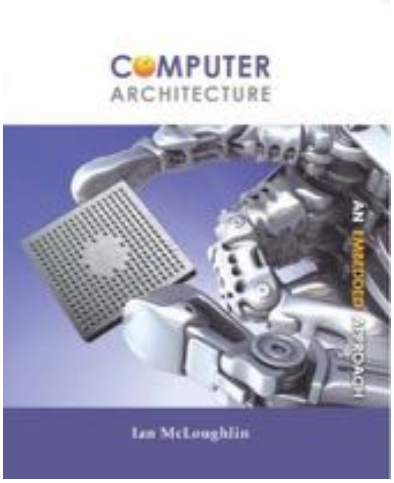
Computer Peripherals

School of Computer Engineering
Nanyang Technological University
Singapore

These notes are part of a 3rd year undergraduate course called "Computer Peripherals", taught at Nanyang Technological University School of Computer Engineering in Singapore, and developed by Associate Professor Kwoh Chee Keong. The course covered various topics relevant to modern computers (at that time), such as displays, buses, printers, keyboards, storage devices etc... The course is no longer running, but these notes have been provided courtesy of him although the material has been compiled from various sources and various people. I do not claim any copyright or ownership of this work; third parties downloading the material agree to not assert any copyright on the material. If you use this for any commercial purpose, I hope you would remember where you found it.

Further reading is suggested at the end of each chapter, however you are recommended to consider a much more modern alternative reference text as follows:



Computer Architecture: ***an embedded approach***
Ian McLoughlin
McGraw-Hill 2011

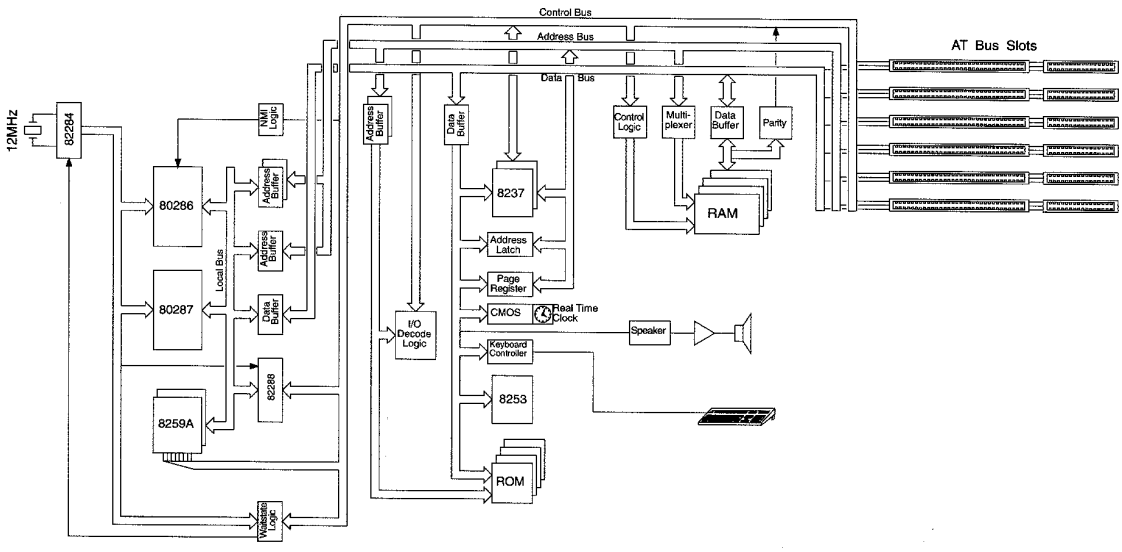
Chapter 1. Computer Buses

1.1. Microcomputer Bus Structure

What Is a Bus? One of the misunderstood features of computers today is the bus. Today one hears about the system bus, the local bus, the SCSI bus, the ISA bus, the PCI bus, the VL-bus, and now USB. These terms are also confused with other terms for slots, ports, connectors, etc. What is a bus, then, and how do these buses, differ?

1.1.1. Bus Definition

First, what is a bus? Basically, it is a means of getting data from one point to another, point A to point B, one device to another device, or one device to multiple devices. The bus includes not only the actual capability to transfer data between devices, but also all appropriate signaling information to ensure complete movement of the data from point A to point B. To avoid loss of data, a bus must include a means of controlling the flow of data between two devices, in order to insure that both devices are ready to send and/or receive information. Finally, both ends must understand the speed with which data is to be exchanged. A bus provides for all of these elements, and it includes a port definition to allow physical interfacing or connecting of two or more devices.



AT bus architecture

The bus has various component parts of a microcomputer connected by a number of different wires, which usually appear in the form of tracks on a Printed Circuit Board (PCB). These wires can be classified into four functional groups: power, address, data and control.

The physically largest PCB tracks are the power rails, since they have to carry the largest electrical currents. These days the voltage levels are usually +3V, +5V,+12V and ground. +5V are used for TTL and compatible whereas +12V supply lines were often required for some peripherals and interface to analog (linear) circuitry.

The address bus consists of a number of parallel lines which travel between CPU, memory and I/O chips. Information on the address bus enables the various memory and I/O devices to be accessed. Local address decoding circuitry is necessary for the memory (I/O device) to recognize the address as its own. The number of address lines varies with the particular microcomputer concerned, but is typically 16 for systems which use an 8-bit CPU, 24 for 16-bit CPUs, and 32 for 32-bit processors. It should be noted that the external address bus is not necessarily the same width as the address bus used internally within the CPU.

The data bus likewise consists of a number of parallel lines, and is either 8, 16, 32 or 64 bits wide. Both instructions and data travel along the data bus.

The control lines do not constitute a bus as such, but rather are a collection of individual lines or wires. Typical control lines are read, write, interrupt request and reset.

Each external device connects to the microcomputer system via its own dedicated peripheral support chip. These chips are controlled using on-chip programmable registers. At the very least, a peripheral chip will contain a control register and a status register. By writing various bit patterns to the control register, the chip can be made to perform its various functions. Similarly, by reading the bit patterns in the status register and responding accordingly, the CPU can effectively interact with the external world.

Because computer buses on a PCB consist of many parallel wires separated by an insulating fibreglass layer, they behave electrically as a long capacitor. Any circuit, such as a CPU which is attempting to drive such a bus, must therefore be capable of driving a capacitive load. TTL circuits are well suited to this task, but MOS circuits have limited drive capability.

Maximum electrical power (the product of voltage and current) is transmitted down the line when the terminating resistor (near or far end) is chosen to match the characteristic impedance of the miniature transmission line formed from the copper track and the insulating fibreglass of the PCB (this characteristic impedance, Z0, depends on the distributed resistance, capacitance and inductance along the PCB track, which is typically around 100 ohms). The bipolar (TTL) output transistor has a built-in resistor. If this collector (or load) resistor is removed, and made external to the gate then it becomes an Open Collector (OC) gate. ICs can incorporate such OC outputs, thus necessitating the connection of an external resistor in order to operate correctly.

Several such OC drivers are needed in order to send data from a CPU to a peripheral support chip located some distance away on the same PCB. Thus bus drivers will be required at the transmitter end, and bus receivers at the receiver end in such a system.

Bi-directional bus transceivers are required on the data bus, since data can travel not only from the CPU to the peripheral device, but also in the opposite direction. Transceivers are required at both the CPU and peripheral device ends, and can be likened to terminators at either end of a long transmission line (in this case the transmission line is formed from the interconnecting tracks on the PCB and the insulating fibreglass on which the tracks have been etched).

Address decoders compare the information present on the address bus at any time with the device's own unique identification (ID) address; when both addresses are identical, the output of the decoder is latched in order to form a Chip Select signal for the peripheral chip in question. In this way only 1-of-2n devices is selected at any particular time, assuming n address lines are used for decoding.

We have seen how several peripheral devices can connect to the same bus, and how, by placing the appropriate information on the address bus, only one such device will be selected at

any particular time. Obviously, we require some method of physically connecting several different devices to a single bus, such that only one device is electrically connected at any one time. There are two widely used methods of achieving this in practice, namely OC and Tri-State (TS).

OC gates were commonly used in minicomputer buses during the 1970s, but have since been superseded by TS bus drivers/transceivers, owing to the high power consumption of the former. TS gates consist internally of an additional output transistor which has the effect of enabling or disabling the gate, depending on the signal applied to a third control input. If a LO is applied to the control input, then the gate is disabled; applying logic levels at the inverter input will produce neither a LO nor a HI at the output. Under these conditions the output is said to be floating, or in its high impedance state; the device output is effectively disconnected. Applying a HI to the control input enables the inverter, such that the usual truth table applies, as indicated.

The individual PCB tracks which constitute a bus will not always be exactly the same physical length. This is not so critical at low speeds (1 or 2 MHz), but becomes increasingly more significant as the system clock is increased (the PCI bus are capable of operating at speeds in excess of 33 MHz and AGP is operated at 66 MHz).These minor differences in distance traveled across the surface of a PCB manifest as differences in the time it takes the different bits of data to travel from the CPU to the peripheral chip(s). A phase difference between adjacent signals is experienced at the distant receiver (or slave) end, when compared with what was transmitted by the bus master.

1.2. Speed of Data transfer

1.2.1. Serial vs. Parallel

One main aspect of a bus is whether the data is transferred in a serial or parallel fashion. In serial mode, the bits of each character are transmitted one at a time, one after another. For example, with each character containing 8 bits, the character is sent between devices, sending the first bit, then the second bit, third bit, and so on until the eighth bit is sent.

Contrast this with parallel transmission, where the bits of a character or data are transferred simultaneously. The parallel interface or transmission contrasts with the serial by allowing the devices to transmit all of a character's bits simultaneously instead of one at a time.

1.2.2. Speeds of Buses

The speed of serial bus is generaly expressed in bits per second (bps). For example, when a port, bus, or interface highlights 56-Kbps capability, the maximum throughput is 56,000 bits per second. To translate that into actual characters (bytes) per second, we need to make a calculation. For a rough estimate of maximum throughput, one can add the start and stop bits of a typical character, totaling 10 bits per character, and divide the interface speed by the number of bits per character. In the case of a 56-Kbps bus or interface, the maximum throughput would be approximately 5600 characters per second. This is only an approximation. Each bus has what is referred to as "overhead" to provide the other highlighted functions of flow control, addressing, etc. The simplicity of the bus will dictate the amount of overhead required. Some interfaces, such as those of devices that are locally connected via RS-232, have little overhead. Hence the actual throughput is relatively close to the maximum speed rating. The RS-232 interface is NOT generally referred to as a bus, but it does have the elements of a bus. (For a complete description of RS-232, refer to CE301.)

The keyword here is maximum, most buses are set up electrically to support a maximum (or burst) throughput rate, as well as a sustained throughput rate. When calculating throughput, use the sustained rate for a closer approximation of speed. The maximum rate is best case rate that a bus, or interface can handle.

1.2.3. Sustained vs. Burst Throughput

A burst rate is the maximum rate at which data can be sent or burst over a bus. A sustained rate is the rate at which data can be continuously sent over the bus. The sustained rate is the rate at which data can be sent over the bus in a consistent manner. It is a better metric than the burst rate for throughput expectations on a bus, as it reflects the typical transmission speed.

1.3. Bus Protocols

Protocol simply refers to the set of rules agreed upon by both the bus master and bus slave as to how data is to be transferred over the bus. Flow control is an important aspect of a protocol. Flow control is used to regulate the flow of information between the devices. When computers are communicating with other devices, flow control must be used to ensure that data is not lost. For example, suppose that two computers are connected in order to upload and download data. Whether the connection involves modems, or the computers are connected back to back, at some point one of the computers will need to store the received information to a disk file. What happens to the incoming data while the file is being written? This defines the requirement for flow control.

Flow control is the ability of a receiving device to regulate the flow of data from the sending device. Protocols break the data into blocks or frames of data, hence the term block size. Some protocols support multiple block sizes, requiring the two communicating devices to agree on the block size before transmission. Typical block sizes are 128, 256, 512, 1024 (1K), and 2048 (2K) bytes. The sending device will send the information a block at a time. In addition, the sending device will perform a calculation on the bits of the data in the block. The result of this calculation is some form of check character or frame-check sequence. The block-check character (BCC) is typically one or two bytes appended to the block of data. The block and appended character(s) are then sent to the other device. The receiving device then performs the same calculation on the data block that it receives and computes a BCC. A comparison is made between the received BCC and the calculated one. If the BCCs are the same, then the block of data was received error free. The receiving entity then notifies the sending device that all was received "OK." If the BCCs are different, then the data was received with an error. The protocols allow for asking for a retransmission of one of more blocks of data.

Factoring in the overhead, one can see where the block size will impact the sustained throughput on a bus. Small block sizes appear to be inherently less efficient, owing to their overhead for framing and flow control, but in reality they can be very efficient in terms of memory usage. Depending on the cache, buffering, and memory sizes and speeds, small block sizes can improve throughput. Conversely, large block sizes require inherently less protocol overhead and may be great for bursts of data. However, improper memory management may take away the gains of larger block sizes. Furthermore, if error conditions exist, then retransmission must occur on a bus. The larger the block size, the longer it takes to retransmit the block of data.

