Universal Serial Bus Specification

Compaq

Digital Equipment Corporation

IBM PC Company

Intel

Microsoft

NEC

Northern Telecom

Revision 1.0
January 15, 1996

Scope of this Revision

The 1.0 revision of the specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

Revision History

Revision	Issue Date	Comments
0.7	November 11, 1994	Supersedes 0.6e.
0.8	December 30, 1994	Revisions to Chapters 3-8, 10, and 11. Added appendixes.
0.9	April 13, 1995	Revisions to all the chapters.
0.99	August 25, 1995	Revisions to all the chapters.
1.0 FDR	November 13, 1995	Revisions to Chapters 1, 2, 5-11.
1.0	January 15, 1996	Edits to Chapters 5, 6, 7, 8, 9, 10, and 11 for consistency.

Please send comments via electronic mail to USB@fes.fm.intel.com
For industry information, refer to the USB Implementers Forum web page at http://www.teleport.com/~USB

Contents

CHAPTER 7 ELECTRICAL

CHAPTER 8 PROTOCOL LAYER

CHAPTER 10 USB HOST: HARDWARE AND SOFTWARE

CHAPTER 11 HUB SPECIFICATION

Chapter 1
Introduction

1.1 Motivation

The motivation for the Universal Serial Bus comes from three interrelated considerations:

- **Connection of the PC to the telephone**
  It is well understood that the merge of computing and communication will be the basis for the next generation of productivity applications. The movement of machine-oriented and human-oriented data types from one location or environment to another depends on ubiquitous and cheap connectivity. Unfortunately, the computing and communication industries have evolved independently. The Universal Serial Bus provides a ubiquitous link that can be used across a wide range of PC to telephone interconnects.

- **Ease of use**
  The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles heel to its further deployment. The combination of user friendly graphical interfaces and the hardware and software mechanisms associated with new generation bus architectures like PCI, PnP ISA, and PCMCIA has made computers less confrontational and easier to reconfigure. However, from the end user point of view, the PC's I/O interfaces such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug and play.

- **Port expansion**
  The addition of external peripherals continues to be constrained by port availability. The lack of a bi-directional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals such as telephone/fax/modem adapters, answering machines, scanners, PDA's, keyboards, mice, etc. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need.

  The Universal Serial Bus is the answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface that is consistent with the requirements of the PC platform of today and tomorrow.

1.2 Objective of the Specification

This document defines an industry standard Universal Serial Bus. The specification describes the bus attributes, the protocol definition, types of transactions, bus management, and the programming interface required to design and build systems and peripherals that are compliant with this standard.

The goal is to enable such devices from different vendors to inter-operate in an open architecture. The specification is intended as an enhancement to the PC architecture spanning portable, business desktops, and home environments. It is intended that the specification allow system OEMs and peripheral developers adequate room for product versatility and market differentiation without the burden of carrying obsolete interfaces or losing compatibility.

1.3 Scope of the Document

- **Target audience**
  The specification is primarily targeted to peripheral developers and system OEMs, but provides valuable information for platform operating system/ BIOS/ device driver, adapter IHVs/ISVs, and platform/adapter controller vendors.
- **Benefit**
  This version of the Universal Serial Bus specification can be used for planning new products, engineering an early prototype, and preliminary software development. All final products are required to be compliant with the Universal Serial Bus Specification 1.0.

1.4 Document Organization

Chapters 1 through 5 provide an overview for all readers, while Chapters 6 through 11 contain detailed technical information defining the Universal Serial Bus.

Peripheral implementers should particularly read Chapters 5 through 11.

Universal Serial Bus Host Controller implementers should particularly read Chapters 5, 6, 7, 8, 10, and 11.

Universal Serial Bus device driver implementers should particularly read Chapters 5, 9, and 10.

This document is complemented and referenced by the following related documents, which will be released shortly:

- The Universal Serial Bus Device Class Specification
- The Universal Serial Bus Design Guide

Please contact the USB Implementers Forum for further details.

Readers are also requested to contact operating system vendors for operating system bindings specific to the USB.

Chapter 2
Terms and Abbreviations

This chapter lists and defines terms and abbreviations used throughout this specification.

Access.bus	The Access.bus is developed by the Access.bus Industry Group, based on the Phillips I^2C technology and a DEC software model. Revision 2.2 specifies the bus for 100 kbs operation, but the technology has headroom to go up to 400 kbs.
ACK	Acknowledgment. Handshake packet indicating a positive acknowledgment.
Active Device	A device that is powered and not in the suspend state.
ADB	See Apple Desktop Bus.
APM	An acronym for Advanced Power Management. APM is a specification for managing suspend and resume operations to conserve power on a host system.
Apple Desktop Bus	An expansion bus used by personal computers manufactured by Apple Computer, Inc.
Asynchronous Data	Data transferred at irregular intervals with relaxed latency requirements.
Asynchronous RA	The incoming data rate, Fs_i, and the outgoing data rate, Fs_o, of the RA process are independent (i.e., no shared master clock).
Asynchronous SRC	The incoming sample rate, Fs_i, and outgoing sample rate, Fs_o, of the SRC process are independent (i.e., no shared master clock).
Audio Device	A device that sources or sinks sampled analog data.
AWG#	The measurement of a wire's cross section as defined by the American Wire Gauge standard.
Babble	Unexpected bus activity that persists beyond a specified point in a frame.
Bandwidth	The amount of data transmitted per unit of time, typically bits per second (bps) or bytes per second (Bps).
Big Endian	A method of storing data that places the most significant byte of multiple byte values at a lower storage addresses. For example, a word stored in big endian format places the least significant byte at the higher address and the most significant byte at the lower address. See Little Endian.
Bit	A unit of information used by digital computers. Represents the smallest piece of addressable memory within a computer. A bit expresses the choice between two possibilities and is typically represented by a logical one (1) or zero (0).
Bit Stuffing	Insertion of a "0" bit into a data stream to cause an electrical transition on the data wires allowing a PLL to remain locked.
bps	Transmission rate expressed in bits per second.

Bps	Transmission rate expressed in bytes per second.
Buffer	Storage used to compensate for a difference in data rates or time of occurrence of events, when transmitting data from one device to another.
Bulk Transfer	Nonperiodic, large bursty communication typically used for a transfer that can use any available bandwidth and also be delayed until bandwidth is available.
Bus Enumeration	Detecting and identifying Universal Serial Bus devices.
Byte	A data element that is eight bits in size.
Capabilities	Those attributes of a Universal Serial Bus device that are administerable by the host.
Characteristics	Those qualities of a Universal Serial Bus device that are unchangeable; for example, the device class is a device characteristic.
CHI	An acronym for Concentration Highway Interface. CHI is a full duplex time division multiplexed serial interface for digitized voice transfers in communications systems. The current specification supports data transfer rates up to 4.096 Mbs.
Client	Software resident on the host that interacts with host software to arrange data transfer between a function and the host. The client is often the data provider and consumer for transferred data.
COM Port	Communications port. On personal computers, an eight-bit asynchronous serial port is typically used.
Configuring Software	The host software responsible for configuring a Universal Serial Bus device. This may be a system configurator or software specific to the device.
Control Pipe	Same as a message pipe.
Control Transfer	One of four Universal Serial Bus Transfer Types. Control transfers support configuration/command/status type communications between client and function.
CRC	See Cyclic Redundancy Check.
CTI	Computer Telephony Integration.
Cyclic Redundancy Check	A check performed on data to see if an error has occurred in transmitting, reading, or writing the data. The result of a CRC is typically stored or transmitted with the checked data. The stored or transmitted result is compared to a CRC calculated for the data to determine if an error has occurred.
Default Address	An address defined by the Universal Serial Bus Specification and used by a Universal Serial Bus device when it is first powered or reset. The default address is 00h.
Default Pipe	The message pipe created by Universal Serial Bus system software to pass control and status information between the host and a Universal Serial Bus device's Endpoint 0.

Device	A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a Universal Serial Bus interface device. At an even higher level, device may refer to the function performed by an entity attached to the Universal Serial Bus; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical. When used as a non-specific reference, a Universal Serial Bus device is either a hub or a function.
Device Address	The address of a device on the Universal Serial Bus. The Device Address is the Default Address when the Universal Serial Bus device is first powered or reset. Hubs and functions are assigned a unique Device Address by Universal Serial Bus software.
Device Endpoint	A uniquely identifiable portion of a Universal Serial Bus device that is the source or sink of information in a communication flow between the host and device.
Device Resources	Resources provided by Universal Serial Bus devices, such as buffer space and endpoints. See Host Resources and Universal Serial Bus Resources.
Device Software	Software that is responsible for using a Universal Serial Bus device. This software may or may not also be responsible for configuring the device for use.
DMI	An acronym for Desktop Management Interface. A method for managing host system components developed by the Desktop Management Task Force.
Downstream	The direction of data flow from the host or away from the host. A downstream port is the port on a hub electrically farthest from the host that generates downstream data traffic from the hub. Downstream ports receive upstream data traffic.
Driver	When referring to hardware, an I/O pad that drives an external load. When referring to software, a program responsible for interfacing to a hardware device; that is, a device driver.
DWORD	Double word. A data element that is 2 words, 4 bytes, or 32 bits in size.
Dynamic Insertion and Removal	The ability to attach and remove devices while the host is in operation.
E^2PROM	See EEPROM.
EEPROM	Electrically Erasable Programmable Read Only Memory. Non-volatile rewritable memory storage technology.
End User	The user of a host.
Endpoint	See Device Endpoint.
Endpoint Address	The combination of a Device Address and an Endpoint Number on a Universal Serial Bus device.
Endpoint Number	A unique pipe endpoint on a Universal Serial Bus device.

EOF1	End of frame timing point #1. Used by the hub to monitor and disconnect bus activity persisting near or past the end of a frame.
EOF2	End of frame timing point #2. Used by hubs to detect bus activity near the end of frame.
EOP	End of packet.
Fs	See Sample Rate.
False EOP	A spurious, usually noise induced, event that is interpreted by a packet receiver as an end of packet.
FireWire	Apple Computer's implementation of the IEEE P1394 bus standard.
Frame	The time from the start of one SOF token to the start of the subsequent SOF token; consists of a series of transactions.
Frame Pattern	A sequence of frames that exhibit a repeating pattern in the number of samples transmitted per frame. For a 44.1 kHz audio transfer, the frame pattern could be nine frames containing 44 samples followed by one frame containing 45 samples.
Full-duplex	Computer data transmission occurring in both directions simultaneously.
Function	A Universal Serial Bus device that provides a capability to the host. For example, an ISDN connection, a digital microphone, or speakers.
GeoPort	A serial bus developed by Apple Computer, Inc. Current specification of the GeoPort supports data transfer rates up to 2 Mbs and provides point to point connectivity over a radius of 4 ft.
Handshake Packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NACK.
Host	The host computer system where the Universal Serial Bus host controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
Host Controller	The host's Universal Serial Bus interface.
Host Controller Driver	The Universal Serial Bus software layer that abstracts the host controller hardware. Host Controller Driver provides an SPI for interaction with a host controller. Host Controller Driver hides the specifics of the host controller hardware implementation.
Host Resources	Resources provided by the host, such as buffer space and interrupts. See Device Resources and Universal Serial Bus Resources.
Hub	A Universal Serial Bus device that provides additional connections to the Universal Serial Bus.
Hub Tier	The level of connect within a USB network topology given as the number of hubs that that the data has to flow through.
I^2C	Acronym for the Inter-Integrated Circuits serial interface. The I^2C interface was invented by Philips Semiconductors.
IEEE P1394	A high performance serial bus. The P1394 is targeted at hard disk and video peripherals, which may require bus bandwidth in excess of 100 Mb/s. The bus protocol supports both isochronous and asynchronous transfers over the same set of four signal wires.

Industry Standard Architecture	The 8 and/or 16 bit expansion bus for IBM AT or XT compatible computers.
Integrated Services Data Network	An internationally accepted standard for voice, data, and signaling using public, switched telephone networks. All transmissions are digital from end-to-end. Includes a standard for out-of-band signaling and delivers significantly higher bandwidth than POTS.
Interrupt Request	A hardware signal that allows a device to request attention from a host. The host typically invokes an interrupt service routine to handle the condition which caused the request.
Interrupt Transfer	One of four Universal Serial Bus Transfer Types. Interrupt transfers characteristics are small data, non periodic, low frequency, bounded latency, device initiated communication typically used to notify the host of device service needs.
IRQ	See Interrupt Request.
ISA	See Industry Standard Architecture.
ISDN	See Integrated Services Data Network.
Isochronous Data	A stream of data whose timing is implied by its delivery rate.
Isochronous Device	An entity with isochronous endpoints, as defined in the USB specification, that sources or sinks sampled analog streams or synchronous data streams.
Isochronous Sink Endpoint	An endpoint that is capable of consuming an isochronous data stream.
Isochronous Source Endpoint	An endpoint that is capable of producing an isochronous data stream.
Isochronous Transfer	One of four Universal Serial Bus Transfer Types. Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device.
Jitter	A tendency toward lack of synchronization caused by mechanical or electrical changes. More specifically, the phase shift of digital pulses over a transmission medium.
kbs	Transmission rate expressed in kilobits per second.
kBs	Transmission rate expressed in kilobytes per second.
Line Printer Port	A port used to access a printer. On most personal computers, an eight-bit parallel interface is typically used.
Little Endian	Method of storing data that places the least significant byte of multiple byte values at lower storage addresses. For example, a word stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address. See Big Endian.
LOA	Loss of bus activity characterized by a start of packet without a corresponding end of packet.
LPT Port	See Line Printer Port.
LSB	Least Significant Bit.
Mbs	Transmission rate expressed in megabits per second.

MBs	Transmission rate expressed in megabytes per second.
Message Pipe	A pipe that transfers data using a request/data/status paradigm. The data has an imposed structure which allows requests to be reliably identified and communicated.
Micro Channel Architecture	A 32 bit expansion bus used on some IBM PS/2 compatible computers.
Modem	An acronym for Modulator/Demodulator. Component that converts signals between analog and digital. Typically used to send digital information from a computer over a telephone network which is usually analog.
MSB	Most Significant Bit.
NACK	Negative Acknowledgment. Handshake packet indicating a negative acknowledgment.
Non Return to Zero Invert	A method of encoding serial data in which ones and zeroes are represented by opposite and alternating high and low voltages where there is no return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
NRZI	See Non Return to Zero Invert.
Object	Host software or data structure representing a Universal Serial Bus entity.
Packet	A bundle of data organized in a group for transmission. Packets typically contain three elements: control information (e.g., source, destination, and length), the data to be transferred, and error detection and correction bits.
Packet Buffer	The logical buffer used by a Universal Serial Bus device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.
Packet ID	A field in a Universal Serial Bus packet that indicates the type of packet, and by inference the format of the packet and the type of error detection applied to the packet.
PBX	See Private Branch eXchange.
PCI	See Peripheral Component Interconnect.
PCMCIA	See Personal Computer Memory Card Industry Association.
Peripheral Component Interconnect	A 32- or 64-bit, processor independent, expansion bus used on personal computers.
Personal Computer Memory Card International Association	The organization that standardizes and promotes PC Card technology.
Phase	A token, data, or handshake packet; a transaction has three phases.
Physical Device	A device that has a physical implementation; e.g. speakers, microphones, and CD players.
PID	See Packet ID.

Pipe	A logical abstraction representing the association between an endpoint on a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (Stream Pipe) or messages (Message Pipe).
Plain Old Telephone Service	Basic service supplying standard single line telephones, telephone lines, and access to public switched networks.
Plug and Play	A technology for configuring I/O devices to use non-conflicting resources in a host. Resources managed by Plug and Play include I/O address ranges, memory address ranges, IRQs, and DMA channels.
PnP	See Plug and Play.
Polling	Asking multiple devices, one at a time, if they have any data to transmit.
POR	See Power On Reset.
Port	Point of access to or from a system or circuit. For Universal Serial Bus, the point where a Universal Serial Bus device is attached.
POTS	See Plain Old Telephone Service.
Power On Reset	Restoring a storage device, register, or memory to a predetermined state when power is applied.
PLL	Phase Locked Loop. A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
Private Branch eXchange	A privately owned telephone switching system which is not regulated as part of the public telephone network.
Programmable Data Rate	Either a fixed data rate (single frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. The exact programming capabilities of an endpoint must be reported in the appropriate class-specific endpoint descriptors.
Protocol	A specific set of rules, procedures, or conventions relating to format and timing of data transmission between two devices.
RA	See Rate Adaptation.
Rate Adaptation	The process by which an incoming data stream, sampled at Fs_i is converted to an outgoing data stream, sampled at Fs_o with a certain loss of quality, determined by the rate adaptation algorithm. Error control mechanisms are required for the process. Fs_i and Fs_o can be different and asynchronous. Fs_i is the input data rate of the RA; Fs_o is the output data rate of the RA.
Request	A request made to a Universal Serial Bus device contained within the data portion of a SETUP packet.
Retire	The action of completing service for a transfer and notifying the appropriate software client of the completion.
Root Hub	A Universal Serial Bus hub directly attached to the host controller. This hub is attached to the host; tier 0.
Root Port	The upstream port on a hub.
Sample	The smallest unit of data on which an endpoint operates; a property of an endpoint.
Sample Rate (Fs)	The number of samples per second, expressed in Hertz.

Sample Rate Conversion	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
SCSI	See Small Computer Systems Interface.
Service	A procedure provided by an SPI.
Service Interval	The period between consecutive requests to a Universal Serial Bus endpoint to send or receive data.
Service Jitter	The deviation of service delivery from its scheduled delivery time.
Service Rate	The number of services to a given endpoint per unit time.
Small Computer Systems Interface	A local I/O bus that allows peripherals to be attached to a host using generic system hardware and software.
SOF	An acronym for Start of Frame. The SOF is the first transaction in each frame. SOF allows endpoints to identify the start of frame and synchronize internal endpoint clocks to the host.
SPI	See System Programming Interface.
SRC	See Sample Rate Conversion.
Stage	One part of the sequence composing a control transfer; i.e., the setup stage, the data stage, and the status stage.
Stream Pipe	A pipe that transfers data as a stream of samples with no defined Universal Serial Bus structure.
Synchronization Type	A classification that characterizes an isochronous endpoint's capability to connect to other isochronous endpoints.
Synchronous RA	The incoming data rate, $Fs_{i,}$ and the outgoing data rate, $Fs_{o,}$ of the RA process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
Synchronous SRC	The incoming sample rate, $Fs_{i,}$ and outgoing sample rate, $Fs_{o,}$ of the SRC process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
System Programming Interface	A defined interface to services provided by system software.
TDM	See Time Division Multiplexing.
Termination	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
Time Division Multiplexing	A method of transmitting multiple signals (data, voice, and/or video) simultaneously over one communications medium by interleaving a piece of each signal one after another.
Time-out	The detection of a lack of bus activity for some predetermined interval.
Token Generator	See Initiator.
Token Packet	A type of packet that identifies what transaction is to be performed on the bus.

Transaction	The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/required based on the transaction type.
Transfer	One or more bus transactions to move information between a software client and its function.
Transfer Type	Determines the characteristics of the data flow between a software client and its function. Four Transfer types are defined: control, interrupt, bulk, and isochronous.
Turnaround Time	The time a device needs to wait to begin transmitting a packet after a packet has been received to prevent collisions on Universal Serial Bus. This time is based on the length and propagation delay characteristics of the cable and the location of the transmitting device in relation to other devices on Universal Serial Bus.
Universal Serial Bus	A collection of Universal Serial Bus devices and the software and hardware that allow them to connect the capabilities provided by functions to the host.
Universal Serial Bus Device	Includes hubs and functions. See device.
Universal Serial Bus Interface	The hardware interface between the Universal Serial Bus cable and a Universal Serial Bus device. This includes the protocol engine required for all Universal Serial Bus devices to be able to receive and send packets.
Universal Serial Bus Resources	Resources provided by Universal Serial Bus, such as bandwidth and power. See Device Resources and Host Resources.
Universal Serial Bus Software	The host-based software responsible for managing the interactions between the host and the attached Universal Serial Bus devices.
USB	See Universal Serial Bus.
USBD	See Universal Serial Bus Driver.
Universal Serial Bus Driver	The host resident software entity responsible for providing common services to clients that are manipulating one or more functions on one or more Host Controllers.
Upstream	The direction of data flow towards the host. An upstream port is the port on a device electrically closest to the host that generates upstream data traffic from the hub. Upstream ports receive downstream data traffic.
Virtual Device	A device that is represented by a software interface layer; e.g., a hard disk with its associated device driver and client software that makes it able to reproduce an audio .WAV file.
WFEOF2	Wait for EOF2 point. One of the four hub repeater states.
WFEOP	Wait for end of packet. One of the four hub repeater states.
WFSOF	Wait for start of frame. One of the four hub repeater states.
WFSOP	Wait for start of packet. One of the four possible hub repeater states.
Word	A data element that is two bytes or 16 bits in size.

Chapter 3
Background

This chapter presents a brief description of the background of the Universal Serial Bus including design goals, features of the bus, and existing technologies.

3.1 Goals for the Universal Serial Bus

The Universal Serial Bus is specified to be an industry standard extension to the PC architecture with a focus on Computer Telephony Integration (CTI), consumer, and productivity applications. The following criteria were applied in defining the architecture for the Universal Serial Bus:

- Ease of use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 12 Mbs
- Full support for the real-time data for voice, audio, and compressed video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehend various PC configurations and form factors
- Provide a standard interface capable of quick diffusion into product
- Enable new classes of devices that augment the PC's capability

3.2 Taxonomy of Application Space

Figure 3-1 describes a taxonomy for the range of data traffic workloads that can be serviced over a Universal Serial Bus. As can be seen, a 12 Mbs bus comprehends the mid-speed and low-speed data ranges. Typically, mid-speed data types are isochronous and low-speed data comes from interactive devices. The Universal Serial Bus being proposed is primarily a desktop bus but can be readily applied to the mobile environment. The software architecture allows for future extension of the Universal Serial Bus by providing support for multiple Universal Serial Bus host controllers.

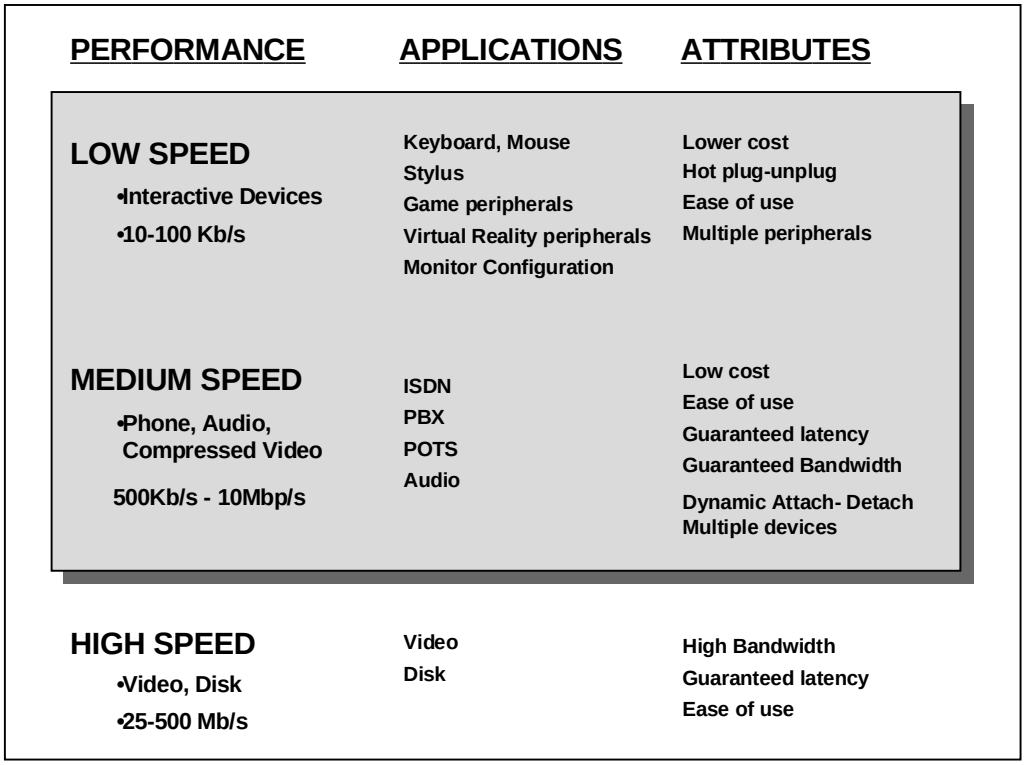


Figure 3-1. Application Space Taxonomy

3.3 Feature List

The Universal Serial Bus specification provides a selection of attributes that can achieve multiple price-performance integration points and can enable functions that allow differentiation at the system and component level. Features are categorized by benefits below:

Easy to use for end user

- Single model for cabling and connectors
- Electrical details isolated from end user; e.g., bus terminations
- Self identifying peripherals, automatic mapping of function to driver, and configuration
- Dynamically attachable and reconfigurable peripherals

Wide range of workloads and applications

- Suitable for device bandwidths ranging from a few kbs to several Mbs
- Supports isochronous as well as asynchronous transfer types over the same set of wires
- Multiple Connections: Support for concurrent operation of many devices
- Support for up to 127 physical devices
- Supports transfer of multiple data and message streams between the host and devices
- Allows compound devices; i.e., peripherals composed of many functions
- Lower protocol overhead resulting in high bus utilization

Isochronous bandwidth

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, etc.
- Isochronous workload may use entire bus bandwidth

Flexibility

- Wide range of packet sizes, allowing a range of device buffering options
- Wide range of device data rates by accommodating packet buffer size and latencies
- Flow control for buffer handling built into protocol

Robustness

- Error handling/fault recovery mechanism built into protocol
- Dynamic insertion and removal of devices identified in user perceived real-time
- Support for identification of faulty devices

Synergy with PC industry

- Simple protocol to implement and integrate
- Consistent with the PC Plug and Play architecture
- Leverages existing operating system interfaces

Low-cost implementation

- Low cost sub-channel at 1.5 Mbs
- Optimized for integration in peripheral and host hardware
- Suitable for development of low cost peripherals
- Low cost cables and connectors
- Utilizes commodity technologies

Upgrade path

- Architecture upgradeable to support multiple Universal Serial Bus host controllers in a system

3.4 Some Existing Technologies

There are several technologies that are commonly considered to be serial buses. Each of these buses were defined for a specific range of application(s). A few of them are listed below:

- **Apple desktop bus (ADB)**
  This is a proprietary minimalist serial interface that provides a simple read/write protocol to up to 16 devices. The cost of hardware interface is estimated to be very low. The ADB supports data rates up to 90 kbs, just enough to communicate with keyboards, pointing devices, or other desktop I/O devices.

- **Access.bus (A.b)**
  The Access.bus is being developed by the Access.bus Industry Group, based on the Philips I^2C technology and a DEC software model. The application space for the Access.bus is primarily keyboards and pointing devices; however, A.b is more versatile then the ADB. The protocol has well defined specifications for the dynamic attach, arbitration, data packets, configuration, and software interface. While addressing is provided for up to 127 devices, the practical loading is limited by cable lengths and power distribution considerations. Revision 2.2 of the A.b specification specifies the bus for 100 kbs operation, but the technology has headroom to go up to 400 kbs using the same separate clock and data wires.

