USB controller chips

In selecting a controller Chip architecture, there are various types of possible configuration. Some controllers have a general purpose CPU on chip, while others take a minimalist approach and interface to an external CPU that handles the non-USB tasks and communicates with the USB controller as needed. All USB controllers have one or more USB ports as well as buffers, registers, and other I/O.A controller chip with a general purpose CPU also has program and data memory on-chip or an interface to these in external memory.

9.1 Chips that interface to a generic microcontroller

These chips enable to add a USB port to any microcontroller. Circuits made from these require two chips and communication can be controlled by an external microcontroller. The chips have external data buses that may use a serial or parallel interface to connect to the CPU, an interrupt pin can signal the CPU when the controller has received data or needs new data to send. The external chip may be slower than maximum speed of USB, making chip suitable only for transferring intermittent data. It can also support DMA mode for faster transfer, e.g. Netchip Semiconductor NET2888 [6].

9.2 Standalone USB controller chips

These chips do not require external controller instead they contain a CPU core and can be programmed like a micro controller (figure 8), they are of two types: -

1) Chips based on popular micro controller families: There are several controllers compatible with popular 8051 family and its variants. The advantage is that large number of development tools and literature is available [6], also developers are familiar with architecture and instruction set, e.g. Intel/Cypress 8x931.
2) Chips designed for USB with different instruction set CPU core: It has a different instruction set because the core is different Cypress Semiconductors has several chips of this type e.g. CY7C63001. The compiler is also available from the chip vendor [2].

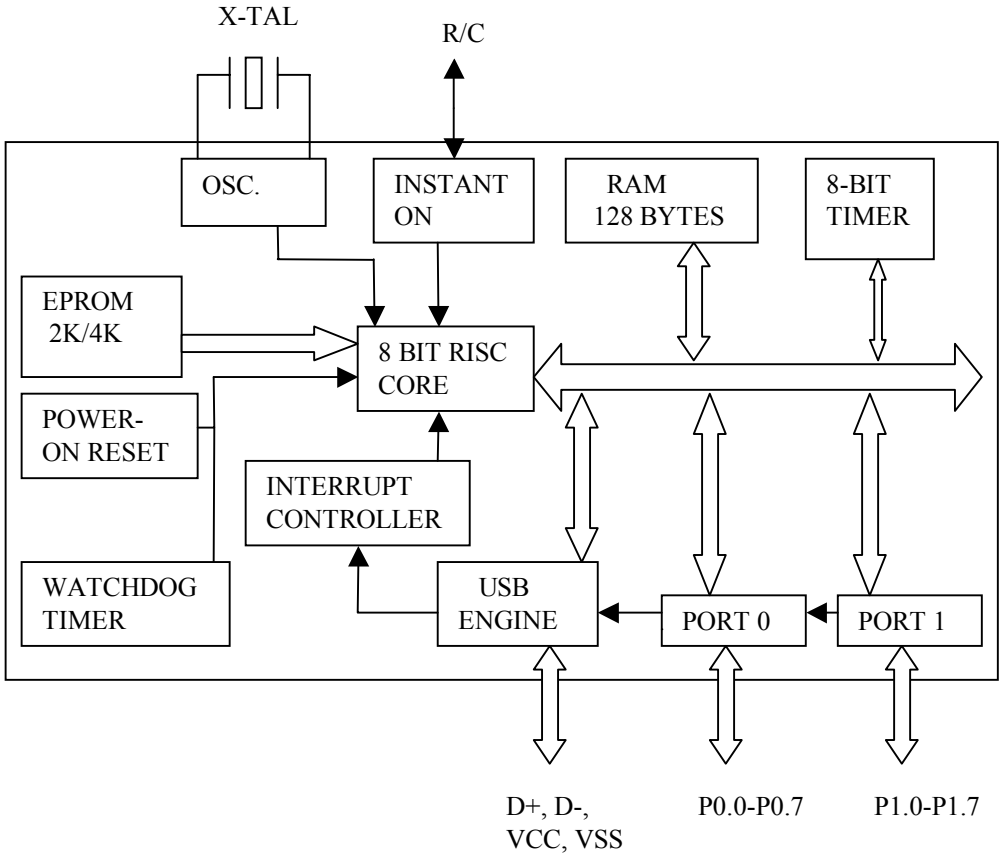

FIGURE 8: ARCHITECTURE OF A USB CONTROLLER CHIP (CYPRESS CY7C63xxx SERIES) [5]