There are a number of different ways of transferring data between CPU and peripheral, and these are usually classified as synchronous or asynchronous, depending on whether or not the transfers bear a relationship to the system clock.

1.3.1. Synchronous Buses

With a synchronous protocol, data transfers occur in relation to successive edges of the system clock. Inherent in this type of protocol is the assumption that data will arrive within a certain time window (if it does not, then the data is lost).

1.3.2. Asynchronous Buses

Asynchronous bus transfers bear no particular timing relation to the system clock; transfers can take place at any time. Additional handshake lines are required in order to guarantee data transfers between master and slave. Synchronous bus transfers, by way of contrast, only depend on the system clock (the protocol being built into the system).

Asynchronous buses are useful when matching the different speeds of the CPU and peripheral chips. For example, a processor can interface to both slow and fast semiconductor memories using an asynchronous protocol. For a write operation, the protocol involves the bus master advising the slave that it has some data ready to send. The slave advises the master upon receipt of the data. The master then sends a message back to the slave acknowledging advice of the successful transfer. Finally, the slave responds to this acknowledgement from the master; a read operation follows a similar protocol. As with the synchronous bus, there will be delays due to address decoding, setup times and skew.

1.3.3. Semi-synchronous Buses

A compromise between the two previous bus protocols is found in the semi-synchronous bus, which approaches the speed of synchronous buses, but allows for interfacing to peripheral devices of varying speeds. The semi-synchronous bus operates essentially as an asynchronous bus until the peripheral device is ready for a transfer, after which the bus becomes synchronous for the duration of the transfer.

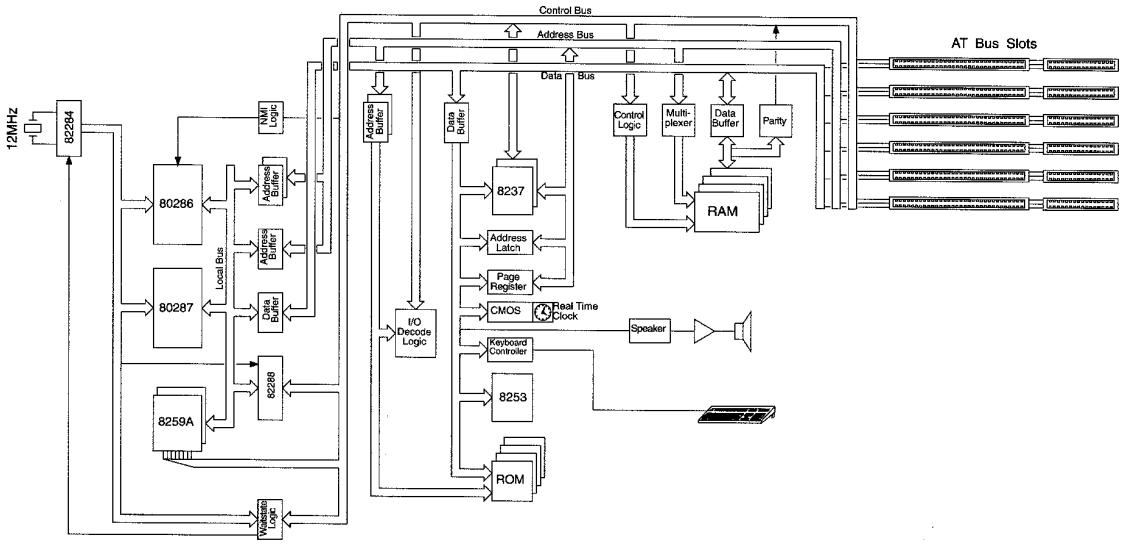
1.4. Buses

A microcomputer system performance can sometime very much dependent on the system bus and combination of buses, for example the PCI bus working in combination with the IDE or SCSI bus as harddisk interface.

We shall only devote our attention here some commonly used buses.

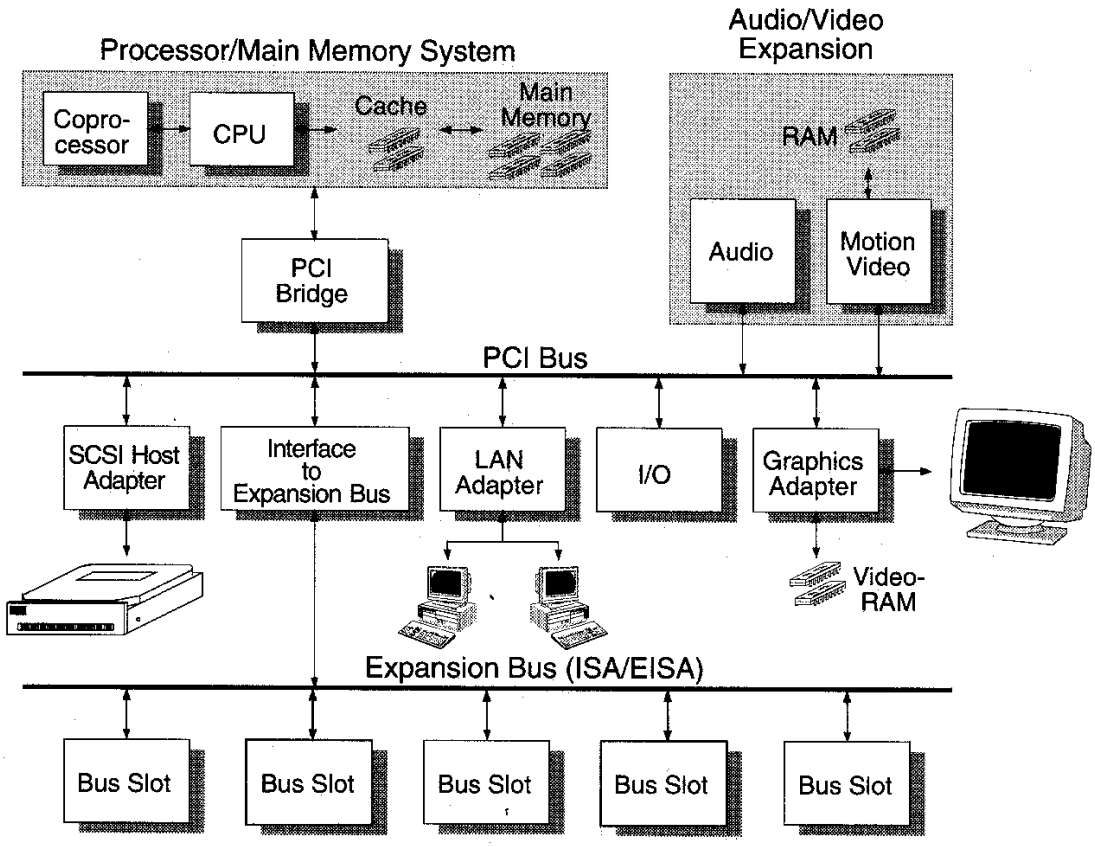
1.4.1. Local Buses

What is a local bus? It is a bus that is local relative to the CPU, hence the name. Computers have had buses in them forever, but recently there has been movement toward industry-standard buses, away from the proprietary buses that existed in computers prior to the advent of the PC. When IBM introduced PC, it defined the XT and AT buses and the move was begun toward open buses.



AT bus architecture

In the early 1990s PCs began to incorporate what is called a *local-bus I/O.* Actually, there were at least three different types of buses-i486 *local bus, VL-bus,* and PCI Bus. The VL-bus is an extension to the 486 bus and has been used mainly for video. The 486 has been replaced by the Pentium line of processor subsystems from Intel. Furthermore, the PCI Bus has taken over as the dominant high-performance local bus due to its openness, performance, and support within the industry.



PCI bus architecture

ISA/EISA/Microchannel Architecture

The original bus in the IBM PC/AT became the ISA bus (Industry Standard Architecture). IBM and Compaq also pushed other buses. IBM introduced a totally new bus in 1987-the Microchannel (MCA). Due to incompatibility with the then-prevalent ISA bus, Microchannel achieved limited acceptance and has given way to PCI. Largely driven by Compaq in 1988, PC system vendors developed an extension to the ISA bus called the *EISA bus,* for *Extended ISA*. EISA-based systems began to appear in mid-1989. Use of EISA tended to be limited to servers and highend desktops. Despite these efforts, ISA remained the dominant I/O bus in PCs. ISA and EISA I/O performance simply could not keep up with the performance demands of PC evolution.

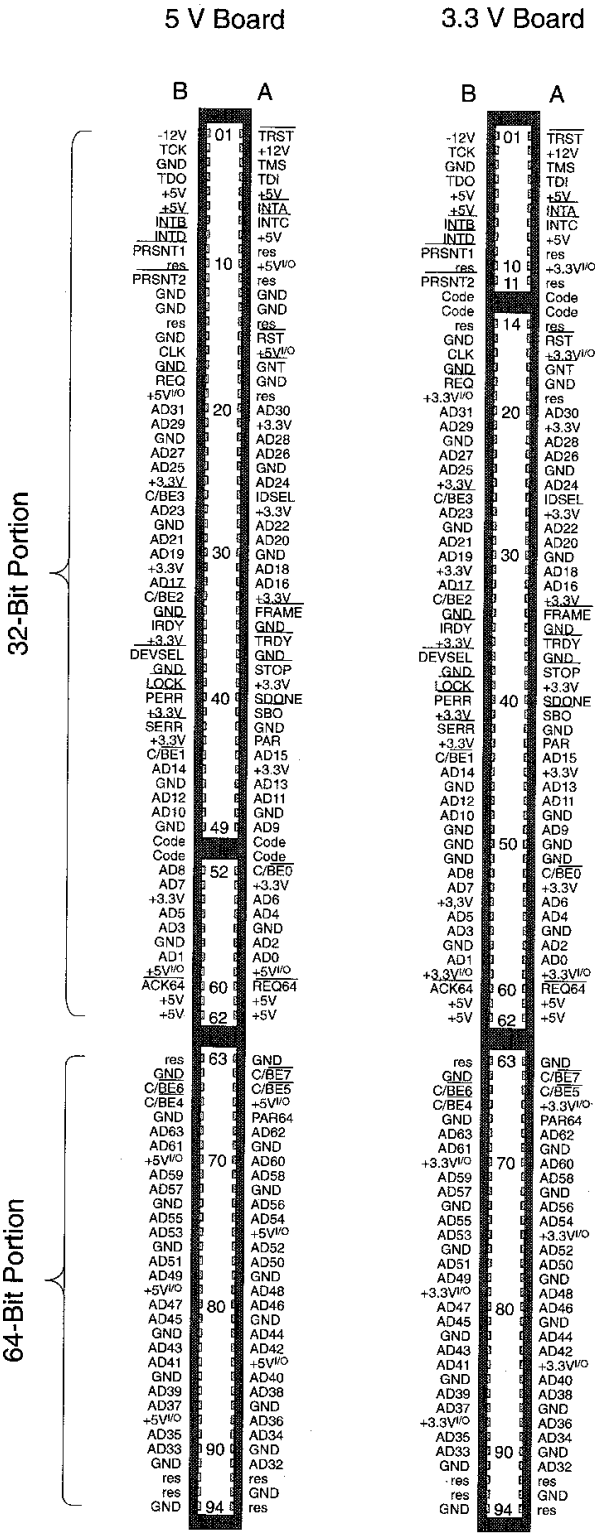
(Video Electronics Standards Association) suggested an extension to the i486 bus to address video performance. The extension consists of using a standard connector on the i486 processor bus. VGA chips were typically mounted on removable cards and attached directly on the 486 local bus. This setup became known as the VL-bus, for Vesa local bus. The connector for the VL-bus is placed directly in line with the normal ISA connector, allowing a single slot to be shared by either an ISA card or a VESA card.

1.4.2. PCI Bus (Peripheral Component Interconnect Bus)

The PCI Bus emerged as the answer to the performance bottleneck. The PCI bus is being used to address all of the problems faced by video, disk (SCSI and IDE), network, etc. However, it is a high-performance bus that is used for peripherals requiring CPU-like performance. PCI is not targeted toward solving the needs of relatively slower devices like a mouse, keyboard, speakers, etc. This is where USB fits.

The typical computer system consisted of a processor chip with an optional high-performance cache memory. This bus consisted of 32 address signals, 32 data signals, and associated control signals. It had a 33 MHz clock (the are also 66 MHz specification) and could transfer 32 bits (or 64 bits) of data every clock cycle. A bridge chipset allowed the connection of the system I/O bus, typically ISA, IDE or SCSI.

The PCI local bus with great performance capabilities met many demands. The PCI Bus is now a well-defined open standard. There are a massive number of PCI-based systems, PCI cards, and chipsets. Today you can find PCI video cards, networking cards, SCSI and IDE controller cards and chips, and others. Furthermore, PCI is processor independent and is used on a number of different CPU-based systems, such as Intel, DEC Alpha, etc. Most systems shipped today include a PCI bus with slots, or at least PCI-based peripherals.