- **IEEE P1394**
  The IEEE P1394 is a high performance serial bus. The application space for P1394 is primarily hard disk and video peripherals, which may require bus bandwidth in excess of 100 Mbs. The protocol supports both isochronous and asynchronous transfers over the same set of four signal wires, broken up as differential pair of clock and data signals. The P1394 specification is very well defined and the first generation devices, based on the IEEE specification, are just coming to market. Current pricing of P1394 solutions is considered competitive relative to SCSI disk interfaces, but not for generic desktop connectivity.

- **CHI**
  The Concentration Highway Interface (CHI) was developed by AT&T for terminals and digital switches. CHI is a full duplex time division multiplexed serial interface for digitized voice transfers in communications systems. The protocol consists of a number of fixed time slots that can carry voice data and control information. The current specification supports data transfer rates up to 4.096 Mbs. The CHI bus has four signal wires: Clock, Framing, Receive data, and Transmit data. Both, the Framing and the Clock signals are generated centrally (i.e., PBX switch).

- **GeoPort**
  The GeoPort was originally developed by Apple Computer, Inc. to primarily enable Macintosh telephony applications. Current specification of the GeoPort supports data transfer rates up to 2 Mbs and provides point to point connectivity over a radius of 4 ft. The standard GeoPort specifies a 9-pin connector (8 pins and an optional 9th power pin) and uses RS-422 signaling. Additionally, Apple has defined an alternate 14-pin connector for extended cable lengths. The GeoPort protocol provides three different operating modes: Beaconing, TDM, and Packetized transfer modes. Apple is currently licensing the GeoPort specification.

Chapter 4
Architectural Overview

This chapter presents an overview of the Universal Serial Bus architecture and key concepts. 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.

Later chapters describe the various components of the USB in greater detail.

4.1 USB System Description

A USB system is described by three definitional areas:

- USB interconnect
- USB devices
- USB host

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.

USB devices and the USB host are described in detail in subsequent sections.

4.1.1 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. Figure 4-1 illustrates the topology of the USB.

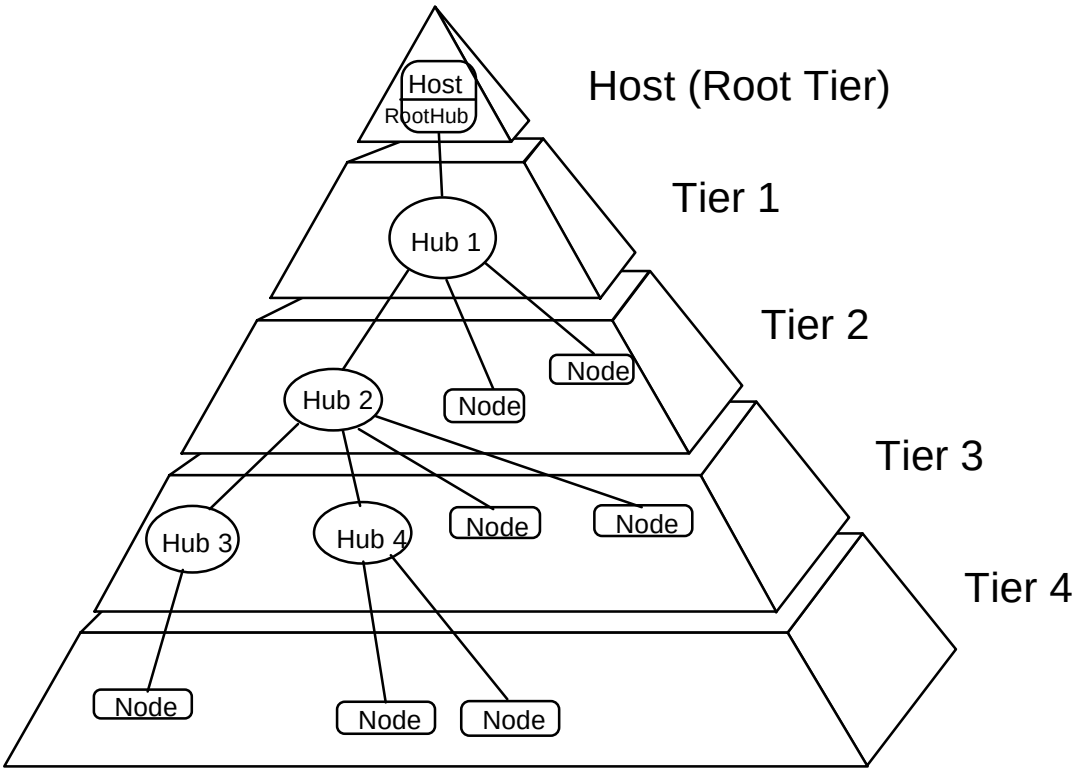


Figure 4-1. Bus Topology

4.1.1.1 The USB Host

There is only one host on any USB system. The USB interface to the host computer system is referred to as the host controller. The host controller may be implemented in a combination of hardware, firmware, or software. A root hub is integrated within the host system to provide one or more attachment points. Additional information concerning the host may be found in Section 4.9 and in Chapter 10, USB Host: Hardware and Software.

4.1.1.2 USB Devices

USB devices are:

- Hubs, which provide additional attachment points to the USB
- Functions, which provide capabilities to the system; for example, an ISDN connection, a digital joystick, or speakers

USB devices present a standard USB interface in terms of their:

- Comprehension of the USB protocol
- Response to standard USB operations such as configuration and reset
- Standard capability descriptive information

Additional information concerning USB devices may be found in Section 4.8 and in Chapter 9, USB Devices.

4.2 Physical Interface

The physical interface of the USB is described in the electrical (Chapter 7) and mechanical (Chapter 6) specifications for the bus.

4.2.1 Electrical

USB transfers signal and power over a four wire cable, shown in Figure 4-2. 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 with bit stuffing to ensure adequate transitions. A SYNC field precedes each packet to allow the receiver(s) to synchronize their bit recovery clocks.

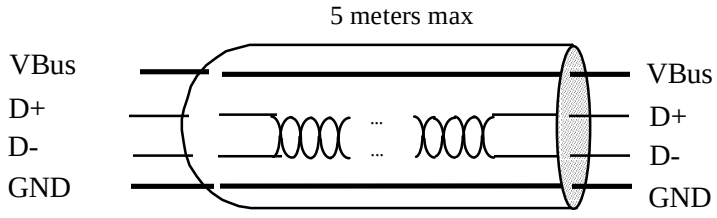


Figure 4-2. USB Cable

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.

4.2.2 Mechanical

The mechanical specifications for cables and connectors are provided in Chapter 6. All devices have an upstream connection. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loopback connections at hubs. The cable has four conductors: a twisted signal pair of standard gauge and a power pair in a range of permitted gauges. The connector is four position, with shielded housing, specified robustness, and ease of attach-detach characteristics.

4.3 Power

The specification covers two aspects of power:

- Power distribution over the USB deals with the issues of how USB devices consume power provided by the host over the USB.
- Power management deals with how USB software and devices fit into the host-based power management system.

4.3.1 Power Distribution

Each USB segment provides a limited amount of power over the cable. The host supplies power for use by USB devices that are directly connected. In addition, any USB device may have its own power supply. USB devices that rely totally on power from the cable are called bus-powered devices. In contrast, those that have an alternate source of power are called self-powered devices. A hub also supplies power for its connected USB devices. The architecture permits bus-powered hubs within certain constraints of topology that are discussed later in Chapter 11. Self-powered devices must implement prescribed power decoupling safety mechanisms. In Figure 4-4, the keyboard, pen, and mouse can all be bus-powered devices.

4.3.2 Power Management

A USB host has a power management system which is independent of the USB. USB system software interacts with the host's power management system to handle system power events such as SUSPEND or RESUME. Additionally, USB devices can carry USB-defined power management information which allow them to be power managed by system software or generic device drivers.

The power distribution and power management features of USB allow it to be designed into power sensitive systems such as battery based notebook computers.

4.4 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.

4.5 Robustness

There are several attributes of the USB that contribute to its robustness:

- Signal integrity using differential drivers, receivers, and shielding
- CRC protection over control and data fields
- Detection of attach and detach and system-level configuration of resources
- Self-recovery in protocol, using time-outs for lost or broken packets
- Flow control for streaming data to ensure isochrony and hardware buffer management
- Data and control pipe constructs for ensuring independence from adverse interactions between functions

4.5.1 Error Detection

The core bit error rate of the USB medium is expected to be close to that of a backplane and any glitches will very likely be transient in nature. To provide protection against such transients, each of these packets includes error protection fields. When data integrity is required, such as with lossless data devices, an error recovery procedure may be invoked in hardware or software.

The protocol includes separate CRCs for control and data fields of each packet. A failed CRC is considered to indicate a corrupted packet. The CRC gives 100% coverage on single and double bit errors.

4.5.2 Error Handling

The protocol optionally allows for error handling in hardware or software. Hardware handling includes reporting and retry of failed transfers. The host controller will retry an error three times before informing the client software of the error. The client software can recover in an implementation specific way.

4.6 System Configuration

The USB supports USB devices attaching to and detaching from the USB at any point in time. Consequently, enumerating the USB is an on-going activity which must accommodate dynamic changes in the physical bus topology.

4.6.1 Attachment of USB Device

All USB devices attach to the USB via a port on specialized USB devices known as hubs. Hubs indicate the attachment or removal of a USB device in its per port status. The host queries the hub to determine the reason for the notification. The hub responds by identifying the port used to attach the USB device. The host enables the port and addresses the USB device with a control pipe using the USB Default

Address. All USB devices are addressed using the USB Default Address when initially connected or after they have been reset.

The host determines if the newly attached USB device is a hub or a function and assigns a unique USB address to the USB device. The host establishes a control pipe for the USB device using the assigned USB address and endpoint number zero.

If the attached USB device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached USB devices.

If the attached USB device is a function, then attachment notifications will be dispatched by USB software to interested host software.

4.6.2 Removal of USB Device

When a USB device has been removed from one of its ports, the hub automatically disables the port and provides an indication of device removal to the host. Then the host removes knowledge of the USB device from any host data structures.

If the removed USB device is a hub, the removal process must be performed for all of the USB devices that were previously attached to the hub.

If the removed USB device is a function, removal notifications are sent to interested host software.

4.6.3 Bus Enumeration

Bus enumeration is the activity that identifies and addresses devices attached to a bus. For many buses, this is done at startup time and the information collected is static. Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity. Additionally, bus enumeration for the USB also includes detection and processing of removals.

4.6.4 Inter-Layer Relationship

USB devices are logically divided into a USB device interface portion, a device portion, and a functional portion. The host is logically partitioned into the USB host interface portion, the aggregate system software portion (USB system software and host system software), and the device software portion.

Each of these portions is defined such that a particular USB task is the responsibility of only one portion. The USB host and USB device portions correspond as shown in Table 4-1.

Table 4-1. Correlation Between Host and Device Layers

USB Host Portion	USB Device Portion
Device Software	Function
System Software	Device
USB Interface	USB Interface

4.7 Data Flow Types

The USB supports functional data and control exchange between the USB host and a USB device as a set of either uni- or bi-directional fashions. USB data transfers take place between host software and a particular endpoint on a USB device. A given USB device may support multiple data transfer endpoints. The USB host treats communications with any endpoint of a USB device independently from any other endpoint. Such associations between the host software and a USB device endpoint are called pipes. As

an example, a given USB device could have an endpoint which would support a pipe for transporting data *to* the USB device and another endpoint which would support a pipe for transporting data *from* the USB device.

The USB architecture comprehends four basic types of data transfers:

- Control transfers that are used to configure a device at attach time and can be used for other device specific purposes
- Bulk data transfers which are generated or consumed in relatively large and bursty quantities and has wide dynamic latitude in transmission constraints
- Interrupt data transfers such as characters or coordinates with human perceptible echo or feedback response characteristics
- Isochronous or streaming real time data transfers which occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency

Any given pipe supports exactly one of the types of transfers described above. The USB data flow model is described in more detail in Chapter 5.

4.7.1 Control Transfers

Control data is used by USB software to configure devices when they are first attached. Other driver software can choose to used control transfers in implementation specific ways. Data delivery is lossless.

4.7.2 Bulk Transfers

Bulk data typically consists of larger amounts of data such as that used for printers or scanners. Bulk data is sequential. Reliable exchange of data is ensured at the hardware level by using error detection in hardware and, optionally, invoking a limited hardware retry. Also, the bandwidth taken up by bulk data can be whatever is available and not being used for other transfer types.

4.7.3 Interrupt Transfers

A small, spontaneous data transfer from a device is referred to as interrupt data. Such data may be presented for transfer by a device at any time and is delivered by the USB at a rate no slower than as is specified by the device.

Interrupt data typically consists of event notification, characters, or coordinates that are organized as one or more bytes. An example of interrupt data is the coordinates from a pointing device. Although an explicit timing rate is not required, interactive data may have response time bounds which the USB must support.

4.7.4 Isochronous Transfers

Isochronous data is continuous and real-time in creation, delivery, and consumption. Timing related information is implied by the steady rate at which isochronous data is received and transferred. Isochronous data must be delivered at the rate received to maintain its timing. In addition to delivery rate, isochronous data may also be sensitive to delivery delays. For isochronous pipes, the bandwidth required is typically based upon the sampling characteristics of the associated function. The latency required is related to the buffering available at each endpoint.

A typical example of isochronous data is voice. If the delivery rate of these data streams is not maintained, glitches in the data stream will occur due to buffer or frame underruns or overruns. Even if data is delivered at the appropriate rate, delivery delays may degrade applications requiring real-time turn around, such as telephony-based audio conferencing.

The timely delivery of isochronous data is ensured at the expense of potential transient losses in the data stream. In other words, any error in electrical transmission is not corrected by hardware mechanisms such as retries. In practice, the core bit error rate of the USB is expected to be small enough not to be an issue. USB isochronous data streams are allocated a dedicated portion of USB bandwidth to ensure that data can be delivered at the desired rate. The USB is also designed for minimal delay of isochronous data transfers.

4.7.5 Allocating USB Bandwidth

USB bandwidth is allocated among pipes. The USB allocates bandwidth for some pipes when a pipe is established. USB devices are required to provide some buffering of data. It is assumed that USB devices requiring more bandwidth are capable of providing larger sized buffers. The goal for the USB architecture is to ensure that buffering induced hardware delay is bounded to within a few milliseconds.

USB's bandwidth capacity can be allocated among many different data streams. This allows a wide range of devices to be attached to the USB. For example, telephony devices ranging from 1B+D all the way up to T1 capacity can be accommodated. Further, different device bit rates, with a wide dynamic range, can be concurrently supported.

USB bandwidth allocation is blocking; i.e., if allocating an additional pipe would disturb preexisting bandwidth or latency allocations, further pipe allocations are denied or blocked. When a pipe is closed, the allocated bandwidth is freed up and may be reallocated to another pipe.

The USB Specification defines the rules for how each transfer type is allowed access to the bus.

4.8 USB Devices

USB devices are divided into device classes such as hub, locator, or text device. The hub device class indicates a specially designated USB device which provides additional USB attachment points (refer to Chapter 11). USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

4.8.1 Device Characterizations

All USB devices are accessed by a unique USB address. Each USB device additionally supports one or more endpoints with which the host may communicate. All USB devices must support a specially designated Endpoint 0 to which the USB device's USB control pipe will be attached.

Associated with Endpoint 0 is the information required to completely describe the USB device. This information falls into the following categories:

- Standard. This is information whose definition is common to all USB devices and includes items such as vendor identification, device class, and power management. Device, configuration, interface, and endpoint descriptions carry configuration related information about the device. Detailed information about these descriptors can be found in Chapter 9.
- Class. The definition of this information varies depending on the device class of the USB device.
- USB Vendor. The vendor of the USB device is free to put any information desired here. The format, however, is not determined by this specification.

Additionally, each USB device carries USB control and status information. All USB devices support a common access method via their USB control pipe.

4.8.2 Device Descriptions

Two major divisions of device classes exist: hubs and functions. Only hubs have the ability to provide additional USB attachment points. Functions provide additional capabilities to the host.

4.8.2.1 Hubs

Hubs are a key element in the plug-and-play architecture of USB. Figure 4-3 shows a typical hub. Hubs serve to simplify USB connectivity from the user's perspective and provide robustness at low cost and complexity.

Hubs are wiring concentrators and enable the multiple attachment characteristics of USB. Attachment points are referred to as ports. Each hub converts a single attachment point into multiple attachment points. The architecture supports concatenation of multiple hubs.

The upstream port of a hub connects the hub towards the host. Each of the other downstream ports of a hub allows connection to another hub or function. Hubs can detect attach and detach at each downstream port and enable the distribution of power to downstream devices. Each downstream port can be individually enabled and configured as either full or low speed. The hub isolates low speed ports from full speed signaling.

A hub consists of two portions: the Hub Controller and the Hub Repeater. The repeater is a protocol controlled switch between the upstream port and downstream ports. It also has hardware support for reset and suspend/resume signaling. The controller provides the interface registers to allow communication to/from the host. Hub specific status and control commands permit the host to configure a hub and to monitor and control its ports.

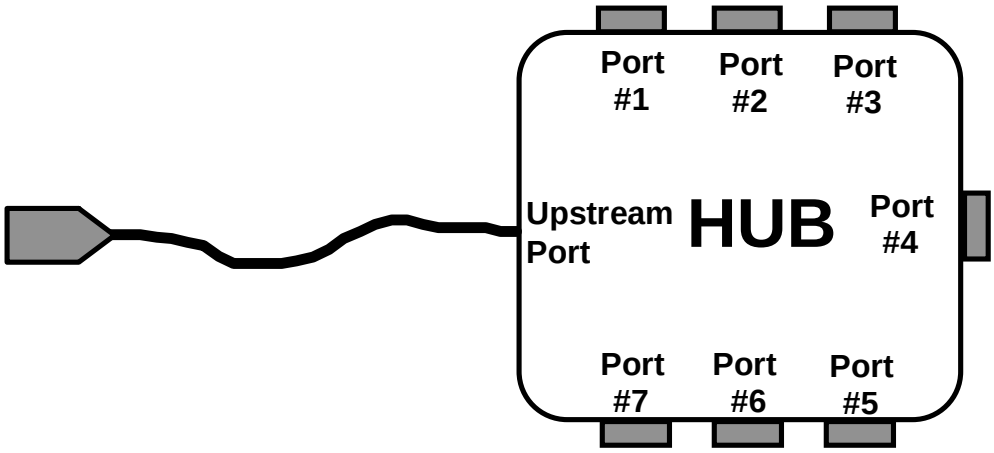


Figure 4-3. A Typical Hub

Figure 4-4 illustrates how hubs provide connectivity in a desktop computer environment.

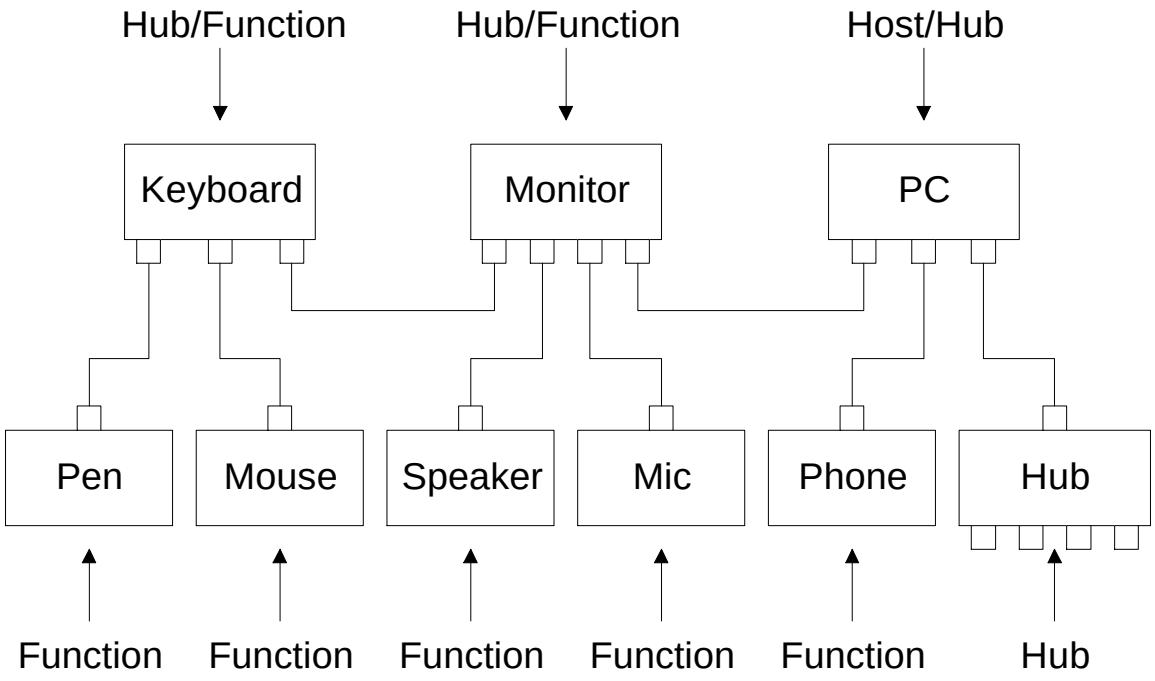


Figure 4-4. Hubs in a Desktop Computer Environment

4.8.2.2 Functions

A function is a USB device that is able to transmit or receive data or control information over the bus. A function is typically implemented as a separate peripheral device with a cable that plugs into a port on a hub. However, a physical package may implement multiple functions and an embedded hub with a single USB cable. This is known as a compound device. A compound device appears to the host as a hub with one or more permanently attached USB devices.

Each function contains configuration information that describes its capabilities and resource requirements. Before a function can be used, it must be configured by the host. This configuration includes allocating USB bandwidth and selecting function specific configuration options.

Examples of functions are:

- A locator device such as a mouse, tablet, or light pen
- An input device such as a keyboard
- An output device such as a printer
- A telephony adapter such as ISDN

4.9 USB Host: Hardware and Software

The USB Host interacts with USB devices through the host controller. The host is responsible for the following:

- Detecting the attachment and removal of USB devices
- Managing control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Providing a limited amount of power to attached USB devices

USB system software on the host manages interactions between USB devices and host-based device software. There are five areas of interactions between USB system software and device software, they are:

- Device enumeration and configuration
- Isochronous data transfers
- Asynchronous data transfers
- Power management
- Device and bus management information

Whenever possible, USB software uses existing host system interfaces to manage the above interactions. For example, if a host system uses Advanced Power Management (APM) for power management, USB system software connects to the APM message broadcast facility to intercept suspend and resume notifications.

4.10 Architectural Extensions

The USB architecture comprehends extensibility at the interface between the Host Controller Driver and USB driver. Implementations with multiple host controllers, and associated Host Controller Drivers, are possible.

Chapter 5
USB Data Flow Model

This chapter presents information about how data is moved across the USB that affects all implementers. The information presented is at a level above the signaling and protocol definitions of the system. Chapter 7, Electrical and Chapter 8, Protocol Layer should be consulted for more details about their respective parts of the USB system. This chapter provides framework information that is further expanded in Chapter 9, USB Device Framework, Chapter 10, USB Host: Hardware and Software, and Chapter 11, Hub Specification. This chapter should be read by all implementers to understand key concepts of the USB.

5.1 Implementer Viewpoints

The USB provides communication services between a host and attached USB devices. However, the simple view an end user sees of attaching one or more USB devices to a host, as in Figure 5-1, is in fact a little more complicated to implement than as indicated by the figure. Different views of the system are required to explain specific USB requirements from the perspective of different implementers. Several important concepts and features must be supported to provide the end user with the reliable operation demanded from today's personal computers. USB is presented in a layered fashion to ease explanation and allow implementers of particular USB products to focus on the details related to their product.

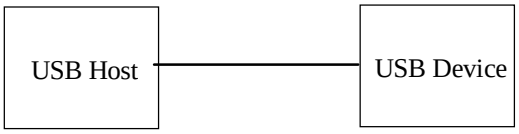


Figure 5-1. Simple USB Host/Device View

Figure 5-2 shows a deeper overview of USB identifying the different layers of the system that will be described in more detail in the remainder of the specification. In particular, there are four focus implementation areas:

- USB Physical Device - A piece of hardware on the end of a USB cable that performs some useful end user function.
- Client Software - Software that executes on the host corresponding to a USB device. This client software is typically supplied with the operating system or provided along with the USB device.
- USB System Software - Software that supports USB in a particular operating system. Typically supplied with the operating system independently of particular USB devices or client software.
- USB Host Controller (Host Side Bus Interface) - The hardware and software that allows USB devices to be attached to a host.

There are shared rights and responsibilities between the four USB system components. The remainder of this specification describes the details required to support robust, reliable communication flows between a function and its client.

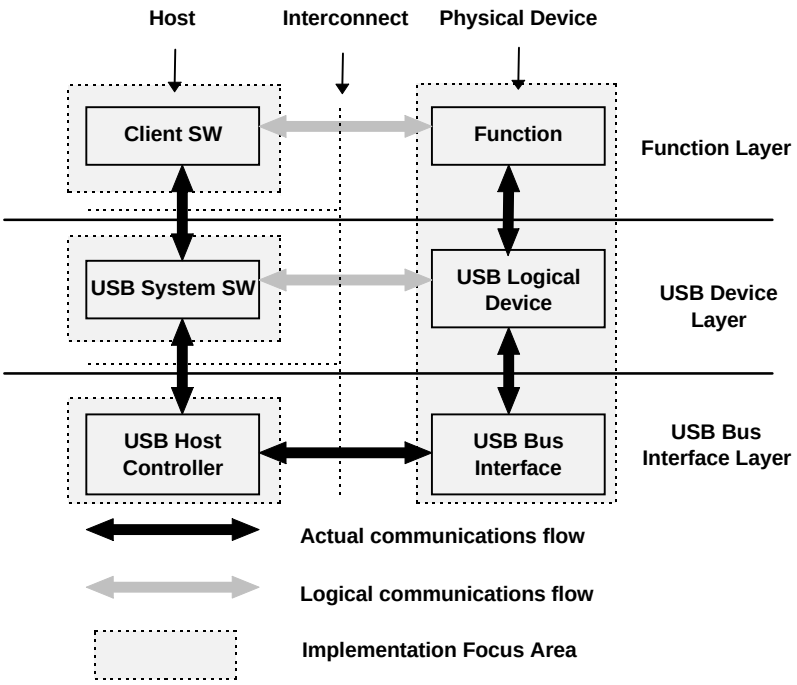


Figure 5-2. USB Implementation Areas

As shown in Figure 5-2, the simple connection of a host to a device requires interaction between a number of layers and entities. The USB Bus Interface layer provides physical/signaling/packet connectivity between the host and a device. The USB Device Layer is the view the USB System software has for performing generic USB operations with a device. The Function Layer provides additional capabilities to the host via an appropriate matched client software layer. The USB Device and Function layers each have a view of logical communication within their layer that actually uses the USB Bus Interface Layer to accomplish data transfer.

The physical view of USB communication as described in Chapters 6, 7, and 8 is related to the logical communication view presented in Chapters 9 and 10. This chapter describes those key concepts that affect USB implementers and should be read by all before proceeding to the remainder of the specification to find those details most relevant to their product.

To describe and manage USB communication, the following concepts are important:

- Bus Topology: Section 5.2 presents the primary physical and logical components of USB and how they interrelate.
- Communication Flow Models: Sections 5.3 through 5.8 describe how communication flows between the host and devices through the USB and defines the four USB transfer types.
- Bus Access Management: Section 5.9 describes how bus access is managed within the host to support a broad range of communication flows by USB devices.
- Special Consideration for Isochronous Transfers: Section 5.10 presents features of USB specific to devices requiring isochronous data transfers. Device implementers for non-isochronous devices will not need to read Section 5.10.