Controller chips vary in the amount of support they provide for USB communications, some require accessing a series of registers to store and retrieve USB data, others may require managing retrieval of descriptors, setting data-toggle values, and ensuring that the appropriate handshake packets are sent. The criteria for selection of chip are: -

1) Rate of data transfer
2) No. And types of endpoints needed
3) Device to be software upgradeable
4) Cable type (flexible, long, etc.)
5) Other hardware features and abilities (timers, program, data memory, I/O)

10. Development and debugging

The development process on device end after hardwiring is writing firmware.

10.1 Embedded code development

This can be done by first choosing language either Assembly or High Level (C-language). Thus, the program can be written in some text editor and Assembler /Compiler is used to create object file through which controller chip can be programmed using a programmer. It could then be tested on development board or emulator.

10.2 Debugging

The debugging is an essential part of development process with USB, being a differential bus and highly complex protocol, if we want to monitor that is going on between device and the host then, probing is not possible by Digital storage oscilloscope alone, like with less complex traditional serial buses, the tools used are: -

USB Protocol Analyzer

Protocol Analyzer is needed to analyze activities on USB and is a standard tool for monitoring activities on bus, they are of two types: -

1) Standalone USB Protocol Analyzers (Don't require connection to logic Analyzer etc)

2) USB Protocol Analyzers/probes connect to existing Logic Analyzer systems.

USBCheck

It is a suite of five applications for USB devices. The application enables developer to view descriptors, send control requests, view the results, and run further tests on hubs, communication class devices and Human interface devices. This is freely available on Implementers Forum website [3].