A PCI slot and its pins

1.4.2.1.PCI Chipsets

Various computer chips are necessary to provide the buses. A **PCI chipset** is a set of chips that provides the PCI capability in a computer system or peripheral. Depending on the system architecture, multiple chipsets may be required. For example, if an ISA bus is going to be provided in the same system with a PCI Bus, there will need to be PCI chipsets to provide both buses. A bridge chip is used to provide connectivity (or a bridge) between two different types of buses. Similarly, PCI chipsets are used to connect the PCI slots and peripherals to the other components of the system, such as CPU, memory, etc. So "PCI chipset" is a generic reference to computer chips that enable PCI within a system. The complexity and number depend on the features planned for a computer system

1.4.3. SCSI Bus and IDE

1.4.3.1. SCSI

SCSI is used to connect peripherals to a computer. As far back as the late 70s, disk-drive manufacturers saw the need to improve the performance and standardization of disk drives and other storage peripherals. Shugart Associates began by designing a new transfer protocol, originally termed SASI (Shugart Associates Systems Interface). There was no ANSI standard for this in the early days, but NCR joined Shugart, and the ANSI committee X3T9.2 was put in place. The resulting protocol became an industry standard called *small computer systems interface* (SCSI). In 1985 the *common command set* (CCS), was added, prior to ANSI's finishing the SCSI standard in 1986. SCSI-II devices were released in 1988 and became a standard in 1994. Here is the evolution of the SCSI bus.

SCSI-I:	Original SCSI protocol. ANSI standard X3.1 31-1996 had a bus speed of 5 MHz and a parallel data path of 8 bits.
SCSI-II:	Added support for CD-ROMs, scanners, and tape drives.
Fast SCSI-II:	Doubled the bus speed to 10 MHz from the original 5 MHz.
Wide SCSI-II:	Doubled the data path to 16 bits instead of the original 8 bits.
Ultra SCSI-III:	Doubled the bus speed to 20 MHz from 10 MHz (4 times the original).

SCSI is used to connect peripherals and computers such as hard disks, tape devices, port expanders, CD-ROMs, CD-R units, scanners, and many other devices. SCSI is similar in functionality to IDA/ATA but can scale better, due to its focus on performance. The SCSI specification also allows for the bus to be extended via a SCSI cable.

1.4.3.2. IDE

Seagate Technologies developed the original ST506 bus to support their 5- and 10-MB disk drives, the ST506 and ST412, respectively. Prior to IDE, Maxtor developed the ESDI *(enhanced small device interface)* bus to replace the ST506. With each evolution, the controller intelligence for the disk drives was moved closer to the CPU system and eventually placed on the motherboard, rather than on cards or within the drives. IDE *(integrated drive electronics),* developed by Compaq and Western Digital, is a standard for connecting disk drives in a common way to PC motherboards. In today's IDE configurations, multiple devices can share the IDE controller. Initially two devices could be put on an IDE controller. Recently this number was doubled to four. A newer version of IDE goes under the name ATA *(AT bus attachment).*

1.4.4. FireWire and USB

1.4.4.1. FireWire

The IEEE-1394 standard, known as FireWire, is a serial SCSI-bus standard supporting transfer rates from 100 to 400 MBps, expanding eventually up to 1.6 Gbps (gigabits per second) or faster. We will come back to this a bit later.

1.4.4.2. Universal Serial Bus

The Universal Serial Bus (USB) standard is for connecting keyboards, monitors, input devices, and digital cameras over a 12-Mbps bus network. USB is designed to simplify the connection of peripheral devices, provide increased I/O capacity, and provide maximum flexibility for the continued evolution of the PC. As connectivity demands increase, USB is to provide for this expansion.

USB should eventually replace the PC's keyboard, serial, and parallel connections with a simple jack architecture, with autodetection capabilities to know when a device is attached or unattached, complete with configuration support. Another goal of USB is to improve the connectivity of new peripherals by placing jacks in convenient locations. With today's computers, the main connection location is on the back of the computer, which is generally not conveniently accessible. With USB, the potential exists to put ports on the front of the computer, on the monitor, keyboard, etc., making it extremely convenient for the typical user to connect more peripherals. It also offers great flexibility and scalability.

Table: Bus Architectures Comparisons

Bus Type	Industry Reference	Bus Width (Bits)	Bus Speed (MHz)	Transfer Rate (MBS)
ISA 8-BIT	Industry Standard Architecture	8	8	4
ISA 16-BIT	Industry Standard Architecture	16	8	8
MCA (32 bit)	Microchannel Architecture	32	8	33
EISA	Extended Industry Standard Architecture	32	8.33	33.3
VL-BUS	VESA LocalBus	33	33 40 50	128-132
PCI (32 bit)	Peripheral Component Interconnect	32	33	132
PCI (64 bit)	Peripheral Component Interconnect	64	33	264
SCSI	Small Computer System Interface	(see above)		
1394	FireWire		100-400	100-400 Mbps
USB	Universal Serial Bus		12	12 Mbps

1.4.4.3. Fire Wire vs. USB

{PRIVATE}Many people confuse 1394 and Universal Serial Bus (USB). It's understandable. Both are emerging technologies that offer a new method of connecting multiple peripherals to a computer. Both permit peripherals to be added to or disconnected from a computer without the need to reboot. Both use thin, flexible cables which employ simple, durable connectors.

Although 1394 and USB cables may look nearly the same, the amount of data flowing through them is quite different. As the chart below shows, the widely different data transfer rate capability of 1394 and USB marks the principal distinction between these two technologies:

{PRIVATE}	**1394 FireWire**	**USB**
Maximum Number Of Devices	62	127
Hot-Swap (Add Or Remove Devices Without Rebooting Computer)	Yes	Yes
Maximum Cable Length Between Devices	4.5m	5m
Data Transfer Rate	**200mbps (25MB/sec)**	**12mbps (1.5MB/sec)**
Bandwidth Roadmap	400mbps (50MB/sec) 800mbps (100MB/sec) 1Gbps+ (125MB/sec+)	None
Macintosh Implementation	Yes	No
Internal Peripheral Connection	Yes	No
Peripheral Devices	- DV Camcorders - High-Resolution Digital Cameras - HDTV - Set-Top Boxes - Hard Disks - DVD-ROM Drives - Printers - Scanners	- Keyboards - Mice - Monitors - Joysticks - Low-Resolution Digital Cameras - Low-Speed CD-ROM Drives - Modems

Firewire offers a data transfer rate that is over 16 times faster than USB. And the speed gap will widen even further in months and years to come. That's because USB was designed with no provision for future improvements in its data transfer capabilities. In contrast, Firewire has a well-defined bandwidth roadmap, with speed increases to 400mbps (50MB/sec) and possibly 800mbps (100MB/sec) expected in 1998, and 1Gbps+ (125MB/sec) and beyond in succeeding years. Such dramatic improvements in data transfer capacity will be required to keep pace with bandwidth hogging devices, such as HDTV, digital set-top boxes and home automation systems, that plan to incorporate 1394 interfaces.

Most industry analysts expect Fire wire and USB to coexist peacefully in computers of the future. Small Firewire and USB connectors will replace the gaggle of connectors found on the back of today's PCs. USB will be reserved for low-bandwidth peripherals (mice, keyboards, modems), while 1394 will be used to connect to the new generation of high-bandwidth computer and consumer electronics products.

1.5. USB

1.5.1. Architectural Overview

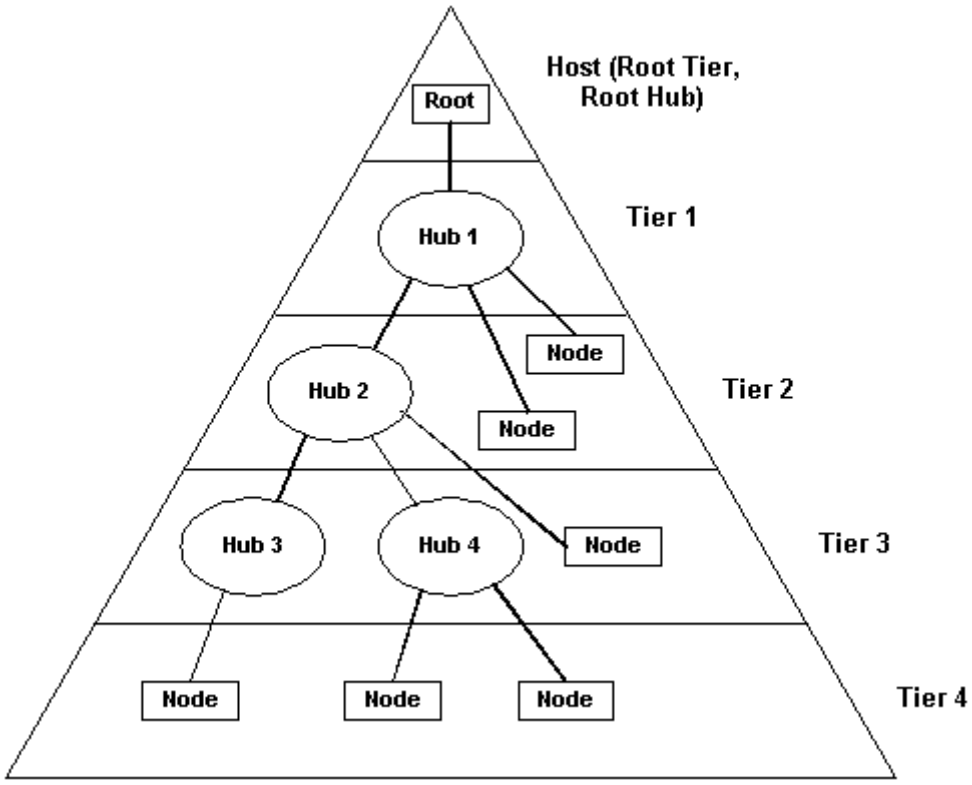
USB is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host scheduled token based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation. This is referred to as dynamic (or hot) attachment and removal.

The USB interconnect is the manner in which USB devices are connected to and communicate with the host. This includes:

- Bus Topology: Connection model between USB devices and the host.
- Inter-layer Relationships: In terms of a capability stack, the USB tasks that are performed at each layer in the system.
- Data Flow Models: The manner in which data moves in the system over the USB between producers and consumers.
- Scheduling the USB: USB provides a shared interconnect. Access to the interconnect is scheduled in order to support isochronous data transfers.

1.5.1.1. Bus Topology

The Universal Serial Bus connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function.



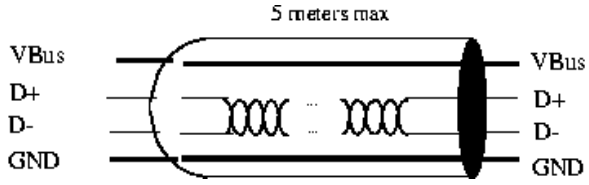
USB topology

1.5.1.2. Electrical specification

USB transfers signal and power over a four wire cable. The signaling occurs over two wires and point-to-point segments. The signals on each segment are differentially driven into a cable of 90 Ω intrinsic impedance. The differential receiver features input sensitivity of at least 200 mV and sufficient common mode rejection.

There are two modes of signaling. The USB full speed signaling bit rate is 12 Mbs. A limited capability low speed signaling mode is also defined at 1.5 Mbs. The low speed method relies on less EMI protection. Both modes can be simultaneously supported in the same USB system by mode switching between transfers in a device transparent manner. The low speed mode is defined to support a limited number of low bandwidth devices such as mice, since more general use would degrade the bus utilization.

The clock is transmitted encoded along with the differential data. The clock encoding scheme is NRZI (which will be discussed later in chapter 7) with bit stuffing to ensure adequate transitions. A SYNC field precedes each packet to allow the receiver(s) to synchronize their bit recovery clocks.



USB cable.

The cable also carries VBus and GND wires on each segment to deliver power to devices. VBus is nominally +5 V at the source. USB allows cable segments of variable lengths up to several meters by choosing the appropriate conductor gauge to match the specified IR drop and other attributes such as device power budget and cable flexibility. In order to provide guaranteed input voltage levels and proper termination impedance, biased terminations are used at each end of the cable. The terminations also permit the detection of attach and detach at each port and differentiate between full speed and low speed devices.

1.5.1.3. Bus Protocol

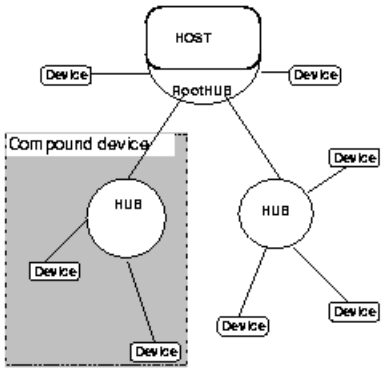
All bus transactions involve the transmission of up to **three packets**. Each transaction begins when the host controller, on a scheduled basis, sends a USB packet describing the type and direction of transaction, the USB device address, and endpoint number. This packet is referred to as the Token Packet. The USB device that is addressed selects itself by decoding the appropriate address fields. In a given transaction, data is transferred either from the host to a device or from a device to the host. The direction of data transfer is specified in the token packet. The source of the transaction then sends a Data Packet or indicates it has no data to transfer. The destination in general responds with a Handshake Packet indicating whether the transfer was successful.