5.2 Bus Topology

There are four main parts to USB topology:

- Host and Devices: The primary components of a USB system.
- Physical Topology: How USB elements are connected.
- Logical Topology: The roles and responsibilities of the various USB elements and how the USB appears from the perspective of the host and a device.
- Client software to function relationships: How client software and its related function interfaces on a USB device view each other.

5.2.1 USB Host

The host's logical composition as shown in Figure 5-3 is:

- The USB host controller
- The aggregate USB system software (USB driver, host controller driver, and host software)
- The client

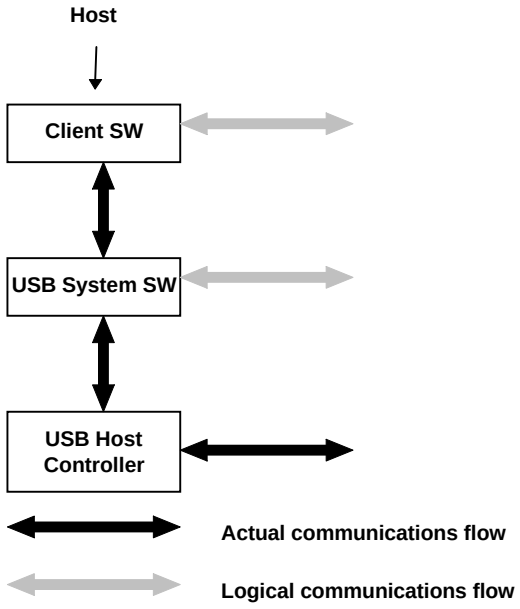


Figure 5-3. Host Composition

The USB host occupies a unique position as the coordinating entity for the USB. In addition to its special physical position, the host has specific responsibilities with regard to the USB and its attached devices. The host controls all access to the USB. A USB device only gains access to the bus by being granted access by the host. The host is also responsible for monitoring the topology of the USB.

For a complete discussion of the host and its duties, refer to Chapter 10, USB Host: Software and Hardware.

5.2.2 USB Devices

A USB physical device's logical composition as shown in Figure 5-4 is:

- USB bus interface
- USB logical device
- Function

USB physical devices provide additional functionality to the host. The types of functionality provided by USB devices vary widely. However, all USB logical devices present the same basic interface to the host. This allows the host to manage the USB-relevant aspects of different USB devices in the same manner.

To assist the host in identifying and configuring USB devices, each device carries and reports configuration related information. Some of the information reported is common among all logical devices. Other information is specific to the functionality provided by the device. The detailed format of this information varies depending on the device class of the device.

For a complete discussion of USB devices, refer to Chapter 9, USB Device Framework.

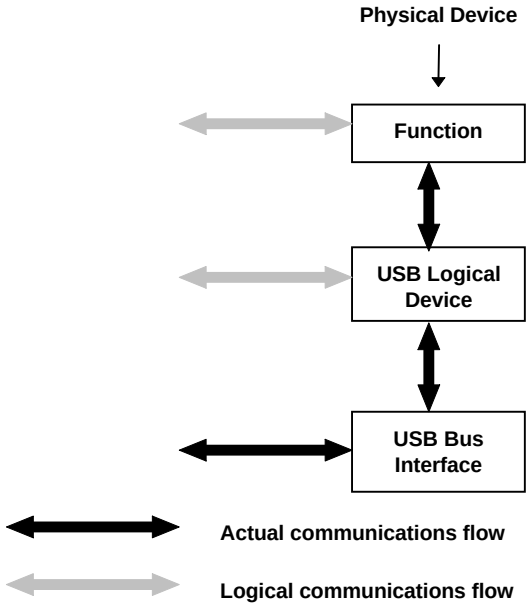


Figure 5-4. Physical Device Composition

5.2.3 Physical Bus Topology

Devices on the USB are physically connected to the host via a tiered star topology, as illustrated in Figure 5-5. 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 Figure 5-5.

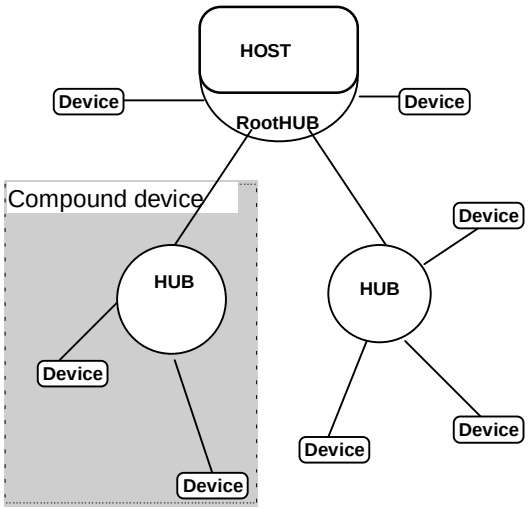


Figure 5-5. USB Physical Bus Topology

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. Figure 5-5 also illustrates a compound device.

5.2.4 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 Figure 5-6 that corresponds to the physical topology shown in Figure 5-5. Hubs are logical devices also, but are not shown in Figure 5-6 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. A more complete discussion of hubs can be found in Chapter 11, Hub Specification.

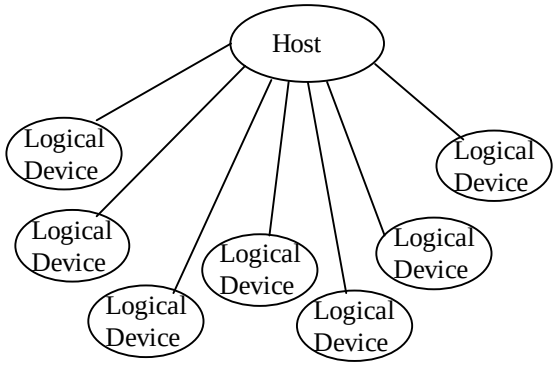


Figure 5-6. USB Logical Bus Topology

5.2.5 Client Software to Function Relationship

Even though the physical and logical topology of the USB reflects the shared nature of the bus, client software(CSw) manipulating a USB function interface is presented with the view that it deals only with its interface(s) of interest. Client software for USB functions must use USB software programming interfaces to manipulate their functions as opposed to directly manipulating their functions via memory or I/O accesses as with other buses (e.g., PCI, EISA, PCMCIA, etc.). During operation, client software should be independent of other devices that may be connected to USB. This allows the designer of the device and client software to focus on the hardware/software interaction design details. Figure 5-7 illustrates a device designer's perspective of the relationships of client software and USB functions with respect to the USB logical topology of Figure 5-6.

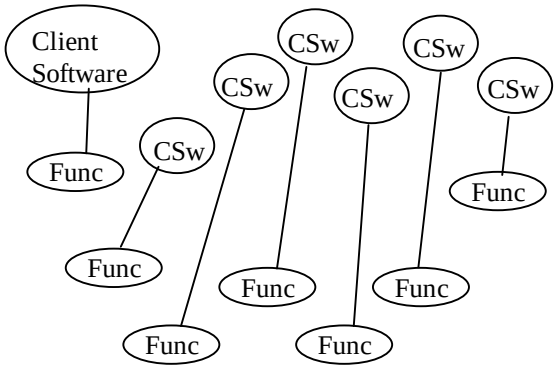


Figure 5-7. Client Software to Function Relationships

5.3 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.

The diagram in Figure 5-8 shows a more detailed view of Figure 5-2. The complete definition of the actual communication flows of Figure 5-2 supports the logical device and function layer communication flows. These actual communication flows cross several interface boundaries. Chapters 6, 7, and 8 describe the mechanical, electrical, and protocol interface definitions of the USB "wire." Chapter 9 describes the USB device programming interface that allows a USB device to be manipulated from the host side of the wire. Chapter 10 describes two host side software interfaces:

- Host Controller Driver (HCD) - the software interface between the USB host controller and USB system software. This interface allows a range of host controller implementations without requiring all host software to be dependent on any particular implementation. One USB Driver can support different host controllers without requiring specific knowledge of a host controller implementation. A host controller implementer provides an HCD implementation that supports the host controller.
- USB Driver (USBD) - the interface between USB system software and the client software. This interface provides clients with convenient functions for manipulating USB devices.

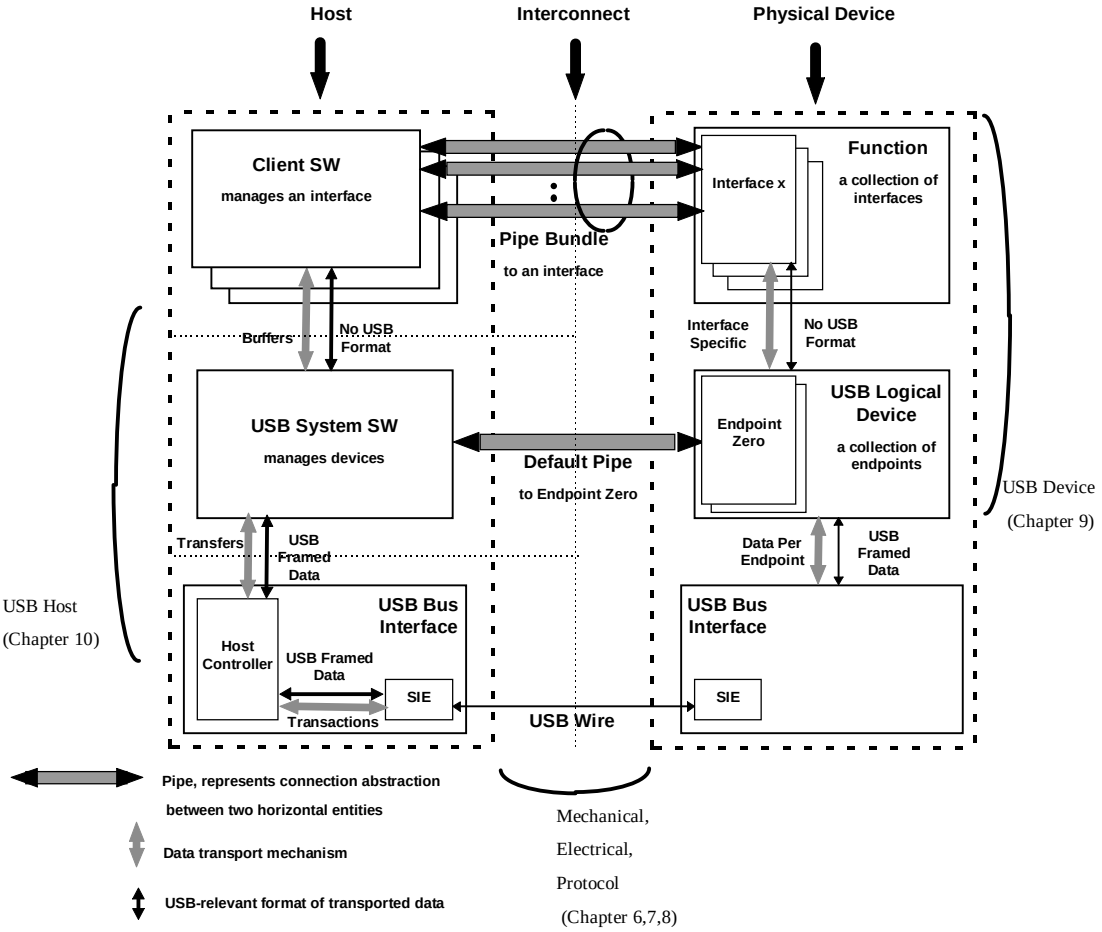


Figure 5-8. USB Host/Device Detailed View

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.

Figure 5-9 illustrates how communication flows are carried over pipes between endpoints and host side memory buffers. The following sections describe endpoints, pipes, and communication flows in more detail.

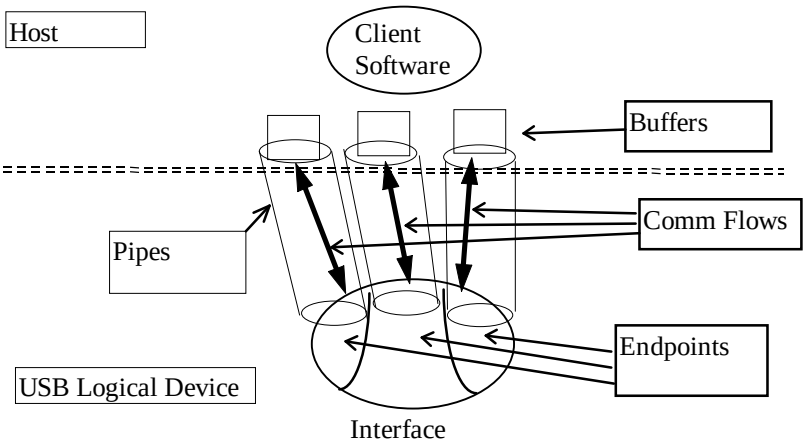


Figure 5-9. USB Communication Flow

Software on the host communicates with a logical device via a set of communication flows. The set of communication flows are selected by the device software/hardware designer(s) to efficiently match the communication requirements of the device to the transfer characteristics provided by USB.

5.3.1 Device Endpoints

An endpoint is a uniquely identifiable portion of a USB device that is the terminus of a communication flow between the host and device. Each USB logical device is composed of a collection of independently operating endpoints. Software may only communicate with a USB device via one or more endpoints. Each logical device has a unique address assigned by the system at device attachment time. Each endpoint on a device has a device (design time) determined unique identifier, the endpoint number. The combination of the device address and the endpoint number allows each endpoint to be uniquely referenced.

An endpoint has characteristics that determine the type of transfer service required between the endpoint and the client software. Endpoints describe themselves by:

- Their bus access frequency/latency requirements
- Their bandwidth requirements
- Their endpoint number
- The error handling behavior requirements
- Maximum packet size that the endpoint is capable of sending or receiving
- The transfer type for the endpoint (refer to Section 5.4 for details)
- For bulk and isochronous transfer types, the direction data is transferred between the endpoint and the host

Endpoints are in an unknown state before being configured. Endpoints must not be accessed by the host before being configured.

5.3.1.1 Endpoint 0 Requirements

All USB devices are required to have an endpoint with endpoint number 0 that is used to initialize and generically manipulate the logical device (e.g., to configure the logical device). Endpoint 0 provides access to the device's configuration information and allows generic USB status and control access. Endpoint 0 supports control transfers as defined in Section 5.5. Endpoint 0 is always configured once a device is attached and powered.

5.3.1.2 Non-endpoint 0 Requirements

Functions can have additional endpoints as required for their implementation. Low speed functions are limited to two optional endpoints beyond the required Endpoint 0. Full speed devices can have additional endpoints only limited by the protocol definition; i.e., a maximum of 16 input endpoints and 16 output endpoints.

An endpoint cannot be used until it is configured. Endpoints, besides Endpoint 0, are configured as a normal part of the device configuration process (refer to Chapter 9).

5.3.2 Pipes

A USB pipe is an association between an endpoint on a device and software on the host. Pipes represent the ability to move data between software on the host via a memory buffer and an endpoint on a device. There are two different, mutually exclusive, pipe communication modes:

- Stream. Data moving through a pipe has no USB defined structure.
- Message. Data moving through a pipe has some USB defined structure.

USB does not interpret the content of data it delivers through a pipe. Even though a message pipe requires that data be structured according to USB definitions, the content of the data is not interpreted by USB.

Additionally, pipes have associated with them:

- A claim on USB bus access and bandwidth usage.
- A transfer type.
- The associated endpoint's characteristics such as directionality and maximum data payload sizes. The data payload is the data that is carried in the data field of a data packet within a bus transaction (as defined in Chapter 8).

Pipes come into existence when a USB device is configured. Since Endpoint 0 is always configured once a device is powered, there is always a pipe for Endpoint 0. This pipe is called the Default Pipe. This pipe is used by system software to determine device identification and configuration requirements, and to configure the device. The Default Pipe can also be used by device specific software after the device is configured. USB system software retains "ownership" of the Default Pipe and mediates use of the pipe by other client software.

A software client normally requests data transfers via I/O Request Packets (IRPs) to a pipe and then either waits or is notified when they are completed. Details about IRPs are defined in an operating system specific manner. This specification uses the term to simply refer to an identifiable request by a software client to move data between itself (on the host) and an endpoint of a device in an appropriate direction. A software client can cause a pipe to return all outstanding IRPs if it desires. The software client is notified that an IRP has completed when the bus transactions associated with it have completed either successfully or due to errors.

If there are no IRPs pending or in progress for a pipe, the pipe is idle and the host controller will take no action with regard to the pipe; i.e., the endpoint for such a pipe will not see any bus transactions directed to it. The only time bus activity is present for a pipe is when IRPs are pending for that pipe.

If a non-isochronous pipe encounters a STALL condition (refer to Chapter 8) or three bus errors are encountered on any packet of an IRP, the IRP is aborted/retired, all outstanding IRPs are also retired, and no further IRPs are accepted until the software client recovers from the condition (in an implementation dependent way) and acknowledges the STALL or error condition via a USBD call. An appropriate status informs the software client of the specific IRP result for error versus STALL (refer to Chapter 10). Isochronous pipe behavior is described in Section 5.6.

An IRP may require multiple data payloads to move the client data over the bus. The data payloads for such a multiple data payload IRP are expected to be maximum packet sized until the last data payload that contains the remainder of the overall IRP. See each transfer type specific description for more details. For such an IRP, short packets (i.e., less than maximum sized data payloads) on input that do not completely fill an IRP data buffer can have one of two possible meanings depending upon the expectations of a client.

- A client can expect a variable sized amount of data in an IRP. In this case, a short packet that does not fill an IRP data buffer can be used simply as an inband delimiter to indicate “end of unit of data.” The IRP should be retired without error and the host controller should advance to the next IRP.
- A client can expect a specific sized amount of data. In this case, a short packet that does not fill an IRP data buffer is an indication of an error. The IRP should be retired, the pipe should be stalled, and any pending IRPs associated with the pipe should also be retired.

Since the host controller must behave differently in the two cases and cannot know on its own which way to behave for a given IRP, it is possible to indicate per IRP which behavior the client desires.

An endpoint can inform the host that it is busy by responding with a NAK. NAKs are not used as a retire condition for returning an IRP to a software client. Any number of NAKs can be encountered during the processing of a given IRP. A NAK response to a transaction does not constitute an error and is not counted as one of the three errors described above.

5.3.2.1 Stream Pipes

Stream pipes deliver data in the data packet portion of bus transactions with no USB required structure on the data content. Data flows in at one end of a stream pipe and out the other end in the same order. Stream pipes are always unidirectional in their communication flow.

Data flowing through a stream pipe is expected to interact from what USB believes is a single client. USB System software is not required to provide synchronization between multiple clients that may be using the same stream pipe. Data presented to a stream pipe is moved through the pipe in sequential order: first-in, first-out.

A stream pipe to a device is bound to a single device endpoint number in the appropriate direction (i.e., corresponding to an IN or OUT token as defined by the protocol layer). The device endpoint number for the opposite direction can be used for some other stream pipe to the device.

Stream pipes support bulk, isochronous, and interrupt transfer types explained below.

5.3.2.2 Message Pipes

Message pipes interact with the endpoint in a different manner than stream pipes. First, a request is sent to the USB device from the host. This request is followed by data transfer(s) in the appropriate direction. Finally, a status stage follows at some later time by a response from the endpoint. In order to accommodate the request/data/status paradigm, message pipes impose a structure on the communication flow which allows commands to be reliably identified and communicated. Message pipes allow

communication flow in both directions although the communication flow may be predominately one way. The pipe for Endpoint 0, the Default Pipe, is always a message pipe.

USB system software ensures that multiple requests are not sent to an endpoint concurrently. An endpoint is only required to service a single message request at a time per endpoint. Multiple software clients on the host can make requests via the Default Pipe, but they are sent to the endpoint in a first-in, first-out order. An endpoint can control the flow of information during the data and status stages based on its ability to respond to the host transactions (refer to Chapter 8 for more details).

An endpoint will not normally be sent the next message from the host until the current message's processing at the endpoint has been completed. However, there are error conditions whereby a message transfer can be aborted by the host and the endpoint can be sent a new message transfer prematurely (from its perspective). From the perspective of the software manipulating a message pipe, an error on some part of an IRP retires the current IRP and all queued IRPs. The software client that requested the IRP is notified of the IRP completion with an appropriate error indication.

A message pipe to a device requires a single device endpoint number in both directions (IN and OUT tokens). USB does not allow a message pipe to be associated with different endpoint numbers for each direction.

Message pipes support the control transfer type explained below.

5.4 Transfer Types

USB transports data through a pipe between a memory buffer associated with a software client on the host and an endpoint on the USB device. Data transported by message pipes is carried in a USB defined structure, but USB allows device specific structured data to be transported within the USB defined message data payload. USB also defines that data moved over the bus is packetized for any pipe (stream or message), but ultimately the formatting and interpretation of the data transported in the data payload of a bus transaction is the responsibility of the client software and function using the pipe. However, USB provides different transfer types that are optimized to more closely match the service requirements of the client software and function using the pipe. An IRP uses one or more bus transactions to move information between a software client and its function.

Each transfer type determines various characteristics of the communication flow including:

- Data format imposed by USB
- Direction of communication flow
- Packet size constraints
- Bus access constraints
- Required data sequences

The designers of a USB device choose the capabilities for the device's endpoints. When a pipe is established for an endpoint, most of the pipe's transfer characteristics are determined and remain fixed for the lifetime of the pipe. Transfer characteristics that can be modified are described for each transfer type.

USB defines four transfer types:

- Control Transfers - Bursty, non-periodic, host software initiated request/response communication typically used for command/status operations.
- Isochronous Transfers - Periodic, continuous communication between host and device typically used for time relevant information. This transfer type also preserves the concept of time encapsulated in the data. This does not imply, however, that the delivery needs of such data is always time-critical.
- Interrupt Transfers - Small data, non-periodic, low frequency, bounded latency, device initiated communication typically used to notify the host of device service needs.
- Bulk Transfers - Non-periodic, large bursty communication typically used for data that can use any available bandwidth and also be delayed until bandwidth is available.

Each transfer type is described in detail in the following four major sections. The data for any IRP is carried by the data field of the data packet as described in Section 8.4.3. Chapter 8 also describes details of the protocol that are affected by use of each particular transfer type.

5.5 Control Transfers

Control transfers allow access to different parts of a device. Control transfers are intended to support configuration/command/status type communication flows between client software and its function. A control transfer is composed of a setup bus transaction moving request information from host to function, zero or more data transactions sending data in the direction indicated by the setup transaction, and a status transaction returning status information from function to host. The status transaction returns "success" when the endpoint has successfully completed processing the requested operation. Section 8.5.2 describes the details of what packets, bus transactions, and transaction sequences are used to accomplish a control transfer. Chapter 9 describes the details of the defined USB command codes.

Each USB device is required to implement Endpoint 0 with a control transfer type. This endpoint is used by the USB system software as a control pipe. Control pipes provide access to the USB device's configuration, status, and control information. A function can provide endpoints for additional control pipes for its own implementation needs.

The USB device framework (refer to Chapter 9) defines standard, device class, or vendor specific requests that can be used to manipulate a device's state. Descriptors are also defined that can be used to contain different information on the device. Control transfers provide the transport mechanism to access device descriptors and make requests of a device to manipulate its behavior.

Control transfers are only carried through message pipes. Consequently, data flows using control transfers must adhere to USB data structure definitions as described in Section 5.5.1.

USB subsystem will make a "best effort" to support delivery of control transfers between the host and devices. A function and its client software cannot request specific bus access frequency or bandwidth for control transfers. USB system software may restrict the bus access and bandwidth that a device may desire for control transfers. These restrictions are defined in Section 5.5.3 and Section 5.5.4.

5.5.1 Data Format

The setup packet has a USB defined structure that accommodates the minimum set of commands required to enable communication between the host and a device. The structure definition allows vendor specific extensions for device specific commands. The data transactions following setup have no USB defined structure. The status transaction also has a USB defined structure. Specific control transfer setup/data definitions are described in Section 8.5.2 and Chapter 9.

5.5.2 Direction

Control transfers are supported via bi-directional communication flow over message pipes.

5.5.3 Packet Size Constraints

An endpoint for control transfers specifies the maximum data payload size that the endpoint can accept from or transmit to the bus. USB defines the allowable maximum control data payload sizes for full speed devices to be only 8, 16, 32, or 64 bytes. Low speed devices are limited to only an 8 byte maximum data payload size. This maximum applies to the data payloads of the data packets following a setup; i.e., the size specified is for the data field of the packet as defined in Chapter 8, not including other protocol required information. A setup packet is always 8 bytes. A control endpoint always uses its MaxPacketSize for data payloads.

All control endpoints are required to support a control data payload maximum size of 8 bytes after reset. An endpoint can be designed to support a larger maximum data payload size. Such an endpoint reports in its configuration information the value for its maximum data payload size. USB does not require that data payloads transmitted be exactly the maximum size; i.e., if a data payload is less than the maximum, it does not need to be padded to the maximum size.

All host controllers are required to have support for 8, 16, 32, and 64 byte maximum data payload sizes for full speed control endpoints and only 8 byte maximum data payload sizes for low speed control endpoints. No host controller is required to support larger or smaller maximum data payload sizes.

During configuration, USB system software reads the endpoint's maximum data payload size and ensures that no data payload will be sent to the endpoint that is larger than the supported size. The host will always use a maximum data payload size of at least 8 bytes.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's *MaxPacketSize* (refer to Chapter 9). When a control transfer involves more data than can fit in one data payload of the currently established maximum size, all data payloads are required to be maximum sized except for the last data payload which will contain the remaining data. If an endpoint wants to transmit less data than expected by the client software, a premature, less than maximum sized data payload will be received by the host controller. This premature, less than maximum sized data payload causes the host controller to advance to the status transaction instead of continuing on with another data transaction or else stall the pipe as was outlined in Section 5.3.2. If a data payload is received that is larger than that expected, the IRP for the control transfer will be aborted/retired and the pipe will stall future IRPs until the condition is corrected and acknowledged.

5.5.4 Bus Access Constraints

Control transfers can be used by full speed and low speed USB devices.

An endpoint has no way to indicate a desired bus access frequency for a control pipe. USB balances the bus access requirements of all control pipes and the specific IRPs that are pending to provide "best effort" delivery of data between client software and functions.

USB requires that part of each frame be reserved to be available for use by control transfers as follows:

- If the control transfers that are attempted (in an implementation dependent fashion) consume less than 10% of the frame time, the remaining time can be used to support bulk transfers (refer to Section 5.8).
- A control transfer that has been attempted and needs to be retried can be retried in the current or a future frame; i.e., it is not required to be retried in the same frame.
- If there are more control transfers than reserved time, but there is additional frame time that is not being used for isochronous or interrupt transfers, a host controller may move additional control transfers as they are available.
- If there are too many pending control transfers than available frame time, control transfers are selected to be moved over the bus as appropriate.
- If there are control transfers pending for multiple endpoints, control transfers for the different endpoints are selected according to a fair access policy that is host controller implementation dependent.
- A transaction of a control transfer that is frequently being retried should not be expected to consume an unfair share of the frame time.

These requirements allow control transfers between host and devices to be regularly moved over the bus with "best effort."

All control transfers pending in a system contend for the same available bus time. Because of this, the bus time made available for control transfers to a particular endpoint can be varied by USB system software at its discretion. An endpoint and its client software cannot assume a specific rate of service for control transfers. Bus time made available to a software client and its endpoint can be changed as other devices are inserted into and removed from the system or also as control transfers are requested for other device endpoints.