11. Circuit

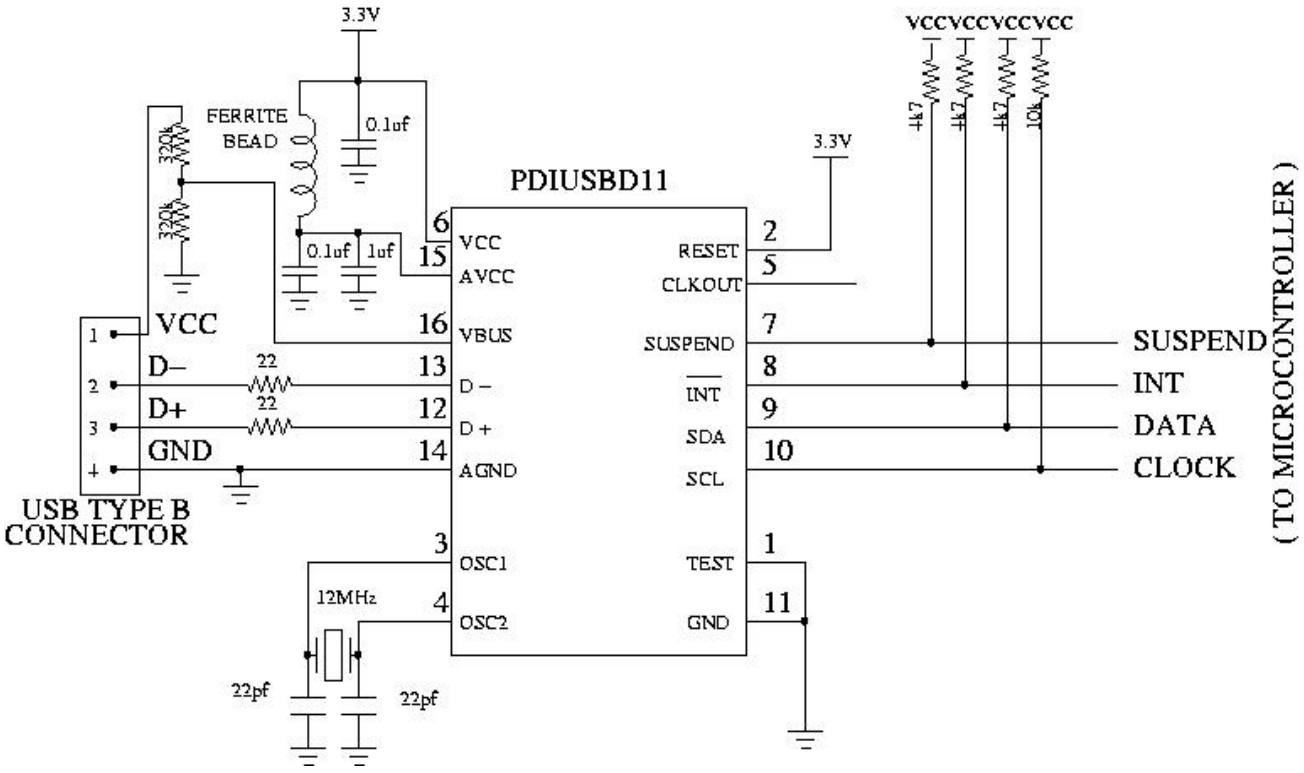

FIGURE 9: USB CHIP PDIUSBD11 WITH INTERFACE TO MICROCONTROLLER CHIP [4]

The circuit of figure 9 uses a USB controller, which can be interfaced to any of the popular family of micro controllers, the various registers are programmed by micro controller through the four-wire I^2C bus. The interrupt pin of this chip is hardwired to one of the micro controller interrupt pins, and also has a suspend pin which is must for a USB chip [4].

On The Go (OTG) Supplement-Point-to-Point Connectivity for USB

It details the "dual role device" which can function both as a device controller (DC) and/or a host controller (HC), aimed at embedded application. Philips have released ISP1161 single chip, integrated host and device controller, it makes an excellent host controller for embedded systems as Linux.

12. RS 232 to USB converter

To make an existing product USB compliant there can be one of the following approaches

1) USB controller chip on device side with the firmware and making use of any standard Device driver on host side so that the device may fit into category of known peripherals list, e.g. Mouse.

2) USB chip with firmware for the case where the device does not have a known driver, for that case, a customized Driver needed to be written using the Driver development program, e.g. Windows Device Developer Kit (DDK) [2].
3) Use of a hardwired chip, which does not require any programming, while they can be used for USB to RS-232 adaptors they are a quick way to make an existing product USB compliant. e.g. Future Technologies Device International (FTDI) chip FT8U232AM

High speed USB Controllers for serial and FIFO applications by FTDI [7]
It is Ultra band Transfer IC with RS-232/RS422 and CPU I/F Options. It has free Virtual Com port Drivers for Windows 98/ME/2000/XP, Linux and MAC.USB to RS-232 Converter, Quad flat pack FT8U232AM in a MQFP 7mm x 7mm to which a USB connector can be soldered. On the PC side VCOM (Virtual Comm. Port) driver supplied royalty free by FTDI, can be run, which emulates a COM port so, the existing application needs not be changed. This is required, if a third party makes the software and there is no control over the interface, or if compatibility is to be maintained with older and existing software.

13. USB host hardware and software

The host is connected to a USB device directly or through a Hub and plugging in or out of device is sensed as a voltage change, subsequently, there is initial exchange of data to know the device details, after which device is categorized by the Host and a unique ID is assigned to it. In the starting High-speed devices also communicates as low speed devices. On host side, the operating system tries to match finer details with information available on its respective system file (*.INF file for Windows operating system) after which it loads the respective driver. After enumeration of device and identification of device driver, the application that access the device must obtain a handle that identifies the device and enables communication with it. Handle is a unique identifier that Windows operating system assigns to the device. An application gets the handle by calling the create API function with a symbolic link that identifies the device [2].