The USB data transfer model between a source or destination on the host and an endpoint on a device is referred to as a **pipe**. There are two types of pipes: stream and message. Stream data has no USB defined structure while message data does. Additionally, pipes have associations of data bandwidth, transfer service type, and endpoint characteristics like directionality and buffer sizes. Pipes come into existence when a USB device is configured. One message pipe, Control Pipe 0, always exists once a device is powered in order to provide access to the device's configuration, status, and control information.

The transaction schedule allows flow control for some stream mode pipes. At the hardware level, this prevents buffers from underrun or overrun situations by using a NACK handshake to throttle the data rate. The token for a NACK'ed transaction is reissued when bus time is available. The flow control mechanism permits the construction of flexible schedules that accommodate concurrent servicing of a heterogeneous mix of stream mode pipes. Thus, multiple stream mode pipes can be serviced at different intervals and with packets of different sizes.

1.5.2. Physical Bus Topology

Devices on the USB are physically connected to the host via a tiered star topology. USB attachment points are provided by a special class of USB device known as a hub. The additional attachment points provided by a hub are called ports. A host includes an embedded hub called the root hub. The host provides one or more attachment points via the root hub. USB devices which provide additional functionality to the host are known as functions. To prevent circular attachments, a tiered ordering is imposed on the star topology of the USB. This results in the tree-like configuration illustrated in the figure below.

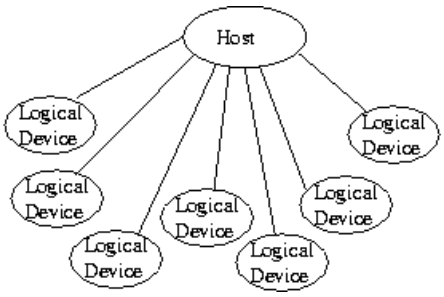


USB Physical Bus

Multiple functions may be packaged together in what appears to be a single physical device. For example, a keyboard and a trackball might be combined in a single package. Inside the package, the individual functions are permanently attached to a hub and it is the internal hub that is connected to the USB. When multiple functions are combined with a hub in a single package, they are referred to as a compound device. From the host's perspective, a compound device is the same as a separate hub with multiple functions attached.

1.5.3. Logical Bus Topology

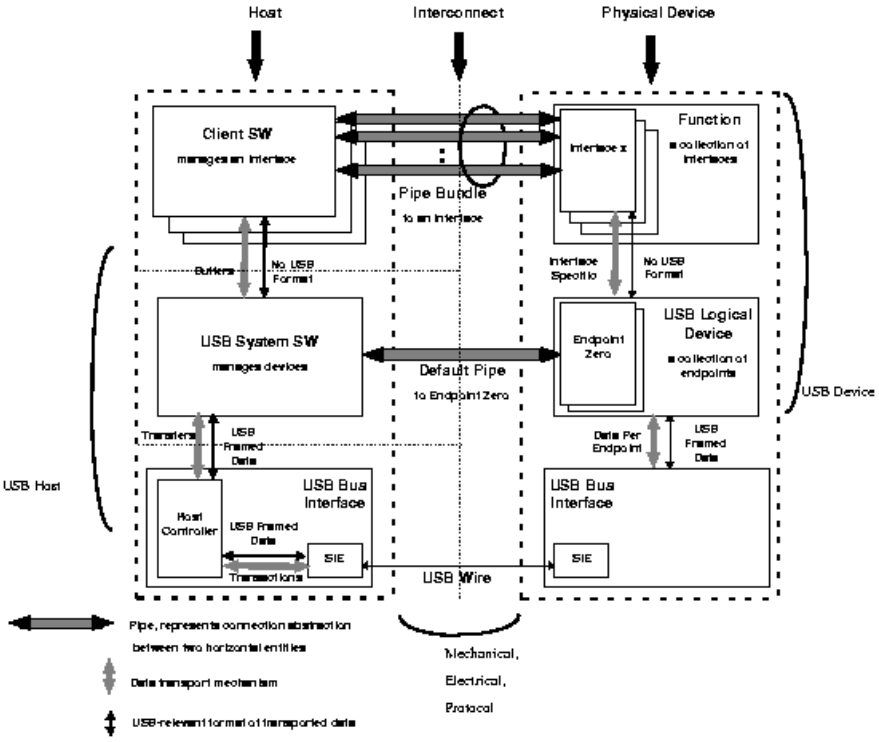
While devices physically attach to the USB in a tiered, star topology, the host communicates with each logical device as if it were directly connected to the root port. This creates the logical view illustrated in the following figure. Hubs are logical devices also, but are not shown to simplify the picture. Even though most host/logical device activities use this logical perspective, the host maintains an awareness of physical topology to support processing the removal of hubs. When a hub is removed, all of the devices attached to the hub must be removed from the host's view of the logical topology.



USB logical bus topology

1.5.4. USB Communication Flow

USB provides a communication service between software on the host and its USB function. Functions can have different communication flow requirements for different client to function interactions. USB provides better overall bus utilization by allowing the separation of the different communication flows to a USB function. Each communication flow makes use of some bus access to accomplish communication between client and function. Each communication flow is terminated at an endpoint on a device. Device endpoints are used to identify aspects of each communication flow.



USB Host/Device View

A USB logical device appears to the USB system as a collection of endpoints. Endpoints are grouped into endpoint sets which implement an Interface. Interfaces are views to the function. System software manages the device using the Default Pipe (associated with Endpoint 0). Client software manages an Interface using pipe bundles (associated with an Endpoint Set).

Client software requests that data be moved across the USB between a buffer on the host and an endpoint on the USB device. The host controller (or USB device depending on transfer direction) packetizes the data to move it over the USB. The host controller also coordinates when bus access is used to move the packet of data over the USB.

1.5.5. USB protocol layer

This section presents a bottom-up view of the protocol starting with field and packet definitions. This is followed by a description of packet transaction formats for different transaction types. Link layer flow control and transaction level fault recovery is then covered. The section finishes with a discussion of retry synchronization, babble, and loss of bus activity recovery.

1.5.5.1 Bit Ordering

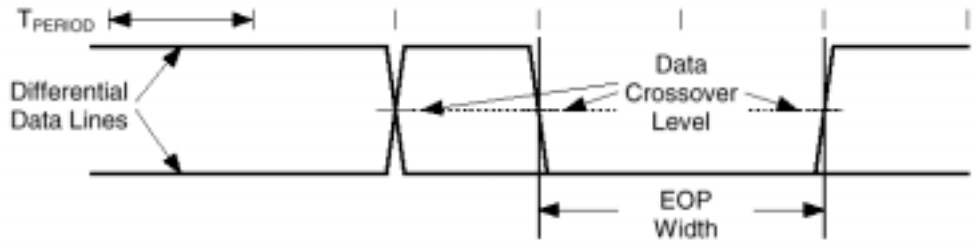
Bits are sent out onto the bus LSB first, followed by next LSB, through to MSB last. In the following diagrams, packets are displayed such that both individual bits and fields are represented (in a left to right reading order) as they would move across the bus.

1.5.5.2 SYNC Field

All packets begin with synchronization (SYNC) field, which is a coded sequence that generates a maximum edge transition density. The SYNC field appears on the bus as IDLE followed by the binary string `KJKJKJKK', in its NRZI encoding. It is used by the input circuitry to align incoming data with the local clock and is defined to be eight bits in length. SYNC serves only as a synchronization mechanism and is not shown in the packet diagrams. The last two bits in the SYNC field are a marker that is used to identify the first bit of the PID. All subsequent bits in the packet must be indexed from this point.

1.5.5.3 EOP Width

The width EOP is about 2 * T_{PERIOD}. The EOP width is measured with the same capacitive load used for maximum rise and fall times and is measured at the same level as the differential signal crossover points of the data lines.



EOP width timing

For full speed transmissions, the EOP width from the transmitter must be between 160 ns and 175 ns. For low speed transmissions, the transmitter's EOP width must be between 1.25 µs and 1.50 µs. These ranges include timing variations due to differential buffer delay and rise/fall time mismatches and to noise and other random effects.

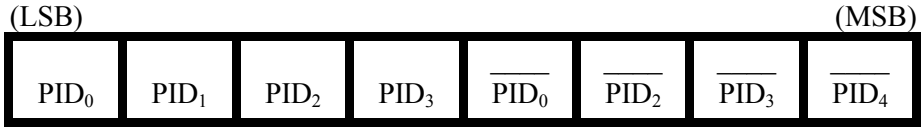
1.5.5.4 Packet Field Formats

Field formats for the token, data, and handshake packets are described in the following section. Packet bit definitions are displayed in unencoded data format. The effects of NRZI coding and bit stuffing have been removed for the sake of clarity. All packets have distinct start and end of

packet delimiters. The start of packet (SOP) is part of the SYNC field, and the end of packet (EOP).

1.5.5.5 Packet Identifier Field

A packet identifier (PID) immediately follows the SYNC field of every USB packet. A PID consists of a four bit packet type field followed by a four-bit check field as shown. The PID indicates the type of packet and, by inference, the format of the packet and the type of error detection applied to the packet. The four-bit check field of the PID insures reliable decoding of the PID so that the remainder of the packet is interpreted correctly. The PID check field is generated by performing a ones complement of the packet type field.



PID Format

The host and all functions must perform a complete decoding of all received PID fields. Any PID received with a failed check field or which decodes to a non-defined value is assumed to be corrupted and it, as well as the remainder of the packet, is ignored by the packet receiver. If a function receives an otherwise valid PID for a transaction type or direction that it does not support, the function must not respond. For example, an IN only endpoint must ignore an OUT token. PID types, codings, and descriptions are listed below

PID types, codings, and descriptions

PID Type	PID Name	PID[3:0]	Description
Token	OUT	b0001	Address + endpoint number in host -> function transaction
	IN	b1001	Address + endpoint number in function -> host transaction
	SOF	b0101	Start of frame marker and frame number
	SETUP	b1101	Address + endpoint number in host -> function transaction for setup to a control endpoint
Data	DATA0	b0011	Data packet PID even
	DATA1	b1011	Data packet PID odd
Handshake	ACK	b0010	Receiver accepts error free data packet
	NAK	b1010	Rx device cannot accept data or Tx device cannot send data
	STALL	b1110	Endpoint is stalled
Special	PRE	b1100	Host-issued preamble. Enables downstream bus traffic to LS devices.

PIDs are divided into four coding groups: token, data, handshake, or special, with the first two transmitted PID bits (PID<1:0>) indicating which group. This accounts for the distribution of PID codes.

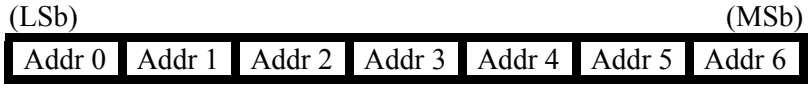
1.5.5.6 Address Fields

Function endpoints are addressed using two fields: the function address field and the endpoint field. A function needs to fully decode both Address and Endpoint fields. Address or endpoint aliasing is not permitted, and a mismatch on either field must cause the token to be ignored. Accesses to non-initialized endpoints will also cause the token to be ignored.

1.5.5.6.1 Address Field

The function address (ADDR) field specifies the function, via its address, that is either the source or destination of a data packet, depending on the value of the token PID. A total of 128 addresses are specified as ADDR<6:0>. The ADDR field is specified for IN, SETUP, and OUT tokens.

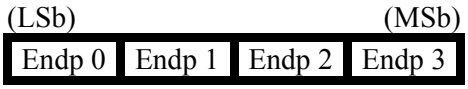
By definition, each ADDR value defines a single function. Upon reset and power-up, a function's address defaults to a value of 0 and must be programmed by the host during the enumeration process. The 0 default address is reserved for default and cannot be assigned for normal operation.



ADDR Field

1.5.5.6.2 Endpoint Field

An additional four-bit endpoint (ENDP) field permits more flexible addressing of functions in which more than one sub-channel is required. Endpoint numbers are function specific. The endpoint field is defined for IN, SETUP, and OUT token PIDs only. All functions must support one control endpoint at 0. Low speed devices support a maximum of two endpoint addresses per function: 0 plus one additional endpoint. Full speed functions may support up to the maximum of 16 endpoints.



Endpoint Field

1.5.5.7 Frame Number Field

The frame number field is an 11-bit field that is incremented by the host on a per frame basis. The frame number field rolls over upon reaching its maximum value of x7FF, and is sent only for SOF tokens at the start of each frame.

1.5.5.8 Data Field

The data field may range from 0 to 1023 bytes and must be an integral numbers of bytes. Data bits within each byte are shifted out LSB first.

Data packet size varies with the transfer type.