The bus frequency and frame timing limit the maximum number of successful control transfers within a frame for any USB system to less than 29 full speed 8 byte data payloads or less than four low speed 8 byte data payloads. Table 5-1 lists information about different sized full speed control transfers and the maximum number of transfers possible in a frame. This table was generated assuming zero length status data stage transaction and one data stage transaction. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload sizes.

Table 5-1. Full Speed Control Transfer Limits

protocol overhead (bytes)						
45	(9-syncs, 9-pids, 6-EP+CRC,6-CRC,8-Setup data					
	7-byte interpacket delay (EOP, etc.))					
data	max Bandwidth	Frame BW	max	Bytes	Bytes/frame	
payload	bytes/sec	per transfer	transfers	Remaining	useful data	
1	32000	3%	32	28	32	
2	62000	3%	31	43	62	
4	120000	3%	30	30	120	
8	224000	4%	28	16	224	
16	384000	4%	24	36	384	
32	608000	5%	19	37	608	
64	832000	7%	13	83	832	
max	1500000				1500	

The 10% frame reservation for control transfers means that in a system with bus time fully allocated, all full speed control transfers in the system contend for a nominal three control transfers per frame. Since the USB subsystem uses control transfers for configuration purposes in addition to whatever other control transfers other client software may be requesting, a given software client and its function should not expect to be able to make use of this full bandwidth for its own control purposes. Host controllers are also free to determine how the individual bus transactions for specific control transfers are moved over the bus within and across frames. An endpoint could see all bus transactions for a control transfer within the same frame or spread across several discontiguous frames. Finally, a host controller, for various implementation reasons, may not be able to provide the theoretical maximum number of control transfers per frame.

Both full speed and low speed control transfers contend for the same available frame time. Low speed control transfers simply take longer to transfer. Table 5-2 lists information about different sized low speed packets and the maximum number of packets possible in a frame. Also for both speeds, since a control transfer is composed of several packets, the packets can be spread over several frames to spread the bus time required across several frames.

Table 5-2. Low Speed Control Transfer Limits

protocol overhead (bytes)						
46						
data	max Bandwidth	Frame BW	max	Bytes	Bytes/frame	
payload	(approx)	per transfer	transfers	Remaining	useful data	
1	3000	25%	3	46	3	
2	6000	26%	3	43	6	
4	12000	27%	3	37	12	
8	24000	29%	3	25	24	
max	187500				187	

5.5.5 Data Sequences

Control transfers require that a setup bus transaction be sent from the host to a device to describe the type of control access that the device should perform. The setup transaction is followed by zero or more control data transactions that carry the specific information for the requested access. Finally, a status transaction completes the control transfer and allows the endpoint to return the status of the control transfer to the client software. After the status transaction for a control transfer is completed, the host can advance to the next control transfer for the endpoint. As described in Section 5.5.4, this next control transfer will be moved over the bus at some host controller implementation defined time in the future.

The endpoint can be busy for a device specific numbers of frames during the data and status transactions of the control transfer. During these times when the endpoint indicates it is busy (refer to Chapter 8 and Chapter 9 for details), the host will retry the transaction at a later time.

If a setup transaction is received by an endpoint before a previously initiated control transfer is completed, the device must abort the current transfer/operation and handle the new control setup transaction. A setup transaction should not normally be sent before the completion of a previous control transfer. However, if a transfer is aborted, for example, due to errors on the bus, the host can send the next setup transaction prematurely from the endpoint's perspective.

After a STALL condition is encountered or an error is detected by the host, a control endpoint is allowed to recover by accepting the next setup PID; i.e., recovery actions via some other pipe are not required for control endpoints, but may be required by implementation for some. For the Default Pipe (Endpoint 0), a device reset (by USBD) will ultimately be required to clear the STALL or error condition if the next setup PID is not accepted.

USB provides robust error detection, recovery/retransmission for errors that occur during control transfers. Transmitters and receivers can remain synchronized with regard to where they are in a control transfer and recover with minimum effort. Retransmission of data and status packets can be detected by a receiver via data retry indicators in the packet. A transmitter can reliably determine that its corresponding receiver has successfully accepted a transmitted packet by information returned in a handshake to the packet. The protocol allows for distinguishing a retransmitted packet from its original packet except for a control setup packet. Setup packets may be retransmitted due to a transmission error; however, setup packets cannot indicate that a packet is an original or a retried transmission.

5.6 Isochronous Transfers

In non-USB environments, isochronous transfers have the general implication of constant-rate, error-tolerant transfers. In the USB environment, requesting an isochronous transfer type provides the requester with the following:

- Guaranteed access to USB bandwidth with bounded latency
- As long as data is provided to the pipe, a constant data rate through the pipe is guaranteed
- In the case of a delivery failure due to error, no retrying of the attempt to deliver the data

While the USB isochronous transfer type is designed to support isochronous sources and destinations, it is not required that software using this transfer type actually be isochronous in order to use the transfer type. Section 5.10 presents more detail on special considerations for handling isochronous data on USB.

5.6.1 Data Format

USB imposes no data content structure on communication flows for isochronous pipes.

5.6.2 Direction

An isochronous pipe is a stream pipe and is, therefore, always unidirectional. An endpoint description identifies whether a given isochronous pipe's communication flow is into or out of the host. If a device requires bi-directional isochronous communication flow, two isochronous pipes must be used, one in each direction.

5.6.3 Packet Size Constraints

An endpoint in a given configuration for an isochronous pipe specifies the maximum size data payload that it can transmit/receive. USB system software uses this information during configuration to ensure that there is sufficient bus time to accommodate this maximum data payload in each frame. If there is sufficient bus time for the maximum data payload, the configuration is established; if not, the configuration is not established. USB system software does not adjust the maximum data payload size for an isochronous pipe as was the case for a control pipe. An isochronous pipe can simply either be supported or not supported in a given USB subsystem configuration.

USB limits the maximum data payload size to 1023 bytes for each isochronous pipe. Table 5-3 lists information about different sized isochronous transactions and the maximum number of transactions possible in a frame.

Table 5-3. Isochronous Transaction Limits

protocol overhead (bytes)						
9	(2-syncs, 2-pids, 2-EP+CRC, 2-CRC,					
	1 byte interpacket delay)					
data	max Bandwidth	Frame BW	max	Bytes	Bytes/frame	
payload		per trans.	trans.	Remaining	useful data	
1	150000	1%	150	0	150	
2	272000	1%	136	4	272	
4	460000	1%	115	5	460	
8	704000	1%	88	4	704	
16	960000	2%	60	0	960	
32	1152000	3%	36	24	1152	
64	1280000	5%	20	40	1280	
128	1280000	9%	10	130	1280	
256	1280000	18%	5	175	1280	
512	1024000	35%	2	458	1024	
1023	1023000	69%	1	468	1023	
max	1500000				1500	

Any given transaction for a isochronous pipe need not be exactly the maximum size specified for the endpoint. The size of a data payload is determined by the transmitter (client software or function) and can vary as required from transaction to transaction. An endpoint can use the optional USB standard sample header to indicate where in the sample stream this packet starts. This allows the receiver to recover from packets lost due to errors. USB ensures that whatever size is presented to the host controller is delivered on the bus. The actual size of a data payload is determined by the data transmitter and may be less than the prenegotiated maximum size. Bus errors can change the actual size seen by the receiver. However, these errors can be detected by either CRC on the data or knowledge the receiver has about the expected size for any transaction.

5.6.4 Bus Access Constraints

Isochronous transfers can only be used by full speed devices.

USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers.

An endpoint for an isochronous pipe does not include information about bus access frequency. All isochronous pipes normally move exactly one data packet each frame (i.e., every 1 ms). Errors on the bus or delays in operating system scheduling of client software can result in no packet being transferred for a frame. An error indication is returned as status to the client software in such a case. A device can also detect this situation by tracking SOF tokens and noticing two SOF tokens without an intervening data packet for an isochronous endpoint.

The bus frequency and frame timing limit the maximum number of successful isochronous transactions within a frame for any USB system to less than 151 full speed 1 byte data payloads. Finally, a host controller, for various implementation reasons, may not be able to provide the theoretical maximum number of isochronous transactions per frame.

5.6.5 Data Sequences

Isochronous transfers do not support data retransmission in response to errors on the bus. A receiver can determine that a transmission error occurred. The low level USB protocol does not allow handshakes to be returned to the transmitter of an isochronous pipe. Normally handshakes would be returned to tell the transmitter whether a packet was successfully received or not. For isochronous transfers, timeliness is more important than correctness/retransmission, and given the low error rates expected on the bus, the protocol is optimized assuming transfers normally succeed. Isochronous receivers can determine whether they missed data during a frame. Also, a receiver can determine how much data was lost. Section 5.10 describes these USB mechanisms in more detail.

An endpoint for isochronous transfers never stalls since there is no handshake to report a STALL condition. The host and client software can never encounter this case. Errors are reported as status associated with the IRP for an isochronous transfer, but the isochronous pipe is not stalled in an error case. If an error is detected, the host continues to process the data associated with the next frame of the transfer. Limited error detection is possible since the protocol for isochronous transactions does not allow per transaction handshakes.

5.7 Interrupt Transfers

The interrupt transfer type is designed to support those devices that need to communicate small amounts of data infrequently, but with bounded service periods. Requesting a pipe with an interrupt transfer type provides the requester with the following:

- Guaranteed maximum service period for the pipe
- Retry of transfer attempts at the next period, in the case of occasional delivery failure due to error on the bus

5.7.1 Data Format

USB imposes no data content structure on communication flows for interrupt pipes.

5.7.2 Direction

An interrupt pipe is a stream pipe and is therefore always unidirectional. Further, an interrupt pipe is only input to the host. Output interrupt pipes are not supported by USB.

5.7.3 Packet Size Constraints

An endpoint for an interrupt pipe specifies the maximum size data payload that it will transmit. The maximum allowable interrupt data payload size is 64 bytes or less for full speed. Low speed devices are limited to 8 bytes or less maximum data payload size. This maximum applies to the data payloads of the data packets; i.e., the size specified is for the data field of the packet as defined in Chapter 8, not including other protocol required information. USB does not require that data packets be exactly the maximum size; i.e., if a data packet is less than the maximum, it does not need to be padded to the maximum size.

All host controllers are required to have support for up to 64 byte maximum data payload sizes for full speed interrupt endpoints and 8 bytes or less maximum data payload sizes for low speed interrupt endpoints. No host controller is required to support larger maximum data payload sizes.

USB system software determines the maximum data payload size that will be used for a interrupt pipe during device configuration. This size remains constant for the lifetime of a device's configuration. USB software uses the maximum data payload size determined during configuration to ensure that there is sufficient bus time to accommodate this maximum data payload in its assigned period. If there is

sufficient bus time, the pipe is established; if not, the pipe is not established. USB software does not adjust the bus time made available to an interrupt pipe as was the case for a control pipe. An interrupt pipe can simply either be supported or not in a given USB subsystem configuration. However, the actual size of a data payload is still determined by the data transmitter and may be less than the maximum size.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's *MaxPacketSize*. A device can move data via an interrupt pipe that is larger than *MaxPacketSize*. A software client can accept this data via an IRP for the interrupt transfer that requires multiple bus transactions without requiring an IRP complete notification per transaction. This can be achieved by specifying a buffer that can hold the desired data size. The size of the buffer is a multiple of *MaxPacketSize* with some remainder. The endpoint must transfer each transaction except the last as *MaxPacketSize* and the last transaction is the remainder. The multiple data transactions are moved over the bus at the period established for the pipe.

When an interrupt transfer involves more data than can fit in one data payload of the currently established maximum size, all data payloads are required to be maximum sized except for the last data payload which will contain the remaining data. If an endpoint wants to transmit less data than expected by the client software, a premature, less than maximum sized data payload will be received by the host controller. This premature, less than maximum sized data payload causes the host controller to retire the current IRP and advance to the next IRP or else stall the pipe as outlined in Section 5.3.2. If a data payload is received that is larger than that expected, the interrupt IRP will be aborted/retired and the pipe will stall future IRPs until the condition is corrected and acknowledged.

5.7.4 Bus Access Constraints

Interrupt transfers can be used by full speed and low speed devices.

USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers.

The bus frequency and frame timing limit the maximum number of successful interrupt transactions within a frame for any USB system to less than 108 full speed 1 byte data payloads or 14 low speed 1 byte data payloads. Finally, a host controller, for various implementation reasons, may not be able to provide the above maximum number of interrupt transactions per frame.

Table 5-4 lists information about different sized full speed interrupt transactions and the maximum number of transactions possible in a frame. Table 5-5 lists similar information for low speed interrupt transactions.

Table 5-4. Full Speed Interrupt Transaction Limits

protocol overhead (bytes)						
13	(3-syncs, 3-pids, 2-EP+CRC, 2-CRC,					
	3 byte interpacket delay)					
data	max Bandwidth	Frame BW	max	Bytes	Bytes/frame	
payload		per trans.	trans.	Remaining	useful data	
1	107000	1%	107	2	107	
2	200000	1%	100	0	200	
4	352000	1%	88	4	352	
8	568000	1%	71	9	568	
16	816000	2%	51	21	816	
32	1056000	3%	33	15	1056	
64	1216000	5%	19	37	1216	
max	1500000				1500	

An endpoint for an interrupt pipe specifies its desired bus access period. A full speed endpoint can specify a desired period from 1 ms to 255 ms. Low speed endpoints are limited to only specifying 10 ms to 255 ms. USB software will use this information during configuration to determine a period that

can be sustained. The period provided by the system may be shorter than that desired by the device up to the shortest period defined by USB. The client software and device can only depend on the fact that the host will ensure that the time duration between two error free transactions (or two transaction attempts) with the endpoint will be no longer than the desired period. Note that errors on the bus can prevent an interrupt transaction from being successfully delivered over the bus and consequently exceed the desired period. The period between any two transaction attempts can also vary over time; although it will never exceed the desired period in error free cases. Also, the endpoint is only polled when the software client has an IRP for an interrupt transfer pending. If the bus time for performing an interrupt transfer arrives and there is no IRP pending, the endpoint will not be given an opportunity to transfer data at that time. Once an IRP is requested, its data will be transferred at the next allocated period.

Table 5-5. Low Speed Interrupt Transaction Limits

protocol overhead (bytes)						
13						
data	max Bandwidth	Frame BW	max	Bytes	Bytes/frame	
payload	(approx)	per trans.	trans.	Remaining	useful data	
1	13000	7%	13	5	13	
2	24000	8%	12	7	24	
4	44000	9%	11	0	44	
8	64000	11%	8	19	64	
max	187500				187	

Interrupt transfers are moved over the USB by accessing an interrupt endpoint every period. The host has no way to determine whether an endpoint will source an interrupt without accessing the endpoint and requesting an interrupt transfer. If the endpoint has no interrupt data to transmit when accessed by the host, it responds with a NAK. An endpoint should only provide interrupt data when it has an interrupt pending to avoid having a software client erroneously notified of IRP complete. A zero length data payload is a valid transfer and may be useful for some implementations.

5.7.5 Data Sequences

Interrupt transactions may use either alternating data toggle bits such that the bits are toggled only upon successful transfer completion or a continuously toggling of data toggle bits. The host in any case must assume that the device is obeying full handshake/retry rules as defined in Chapter 8. A device may choose to always toggle DATA0/DATA1 PIDs so that it can ignore handshakes from the host. However, in this case, the client software can miss some data packets when an error occurs because the host controller interprets the next packet as a retry of a missed packet.

If a stall condition is detected on an interrupt pipe due to transmission errors or a STALL handshake being returned from the endpoint, all pending IRPs are retired. Removal of the STALL condition is achieved via software intervention through a separate control pipe. This recovery must also reset the data toggle bit to DATA0 for the endpoint. The software client must also call a USBD Function to reset the host's data toggle to DATA0, acknowledge, and clear the stall condition on the host.

Interrupt transactions are retried due to errors detected on the bus that affect a given transfer.

5.8 Bulk Transfers

The bulk transfer type is designed to support devices that need to communicate relatively large amounts of data at highly variable times where the transfer can be deferred until bandwidth is available. Requesting a pipe with a bulk transfer type provides the requester with the following:

- Access to the USB on a bandwidth available basis
- Retry of transfers, in the case of occasional delivery failure due to error on the bus
- Guaranteed delivery of data, but no guarantees of bandwidth or latency

Bulk transfers occur only on a bandwidth available basis. For a USB with large amounts of free bandwidth, bulk transfers may happen relatively quickly; while for a USB with little bandwidth available, bulk transfers may trickle out over a relatively long period of time.

5.8.1 Data Format

USB imposes no data content structure on communication flows for bulk pipes.

5.8.2 Direction

A bulk pipe is a stream pipe and, therefore, always has communication flowing either into or out of the host for a given pipe. If a device requires bi-directional bulk communication flow, two bulk pipes must be used, one in each direction.

5.8.3 Packet Size Constraints

An endpoint for bulk transfers specifies the maximum data payload size that the endpoint can accept from or transmit to the bus. USB defines the allowable maximum bulk data payload sizes to be only 8, 16, 32, or 64 bytes. This maximum applies to the data payloads of the data packets; i.e.; the size specified is for the data field of the packet as defined in Chapter 8, not including other protocol required information.

A bulk endpoint is designed to support a maximum data payload size. A bulk endpoint reports in its configuration information the value for its maximum data payload size. USB does not require that data payloads be transmitted exactly the maximum size; i.e., if a data payload is less than the maximum, it does not need to be padded to the maximum size.

All host controllers are required to have support for 8, 16, 32, and 64 byte maximum packet sizes for bulk endpoints. No host controller is required to support larger or smaller maximum packet sizes.

During configuration, USB system software reads the endpoint's maximum data payload size and ensures that no data payload will be sent to the endpoint that is larger than the supported size.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's reported *MaxPacketSize*. When a bulk IRP involves more data than can fit in one maximum sized data payload, all data payloads are required to be maximum size except for the last data payload which will contain the remaining data. If an endpoint transmits less data than expected by the client software, a premature, less than maximum sized data payload will be received by the host controller. This premature, less than maximum sized data payload causes the host controller to retire the current IRP and advance to the next IRP or else stall the pipe as was outlined in Section 5.3.2. If a data payload is received that is larger than that expected, the pipe will stall and all pending bulk IRPs for that endpoint will be aborted/retired.

5.8.4 Bus Access Constraints

Bulk transfers can only be used by full speed devices.

An endpoint has no way to indicate a desired bus access frequency for a bulk pipe. USB balances the bus access requirements of all bulk pipes and the specific IRPs that are pending to provide "good effort" delivery of data between client software and functions. Moving control transfers over the bus has priority over moving bulk transfers.

There is no frame time guaranteed to be available for bulk transfers as there is for control transfers. Bulk transfers are only moved over the bus on a bandwidth available basis. If there is frame time that is not being used for other purposes, bulk transfers will be moved over the bus. If there is no time in a frame available for bulk transfers, no bulk transfers will be moved in that frame. If there are too many pending bulk transfers for the available frame time, bulk transfers are selected to be moved over the bus as

appropriate. If there are bulk transfers pending for multiple endpoints, bulk transfers for the different endpoints are selected according to a fair access policy that is host controller implementation dependent.

All bulk transfers pending in a system contend for the same available bus time. Because of this, the bus time made available for bulk transfers to a particular endpoint can be varied by USB system software at its discretion. An endpoint and its client software cannot assume a specific rate of service for bulk transfers. Bus time made available to a software client and its endpoint can be changed as other devices are inserted into and removed from the system or also as bulk transfers are requested for other device endpoints. Client software cannot assume ordering between bulk and control transfers; i.e., in some situations bulk transfers can be delivered ahead of control transfers.

The bus frequency and frame timing limit the maximum number of successful bulk transactions within a frame for any USB system to less than 72 8 byte data payloads. Table 5-6 lists information about different sized bulk transactions and the maximum number of transactions possible in a frame. Host controllers are free to determine how the individual bus transactions for specific bulk transfers are moved over the bus within and across frames. An endpoint could see all bus transactions for a bulk transfer within the same frame or spread across several discontiguous frames. Finally, a host controller, for various implementation reasons, may not be able to provide the above maximum number of transactions per frame.

Table 5-6. Bulk Transaction Limits

protocol overhead (bytes)						
13	(3-syncs, 3-pids, 2-EP+CRC, 2-CRC					
	3 byte interpacket delay)					
data	max Bandwidth	Bandwidth	max	Bytes	Bytes/frame	
payload	bytes/sec	per trans.	trans.	Remaining	useful data	
1	107000	1%	107	2	107	
2	200000	1%	100	0	200	
4	352000	1%	88	4	352	
8	568000	1%	71	9	568	
16	816000	2%	51	21	816	
32	1056000	3%	33	15	1056	
64	1216000	5%	19	37	1216	
max	1500000				1500	

5.8.5 Data Sequences

Bulk transactions use data toggle bits that are toggled only upon successful transaction completion to preserve synchronization between transmitter and receiver when transactions are retried due to errors. Bulk transactions are initialized to DATA0 when the endpoint is configured by an appropriate control transfer. The host will also start the first bulk transaction with DATA0. If a bulk pipe is stalled, the data toggle on the host is reset to DATA0 when the stall is acknowledged by the software client via a USBD function. The endpoint has its stall condition cleared via an appropriate control transfer. That action also resets the endpoint's data toggle to DATA0.

Bulk transactions are retried due to errors detected on the bus that affect a given transaction.

5.9 Bus Access for Transfers

Accomplishing any data transfer between the host and a USB device requires some use of the USB bandwidth. Supporting a wide variety of isochronous and asynchronous devices requires that each device's transfer requirements are accommodated. The process of assigning bus bandwidth to devices is called Transfer Management. There are several entities on the host that coordinate the information flowing over USB: Client software, the USB Driver (USBD), and the Host Controller Driver (HCD). Implementers of these entities need to know the key concepts related to bus access:

- Transfer Management - The entities and the objects that support communication flow over USB.
- Transaction Tracking - The USB mechanisms that are used to track transactions as they move through the USB system.
- Bus Time - The time it takes to move a packet of information over the bus.
- Device/Software Buffer Size - The space required to support a bus transaction.
- Bus Bandwidth Reclamation - Conditions where bandwidth that was allocated to other transfers but was not used and can now be possibly reused by control and bulk transfers.

The previous sections focused on how client software relates to a function and what the logical flows are over a pipe between the two entities. This section focuses on the different parts of the host and how they must interact to support moving data over the USB. This information may also be of interest to device implementers to understand aspects of what the host is doing when a client requests a transfer and how that transfer is presented to the device.

5.9.1 Transfer Management

Transfer Management involves several entities that operate on different objects in order to move transactions over the bus:

- Client Software - Consumes/Generates function specific data to/from a function endpoint via calls and callbacks requesting IRPs with USBD interface.
- USB Driver (USBD) - Converts data in client IRPs to/from device endpoint via calls/callbacks with appropriate HCD. A single client IRP may involve one or more transfers.
- Host Controller Driver (HCD) - Converts IRPs to/from transactions (as required by a host controller implementation) and organizes them for manipulation by the host controller. Interactions between the host controller driver and its hardware is implementation dependent and outside the scope of the USB specification.
- Host Controller - Takes transactions and generates bus activity via packets to move function specific data across the bus for each transaction.

Figure 5-10 shows how the entities are organized as information flows between client software and the USB. The objects of primary interest to each entity are shown at the interfaces between entities.

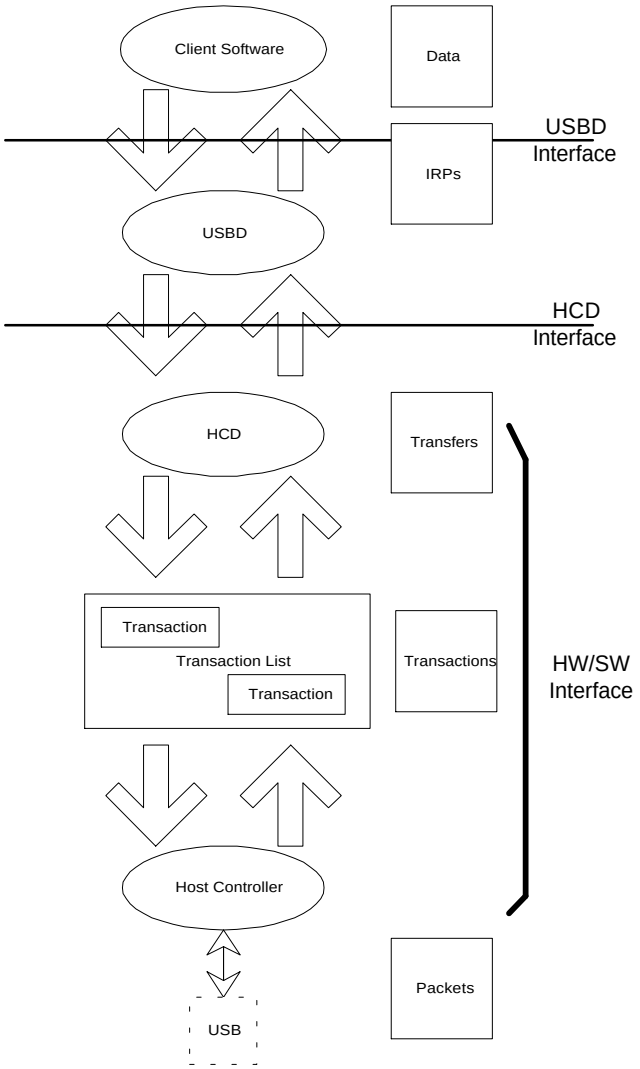


Figure 5-10. USB Information Conversion From Client Software to Bus

5.9.1.1 Client Software

Client software determines what transfers need to be made with a function. It uses appropriate operating system specific interfaces to request IRPs. Client software is only aware of the set of pipes (i.e., the interface) it needs to manipulate its function. The client is aware and adheres to all bus access and bandwidth constraints as described previously for each transfer type. The requests made by the client software are presented via the USBD interface.

Some clients may manipulate USB functions via other device class interfaces defined by the operating system and may themselves not make direct USBD calls. However, there is always some lowest level client that makes USBD calls to pass IRPs to USBD. All IRPs presented are required to adhere to the prenegotiated bandwidth constraints set when the device was attached to the bus and configured. If a function is moved from a non-USB environment to USB, the driver that would have directly manipulated the function hardware via memory or I/O accesses is the lowest client software in the USB environment that now interacts with USBD to manipulate its USB function.

After client software has requested a transfer of its function and the request has been serviced, the client software gets notified of the completion status of the IRP. If the transfer involved function to host data transfer, the client software can access the data in the data buffer associated with the completed IRP.

The USBD interface is defined in Chapter 10.

5.9.1.2 USB Driver

USBD is involved in mediating bus access at two general times while a device is attached to the bus during configuration and normal transfers. When a device is attached and configured, USBD is involved to ensure that the desired device configuration can be accommodated on the bus. The USBD receives configuration requests from the configuring software which describe the desired device configuration: endpoint(s), transfer type(s), transfer period(s), data size(s), etc. USBD either accepts or rejects a configuration request based on bandwidth availability and the ability to accommodate that request type on the bus. If it accepts the request, USBD creates a pipe for the requester of the desired type and with appropriate constraints as defined for the transfer type.

The configuration aspects of USBD are typically operating system environment specific and heavily leverage the configuration features of the operating system to avoid defining additional (redundant) interfaces.

Once a device is configured, the software client can request IRPs to move data between it and its function endpoints.

5.9.1.3 Host Controller Driver

HCD is responsible for tracking the IRPs in progress and ensuring that USB bandwidth and frame time maximums are never exceeded. When IRPs are made for a pipe, HCD adds them to the transaction list. When an IRP is complete, HCD notifies the requesting software client of the completion status for the IRP. If the IRP involved data transfer from the function to the software client, the data was placed in the client-indicated data buffer.