1) User's role: When a device is attached and ready to transfer data, the host initiates a transfer by requesting it; the user might click a button in a chosen application to cause application to request data from the device.
2) Application's role: After the user requests a transfer, the application begins communication with the device. The Windows operating system API offers two ways of accessing, using read file / writes file pair or deviceIOcontrol, a driver may use one or both.
3) Device Driver's Role: When an application makes an API call, windows pass call to the appropriate device driver. The device driver converts the request to a format the USB bus class driver can understand.
4) Hub Driver's Role: Host's hub driver resides between a device specific or USB class driver and the USB bus class driver the hub Driver handles the initializing of the root hub's ports and devices downstream of the port. This driver requires no programming by the device developers.
5) Bus class Driver's Role: Bus class driver translates communications requests between the hub Driver and host controller driver. It handles the bus enumeration, power management, and some aspects of USB transactions. These communications requires no programming by the device developers.
6) Host controller Driver's Role: The host controller Driver communicates with the host controller hardware, which in turn connects to the bus. The host controller Driver requires no programming by the device developers.
7) Device's Role: From the host's port, data may pass through additional hubs; eventually the data reaches the hub that connects to the device. The device recognizes its address, reading the incoming data and takes the appropriate action.

Many communications require response, which may contain data sent in response to the request or just a packet with a handshake code. This information travels back to the host in the reverse order: through the device hub, onto the bus, and to the PC's hardware and software. A device driver may pass a response on to an application, which may display the result or take other action. For ending

communication, when an application closes or decides that it no longer needs to access the device, it uses the API function Close Handle to free system resources.

14. Conclusion

The number of devices, which are coming for USB, is large and increasing. Currently, Universal Serial Bus has with the full speed mode has paralleled apple fire wire (IEEE 1394) bus and is applicable for wide variety of devices. The three modes allow USB to cover a broad category of peripherals "high to low speed". USB plays a key role in fast growing areas like digital imaging, PC telephony, and multimedia games. The presence of USB as a standard offers a high degree of reliability in these exciting new areas. USB opens the door to new levels of innovation and ease of use for input devices, such as new generation of "force-feedback" digital joysticks. USB data rate accommodates MPEG-2 video–base products, data gloves and digitizers. Computer telephony integration is expected to be a big growth area for PCs. There are also opportunities for all types of peripheral from printers to scanners to high-speed communications such as Ethernet, Integrated Services Digital Network (ISDN) and satellite communications.

References

[1] Raja Sekhar B., "Universal Serial Bus", M. Tech. seminar, ES Group, IIT Bombay, http://www.ee.iitb.ac.in/~esgroup/es_mtech02_sem/es02_sem_rep_rajasekhar.pdf, site of EE Dept, IIT Bombay, last accessed on Nov 9, 2003.

[2] Axelson, J. "USB Complete" Penram International Publishing (India), Mumbai 1999.

[3] USB Implementers Forum, "USB 1.1 Spec", http://www.usb.org/developers/vendor.html, site sponsored by USB Implementers Forum, Inc. creators of USB technology last accessed on Nov 5, 2003.

[4] Craig, P. "Universal Serial Bus", http://www.beyondlogic.org, owner Craig peacock, last accessed on Nov 5, 2003.

[5] Cypress Semiconductors, "Universal Serial Bus", http://www.cypress.com/index.cfm, site of Cypress Semiconductors Corporation, last accessed on Nov 5, 2003.

[6] Axelson, J. "USB", http://www.lvr.com/index.html, site of Lakeview Research last accessed on Oct 20, 2003.

[7] Gigatechnology, "USB Modules", http://www.gigatechnology.com/usbmodprod.html, site of Gigatechnology Pvt Ltd. last accessed on Oct 20, 2003.