1.5.5.9 Cyclic Redundancy Checks

Cyclic redundancy checks (CRCs) are used to protect the all non-PID fields in token and data packets. In this context, these fields are considered to be protected fields. The PID is not included in the CRC check of a packet containing a CRC. All CRCs are generated over their respective fields in the transmitter before bit stuffing is performed. Similarly, CRCs are

decoded in the receiver after stuffed bits have been removed. Token and data packet CRCs provide 100% coverage for all single and double bit errors. A failed CRC is considered to indicate that one or more of the protected fields is corrupted and causes the receiver to ignore those fields, and, in most cases, the entire packet. For CRC generation and checking, the shift registers in the generator and checker are seeded with an all ones pattern.

1.5.5.9.1 Token CRCs

A five-bit CRC field is provided for tokens and covers the ADDR and ENDP fields of IN, SETUP, and OUT tokens or the time stamp field of an SOF token. The generator polynomial is:

$$G(X) = X^5 + X^2 + 1$$

1.5.6.9.2 Data CRCs

The data CRC is a 16-bit polynomial applied over the data field of a data packet. The generating polynomial is:

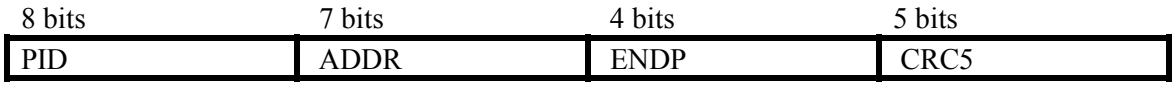
$$G(X) = X^{16} + X^{15} + X^2 + 1$$

1.5.5.10 Packet Formats

This section shows packet formats for token, data, and handshake packets. Fields within a packet are displayed in the order in which bits are shifted out onto the bus in the order shown in the figures.

1.5.5.10.1 Token Packets

A token consists of a PID, specifying either IN, OUT, or SETUP packet type, and ADDR and ENDP fields. For OUT and SETUP transactions, the address and endpoint fields uniquely identify the endpoint that will receive the subsequent data packet. For IN transactions, these fields uniquely identify which endpoint should transmit a data packet. Only the host can issue token packets. IN PIDs define a data transaction from a function to the host. OUT and SETUP PIDs define data transactions from the host to a function.



Token Format

Token packets have a five-bit CRC which covers the address and endpoint fields as shown above. The CRC does not cover the PID, which has its own check field. Token and SOF packets are delimited by an EOP after three bytes of packet field data. If a packet decodes as an otherwise valid token or SOF but does not terminate with an EOP after three bytes, it must be considered invalid and ignored by the receiver.

1.5.5.10.2 Start of Frame Packets

Start of Frame (SOF) packets are issued by the host at a nominal rate of once every 1.00 ms ±0.05. SOF packets consist of a PID indicating packet type followed by an 11-bit frame number field.

SOF Packet

The SOF token comprises the token-only transaction that distributes a start of frame marker and accompanying frame number at precisely timed intervals corresponding to the start of each frame. All full speed functions, including hubs, must receive and decode the SOF packet. The SOF token does not cause any receiving function to generate a return packet; therefore, SOF delivery to any given function cannot be guaranteed. The SOF packet delivers two pieces of timing information. A function is informed that a start of frame has occurred when it detects the SOF PID. Frame timing sensitive functions, which do not need to keep track of frame number, need only decode the SOF PID; they can ignore the frame number and its CRC. If a function needs to track frame number, then it must comprehend both the PID and the time stamp.

1.5.5.10.3 Data Packets

A data packet consists of a PID, a data field, and a CRC. There are two types of data packets, identified by differing PIDs: DATA0 and DATA1. Two data packet PIDs are defined to support data toggle synchronization.

Data Packet Format

Data must always be sent in integral numbers of bytes. The data CRC is computed over only the data field in the packet and does not include the PID, which has its own check field.

1.5.5.10.4 Handshake Packets

Handshake packets consist of only a PID. Handshake packets are used to report the status of a data transaction and can return values indicating successful reception of data, flow control, and stall conditions. Only transaction types that support flow control can return handshakes. Handshakes are always returned in the handshake phase of a transaction and may be returned, instead of data, in the data phase. Handshake packets are delimited by an EOP after one byte of packet field. If a packet decodes as an otherwise valid handshake but does not terminate with an EOP after one byte, it must be considered invalid and ignored by the receiver.

Handshake Packet

There are three types of handshake packets:

- **ACK** indicates that the data packet was received without bit stuff or CRC errors over the data field and that the data PID was received correctly. ACK may be issued either when sequence bits match and the receiver can accept data or when sequence bits mismatch and the sender and receiver must resynchronize to each other. An ACK handshake is applicable only in transactions which data has been transmitted and where a handshake is expected. ACK can be returned by the host for IN transactions and by a function for OUT transactions.
- **NAK** indicates that a function was unable to accept data from the host (OUT) or that a function has no data to transmit to the host (IN). NAK can only be returned by functions in the data phase of IN transactions or the handshake phase of OUT transactions, and the host can never issue a NAK. NAK is used for flow control purposes to indicate that a function is

temporarily unable to transmit or receive data, but will eventually be able to do so without need of host intervention. NAK is also used by interrupt endpoints to indicate that no interrupt is pending.

- **STALL** is returned by a function in response to an IN token or after the data phase of an OUT. STALL indicates that a function is unable to transmit or receive data, and that the condition requires host intervention to remove the stall. Once a function's endpoint is stalled, the function must continue returning STALL until the condition causing the stall has been cleared through host intervention. The host is not permitted to return a STALL under any condition.

15.5.10.5 Handshake Responses

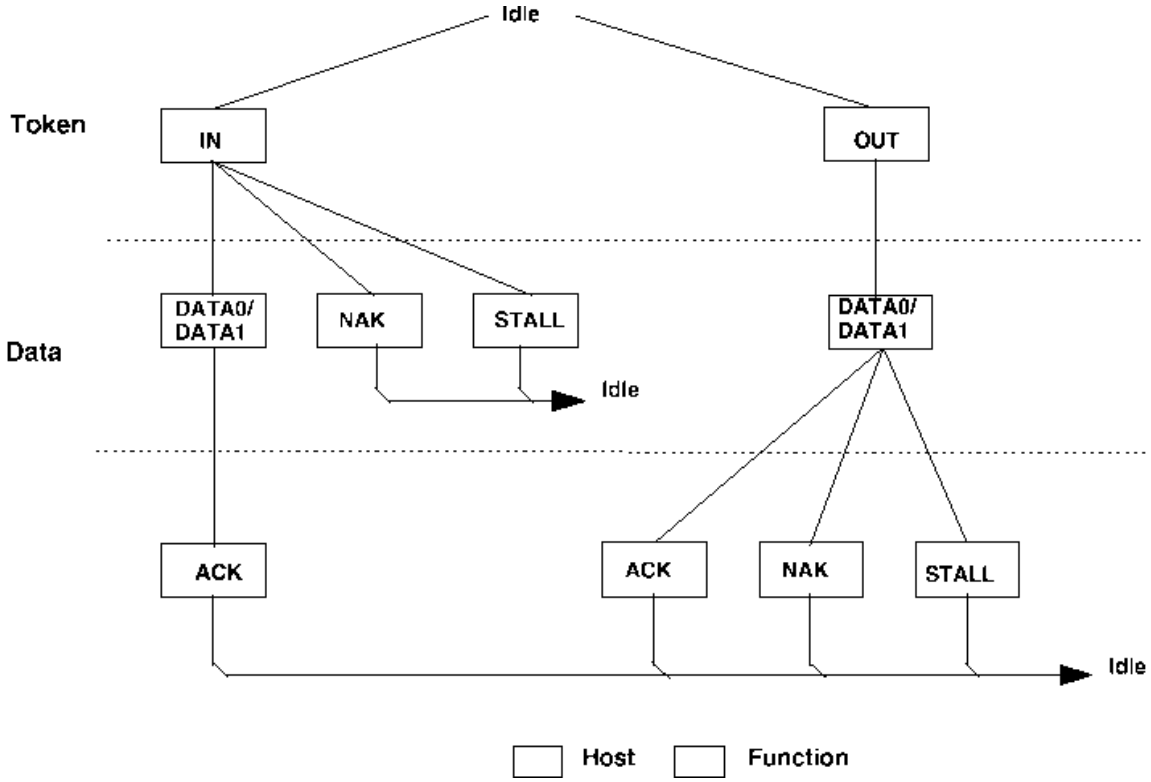
Transmitting and receiving functions must return handshakes based upon an order of precedence. Not all handshakes are allowed, depending on the transaction type and whether the handshake is being issued by a function or the host. For more details, refer to the USB specification.

1.5.6 Transaction Formats

Packet transaction format varies depending on the endpoint type. There are four endpoint types: **bulk, control, interrupt, and isochronous.**

1.5.6.1 Bulk Transactions

Bulk transaction types are characterized by the ability to guarantee error free delivery of data between the host and a function by means of error detection and retry. Bulk transactions use a three phase transaction consisting of token, data, and handshake packets. Under certain flow control and stall conditions, the data phase may be replaced with a handshake resulting in a two phase transaction in which no data is transmitted.



Bulk Transaction Format

When the host wishes to receive bulk data, it issues an IN token. The function endpoint responds by returning either a DATA packet or, should it be unable to return data, a NAK or STALL handshake. A NAK indicates that the function is temporarily unable to return data, while a STALL indicates that the endpoint is permanently stalled and requires host software intervention. If the host receives a valid data packet, it responds with an ACK handshake. If the host detects an error while receiving data, it returns no handshake packet to the function.
When the host wishes to transmit bulk data, it first issues an OUT token packet followed by a data packet. The function then returns one of three handshakes. ACK indicates that the data packet was received without errors and informs the host that that it may send the next packet in the sequence. NAK indicates that the data was received without error but that the host should resend the data because the function was in a temporary condition preventing it from accepting the data at this time (e.g., buffer full). If the endpoint was stalled, STALL is returned to indicate that the host should not retry the transmission because there is an error condition on the function. If the data packet was received with a CRC or bit stuff error, no handshake is returned.

Data packet synchronization is achieved via use of the data sequence toggle bits and the DATA0/DATA1 PIDs. Bulk endpoints must have their toggle sequence bits initialized via a separate control endpoint.

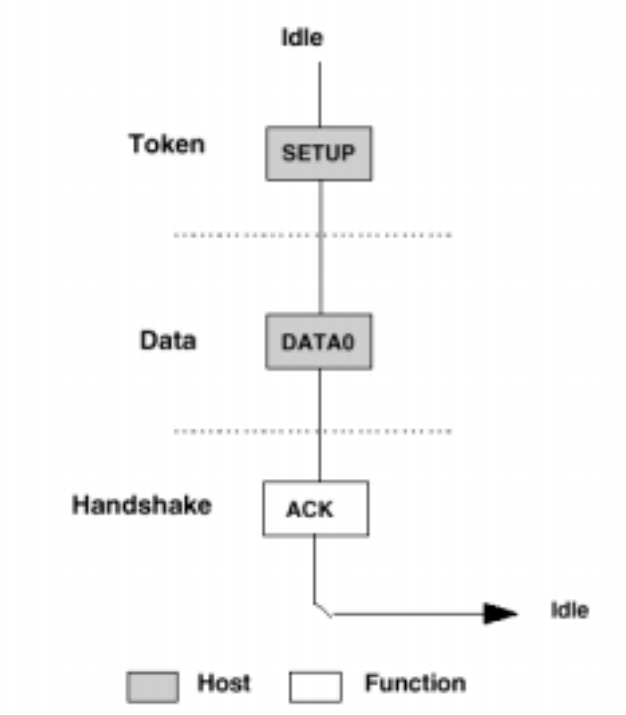
Bulk write	OUT (0)	OUT (1)	...	OUT (0/1)
	DATA0	DATA1	...	DATA0/1

Bulk Reads and Writes

The host always initializes the first transaction of a bus transfer to the DATA0 PID. The second transaction uses a DATA1 PID, and successive data transfers alternate for the remainder of the bulk transfer. The data packet transmitter toggles upon receipt of ACK, and the receiver toggles upon receipt and acceptance of a valid data packet.

1.5.6.2 Control Transfers

Control transfers minimally have two transaction stages: Setup and Status. A control transfer may optionally contain a data stage between the setup and status stages. During the Setup stage, a Setup transaction is used to transmit information to the control endpoint of a function. Setup transactions are similar in format to an OUT, but use a SETUP rather than on OUT PID. A Setup always uses a DATA0 PID for the data field of the Setup transaction. The function receiving a Setup must accept the Setup data and respond with an ACK handshake or, if the data is corrupted, discard the data and return no handshake.