IRPs are defined in an operating system dependent manner.

5.9.1.4 Transaction List

The transaction list is a host controller implementation dependent description of the current outstanding set of bus transactions that need to be run on the bus. A typical transaction list consists of a series of frame descriptions in some host controller implementation dependent representation. Only HCD and its host controller have access to the specific representation. Each frame description contains transaction descriptions in which parameters such as data size in bytes, the device address and endpoint number, and the memory area to which data is to be sent or received are identified.

A transaction list and the interface between HCD and its host controller is typically represented in an implementation dependent fashion and is not defined explicitly as part of the USB specification.

5.9.1.5 Host Controller

The host controller has access to the transaction list and translates it into bus activity. In addition, the host controller provides a reporting mechanism whereby the status of a transaction (done, pending, stalled, etc.) can be obtained. The host controller converts transactions into appropriate implementation dependent activities that result in USB packets moving over the bus topology rooted in the root hub.

The host controller ensures that the bus access rules defined by the protocol are obeyed; e.g., inter-packet timings, time-outs, babble, etc. The HCD interface provides a way for the host controller to participate in whether a new pipe is allowed access to the bus. This is done because host controller implementations can have restrictions/constraints on the minimum inter-transaction times they may support for combinations of bus transactions.

The interface between the transaction list and the host controller is hidden within an HCD and host controller implementation. The host controller is typically implemented in hardware.

5.9.2 Transaction Tracking

A USB function sees data flowing across the bus in packets as described in Chapter 8. The host controller uses some implementation dependent representation to track what packets to transfer to/from what endpoints at what time or in what order. Most client software does not want to deal with packetized communication flows since this involves a degree of complexity and interconnect dependency that limits the implementation. USB system software (USBD and HCD) provides support for matching data movement requirements of a client to packets on the bus. The host controller hardware and software uses IRPs to track information about one or more transactions that combine to deliver a transfer of information between the client software and the function. Figure 5-11 summarizes how transactions are organized into IRPs for the four transfer types. Detailed protocol information for each transfer type can be found in Chapter 8. More information about client software views of IRPs can be found in Chapter 10 and in the operating system specific information for a particular operating system.

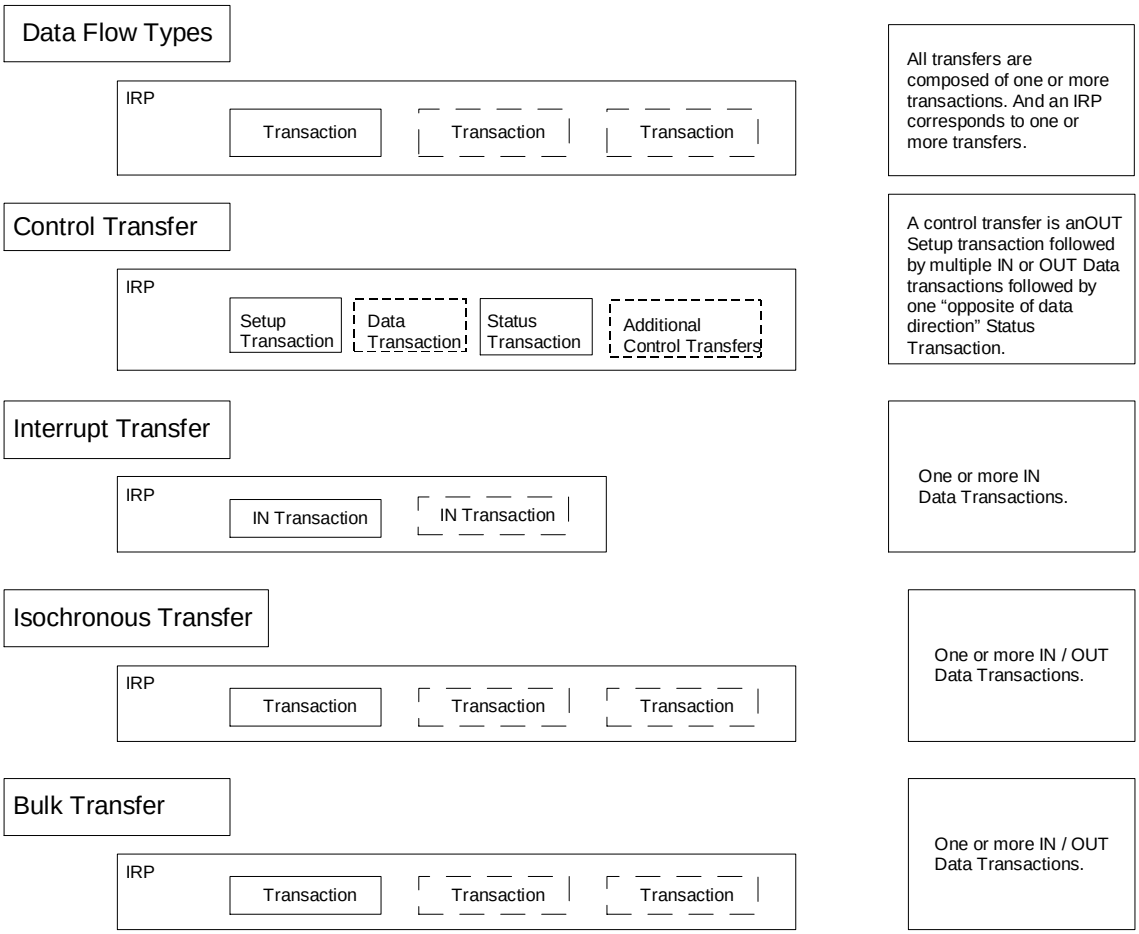


Figure 5-11. Transfers for Communication Flows

Even though IRPs track the bus transactions that need to occur to move a specific data flow over USB, host controllers are free to choose how the particular bus transactions are moved over the bus subject to the USB defined constraints; e.g., exactly one transaction per frame for isochronous transfers. In any case, an endpoint will see transactions in the order they appear within an IRP unless errors occur. For example, Figure 5-12 shows two IRPs, one each for two pipes where each IRP contains three transactions. For any transfer type, a host controller is free to move the first transaction of the first IRP followed by the first transaction of the second IRP somewhere in Frame 1, while moving the second transactions of each IRP in opposite order somewhere in Frame 2. If these are isochronous transfer types, that is the only degree of freedom a host controller has. If these are control or bulk transfers, a host

controller could further move more or less transactions from either IRP within either frame. Functions cannot depend on seeing transactions within an IRP back to back within a frame.

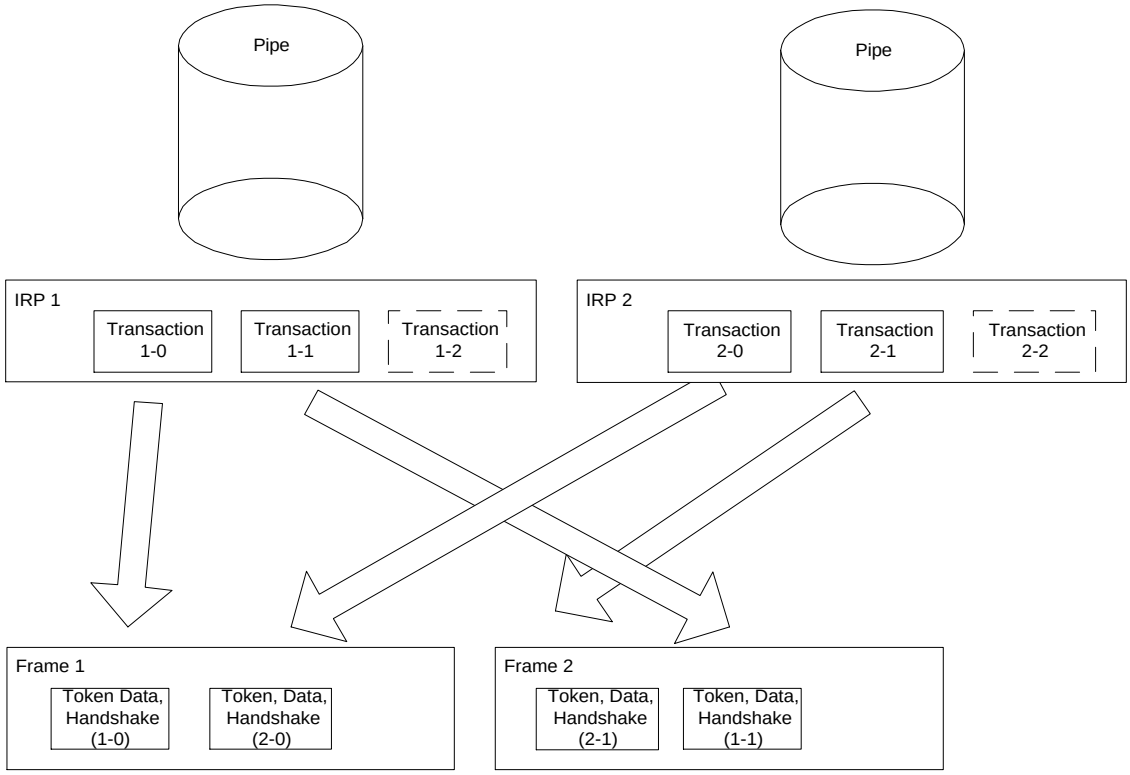


Figure 5-12. Arrangement of IRPs to Transactions/Frames

5.9.3 Calculating Bus Transaction Times

When the USB system software allows a new pipe to be created for the bus, it must calculate how much bus time is required for a given transaction. That bus time is based on the maximum packet size information reported for an endpoint, the protocol overhead for the specific transaction type request, the overhead due to signaling imposed bit-stuffing, inter-packet timings required by the protocol, inter-transaction timings, etc. These calculations are required to ensure that the time available in a frame is not exceeded. The equations used to determine transaction bus time are:

KEY

Data_bc	Byte count of data payload
Host_Delay	Time required for the host to prepare for or recover from the transmission; host controller implementation specific
Floor()	Integer portion of argument
Hub_LS_Setup	The time provided by the host controller for hubs to enable low speed ports; Measured as the delay from end of PRE PID to start of low speed SYNC; minimum of 4 full speed bit times.
BitStuffTime	Function that calculates theoretical additional time required due to bit stuffing in signaling; worst case is (1.1667*8*Data_bc)

Full Speed (Input)

Non-Isochronous Transfer (Handshake Included)

= 9107 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Isochronous Transfer (No Handshake)

= 7268 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Full Speed (Output)

Non-Isochronous Transfer (Handshake Included)

= 9107 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Isochronous Transfer (No Handshake)

= 6265 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Low Speed (Input)

= 64060 + (2 * Hub_LS_Setup) +
(676.67 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Low Speed (Output)

= 64107 + (2 * Hub_LS_Setup) +
(667.0 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

The Bus Times in the above equations are in nanoseconds and take into account propagation delays due to distance the device is from the host. These are typical equations that can be used to calculate Bus Time; however, different implementations may choose to use coarser approximations of these times.

The actual bus time taken for a given transaction will almost always be less than that calculated since bit stuffing overhead is data dependent. Worst case bit stuffing is calculated as 1.1667 times the raw time (i.e., the BitStuffTime function multiplies the Data_bc by 8*1.1667 in the equations). This means that there will almost always be time unused on the bus (subject to data pattern specifics) after all regularly scheduled transactions have completed. By placing all bulk/control transactions at the end of a frame, bit stuffing time can be calculated as less than worst case. This more aggressive calculation comes at the cost of having some bulk/control transfer's transaction fail in a given frame every once in a while due to exceeding the frame time when enough of the previous transactions require worst case bit stuffing. The failed transaction can be retried, will seldom happen given random data patterns, and can result in a better allocation estimate for isochronous and interrupt transfer times. In any case, the bus time made available due to less bit stuffing can be reused as discussed in Section 5.9.5.

The Host_Delay term in the equations is host controller and system dependent and allows for additional time a host controller may require due to delays in gaining access to memory or other implementation dependencies. This term is incorporated into an implementation of these equations by using the Transfer Constraint Management functions provided by the HCD interface. These equations are typically implemented by a combination of USBD and HCD software working in cooperation. The results of these calculations are used to determine whether a transfer or pipe creation can be supported in a given USB configuration.

5.9.4 Calculating Buffer Sizes in Functions/Software

Client software and functions both need to provide buffer space for pending data transactions awaiting their turn on the bus. For non-isochronous pipes, this buffer space only needs to be large enough to hold the next data packet. If more than one transaction request is pending for a given endpoint, the buffering for each transaction must be supplied. Methods to calculate precise absolute minimum buffering a function may require because of specific interactions defined between its client software and the function are outside the scope of the USB specification.

The host controller is expected to be able to support an unlimited number of transactions pending for the bus subject to available system memory for buffer and descriptor space, etc. Host controllers are allowed to limit how many frames into the future they allow a transaction to be requested.

For isochronous pipes, Section 5.10.4 describes details affecting host side and device side buffering requirements. In general, buffers need to be provided to hold approximately twice the amount of data that can be transferred in 1 ms.

5.9.5 Bus Bandwidth Reclamation

USB bandwidth and bus access are granted based on a calculation of worst case bus transmission time and required latencies. However, due to the constraints placed on different transfer types and the fact that the bit stuffing bus time contribution is calculated as a constant but is data dependent, there will frequently be bus time remaining in each frame time versus what the frame transmission time was calculated to be. In order to support the most efficient use of the bus bandwidth, control and bulk transfers are candidates to be moved over the bus as bus time becomes available. Exactly how a host controller supports this is implementation dependent. A host controller can take into account the transfer types of pending IRPs and implementation specific knowledge of remaining frame time to reuse reclaimed bandwidth.

5.10 Special Considerations for Isochronous Transfers

Support for isochronous data movement between the host and a device is one of the system capabilities supported by USB. Delivering isochronous data reliably over USB requires careful attention to detail. The responsibility for reliable delivery is shared by several USB entities: the device/function, the bus, the host controller, and one or more software agents. Since time is a key part of an isochronous transfer, it is important for USB designers to understand how time is dealt within USB by these different entities.

All isochronous devices must report their capabilities in the form of device specific descriptors. The capabilities should also be provided in a form that the potential customer can use to decide whether the device offers a solution to his problem(s). The specific capabilities of a device can justify price differences.

In any communication system, the transmitter and receiver must be synchronized enough to deliver data robustly. In an asynchronous communication system, data can be delivered robustly by allowing the transmitter to detect that the receiver has not received a data item correctly and simply retrying transmission of the data.

In an isochronous communication system, the transmitter and receiver remain time and data synchronized to deliver data robustly. USB does not support transmission retry of isochronous data so that minimal bandwidth can be allocated to isochronous transfers and time synchronization is not lost due to a retry delay. However, it is critical that a USB isochronous transmitter/receiver pair still remain synchronized both in normal data transmission cases and in cases where errors occur on the bus.

In many systems that deal with isochronous data, a single global clock is used to which all entities in the system synchronize; e.g., the PSTN - Public Switched Telephone Network. Given that a broad variety of devices with different natural frequencies may be attached to USB, no single clock can provide all the features required to satisfy the synchronization requirements of all devices and software while still

supporting the cost targets of mass market PC products. USB defines a clock model that allows a broad range of devices to coexist on the bus and have reasonable cost implementations.

This section presents options or features that can be used by isochronous endpoints to minimize behavior differences between a non-USB implemented function and a USB version of the function. An example is included to illustrate the similarities and differences of non-USB and USB versions of a function.

The remainder of the section presents key concepts of:

- USB Clock Model - What clocks are present in a USB subsystem that have impact on isochronous data transfers.
- USB Frame Clock to Function Clock Synchronization Options - How the USB Frame clock can relate to a function clock.
- Start of Frame Tracking - Responsibilities/Opportunities of isochronous endpoints with respect to the Start of Frame (SOF) token and USB Frames.
- Data Prebuffering - Requirements on accumulating data before generation/transmission/consumption.
- Error Handling - Isochronous specific details for error handling.
- Buffering for Rate Matching - Equations that can be used to calculate buffer space required for isochronous endpoints.

5.10.1 Example Non-USB Isochronous Application

The example used is a reasonably general case example. Other simpler or more complex cases are possible and the relevant USB features identified can be used or not as appropriate.

The example consists of an 8 kHz mono microphone connected through a mixer driver that sends the input data stream to 44 kHz stereo speakers. The mixer expects the data to be received and transmitted at some sample rate and encoding. A rate matcher driver on input and output converts the sample rate and encoding from the natural rate and encoding of the device to the rate and encoding expected by the mixer. Figure 5-13 illustrates this example.

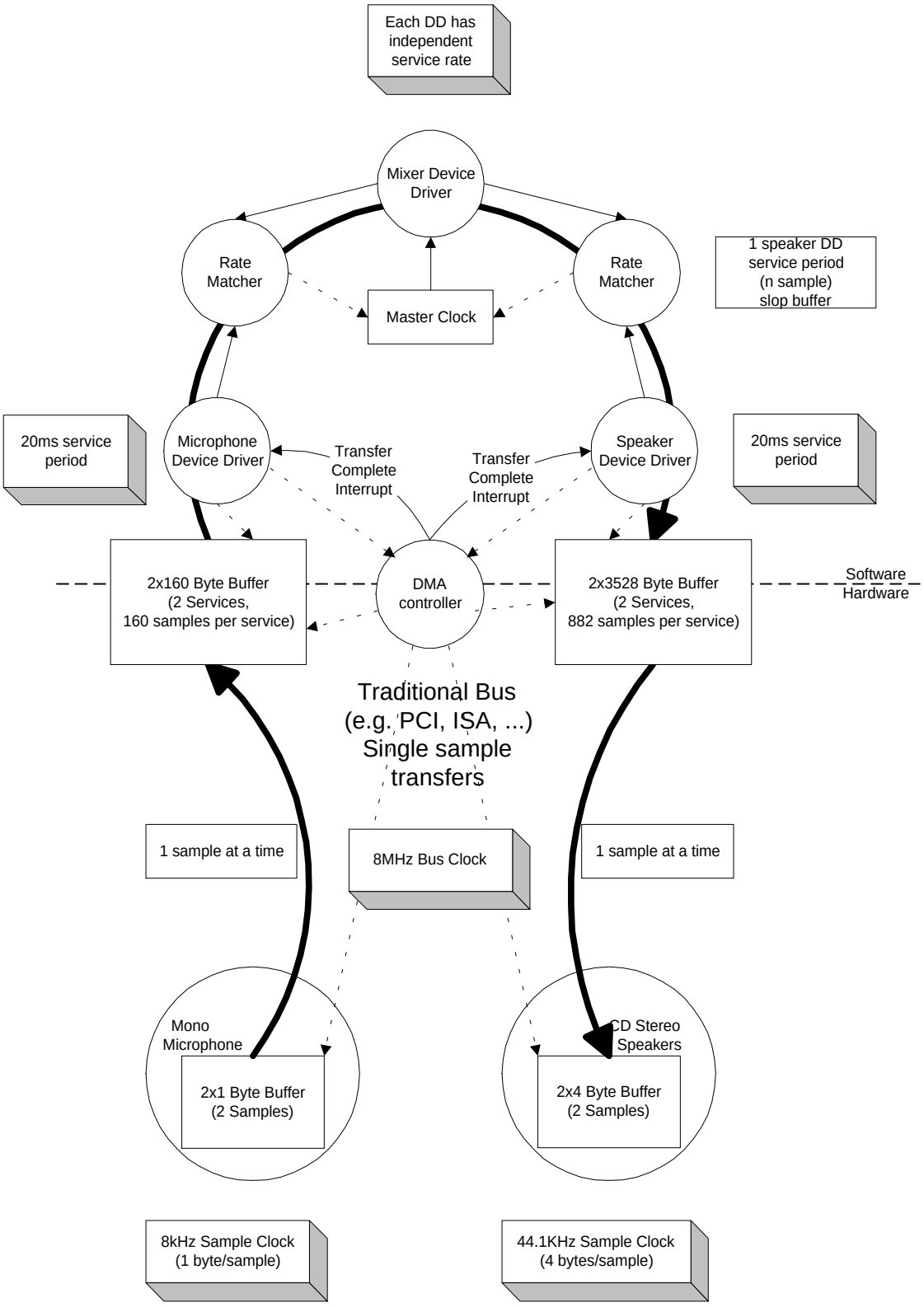


Figure 5-13. Non-USB Isochronous Example

A master clock (can be provided by software driven from the real time clock) in the PC is used to awaken the mixer to ask the input source for input data and to provide output data to the output sink. In this example, assume it awakens every 20 ms. The microphone and speakers each have their own sample clocks that are unsynchronized with respect to each other or the master mixer clock. The microphone produces data at its natural rate (1 byte samples, 8,000 times a second) and the speakers consume data at their natural rate (4 byte samples, 44,100 times a second). The three clocks in the system can drift and jitter with respect to each other. Each rate matcher may also be running at a different natural rate than either the mixer driver, the input source/driver, or output sink/driver.

The rate matchers also monitor the long term data rate of their device compared to the master mixer clock and interpolate an additional sample or merge two samples to adjust the data rate of their device to the data rate of the mixer. This adjustment may be required every couple of seconds, but typically occurs infrequently. The rate matchers provide some additional buffering to carry through a rate match.

Note that some other application might not be able to tolerate sample adjustment and would need some other means of accommodating master clock to device clock drift or else would require some means of synchronizing the clocks to ensure that no drift could occur.

The mixer always expects to receive exactly a service period of data (20 ms service period) from its input device and produce exactly a service period of data for its output device. The mixer can be delayed up to less than a service period if data or space is not available from its input/output device. The mixer assumes that such delays don't accumulate.

The input and output devices and their drivers expect to be able to put/get data in response to a hardware interrupt from the DMA controller when their transducer has processed one service period of data. They expect to get/put exactly one service period of data. The input device produces 160 bytes (10 samples) every service period of 20 ms. The output device consumes 3528 bytes (882 samples) every 20 ms service period. The DMA controller can move a single sample between the device and the host buffer at a rate much faster than the sample rate of either device.

The input and output device drivers provide two service periods of system buffering. One buffer is always being processed by the DMA controller. The other buffer is guaranteed to be ready before the current buffer is exhausted. When the current buffer is emptied, the hardware interrupt awakens the device driver and it calls the rate matcher to give it the buffer. The device driver requests a new IRP with the buffer before the current buffer is exhausted.

The devices can provide two samples of data buffering to ensure that they always have a sample to process for the next sample period while the system is reacting to the previous/next sample.

The service periods of the drivers are chosen to survive interrupt latency variabilities that may be present in the operating system environment. Different operating system environments will require different service periods for reliable operation. The service periods are also selected to place a minimum interrupt load on the system since there may be other software in the system that requires processing time.

5.10.2 USB Clock Model

Time is present in the USB system via clocks. In fact, there are multiple clocks in a USB system that must be understood:

- Sample clock - This clock determines the natural data rate of samples moving between client software on the host and the function. This clock does not need to be different between non-USB and USB implementations.
- Bus clock - This clock runs at a 1.000 ms period (1 kHz frequency) and is indicated by the rate of Start of Frame(SOF) packets on the bus. This clock is somewhat equivalent to the 8 MHz clock in the non-USB example. In the USB case, the bus clock is often a lower frequency clock than the sample clock, whereas the bus clock is almost always a higher frequency clock than the sample clock in a non-USB case.
- Service clock - This clock is determined by the rate at which client software runs to service IRPs that may have accumulated between executions. This clock also can be the same in the USB and non-USB cases.

In most operating system environments that exist today, it is not possible to support a broad range of isochronous communication flows if each device driver must be interrupted for each sample for fast sample rates. Therefore, multiple samples, if not multiple packets, will be processed by client software and then given to the host controller to sequence over the bus according to the prenegotiated bus access requirements. Figure 5-14 presents an example for a reasonable USB clock environment equivalent to the non-USB example in Figure 5-13.

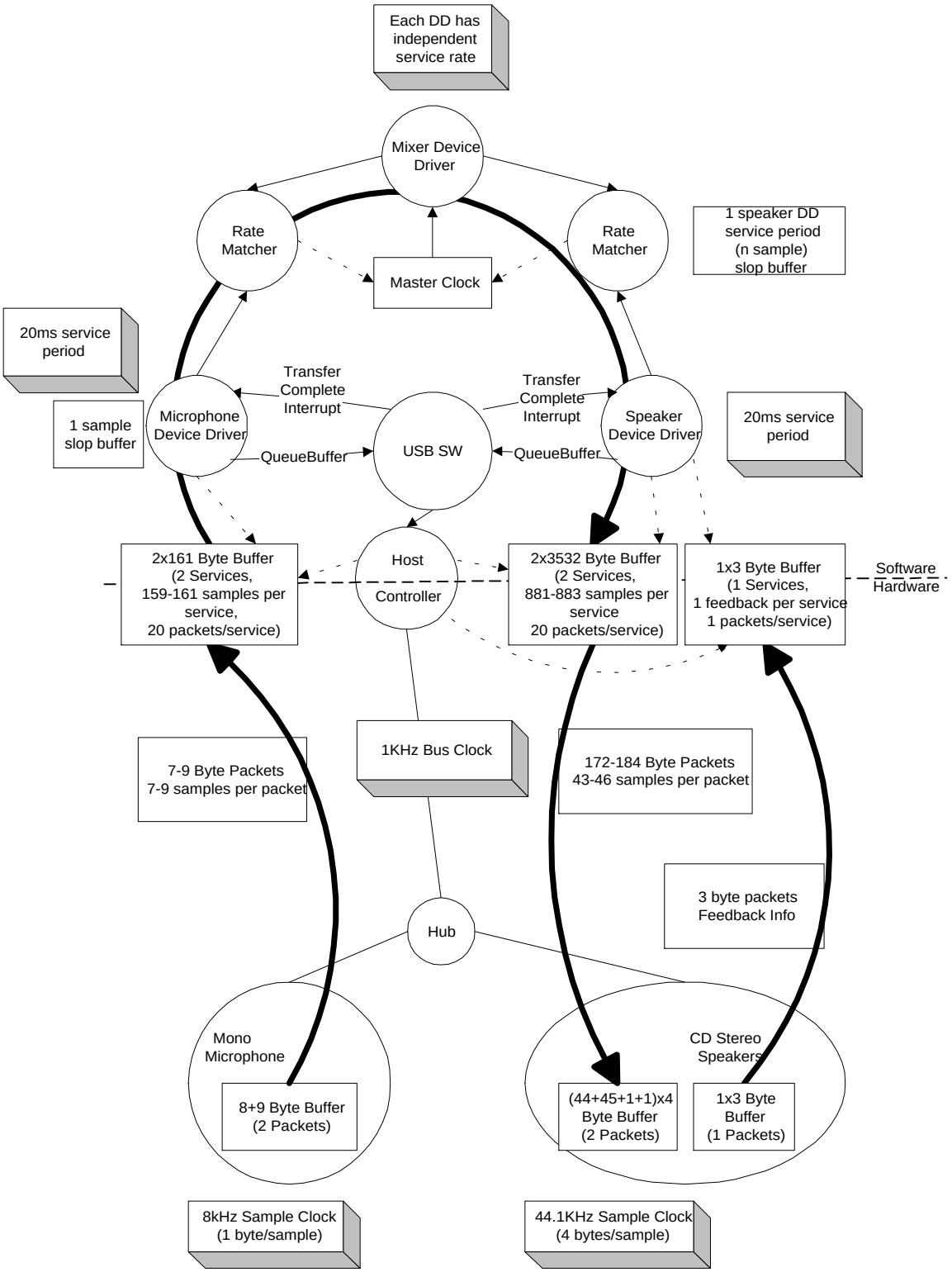


Figure 5-14. USB Isochronous Application

Figure 5-14 shows a typical round trip path of information from a microphone as an input device to a speaker as an output device. The clocks, packets, and buffering involved are also shown. Figure 5-14 will be explored in more detail in the following sections.

The focus of this example is to identify the differences introduced by USB compared to the previous non-USB example. The differences are in the areas of buffering, synchronization given the existence of a USB bus clock, and delay. The client software above the device drivers can be unaffected in most cases.

5.10.3 Clock Synchronization

In order for isochronous data to be manipulated reliably, the three clocks identified above must be synchronized in some fashion. If the clocks are not synchronized, several clock to clock attributes can be present that can be undesirable:

- Clock drift - Two clocks that are nominally running at the same rate can, in fact, have implementation differences that result in one clock running faster or slower than the other over long periods of time. If uncorrected, this variation of one clock compared to the other can lead to having too much or too little data when data is expected to always be present at the time required.
- Clock jitter - A clock may vary its frequency over time due to changes in temperature, etc. This may also alter when data is actually delivered compared to when it is expected to be delivered.
- Clock to clock phase differences - If two clocks are not phase locked, different amounts of data may be available at different points in time as the beat frequency of the clocks cycle out over time. This can lead to quantization/sampling related artifacts.

The bus clock provides a central clock with which USB hardware devices and software can synchronize to one degree or another. However, the software will, in general, not be able to phase or frequency lock precisely to the bus clock given the current support for “real time-like” operating system scheduling support in most PC operating systems. Software running in the host can, however, know that data moved over USB is packetized. For isochronous transfer types, a single packet of data is moved exactly once per frame and the frame clock is reasonably precise. Providing the software with this information allows it to adjust the amount of data it processes to the actual frame time that has passed.

5.10.4 Isochronous Devices

USB includes a framework for isochronous devices which defines Synchronization Types, how isochronous endpoints provide data rate feedback, and how they can be connected together. Isochronous devices include sampled analog devices; for example, audio and telephony devices, and synchronous data devices. Synchronization Type classifies an endpoint according to its capability to synchronize its data rate to the data rate of the endpoint that it is connected to. Feedback is provided by indicating accurately what the required data rate is, relative to the SOF frequency. The ability to make connections depends on the quality of connection that is required, the endpoint synchronization type, and the capabilities of the host application which is making the connection. Additional Device Class-specific information may be required, depending on the application.

Note that the term “data” is used very generally, and may refer to data which represents sampled analog information (like audio), or it may be more abstract information. “Data rate” refers to the rate at which analog information is sampled, or the rate at which data is clocked.

The following information is required in order to determine how to connect isochronous endpoints:

- Synchronization Type
  - Asynchronous - unsynchronized, although sinks provide data rate feedback
  - Synchronous - synchronized to USB's SOF
  - Adaptive - synchronized using feedback or feedforward data rate information
- Available data rates
- Available data formats

Synchronization Type and data rate information are needed to determine if an exact data rate match exists between source and sink, or if an acceptable conversion process exists which would allow the source to be connected to the sink. It is the responsibility of the application to determine whether the connection can be supported within available processing resources and other constraints (like delay). Specific USB device classes define how to describe Synchronization Type and data rate information.

Data format matching and conversion is also required for a connection, but it is not a unique requirement for isochronous connections. Details about format conversion can be found in other documents related to specific formats.

5.10.4.1 Synchronization Type

Three distinct synchronization types are defined. Table 5-7 presents an overview of endpoint synchronization characteristics for both source and sink endpoints. The types are presented in order of increasing capability.

Table 5-7. Synchronization Characteristics

	Source	**Sink**
Asynchronous	Free running Fs Provides implicit feedforward (data stream)	Free running Fs Provides explicit feedback (interrupt pipe)
Synchronous	Fs locked to SOF Uses implicit feedback (SOF)	Fs locked to SOF Uses implicit feedback (SOF)
Adaptive	Fs locked to sink Uses explicit feedback (control pipe)	Fs locked to data flow Uses implicit feedforward (data stream)

5.10.4.1.1 Asynchronous

Asynchronous endpoints cannot synchronize to SOF or any other clock in the USB domain. They source or sink an isochronous data stream at either a fixed data rate (single frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. If the data rate is programmable, it is set during initialization of the isochronous endpoint. Asynchronous devices must report their programming capabilities in the class specific endpoint descriptor as described in their Device Class specification. The data rate is locked to a clock external to USB or to a free running internal clock. These devices place the burden of data rate matching elsewhere in the USB environment. Asynchronous source endpoints carry their data rate information implicitly in the number of samples they produce per frame. Asynchronous sink endpoints must provide explicit feedback information to an adaptive driver (refer to Section 5.10.4.2).

An example of an asynchronous source is a CD-audio player that provides its data based on an internal clock or resonator. Another example is a Digital Audio Broadcast (DAB) receiver or a Digital Satellite Receiver (DSR). Here too, the sample rate is fixed at the broadcasting side and is beyond USB control.

Asynchronous sink endpoints could be low cost speakers, running off of their internal sample clock.

Another case arises when there are two or more devices present on USB that need to have mastership control over SOF generation in order to operate as synchronous devices. This could happen if there were two telephony devices, each locked to a different external clock. One telephony device could be digitally connected to a PBX which is not synchronized to the ISDN. The other device could be connected directly to the ISDN. Each device will source or sink data to/from the network side at an externally driven rate. Since only one of the devices can take mastership over SOF, the other will sink or source data at a rate which is asynchronous to SOF. This example indicates that every device capable of SOF mastership may be forced to operate as an asynchronous device.

5.10.4.1.2 Synchronous

Synchronous endpoints can have their clock system (their notion of time) controlled externally through SOF synchronization. These endpoints must be doing one of the following:

- Slaving their sample clock to the 1 ms SOF tick (by means of a programmable PLL).
- Controlling the rate of USB SOF generation so that their data rate becomes automatically locked to SOF. In case these endpoints are not granted SOF mastership, they must degenerate to the asynchronous mode of operation (refer to the asynchronous example).

Synchronous endpoints may source or sink isochronous data streams at either a fixed data rate (single frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. If programmable, the operating data rate is set during initialization of the isochronous endpoint. The number of samples or data units generated in a series of USB frames is deterministic and periodic. Synchronous devices must report their programming capabilities in the class specific endpoint descriptor as described in their Device Class specification.

An example of a synchronous source is a digital microphone that synthesizes its sample clock from SOF and produces a fixed number of audio samples every USB frame. Another possibility is a 64 kb/s bit-stream from an ISDN “modem.” If the USB SOF generation is locked to the PSTN clock (perhaps through the same ISDN device), the data generation will also be locked to SOF and the endpoint will produce a stable 64 kb/s data stream, referenced to the SOF time notion.

5.10.4.1.3 Adaptive

Adaptive endpoints are the most capable endpoints possible. They are able to source or sink data at any rate within their operating range. Adaptive source endpoints produce data at a rate which is controlled by the data sink. The sink provides feedback (refer to Section 5.10.4.2) to the source which allows the source to know the desired data rate of the sink. Adaptive endpoints can communicate with all types of sink endpoints. For adaptive sink endpoints, the data rate information is embedded in the data stream. The average number of samples received during a certain averaging time determines the instantaneous data rate. If this number changes during operation, the data rate is adjusted accordingly.

The data rate operating range may center around one rate (e.g., 8 kHz), select between several programmable or auto detecting data rates (32 kHz, 44.1 kHz, 48 kHz, …), or may be within one or more ranges (e.g., 5 kHz to 12 kHz, 44 kHz to 49 kHz). Adaptive devices must report their programming capabilities in the class specific endpoint descriptor as described in their Device Class specification

An example of an adaptive source is a CD player that contains a fully adaptive sample rate converter (SRC) so that the output sample frequency no longer needs to be 44.1 kHz but can be anything within the operating range of the SRC. Adaptive sinks include such endpoints as high-end digital speakers, headsets, etc.

5.10.4.2 Feedback

An asynchronous sink provides feedback to an adaptive source by indicating accurately what its desired data rate (F_f) is, relative to the USB SOF frequency. The required data rate is accurate to better than one sample per second (1 Hz) in order to allow a high quality source rate to be created and to tolerate delays and errors in the feedback loop.

The F_f value consists of a fractional part, in order to get the required resolution with 1 kHz frames, and an integer part, which gives the minimum number of samples per frame. Ten bits are required to resolve one sample within a 1 kHz frame frequency (1000 / 2^10 = 0.98). This is a 10 bit fraction, represented in unsigned fixed binary point 0.10 format. The integer part needs 10 bits (2^10 = 1024) to encode up to 1023 1-byte samples per frame. The 10 bit integer is represented in unsigned fixed binary point 10.0 format. The combined F_f value can be coded in unsigned fixed binary point 10.10 format, which fits into three bytes (24 bits). Since the maximum integer value is fixed to 1023, the 10.10 number will be left-justified in the 24 bits, so that it has a 10.14 format. Only the first 10 bits behind the binary point are required. The lower 4 bits may be optionally used to extend the precision of F_f; otherwise, they shall be reported as 0. The bit and byte ordering follows the definitions of other multi-byte fields contained in Chapter 8.

Each frame, the adaptive source adds F_f to any remaining fractional sample count from the previous frame, sources the number of samples in the integer part of the sum, and retains the fractional sample count for the next frame. The source can look at the behavior of F_f over many frames to determine an even more accurate rate, if it needs to.

The sink can determine F_f by counting cycles of a clock with a frequency of Fs * 2^P for a period of 2^(10-P) frames, where P is an integer. P is practically bound to be in the range [0,10] because there is no point in using a clock slower than Fs, and no point in trying to update more than once a frame. The counter is read into F_f and reset every 2^(10-P) frames. As long as no clock cycles are skipped, the count will be accurate over the long term. An endpoint only needs to implement the number of counter bits that it requires for its maximum F_f.

A digital telephony endpoint, for example, will usually derive its 8 kHz Fs by dividing down the 64 kHz clock (P=3) which it uses to serialize the data stream. The 64 kHz clock phase can also give an additional 1 bit of accuracy, effectively giving P=4. This would give F_f updates every 2^(10-4) = 64 frames. A 13-bit counter would be required to obtain F_f, with 3 bits for 8 samples per frame, and 10 bits for the fractional part. The 13 bits would provide a 3.10 field within the 10.14 F_f value, with the remaining bits set to 0.

The choice of P is endpoint specific, and should be between 1 and 9, inclusive. Larger values of P are preferred, since they reduce the size of the frame counter and increase the rate at which F_f is updated. More frequent updates result in a tighter control of the source data rate, which reduces the buffer space required to handle F_f changes. P should be less than 10 so that F_f is averaged across at least two frames in order to reduce SOF jitter effects. P should not be 0 in order to keep the deviation in the number of samples sourced to less than 1 in the event of a lost F_f value.

Interrupt transfers are used to read F_f from the feedback register at periodic intervals. The desired reporting rate for the feedback should be 2^(10-P) ms (frames). F_f will be reported at most once per update period. There is nothing to be gained by reporting the same F_f value more than once per update period. The endpoint may choose to only report F_f if the updated value has changed from the previous F_f value.

It is possible that the source will deliver one too many or one too few samples over a long period, due to errors or accumulated inaccuracies in measuring F_f. The sink must have sufficient buffer capability to

accommodate this. When the sink recognizes this condition, it should adjust the reported F_f value to correct it. This may also be necessary to compensate for relative clock drifts. The implementation of this correction process is endpoint specific and is not specified.

An adaptive source may obtain the sink data rate information from an adaptive sink which is locked to the same clock as the sink, as would be the case for a two-way speech connection. In this case, the feedback pipe is not needed.

5.10.4.3 Connectivity

In order to fully describe the source-to-sink connectivity process, an interconnect model is presented. The model indicates the different components involved and how they interact to establish the connection.

The model provides for multi-source/multi-sink situations. Figure 5-15 illustrates a typical situation (highly condensed and incomplete). A physical device is connected to the host application software through different hardware and software layers as described in the USB specification. At the client interface level, a “virtualized” device is presented to the application. From the application standpoint, only virtual devices exist. It is up to the device driver and client software to decide what the exact relation is between physical and virtual device.

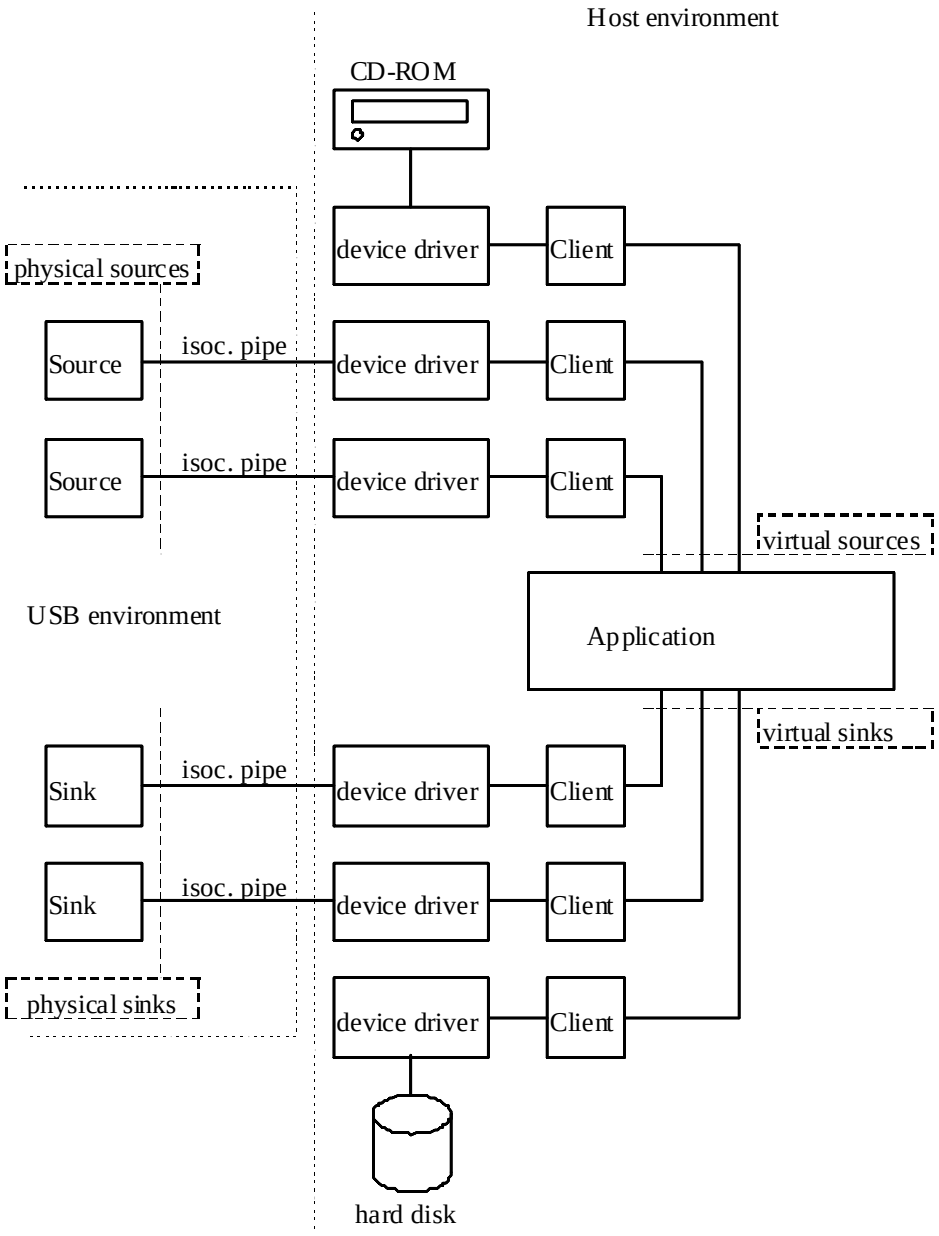


Figure 5-15. Example Source/Sink Connectivity

Device manufacturers (or operating system vendors) must provide the necessary device driver software and client interface software to convert their device from the physical implementation to a USB compliant software implementation, the virtual device. As stated before, depending on the capabilities built into this software, the virtual device can exhibit different synchronization behavior from the physical device. However, the synchronization classification equally applies to both physical and virtual devices. All physical devices belong to one of the three possible synchronization types. Therefore, the capabilities that have to be built into the device driver and/or client software are the same as the capabilities of a physical device. The word “application” must be replaced by “device driver/client software.” In the case of a physical source to virtual source connection, “virtual source device” must be replaced by “physical source device” and “virtual sink device” must be replaced by “virtual source device.” In the case of a virtual sink to physical sink connection, “virtual source device” must be replaced by “virtual sink device” and “virtual sink device” must be replaced by “physical sink device.”

Placing the rate adaptation functionality into the device driver/client software layer has the distinct advantage of isolating all applications, using the device from the specifics and problems associated with rate adaptation. Applications that would otherwise be multi-rate degenerate to simpler mono-rate systems.

Note that the model is not limited to only USB devices. A CD-ROM drive, for example, containing 44.1 kHz audio can appear as either an asynchronous, synchronous, or adaptive source. Asynchronous operation means that the CD-ROM fills its buffer at the rate that it reads data from the disk, and the driver empties the buffer according to its USB service interval. Synchronous operation means that the driver uses the USB service interval (e.g., 10 ms) and nominal sample rate of the data (44.1 kHz) to determine to put out 441 samples every USB service interval. Adaptive operation would build in a sample rate converter to match the CD-ROM output rate to different sink sampling rates.

Using this reference model, it is possible to define what operations are necessary to establish connections between various sources and sinks. Furthermore, the model indicates at what level these operations must or can take place. First there is the stage where physical devices are mapped onto virtual devices and vice versa. This is accomplished by the driver and/or client software. Depending on the capabilities included in this software, a physical device can be transformed into a virtual device of an entirely different synchronization type. The second stage is the application that uses the virtual devices. Placing rate matching capabilities at the driver/client level of the software stack relieves applications communicating with virtual devices from the burden of performing rate matching for every device that is attached to them. Once the virtual device characteristics are decided, the actual device characteristics are not any more interesting than the actual physical device characteristics of another driver.

As an example, consider a mixer application that connects at the source side to different sources, each running at their own frequencies and clocks. Before mixing can take place, all streams must be converted to a common frequency and locked to a common clock reference. This action can be performed in the physical to virtual mapping layer or it can be handled by the application itself for each source device independently. Similar actions must be performed at the sink side. If the application sends the mixed data stream out to different sink devices, it can either do the rate matching for each device itself or it can rely on the driver/client software to do that if possible.

Table 5-8 indicates at the intersections what actions the application must perform to connect a source endpoint to a sink endpoint.

Table 5-8 Connection Requirements

	Source Endpoint		
Sink Endpoint	**Asynchronous**	**Synchronous**	**Adaptive**
Asynchronous	Async Source/Sink RA See Note 1.	Async SOF/Sink RA See Note 2.	Data + Feedback Feedthrough See Note 3.
Synchronous	Async Source/SOF RA See Note 4.	Sync RA See Note 5.	Data Feedthrough + Application Feedback See Note 6.
Adaptive	Data Feedthrough See Note 7.	Data Feedthrough See Note 8.	Data Feedthrough See Note 9.

Notes:

1. Asynchronous RA in the application. Fs_i is determined by the source, using the feedforward information embedded in the data stream. Fs_O is determined by the sink, based on feedback information from the sink. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
2. Asynchronous RA in the application. Fs_i is determined by the source but locked to SOF. Fs_O is determined by the sink, based on feedback information from the sink. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
3. If Fs_O falls within the locking range of the adaptive source, a feedthrough connection can be established. $Fs_i = Fs_O$, and both are determined by the asynchronous sink, based on feedback information from the sink. If Fs_O falls outside the locking range of the adaptive source, the adaptive source is switched to synchronous mode and Note 2 applies.
4. Asynchronous RA in the application. Fs_i is determined by the source. Fs_O is determined by the sink and locked to SOF. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
5. Synchronous RA in the application. Fs_i is determined by the source and locked to SOF. Fs_O is determined by the sink and locked to SOF. If $Fs_i = Fs_O$, the process degenerates to a loss-free feedthrough connection.
6. The application will provide feedback to synchronize the source to SOF. Then the adaptive source appears to be a synchronous endpoint and Note 5 applies.
7. If Fs_i falls within the locking range of the adaptive sink, a feedthrough connection can be established. $Fs_i = Fs_O$ and are determined by and locked to the source.
   If Fs_i falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an Fs_O that is within the locking range of the adaptive sink.
8. If Fs_i falls within the locking range of the adaptive sink, a feedthrough connection can be established. $Fs_O = Fs_i$ and are determined by the source and locked to SOF.
   If Fs_i falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an Fs_O that is within the locking range of the adaptive sink.
9. The application will use feedback control to set Fs_O of the adaptive source when the connection is set up. The adaptive source operates as an asynchronous source in the absence of ongoing feedback information and Note 7 applies.

In cases where RA is needed but not available, the rate adaptation process could be mimicked by sample dropping/stuffing. The connection could then still be made, possibly with a warning about poor quality; otherwise, the connection cannot be made.

5.10.4.3.1 Audio Connectivity

When the above is applied to audio data streams, the rate adaptation process is replaced by sample rate conversion, which is a specialized form of rate adaptation. Instead of error control, some form of sample interpolation is used to match incoming and outgoing sample rates. Depending on the interpolation techniques used, the audio quality (distortion, signal to noise ratio, etc.) of the conversion can vary significantly. In general, higher quality requires more processing power.

5.10.4.3.2 Synchronous Data Connectivity

For the synchronous data case, rate adaptation is used. Occasional slips/stuffs may be acceptable to many applications which implement some form of error control. Error control includes error detection and discard, error detection and retransmit, or forward error correction. The rate of slips/stuffs will depend on the clock mismatch between the source and sink, and may be the dominant error source of the channel. If the error control is sufficient, then the connection can still be made.

5.10.5 Data Prebuffering

USB requires that devices prebuffer data before processing/transmission to allow the host more flexibility in managing when each pipe's transaction is moved over the bus from frame to frame.

For transfers from function to host, the endpoint must accumulate samples during frame X until it receives the Start of Frame (SOF) token packet for frame X+1. It "latches" the data from frame X into its packet buffer and is now ready to send the packet containing those samples during frame X+1. When it will send that data during the frame is determined solely by the host controller and can vary from frame to frame.

For transfers from host to function, the endpoint will accept a packet from the host sometime during frame Y. When it receives the SOF for frame Y+1, it can then start processing the data received in frame Y.

This approach allows an endpoint to use the SOF token as a stable clock with very little jitter/drift when the host controller moves the packet over the bus while also allowing the host controller to vary within a frame precisely when the packet is actually moved over the bus. This prebuffering introduces some additional delay between when a sample is available at an endpoint and when it moves over the bus compared to an environment where the bus access is at exactly the same time offset from SOF from frame to frame.

Figure 5-16 shows the time sequence where for a function to host transfer (IN process), data D_0 is accumulated during frame F_i at time T_i, and transmitted to the host during frame F_{I+1}. Similarly, for a host to function transfer (OUT process), data D_0 is received by the endpoint during frame F_{i+1} and processed during frame F_{i+2}.

Time:	T_i	T_{i+1}	T_{i+2}	T_{i+3}	...	T_m	T_{m+1}	...
Frame:	F_i	F_{i+1}	F_{i+2}	F_{i+3}	...	F_m	F_{m+1}	...
Data on Bus:		D_0	D_1	D_2	...	D_0	D_1	...
OUT Process:			D_0	D_1	...		D_0	...
IN Process	D_0	D_1	...		D_0	...		

Figure 5-16. Data Prebuffering

5.10.6 SOF Tracking

Functions supporting isochronous pipes must receive and comprehend the SOF token to support prebuffering as previously described. Given that SOFs can be corrupted, a device must be prepared to recover from a corrupted SOF. These requirements limit isochronous transfers to full speed devices only, since low speed devices do not see SOFs on the bus. Also, since SOF packets can be damaged in transmission, devices that support isochronous transfers need to be able to synthesize the existence of an SOF that they may not see due to a bus error.

Isochronous transfers require the appropriate data to be transmitted in the corresponding frame. USB requires that when an isochronous transfer is presented to the host controller, it identifies the frame number for the first frame. The host controller must not transmit the first transaction before the indicated frame number. Each subsequent transaction in the IRP must be transmitted in succeeding frames. If there are no transactions pending for the current frame, then the host controller must not transmit anything for an isochronous pipe. If the indicated frame number is passed, the host controller must skip (i.e., not transmit) all transactions until the one corresponding to the current frame is reached.

5.10.7 Error Handling

Isochronous transfers provide no data packet retries (i.e., no handshakes are returned to a transmitter by a receiver) so that timeliness of data delivery is not perturbed. However, it is still important for the agents responsible for data transport to know when an error occurs and how the error affects the communication flow. In particular, for a sequence of data packets (A,B,C,D), USB allows sufficient information such that a missing packet (A,_,C,D) can be detected and will not unknowingly be turned into an incorrect data or time sequence (A,C,D or A,_,B,C,D). The protocol provides four mechanisms that support this: exactly one packet per frame, SOF, CRC, and bus transaction timeout.

Isochronous transfers require exactly 1 data transaction every frame for normal operation. USB does not dictate what data is transmitted in each frame. The data transmitter/source determines specifically what data to provide. This regular data per frame provides a framework that is fundamental to detecting missing data errors. Any phase of a transaction can be damaged during transmission on the bus. Chapter 8 describes how each error case affects the protocol.

Since every frame is preceded by an SOF packet and a receiver can see SOFs on the bus, a receiver can determine that its expected transaction did not occur between two SOFs. Additionally, since even an SOF packet can be damaged, a device must be able to reconstruct the existence of a missed SOF as described in Section 5.10.6.

A data packet may be corrupted on the bus; therefore, CRC protection allows a receiver to determine that the data packet it received was corrupted.

Finally, the protocol defines the details that allow a receiver to determine via bus transaction timeout that it is not going to receive its data packet after it has successfully seen its token packet.

Once a receiver has determined that a data packet was not received, it may need to know the size of the data that was missed in order to recover from the error with regard to its functional behavior. If the communication flow is always the same data size per frame, then the size is always a known constant. However, in some cases the data size can vary from frame to frame. In this case, the receiver and transmitter have an implementation dependent mechanism to determine the size of the lost packet.

In summary, whether a transaction is actually moved successfully over the bus or not, the transmitter and receiver always advance their data/buffer streams one transaction per frame to keep data per time synchronization. The detailed mechanisms described above allow detection, tracking, and reporting of damaged transactions so that a function or its client software can react to the damage in a function appropriate fashion. The details of that function/application specific reaction are outside the scope of the USB specification.

5.10.8 Buffering for Rate Matching

Given that there are multiple clocks that affect isochronous communication flows in USB, buffering is required to rate match the communication flow across USB. There must be buffer space available both in the device per endpoint and on the host side on behalf of the client software. These buffers provide space for data to accumulate until it is time for a transfer to move over USB. Given the natural data rates of the device, the maximum size of the data packets that move over the bus can also be calculated. Figure 5-17 shows the equations used to determine buffer size on the device and host and maximum packet size that must be requested to support a desired data rate. These equations allow a device and client software design time determined service clock rate (variable X), sample clock rate (variable C), and sample size (variable S). USB only allows one transaction per bus clock. These equations should provide design information for selecting the appropriate packet size that an endpoint will report in its characteristic information and the appropriate buffer requirements for the device/endpoint and its client software. Figure 5-14 shows actual buffer, packet, and clock values for a typical isochronous example.