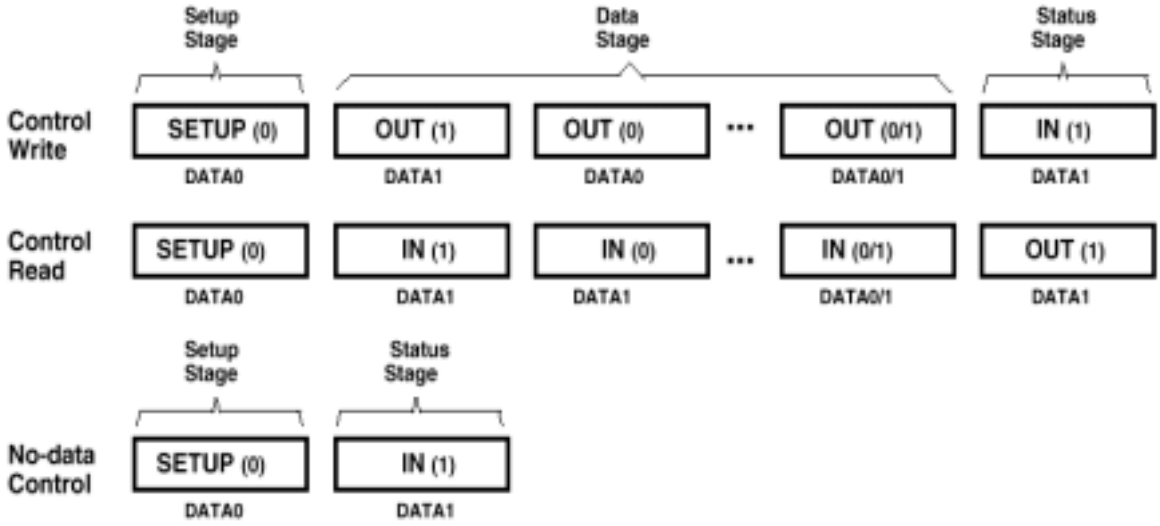


Control Setup Transaction

The data stage, if present, of a control transfer consists of one or more IN or OUT transactions and follows the same protocol rules as bulk transfers. All the transactions in the data stage must be in the same direction, i.e., all INs or all OUTs. The amount of data to be sent during the data phase and its direction are specified during the Setup stage. If the amount of data exceeds the

prenegotiated data packet size, the data is sent in multiple transactions (INs or OUTs) which carry the maximum packet size. Any remaining data is sent as a residual in the last transaction.

The status stage of a control transfer is the last operation in the sequence. A status stage is delineated by a change in direction of data flow from the previous stage and always uses a DATA1 PID. If, for example, the data stage consists of OUTs, the status is a single IN transaction. If the control sequence has no data stage, then it consists of a Setup stage followed by a Status stage consisting of an IN transaction. The following Figure shows the transaction order, the data sequence bit value and the data PID types for control read and write sequences. The sequence bits are displayed in parentheses.



Control Read and Write Sequences

1.5.6.2.1 Reporting Status Results

The status stage reports to the host the outcome of the previous setup and data stages of the transfer. Three possible results may be returned:

- The command sequence completed successfully
- The command sequence failed to complete
- The function is still busy completing command

Status reporting is always in the function to host direction.

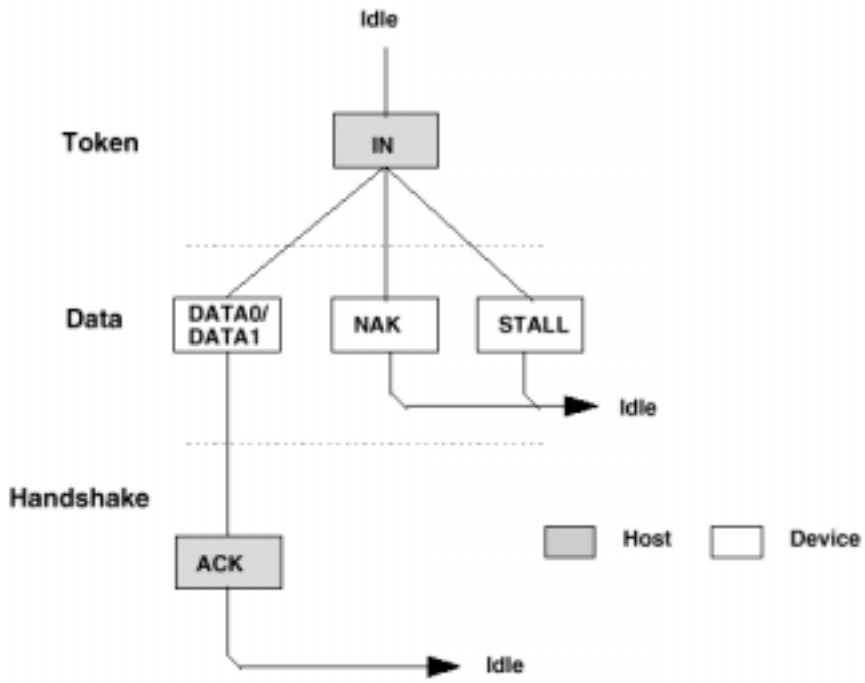
1.5.6.2.2 Error Handling on the Last Data Transaction

If the ACK handshake on an IN transaction gets corrupted, the function and the host will temporarily disagree on whether the transaction was successful. If the transaction is followed by another IN, the toggle retry mechanism will detect the mismatch and recover from the error. If the ACK was on the last IN of a control transfer, then the toggle retry mechanism cannot be used and an alternative scheme must be used. The host which successfully received the data of the last IN, issues an OUT setup transfer, and the function, upon seeing that the token direction has toggled, interprets this action as proof that the host successfully received the data. In other words, the function interprets the toggling of the token direction as implicit proof of the host's successful receipt of the last ACK handshake. Therefore, when the function sees the OUT setup transaction, it advances to the status phase.

Control writes do not have this ambiguity. The host, by virtue of receiving the handshake, knows for sure if the last transaction was successful. If an ACK handshake on an OUT gets corrupted, the host does not advance to the status phase and retries the last data instead.

1.5.6.3 Interrupt Transactions

Interrupt transactions consist solely of IN. Upon receipt of an IN token, a function may return data, NAK, or STALL. If the endpoint has no new interrupt information to return, i.e., no interrupt is pending, the function returns a NAK handshake during the data phase. A stalled interrupt endpoint causes the function to return a STALL handshake if it is permanently stalled and requires software intervention by the host. If an interrupt is pending, the function returns the interrupt information as a data packet. The host, in response to receipt of the data packet, issues either an ACK handshake if data was received error free or returns no handshake if the data packet was received corrupted.

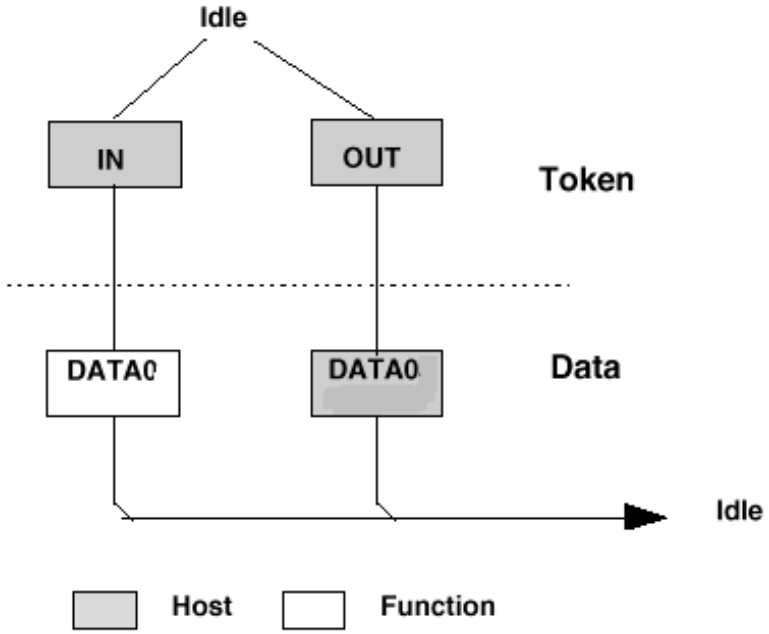


Interrupt Transaction Format

When an endpoint is using the Interrupt transfer mechanism for actual interrupt data, the data toggle protocol must be followed. This allows the function to know that the data has been received by the host and the event condition may be cleared. This `guaranteed' delivery of events allows the function to only send the interrupt information until it has been received by the host rather than having to send the interrupt data every time the function is polled and until host software clears the interrupt condition. When used in the toggle mode, an interrupt endpoint is initialized to the DATA0 PID and behaves the same as the bulk IN transaction. An Interrupt endpoint may also be used to communicate rate feedback information for certain types of isochronous functions. When used in this mode, the data toggle bits should be changed after each data packet is sent to the host without regard to the presence or type of handshake packet.

1.5.6.4 Isochronous Transactions

ISO transactions have a token and data phase, but no handshake phase. The host issues either an IN or an OUT token followed by the data phase in which the endpoint (for INs) or the host (for OUTs) transmits data. ISO transactions do not support a handshake phase or retry capability.



Isochronous Transaction Format

ISO transactions do not support toggle sequencing, and the data PID is always DATA0. The packet receiver does not examine the data PID.

1.5.7 Data Toggle Synchronization and Retry

USB provides a mechanism to guarantee data sequence synchronization between data transmitter and receiver across multiple transactions. This mechanism provides a means of guaranteeing that the handshake phase of a transaction was interpreted correctly by both the transmitter and receiver. Synchronization is achieved via use of the DATA0 and DATA1 PIDs and separate data toggle sequence bits for the data transmitter and receiver. Receiver sequence bits toggle only when the receiver is able to accept data and receives an error free data packet with the correct data PID. Transmitter sequence bits toggle only when the data transmitter receives a valid ACK handshake. The data transmitter and receiver must have their sequence bits synchronized at the start of a transaction, and the mechanism for doing this varies with the transaction type. Data toggle synchronization is not supported for ISO transfers.

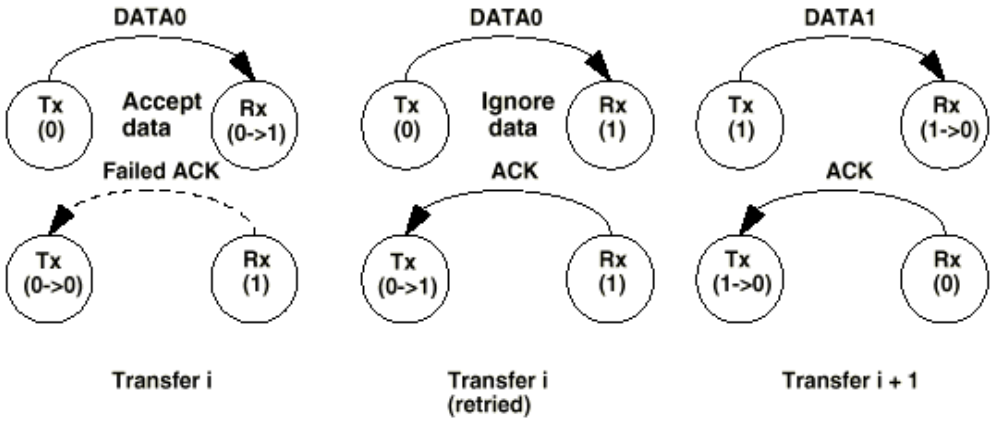
1.5.7.1 Initialization via SETUP Token

Control transfers use the SETUP token for initializing host and function sequence bits. Figure below shows the host issuing a SETUP packet to a function followed by an OUT. The numbers in the circles represent the transmitter and receiver sequence bits. The function must accept the data and ACK the transaction. When the function accepts the transaction, it must reset its sequence bit so that both the hosts and function's sequence bits are equal to `1' at the end of the SETUP transaction.

1.5.7.2 Data Corrupted or Not Accepted

If data cannot be accepted or the received data packet is corrupted, the receiver will issue a NAK or STALL handshake, or will time out, depending on the circumstances, and the receiver will not toggle its sequence bit. Where a transaction is NAKed and then retried. Any non-ACK handshake or time out will generate similar retry behavior. The transmitter, having not received an ACK handshake, will not toggle its sequence bit. As a result, a failed data packet transaction leaves the transmitter's and receiver's sequence bits synchronized and untoggled. The transaction will then be retried and, if successful, will cause both transmitter and receiver sequence bits to toggle.