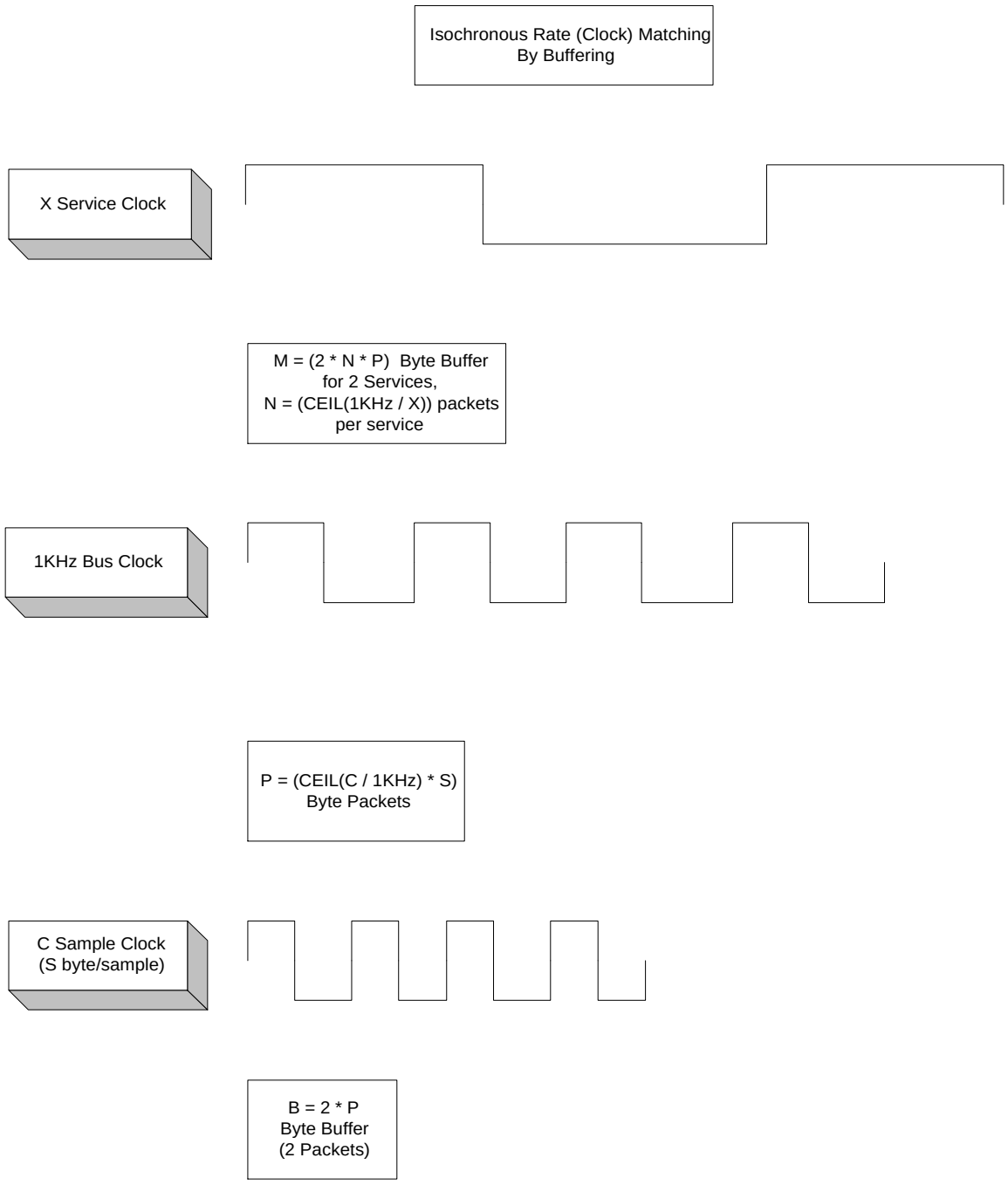


Figure 5-17. Packet and Buffer Size Formulas for Rate Matched Isochronous Transfers

The USB data model assumes that devices have some natural sample size and rate. USB supports the transmission of packets that are multiples of sample size to make error recovery handling easier when isochronous transactions are damaged on the bus. If a device has no natural sample size or if its samples are larger than a packet, it should describe its sample size as being one byte. If a sample is split across a data packet, the error recovery can be harder when an arbitrary transaction is lost. In some cases, data synchronization can be lost unless the receiver knows in what frame number each partial sample is transmitted. Furthermore, if the number of samples can vary due to clock correction (e.g., for a non-derived device clock), it may be difficult or inefficient to know when a partial sample is transmitted. Therefore, USB does not split samples across packets.

Chapter 6
Mechanical

This chapter provides the mechanical specification for the cables and connectors for USB hubs, functions, and hosts. The specification includes the dimensions, materials, electrical, and reliability requirements.

6.1 Architectural Overview

The physical topology of a USB channel consists of connecting a hub or function to another hub, function, or host. There are two speeds at which the channel can operate. The fully rated speed of 12 Mbs requires the use of a shielded cable with two internal power conductors and two internal signal conductors. For lower cost and lower speed, a sub-channel at 1.5 Mbs is allowed by the specification with the use of unshielded cabling.

Two types of plugs and receptacles are specified for USB: series A and series B. Series A plugs and receptacles are to be used for those devices on which the external cable is permanently attached to devices such as keyboards, mice, and hubs. Series B plugs and receptacles are to be used for those devices that require an external connector so that the USB cabling is detachable, such as printers, scanners, and modems. There may be internal connectors that will need to meet the electrical requirements of the USB specification, but the mechanical aspects of the internal connector are not part of the USB specification. All cables that have a series A or series B connector should meet the construction requirements of the fully rated channel and have maximum gauge power conductors.

Series A and B connectors cannot be interchanged; therefore, there is no possibility that the integrity of the bus will be compromised.

6.2 Dimensioning Requirements

Default tolerances are listed in Table 6-1, unless otherwise specified. The dimensions are in millimeters.

Table 6-1. Default Tolerances

Over 1 to 5	Over 5 to 30	Over 30 to 100	Over 100 to 300	Over 300 to 1000	Over 1000 to 3000	Over 3000 to 5000
±0.3	±0.4	±0.6	±0.8	±1.6	±2.5	±10

6.3 Cable

All hubs and functions as defined in this specification will have one permanently attached cable or be terminated with a series B connector.

The standard USB cable will consist of one pair of 20-28 AWG wire for power distribution with another 28 AWG pair twisted, with a shield and overall jacket. This will be used for typical peripherals operating at the rated 12 Mbs signaling.

An alternative cable of identical gauge but without the twisted conductors and shield can be used for 1.5 Mbs signaling. This will be used in a sub-channel application where the wider bandwidth is not needed.

In all other respects, the mechanical specifications for the sub-channel will be identical to the fully rated specification.

6.3.1 Cable Specification

This specification defines the detailed requirements of a twisted pair, 28 AWG, PVC, round cable with two power leads (non twisted) for fully rated devices as well as a four-conductor cable with an overall jacket for the sub-channel devices.

6.3.1.1 Applicable Documents

Underwriters' Laboratory, Inc.

UL-STD-94	Tests for Flammability of Plastic Materials for Parts in Devices and Appliances
UL-Subject-444	Communication Cables

American Standard Test Materials

ASTM-D-4565	Physical and Environmental Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, Test Standard Method
ASTM-D-4566	Electrical Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, Test Standard Method

6.3.1.2 Requirements

Mechanical

Material/Finish:

Outer Jacket: Polyvinyl chloride (PVC)

Color: Recommended; frost white

Conductor Insulation: Semi-rigid PVC for power conductors and polyethylene (optionally foamed) or equivalent meeting the requirements of Table 6-4 for the signal pair (fully rated 12 Mbs only).

Conductors: Refer to Table 6-2 for power distribution conductors. The signaling conductor pair is 28 AWG.

Table 6-2. Conductors - Pair for Power Distribution

Gauge and Conductor Outer Diameter
28 AWG - .84 ±.05 mm
26 AWG - 1.00 ±.05 mm
24 AWG - 1.10 ±.07 mm
22 AWG - 1.30 ±.07 mm
20 AWG - 1.50 ±.08 mm

Cable Construction:

Fully Rated: Cable shall consist of four conductors; one twisted pair with 28 AWG conductors (data pair), one non-twisted pair (power distribution pair) with an overall jacket. The twisted pair shall have one twist per 6-8 cm.

Sub-Channel: Cable shall consist of four conductors: one pair with 28 AWG conductors (data pair) and one pair for power distribution with an overall jacket.

Outer Jacket:
A. Outside Diameter:
Fully rated and sub-channel: 3.4 to 5.3 mm.

B. Color: Frost white recommended.

Conductor Insulation:
A. Outside Diameter: Refer to Table 6-2.
B. Color: Refer to Table 6-5.

Conductors:

Fully Rated:

A. 28 AWG stranded - twisted pair
B. Non twist - One pair per Table 6-2 stranded selected as needed for proper DC power distribution.
C. Shield: Required for EMI compliance. Suggest aluminized mylar wrap with a 28 AWG drain wire and 65% minimum. coverage tinned copper mesh over the foil.

Sub-Channel:

A. 28 AWG stranded - One pair.
B. Non Twist - One pair per Table 6-2 stranded selected as needed for proper DC power distribution.

Break Strength: 45 Newtons minimum when tested in accordance with ASTM-D-4565.

Electrical:

Voltage Rating: 30 V (rms) maximum.

Conductor Resistance: As shown in Table 6-3, when tested in accordance with ASTM-D-4565. Refer to Section 6.4 for limitations on DC voltage drop.

Table 6-3. Conductor Resistance

Gauge	DC Resistance (max.)
28	0.232 Ω/m
26	0.145 Ω/m
24	0.0909 Ω/m
22	0.0574 Ω/m
20	0.0358 Ω/m

Resistance Unbalance: The resistance unbalance between the two conductors shall not exceed 5% when tested in accordance with ASTM-D-4566.

Length: Maximum cable length shall not exceed 3 meters for the sub-channel and 5 meters for the fully rated channel.

Fully rated only:
Attenuation: The attenuation of the signal pair measured in accordance with ASTM-D-4566 shall not exceed the values in Table 6-4.

Characteristic Impedance: The characteristic impedance of the signal pair shall be 90 Ω ±15%, when measured in accordance with ASTM-D-4566 over the frequency range of 1-16 MHz.

Propagation delay of the fully rated twisted signal pair must be equal to or less than 30 ns over the length of cable used in the frequency range of 1-16 MHz. Refer to Section 6.5 if the cabling cannot meet this requirement. Note that the signal pair might require a foamed polyethylene insulation on the signal pair to meet the propagation delay requirements for a full length cable.

Skew: The maximum skew between the two signal pair must be less than 65.6 ps/m at 10 kHz.

Table 6-4. Signal Attenuation

Frequency (MHz)	Attenuation (maximum) dB/305 m
0.064	4.80
0.256	6.70
0.512	8.20
0.772	9.40
1.000	12.0
4.000	24.0
8.000	35.0
10.000	38.0
16.000	48.0

Environmental:

Temperature Rating: -40 °C to 60 °C storage; 0 °C to 40 °C operating.

Laboratory Approvals: Item shall be UL listed per UL Subject 444. Class 2, Type CM for Communication Cable Requirements.

Flammability: Plastic material used in the construction of this item shall meet the flammability requirements of NEC Article 800.

Marking: Item shall be legibly and permanently marked with the vendor name or symbol, UL File Number, Type CM (UL).

Qualification: All suppliers, when requested, must be able to supply appropriate documentation to show conformance to the requirements of this chapter.

All electrical measurements should be made with a sample cable removed from the reel or container. The cable must rest on a non-conductive surface or be on aerial supports.

Table 6-5. Cable Color Code

Wire	Color
+ Data	Green
- Data	White
VCC	Red
Ground	Black

6.3.2 Connector (Series A)

Figures 6-1 through 6-7 describe the Series A connector.

6.3.2.1 Plug (Series A)

The USB (Series A) plug is a four-position plug with a shielded housing compatible with the cabling as described in Section 6.3. The following guidelines ensure intermateability. The recommended color is frost white for the overmold. Internal plastic features can be frost white or equivalent.

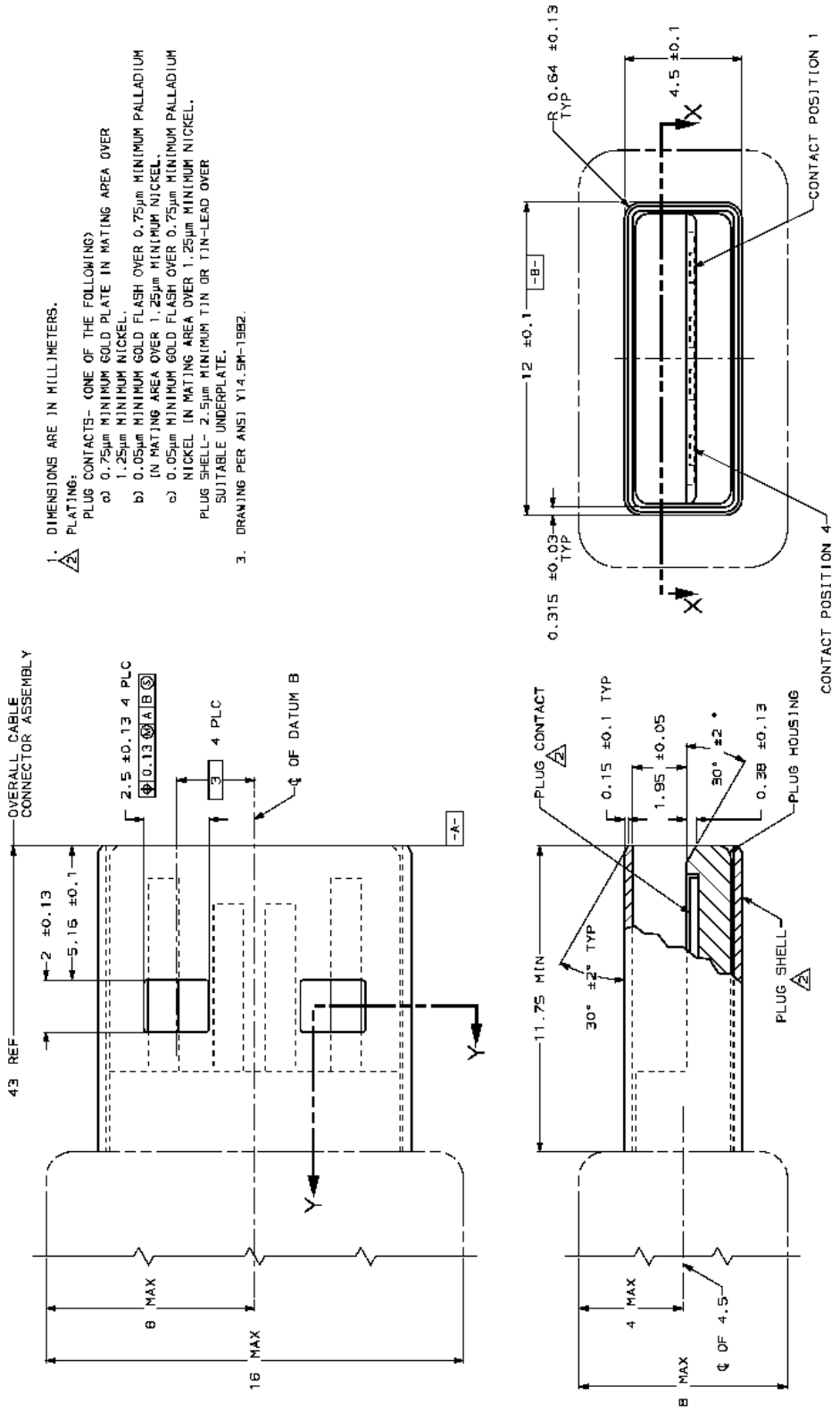


Figure 6-1. Plug Connector (Series A)

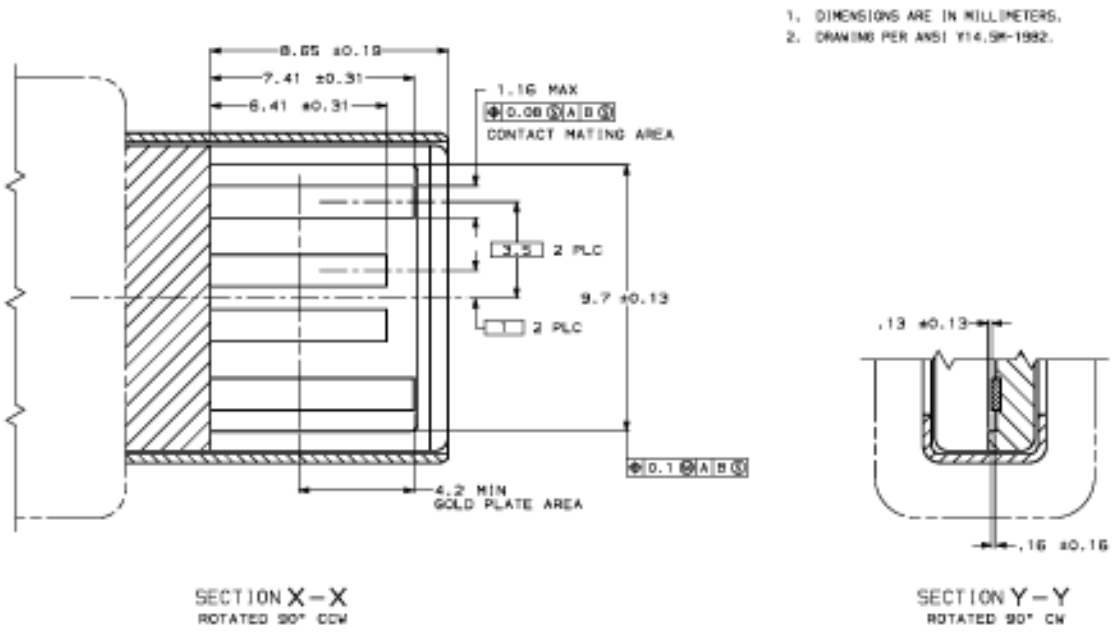


Figure 6-2. Plug Contact Detail (Series A)

The termination of the conductors to the plug contacts may be done as deemed appropriate by the connector's manufacturing process.

6.3.2.2 Receptacle (Series A)

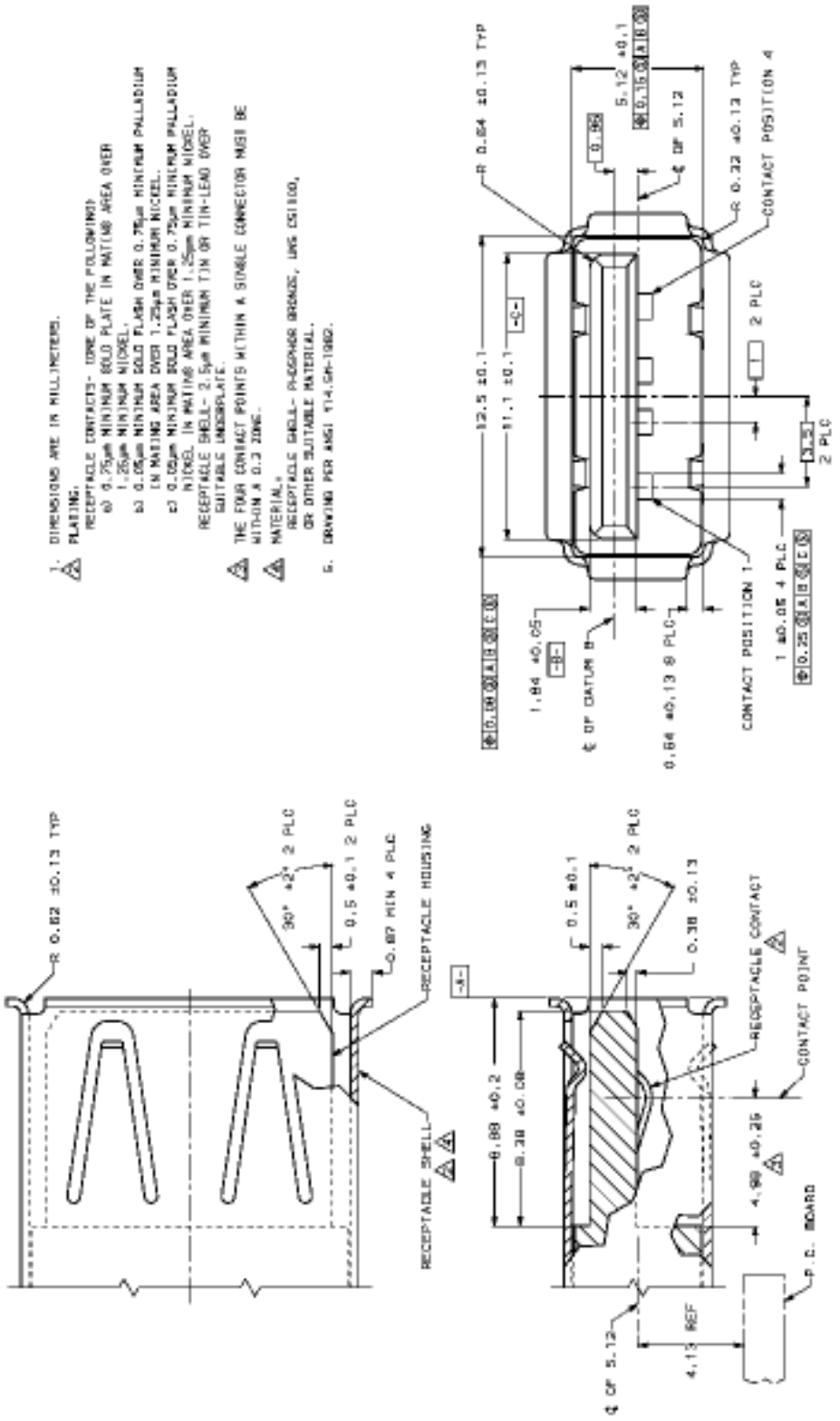


Figure 6-3. Receptacle (Series A)

There are four variants of the receptacle available for general use. They are vertical, right angled, panel mount, and stacked right angled with SMT as well as through hole variants. However, as long as the interface requirements of the specification are met, it is up to the implementer as to what form the receptacle will take. Internal plastic features should be frost white or equivalent.

6.3.2.3 Connector Mating Features (Series A)

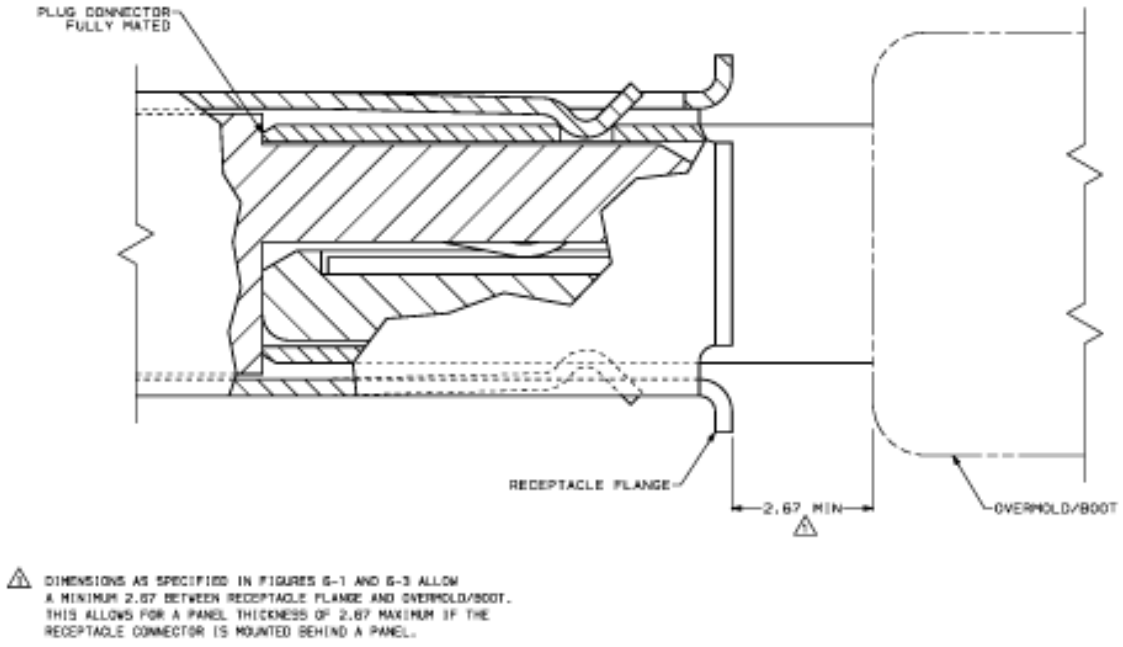
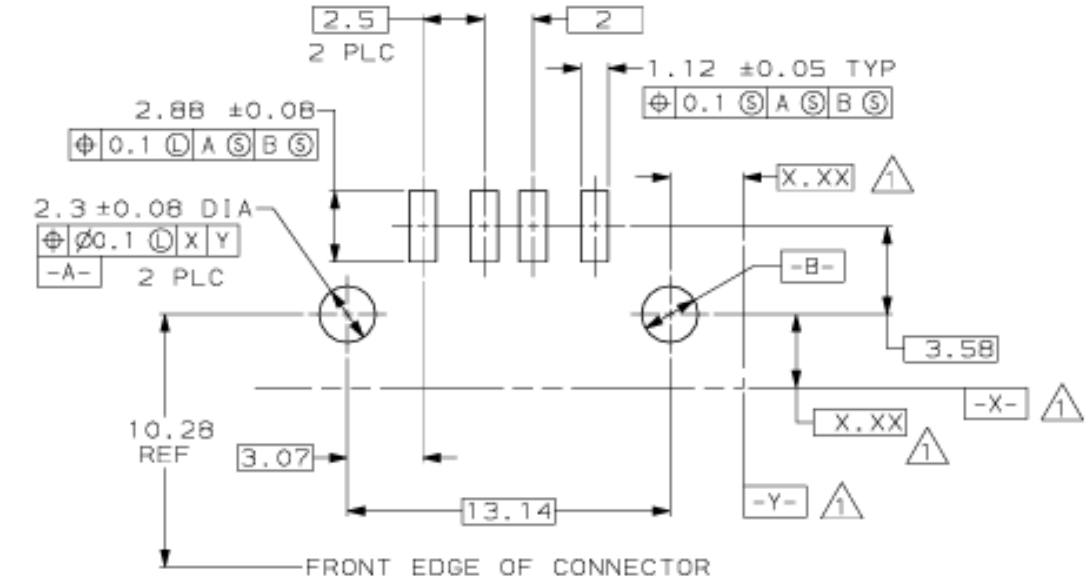


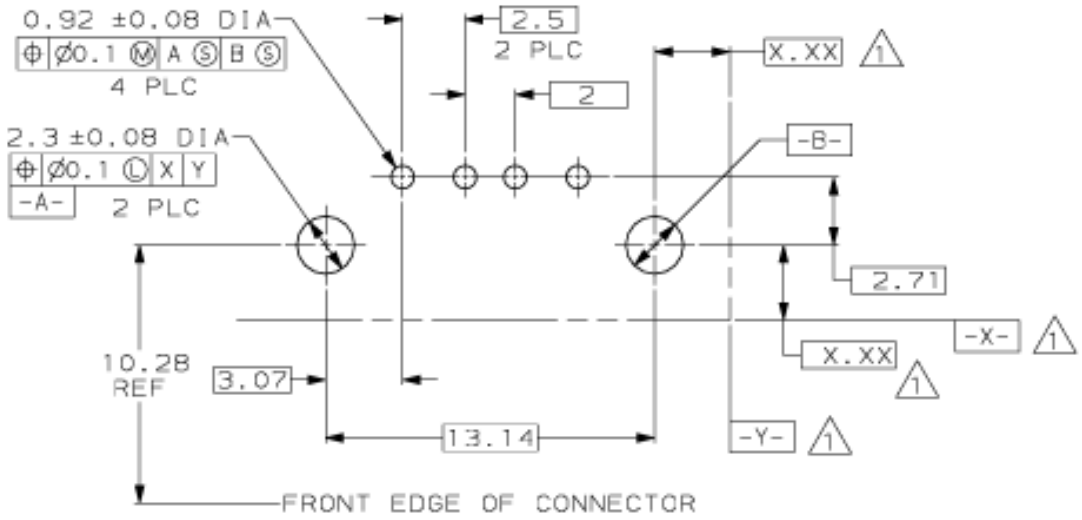
Figure 6-4. Connector Mating Features (Series A)

6.3.2.4 Receptacle PWB Foot Print (Series A)



1. DATUM AND BASIC DIMENSIONS ESTABLISHED BY CUSTOMER.
2. RECOMMENDED PC BOARD THICKNESS OF 1.57
3. DRAWING PER ANSI Y14.5M-1982.