1.5.7.3 Corrupted ACK Handshake

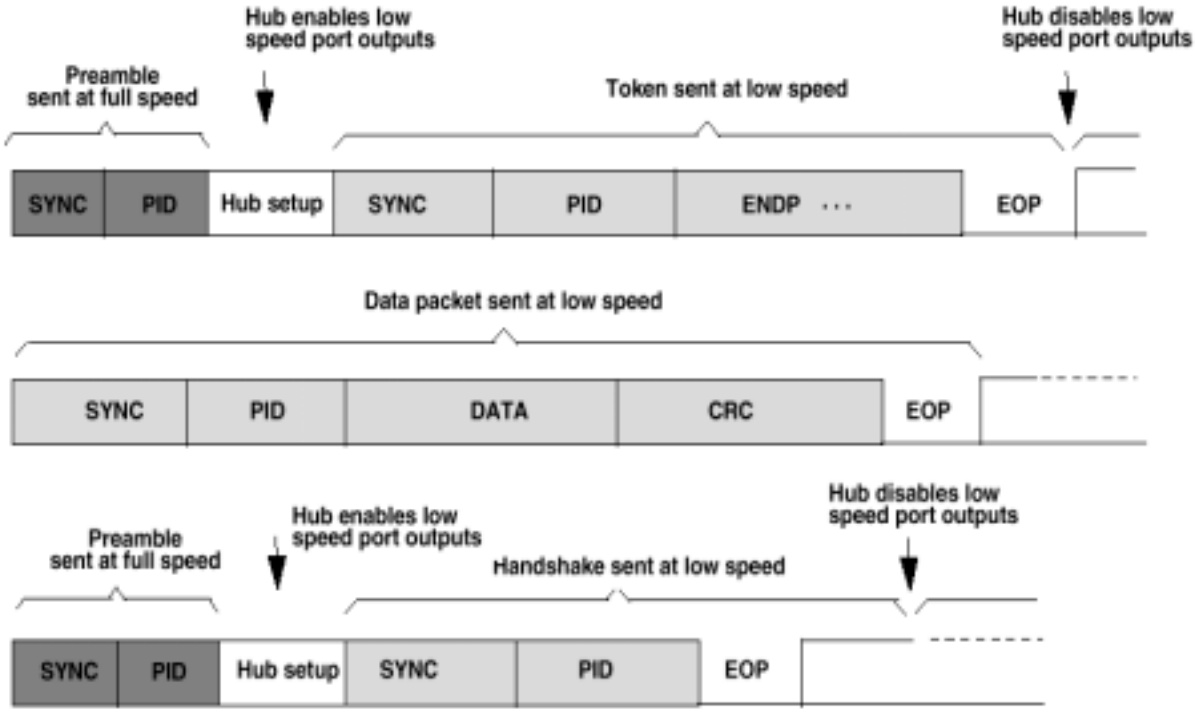


The transmitter is the last and only agent to know for sure whether a transaction has been successful, due to its receiving an ACK handshake. A lost or corrupted ACK handshake can lead to a temporary loss of synchronization between transmitter and receiver. Here the transmitter issues a valid data packet, which is successfully acquired by the receiver; however, the ACK handshake is corrupted.

At the end of transaction <i>, there is a temporary loss of coherency between transmitter and receiver, as evidenced by the mismatch between their respective sequence bits. The receiver has received good data, but the transmitter does not know whether it has successfully sent data. On the next transaction, the transmitter will resend the previous data using the previous DATA0 PID. The receiver's sequence bit and the data PID will not match, so the receiver knows that it has previously accepted this data. Consequently, it discards the incoming data packet and does not toggle its sequence bit. The receiver then issues an ACK, which causes the transmitter to regard the retried transaction as successful. Receipt of ACK causes the transmitter to toggle its sequence bit. At the beginning of transaction <i+1>, the sequence bits have toggled and are again synchronized. The data transmitter must guarantee that any retried data packet be identical in length to that sent in the original transaction. If the data transmitter is unable, because of problems such as a buffer underrun condition, to transmit the identical amount of data as was in the original data packet, it must abort the transaction by generating a bit stuffing violation. This causes a detectable error at the receiver and guarantees that a partial packet will not be interpreted as a good packet. The transmitter should not try to force an error at the receiver by sending a known bad CRC. A combination of a bad packet with a ``bad'' CRC may be interpreted by the receiver as a good packet.

1.5.8 Low Speed Transactions

USB supports signaling at two speeds: full speed (FS) signaling at 12.0 Mbs and low speed (LS) signaling at 1.5 Mbs. Hubs disable downstream bus traffic to all ports to which LS devices are attached during full speed downstream signaling. This is required both for EMI reasons and to prevent any possibility that an LS device might misinterpret downstream a FS packet as being addressed to it. Figure 8-19 shows an IN LS transaction in which the host issues a token and handshake and receives a data packet.



Low Speed Transaction

All downstream packets transmitted to LS devices require a preamble. The preamble consists of a SYNC followed by a PID, both sent at full speed. Hubs must comprehend the PRE PID; all other USB devices must ignore it and treat it as undefined. After the end of the preamble PID, the host must wait at least 4 full speed bit times during which hubs must complete the process of configuring their repeater sections to accept LS signaling. During this hub setup interval, hubs must drive their FS and LS ports to their respective idle states. Hubs must be ready to accept low speed signaling from the host before the end of the hub setup interval. Low speed connectivity rules are summarized below:

1. Low speed devices are identified during the connection process and the hub ports to which they are connected are identified as low speed.
2. All downstream low speed packets must be prefaced with a preamble (sent at full speed) which turns on the output buffers on low speed hub ports.
3. Low speed hub port output buffers are turned off upon receipt of EOP and are not turned on again until a preamble PID is detected.
4. Upstream connectivity is not affected by whether a hub port is full or low speed.

The start of LS signaling commences with the host issuing SYNC at low speed, followed by the remainder of the packet. The end of packet is identified by End of Packet (EOP), at which time all hubs tear down connectivity and disable any ports to which LS devices are connected. Hubs

do not switch ports for upstream signaling; LS ports remain enabled in the upstream direction for both LS and FS signaling. LS and FS transactions maintain a high degree of protocol commonality. However, LS signaling does have certain limitations which include:

- Data payload limited to eight bytes, maximum
- LS only supports Interrupt and Control types of transfers
- The SOF packet is not received by LS devices

1.5.9 Error Detection and Recovery

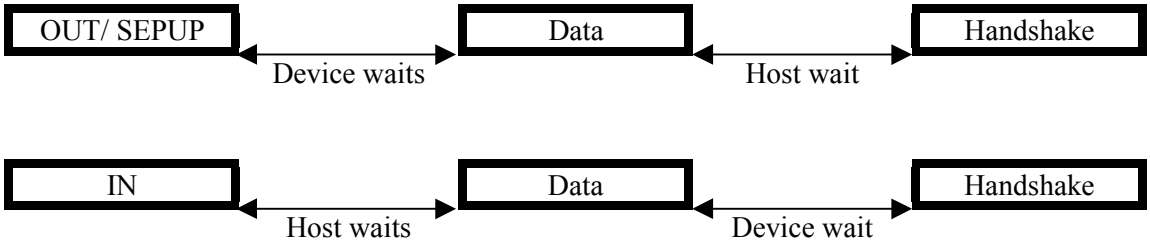
USB is designed to permit reliable end to end communication in the presence of errors on the physical signaling layer. This includes the ability to reliably detect the vast majority of possible errors and to recover from errors on a transaction type basis. Control transactions, for example, require a high degree of data reliability; they support end to end data integrity using error detection and retry. ISO transactions, by virtue of their bandwidth and latency requirements, do not permit retries and must tolerate a higher incidence of uncorrected errors.

1.5.9.1 Packet Error Categories

USB employs three error detection mechanisms: bit stuff violations, PID check bits, and CRCs.

1.5.9.2 Bus Turnaround Timing

The host and USB function need to keep track of how much time has elapsed from when the transmitter completes sending a packet until it begins to receive a packet back. This time is referred to as the bus turnaround time and is tracked by the packet transmitter's bus turnaround timer. Both devices and the host require turnaround timers. The device bus turnaround time is defined by the worst case round trip delay plus the maximum device response. USB devices cannot time out earlier than 16 bit times after the end of the previous EOP and they must time out by 18 bit times. If the host wishes to indicate an error condition via a timeout, it must wait at least 18 bit times before issuing the next token to insure that all downstream devices have timed out.



Bus Turnaround Timer Usage

As shown above, the device uses its bus turnaround timer between token and data or data and handshake phases. The host uses its timer between data and handshake or token and data phases. If the host receives a corrupted data packet, it must wait before sending out the next token. This wait interval guarantees that the host does not attempt to issue a token immediately after a false EOP.

1.5.9.3 False EOPs

If such an event were to occur, it would constitute a bus collision and have the ability to corrupt up to two consecutive transactions.

1.5.9.4 Babble and Loss of Activity Recovery

USB must be able to detect and recover from conditions which leave it waiting indefinitely for an end of packet or which leave the bus in something other than the idle state at the end of a frame. Loss of activity (LOA) is characterized by start of packet (SOP) followed by lack of bus activity and no end of packet (EOP) at the end of a frame. Babble is characterized by an SOP followed by the presence of bus activity past the end of a frame. LOA and babble have the potential to either deadlock the bus or force out the beginning of the next frame. Neither condition is acceptable, and both must be prevented from occurring. As the USB component responsible for controlling connectivity, hubs are responsible for babble/LOA detection and recovery. All USB devices that fail to complete their transmission at the end of a frame are prevented from transmitting past a frame's end by having the nearest hub disable the port to which the offending device is attached. Details of the hub babble/LOA recovery mechanism appear in Section 11.4.3 of the USB specification.

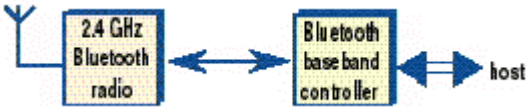
1.6. Bluetooth:

1.6.1.Technology Overview

The technology is an open specification for wireless communication of data and voice. It is based on a low-cost short-range radio link, built into a 9 x 9 mm microchip, facilitating protected ad hoc connections for stationary and mobile communication environments. Bluetooth - A Global Specification for Wireless Connectivity. Bluetooth technology allows for the replacement of the many proprietary cables that connect one device to another with one universal short-range radio link. For instance, Bluetooth radio technology built into both the cellular telephone and the laptop would replace the cumbersome cable used today to connect a laptop to a cellular telephone. Printers, PDA's, desktops, fax machines, keyboards, joysticks and virtually any other digital device can be part of the Bluetooth system. But beyond untethering devices by replacing the cables, Bluetooth radio technology provides a universal bridge to existing data networks, a peripheral interface, and a mechanism to form small private ad hoc groupings of connected devices away from fixed network infrastructures. Designed to operate in a noisy radio frequency environment, the Bluetooth radio uses a fast acknowledgement and frequency hopping scheme to make the link robust. Bluetooth radio modules avoid interference from other signals by hopping to a new frequency after transmitting or receiving a packet. Compared with other systems operating in the same frequency band, the Bluetooth radio typically hops faster and uses shorter packets. This makes the Bluetooth radio more robust than other systems. Short packages and fast hopping also limit the impact of domestic and professional microwave ovens. Use of Forward Error Correction (FEC) limits the impact of random noise on long-distance links. The encoding is optimized for an uncoordinated environment. Bluetooth radios operate in the unlicensed ISM band at 2.4 GHz. A frequency hop transceiver is applied to combat interference and fading. A shaped, binary FM modulation is applied to minimize transceiver complexity. The gross data rate is 1Mb/s. A Time-Division Duplex scheme is used for full-duplex transmission. The Bluetooth baseband protocol is a combination of circuit and packet switching. Slots can be reserved for synchronous packets.

Each packet is transmitted in a different hop frequency. A packet nominally covers a single slot, but can be extended to cover up to five slots. Bluetooth can support an asynchronous data channel, up to three simultaneous synchronous voice channels, or a channel which simultaneously supports asynchronous data and synchronous voice. Each voice channel supports 64 kb/s synchronous (voice) link. The asynchronous channel can support an asymmetric link of maximally 721 kb/s in either direction while permitting 57.6 kb/s in the return direction, or a 432.6 kb/s symmetric link. The different functions in the Bluetooth system are:

- a radio unit
- a link control unit
- link management
- software functions

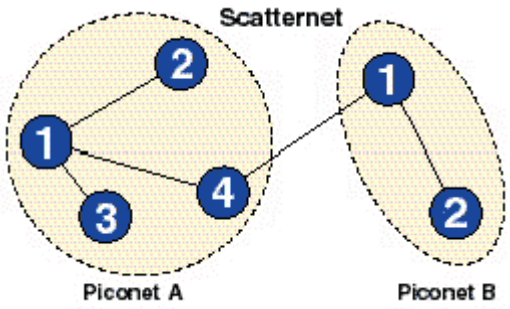


1.6.2. Definitions

- Piconet: a collection of devices connected via Bluetooth technology in an ad hoc fashion. A piconet starts with two connected devices, such as a portable PC and cellular phone, and may grow to eight connected devices.All Bluetooth devices are peer units and have identical implementations. However, when establishing a piconet, one unit will act as a master and the other(s) as slave(s) for the duration of the piconet connection.
- Scatternet: Multiple independent and non-synchronized piconets form a scatternet
- Master unit: the device in a piconet whose clock and hopping sequence are used to synchronize all other devices in the piconet.
- Slave units: all devices in a piconet that are not the master.
- Mac address: 3-bit address to distinguish between units participating in the piconet.
- Parked units: devices in a piconet which are synchronized but do not have a MAC addresses.
- Sniff and hold mode: devices synchronized to a piconet can enter power-saving modes in which device activity is lowered.

1.6.3. Network topology

The Bluetooth system supports both point-to-point and point-to-multi-point connections. Several piconets can be established and linked together ad hoc, where each piconet is identified by a different frequency hopping sequence. All users participating on the same piconet are synchronized to this hopping sequence. The topology can best be described as a multiple piconet structure.



The full-duplex data rate within a multiple piconet structure with 10 fully-loaded, independent piconets is more than 6 Mb/s.

1.6.3.1.Voice

Voice channels use the Continuous Variable Slope Delta Modulation (CVSD) voice coding scheme, and never retransmit voice packets. The CVSD method was chosen for its robustness in handling dropped and damaged voice samples. Rising interference levels are experienced as increased background noise: even at bit error rates up 4%, the CVSD coded voice is quite audible.

1.6.3.2. Radio

The Bluetooth air interface is based on a nominal antenna power of 0dBm. The air interface complies with the FCC rules for the ISM band at power levels up to 0dBm. Spectrum spreading has been added to facilitate optional operation at power levels up to 100 mW worldwide. Spectrum spreading is accomplished by frequency hopping in 79 hops displaced by 1 MHz, starting at 2.402 GHz and stopping at 2.480 GHz. Due to local regulations the bandwidth is reduced in Japan, France and Spain. This is handled by an internal software switch. The maximum frequency hopping rate is 1600 hops/s. The nominal link range is 10 centimeters to 10 meters, but can be extended to more than 100 meters by increasing the transmit power.
Baseband

The baseband describes the specifications of the digital signal processing part of the hardware - the Bluetooth link controller, which carries out the baseband protocols and other low-level link routines. Establishing network connections

Before any connections in a piconet are created, all devices are in STANDBY mode. In this mode, an unconnected unit periodically "listens" for messages every 1.28 seconds. Each time a device wakes up, it listens on a set of 32 hop frequencies defined for that unit. The number of hop frequencies varies in different geographic regions; 32 is the number for most countries (except Japan, Spain and France).

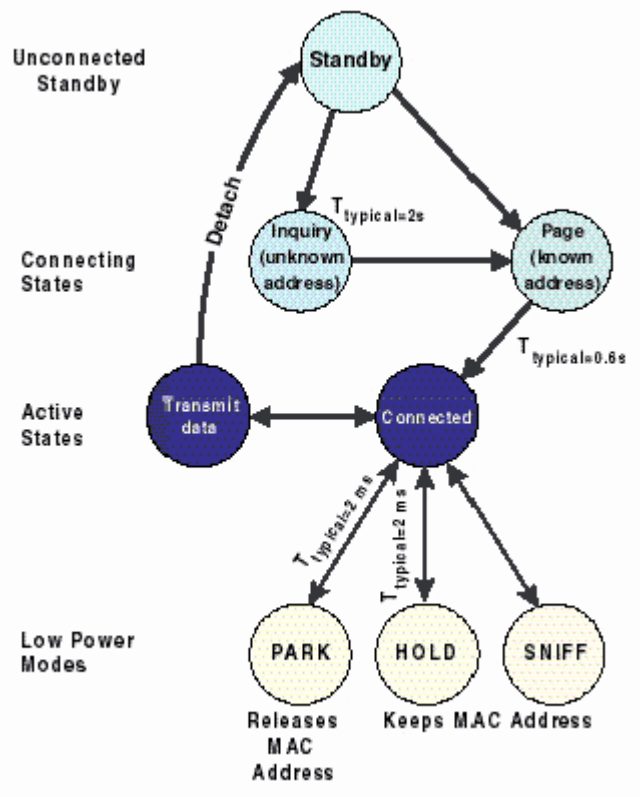
The connection procedure is initiated by any of the devices which then becomes master. A connection is made by a PAGE message if the address is already known, or by an INQUIRY message followed by a subsequent PAGE message if the address is unknown. In the initial PAGE state, the master unit will send a train of 16 identical page messages on 16 different hop frequencies defined for the device to be paged (slave unit). If no response, the master transmits a train on the remaining 16 hop frequencies in the wake-up sequence. The maximum delay

before the master reaches the slave is twice the wakeup period (2.56 seconds) while the average delay is half the wakeup period (0.64 seconds).

The INQUIRY message is typically used for finding Bluetooth devices, including public printers, fax machines and similar devices with an unknown address. The INQUIRY message is very similar to the page message, but may require one additional train period to collect all the responses. A power saving mode can be used for connected units in a piconet if no data needs to be transmitted.

The master unit can put slave units into HOLD mode, where only an internal timer is running. Slave units can also demand to be put into HOLD mode. Data transfer restarts instantly when units transition out of HOLD mode. The HOLD is used when connecting several piconets or managing a low power device such as a temperature sensor.

Two more low power modes are available, the SNIFF mode and the PARK mode. In the SNIFF mode, a slave device listens to the piconet at reduced rate, thus reducing its duty cycle. The SNIFF interval is programmable and depends on the application. In the PARK mode, a device is still synchronized to the piconet but does not participate in the traffic. Parked devices have given up their MAC address and occasional listen to the traffic of the master to re-synchronize and check on broadcast messages. If we list the modes in increasing order of power efficiency, then the SNIFF mode has the higher duty cycle, followed by the HOLD mode with a lower duty cycle, and finishing with the PARK mode with the lowest duty cycle. Link types and packet types



The link type defines what type of packets can be used on a particular link. The Bluetooth baseband technology supports two link types:

- Synchronous Connection Oriented (SCO) type (used primarily for voice)

- Asynchronous Connectionless (ACL) type (used primarily for packet data)

Different master-slave pairs of the same piconet can use different link types, and the link type may change arbitrarily during a session. Each link type supports up to sixteen different packet types. Four of these are control packets and are common for both SCO and ACL links. Both link types use a Time Division Duplex (TDD) scheme for full-duplex transmissions. The SCO link is symmetric and typically supports time-bounded voice traffic. SCO packets are transmitted over reserved intervals. Once the connection is established, both master and slave units may send SCO packets without being polled. One SCO packet types allows both voice and data transmission - with only the data portion being retransmitted when corrupted. The ACL link is packet oriented and supports both symmetric and asymmetric traffic. The master unit controls the link bandwidth and decides how much piconet bandwidth is given to each slave, and the symmetry of the traffic. Slaves must be polled before they can transmit data. The ACL link also supports broadcast messages from the master to all slaves in the piconet. Error correction There are three error-correction schemes defined for Bluetooth baseband controllers:

- 1/3 rate forward error correction code (FEC)
- 2/3 rate forward error correction code FEC
- Automatic repeat request (ARQ) scheme for data.

The purpose of the FEC scheme on the data payload is to reduce the number of retransmissions. However, in a reasonably error-free environment, FEC creates unnecessary overhead that reduces the throughput. Therefore, the packet definitions have been kept flexible as to whether or not to use FEC in the payload. The packet header is always protected by a 1/3 rate FEC; it contains valuable link information and should survive bit errors. An unnumbered ARQ scheme is applied in which data transmitted in one slot is directly acknowledged by the recipient in the next slot. For a data transmission to be acknowledged both the header error check and the cyclic redundancy check must be okay; otherwise a negative acknowledge is returned. Authentication and Privacy The Bluetooth baseband provides user protection and information privacy mechanisms at the physical layer. Authentication and encryption is implemented in the same way in each Bluetooth device, appropriate for the ad hoc nature of the network. Connections may require a one-way, two-way, or no authentication. Authentication is based on a challenge-response algorithm. Authentication is a key component of any Bluetooth system, allowing the user to develop a domain of trust between a personal Bluetooth device, such as allowing only the owner's notebook computer to communicate through the owner's cellular telephone. Encryption is used to protect the privacy of the connection. Bluetooth uses a stream cipher well suited for a silicon implementation with secret key lengths of 0, 40, or 64 bits. Key management is left to higher layer software. The goal of Bluetooth's security mechanisms is to provide an appropriate level of protection for Bluetooth's short-range nature and use in a global environment. Users requiring stalwart protection are encouraged to use stronger security mechanisms available in network transport protocols and application programs.

1.6.4. Link Management

The Link Manager (LM) software entity carries out link setup, authentication, link configuration, and other protocols. The Link Manager discovers other remote LM's and communicates with them via the Link Manager Protocol (LMP). To perform its service provider role, the LM uses the services of the underlying Link Controller (LC). Services provided:

- Sending and receiving of data

- Name request. The Link Manager has an efficient means to inquire and report a name or device ID upto 16 characters in length.
- Link address inquiries.
- Connection set-up.
- Authentication.
- Link mode negotiation and set-up, e.g. data or data/voice. This may be changed during a connection.
- The Link Manager decides the actual frame type on a packet-by-packet basis.
- Setting a device in sniff mode. In sniff mode, the duty cycle of the slave is reduced: it listens only every M slots where M is negotiated at the Link manager. The master can only start transmission in specified time slots spaced at regular intervals.
- Setting a link device on hold. In hold mode, turning off the receiver for longer periods saves power. Any device can wake up the link again, with an average latency of 4 seconds. This is defined by the Link Manager and handled by the Link Controller.
- Setting a device in park mode when it does not need to participate on the channel but wants to stay synchronized. It wakes up at regular intervals to listen to the channel in order to re-synchronize with the rest of the piconet, and to check for page messages.

1.6.5. Software Framework

Bluetooth devices will be required to support baseline interoperability feature requirements to create a positive consumer experience. For some devices, these requirements will extend from radio module compliance and air protocols, up to application-level protocols and object exchange formats. For other devices, such as a headset, the feature requirements will be significantly less. Ensuring that any device displaying the Bluetooth "logo" interoperates with other Bluetooth devices is a goal of the Bluetooth program. Software interoperability begins with the Bluetooth link level protocol responsible for protocol multiplexing, device and service discovery, and segmentation and reassembly. Bluetooth devices must be able to recognise each other and load the appropriate software to discover the higher level abilities each device supports. Interoperability at the application level requires identical protocol stacks. Different classes of Bluetooth devices (PC's, handhelds, headsets, cellular telephones) have different compliance requirements. For example, you would never expect a Bluetooth headset to contain an address book. Headsets compliance implies Bluetooth radio compliance, audio capability, and device discovery protocols. More functionality would be expected from cellular phones, handheld and notebook computers. To obtain this functionality, the Bluetooth software framework will reuse existing specifications such as OBEX, vCard/vCalendar, Human Interface Device (HID), and TCP/IP rather than invent yet another set of new specifications. Device compliance will require conformance to both the Bluetooth Specification and existing protocols. The Software Framework is contemplating the following functions:

- Configuration and diagnosis utility
- Device discovery
- Cable emulation
- Peripheral communication
- Audio communication and call control
- Object exchange for business cards and phone books Networking protocol

1.6.6. PC General

Usage models and implementation examples with a notebook PC focus are described in this section. The Bluetooth Specification contemplates interfaces where the radio modules may be integrated into notebook personal computers or attached using PC-Card or USB. Notebook PC usage models include:

- Remote networking using a Bluetooth cellular phone.
- Speakerphone applications using a Bluetooth cellular phone
- Business card exchange between Bluetooth notebooks, handhelds, and phones.
- Calendar synchronisation between Bluetooth notebooks, handhelds, and phones.

Bluetooth technology is operating system independent and not tied to any specific operating system. Implementations of the Bluetooth Specification for several commercial operating systems are in development. For notebook computers, the implementation of the Bluetooth Specification in Microsoft Windows98 and NT 5.0 using WDM and NDIS drivers is being contemplated.* Customer-visible interoperability is promoted by requiring minimal levels of software functionality, such as speakerphone, on notebook computers.) *Third-party brands and names are the property of their respective owners.
Telephone

Usage models and implementation examples focused on the digital cellular phone are described in this section. The Bluetooth Specification contemplates interfaces where the radio modules may be integrated directly into cellular handsets or attached using an add-on device. Phone usage models include (are not constrained to):

- Wireless hands-free operation using a Bluetooth headset.
- Cable-free remote networking with a Bluetooth notebook or handheld computer.
- Business card exchange with other Bluetooth phones, notebook or handheld computers.
- Automatic address book synchronisation with trusted Bluetooth notebooks or handheld computers.

The Bluetooth compliance document will require digital cellular phones to support some subset of the Bluetooth Specification. The Bluetooth contingents within the telephony Promoter companies are working with their fellow employees involved in the Wireless Application Protocol (WAP) Forum to investigate how the two technologies can benefit from each other.
Others

Usage models and implementation examples centered on other contemplated Bluetooth devices include:

- Headsets
- Handheld and wearable devices
- Human Interface Device (HID) compliant peripherals
- Data and voice access points

The wireless headset will support untethered audio for phones and provide phone-quality audio for notebook computers operating in sound-sensitive environments. The Bluetooth compliance document will specify the various parts of the Specification and existing specifications required by different classes of peripherals.

1.7. Reference:

Messmer, Hans-Peter., "The indispensable PC hardware book : your hardware questions answered." Addison-Wesley, 1994.

McDowell, Steven and Seyer, Martin D., "USB explained" Prentice Hall PTR.