Figure 6-5. PWB Footprint for Receptacle, SMT (Series A)



1. DATUM AND BASIC DIMENSIONS ESTABLISHED BY CUSTOMER.
2. RECOMMENDED PC BOARD THICKNESS OF 1.57
3. DRAWING PER ANSI Y14.5M-1982.

Figure 6-6. PWB Footprint for Receptacle, Throughhole (Series A)

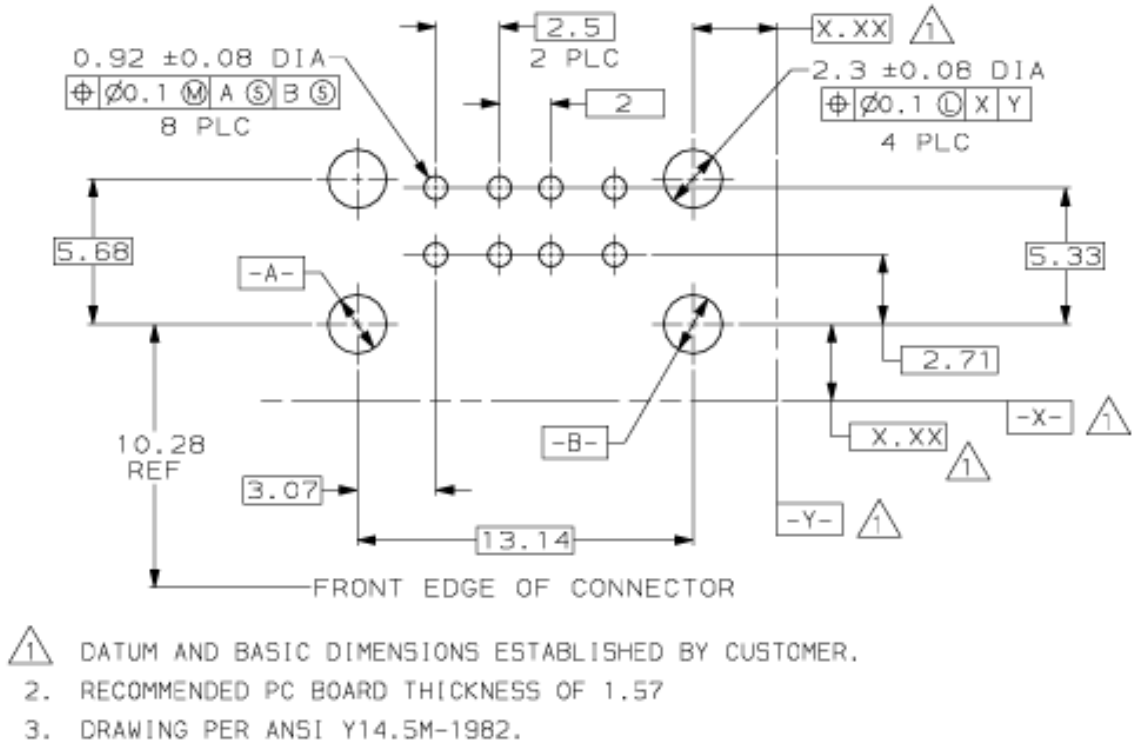


Figure 6-7. PWB Footprint for Receptacle, Stacked Right Angle (Series A)

6.3.3 Connector (Series B)

Figures 6-8 through 6-12 describe the Series B connector.

6.3.3.1 Plug (Series B)

The USB (Series B) plug is a four position plug with shielded housing compatible with the cabling as described in Section 6.3. The following guidelines ensure intermateability. The recommended color is frost white for the overmold. The internal features can be frost white or equivalent.

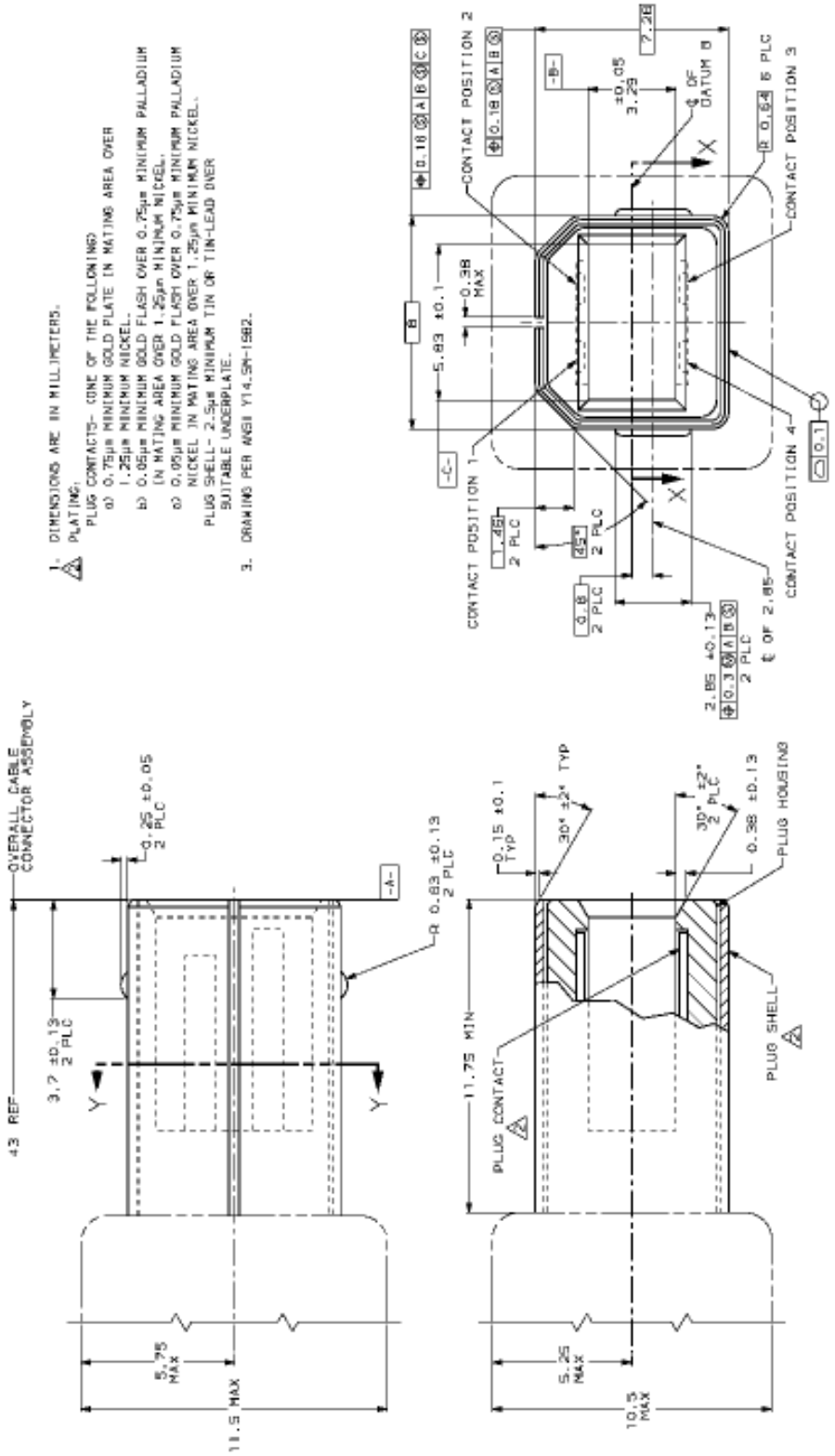


Figure 6-8. Plug Connector (Series B)

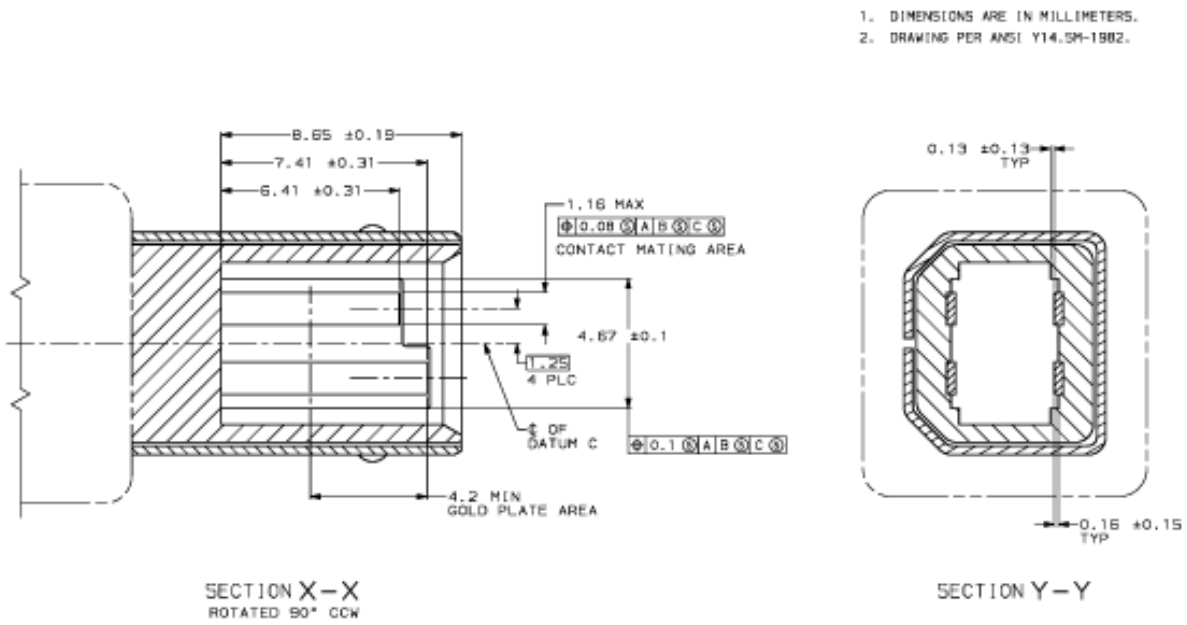


Figure 6-9. Plug Contact Detail (Series B)

6.3.3.2 Receptacle (Series B)

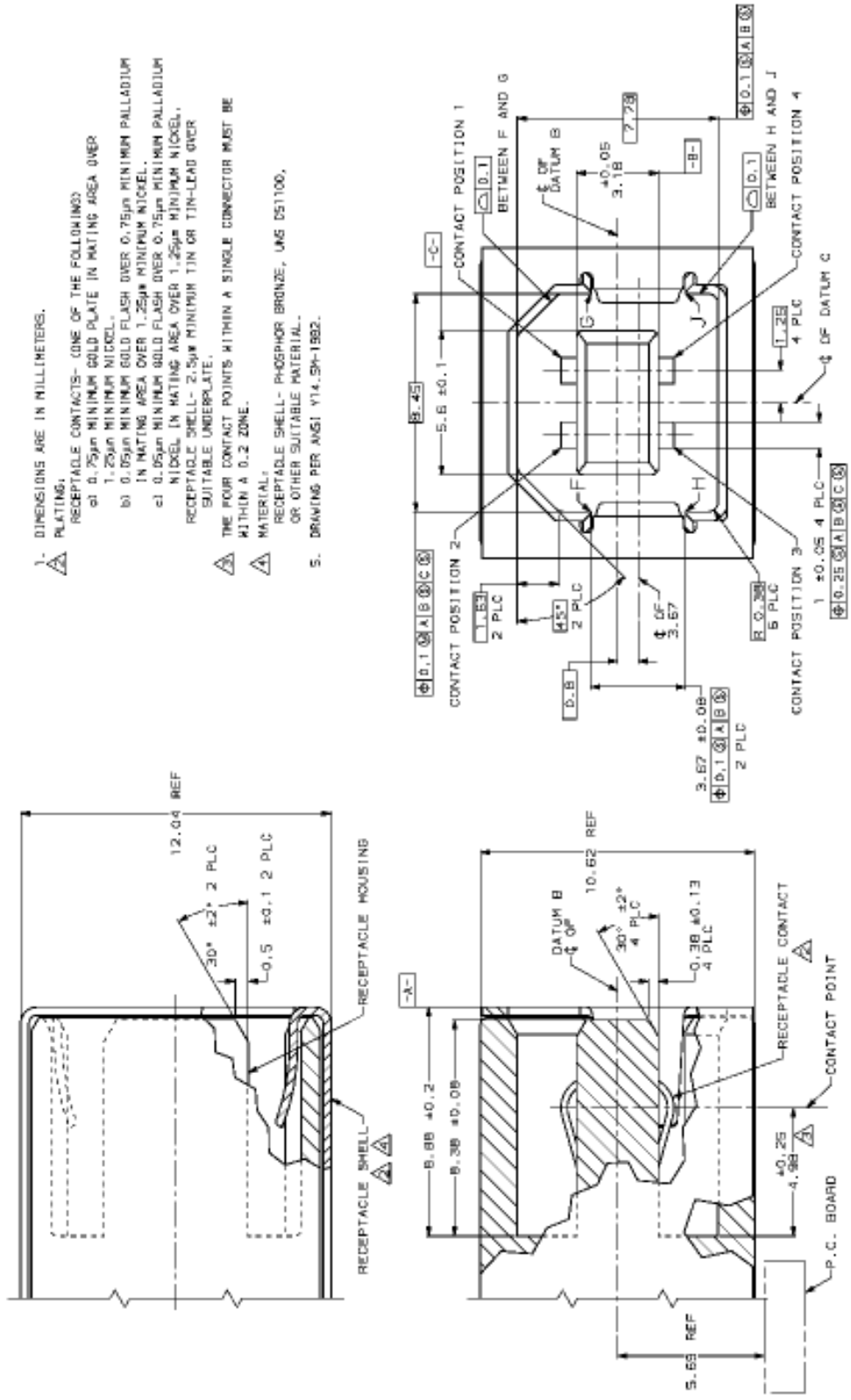


Figure 6-10. Receptacle (Series B)

6.3.3.3 Connector Mating Features (Series B)

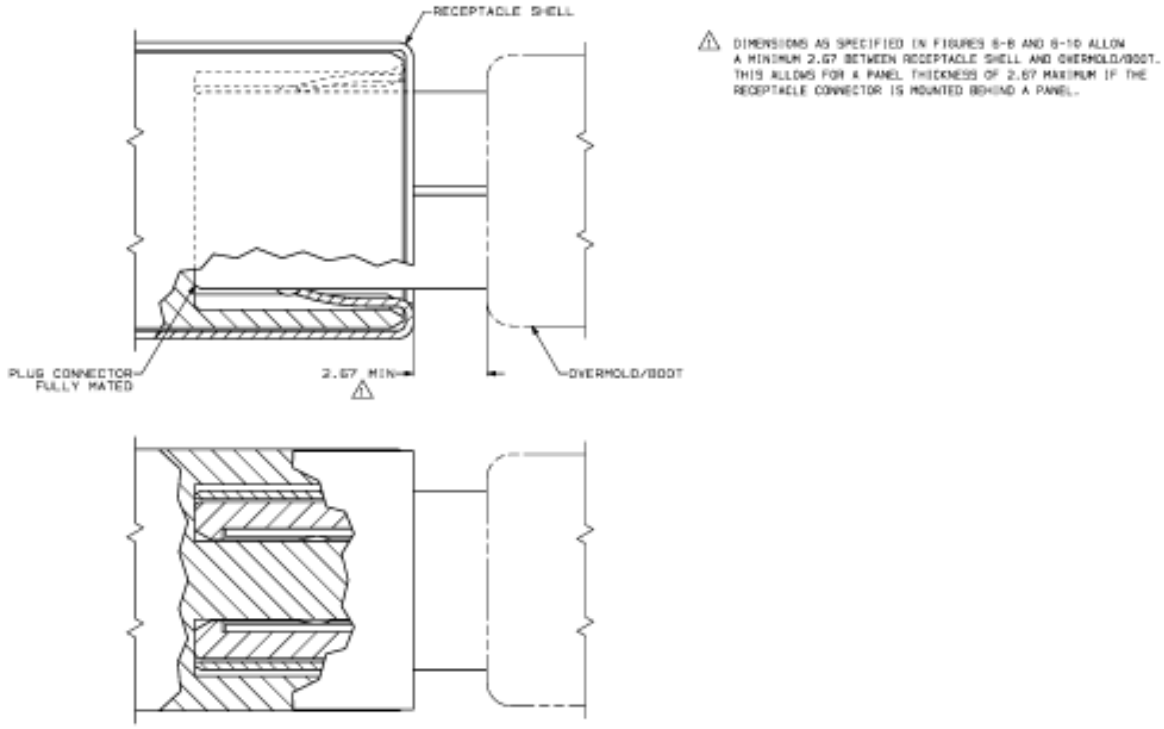
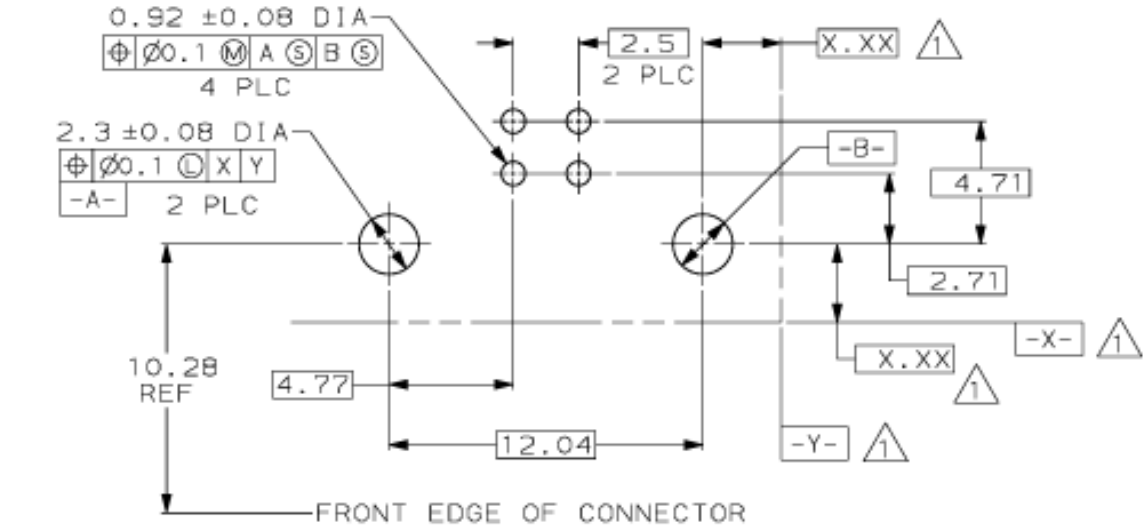


Figure 6-11. Connector Mating Features (Series B)

6.3.3.4 Receptacle PWB Foot Print (Series B)



1. DATUM AND BASIC DIMENSIONS ESTABLISHED BY CUSTOMER.
2. RECOMMENDED PC BOARD THICKNESS OF 1.57
3. DRAWING PER ANSI Y14.5M-1982.

Figure 6-12. PWB Footprint for Receptacle, Throughhole (Series B)

6.3.4 Serial Bus Icon

The USB icon, shown in Figure 6-13, should be molded into the connector and also placed on the product for ease of identifying the USB port. It is recommended that the icon on the product and the one on the plug be adjacent to each other when the plug and receptacle are mated. This icon can be used for both series A and B connector schemes. On the plug, there should be a 0.635 mm rectangular recessed area around the icon such that there is a perceptible feel of the icon.

Figure 6-13. USB Icon Artwork

6.3.5 Plug/Receptacle Mechanical and Electrical Requirements

6.3.5.1 Contact Numbering (Series A and B)

Table 6-6. Contact Numbering

Contact Number	Signal Name	Comment
1	VCC	Cable power
2	- Data	
3	+ Data	
4	Ground	Cable ground

6.3.5.2 Ratings

Voltage: 30 Vac (rms).

Current: 1 A maximum per contact not to exceed 30 °C temperature rise.

Temperature: -40 °C to 60 °C storage; 0 °C to 40 °C operating.

6.3.5.3 Performance and Test Description

Product is designed to meet electrical, mechanical, and environmental performance requirements specified in Table 6-7. Unless otherwise specified, all tests shall be performed at ambient environmental conditions. Cable construction and/or part number used for testing must be included with test report.

Table 6-7. Test Requirements and Procedures Summary

Test Description	Requirement	Procedure
Examination of product	Meets requirements of Section 6.3	Visual, dimensional, and functional compliance
ELECTRICAL		
Termination resistance	30 mΩ maximum	EIA 364-23 Subject mated contacts assembled in housing to 20 mV maximum open circuit at 100 mA maximum. See Figure 6-14.
Insulation resistance	1000 MΩ minimum	EIA 364-21 Test between adjacent contacts of mated and unmated connector assemblies
Dielectric withstanding voltage	750 Vac at sea level	EIA 364-20 Test between adjacent contacts of mated and unmated connector assemblies
Capacitance	2 pF maximum	EIA 364-30 Test between adjacent circuits of unmated connectors at 1 kHz
MECHANICAL		
Vibration, random	No discontinuities of 1 μs or longer duration. See Note.	EIA 364-28 Condition V Test letter A. Subject mated connectors to 5.35 G's rms. Fifteen minutes in each of three mutually perpendicular planes. See Figure 6-15.
Physical shock	No discontinuities of 1 μs or longer duration. See Note.	EIA 364-27 Condition H. Subject mated connectors to 30 G's half-sine shock pulses of 11 ms duration. Three shocks in each direction applied along three mutually perpendicular planes, 18 total shocks. See Figure 6-15 for the test setup.
Durability	See Note.	EIA 364-09 Mate and unmate connector assemblies for 1500 cycles at maximum rate of 200 cycles per hour

Table 6-7. Test Requirements and Procedures Summary (Continued)

Test Description	Requirement	Procedure
Mating force	35 Newtons maximum	EIA 364-13 Measure force necessary to mate connector assemblies at maximum rate of 12.5 mm per minute.
Unmating force	10 Newtons minimum	EIA 364-13 Measure force necessary to unmate connector assemblies at maximum rate of 12.5 mm per minute.
Cable Retention	Cable shall not dislodge from cable crimp.	Apply axial load of 25 Newtons to the cable.
ENVIRONMENTAL		
Thermal shock	See Note.	EIA 364-32 Test Condition I. Subject mated connectors to five cycles between -55 °C and 85 °C.
Humidity	See Note.	EIA-364-31 Method II Test Condition A. Subject mated connectors to 96 hours at 40 °C with 90 to 95% RH.
Temperature life	See Note.	EIA-364-17 Test Condition 3 Method A. Subject mated connectors to temperature life at 85 °C for 250 hours.

Note:

Shall meet visual requirements, show no physical damage, and shall meet requirements of additional tests as specified in the test sequence listed in Table 6-8.

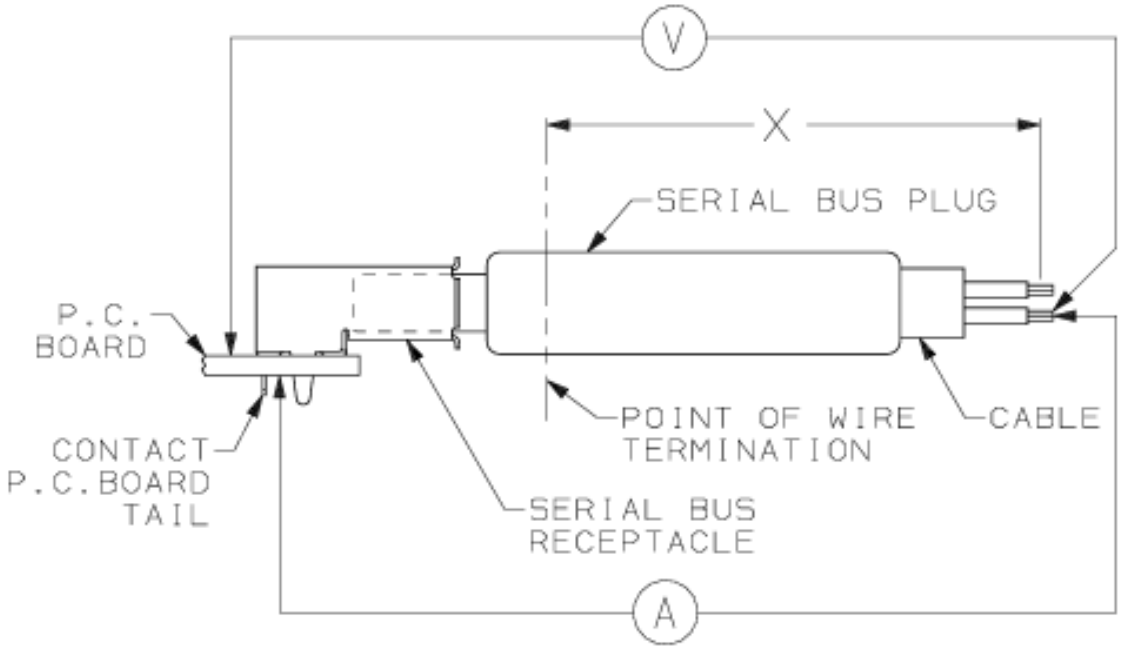
Table 6-8. Product Qualification Test Sequence

Test or Examination	Test Group (a)		
	1	2	3
	Test Sequence (b)		
Examination of product	1,10	1,5	1,9
Termination resistance	3,7	2,4	
Insulation resistance			3,7
Dielectric withstanding voltage			4,8
Capacitance			2
Vibration	5		
Physical shock	6		
Durability	4		
Mating force	2		
Unmating force	8		
Thermal shock			5
Humidity			6
Cable Retention	9		
Temperature life		3(c)	

Notes:
(a) Refer to Section 6.3.5.4.
(b) Numbers indicate sequence in which tests are performed.
(c) Precondition samples with 10 cycles durability.

6.3.5.4 Sample Selection

Samples shall be prepared in accordance with applicable manufacturers' instructions and shall be selected at random from current production. Test groups 1, 2, and 3 shall consist of a minimum of eight connectors. A minimum of 30 contacts shall be selected and identified. Unless otherwise specified, these contacts shall be used for all measurements.



1. RESISTANCE DUE TO X INCHES OF WIRE IS TO BE REMOVED FROM ALL READINGS.

Figure 6-14. Termination Resistance Measurement Points

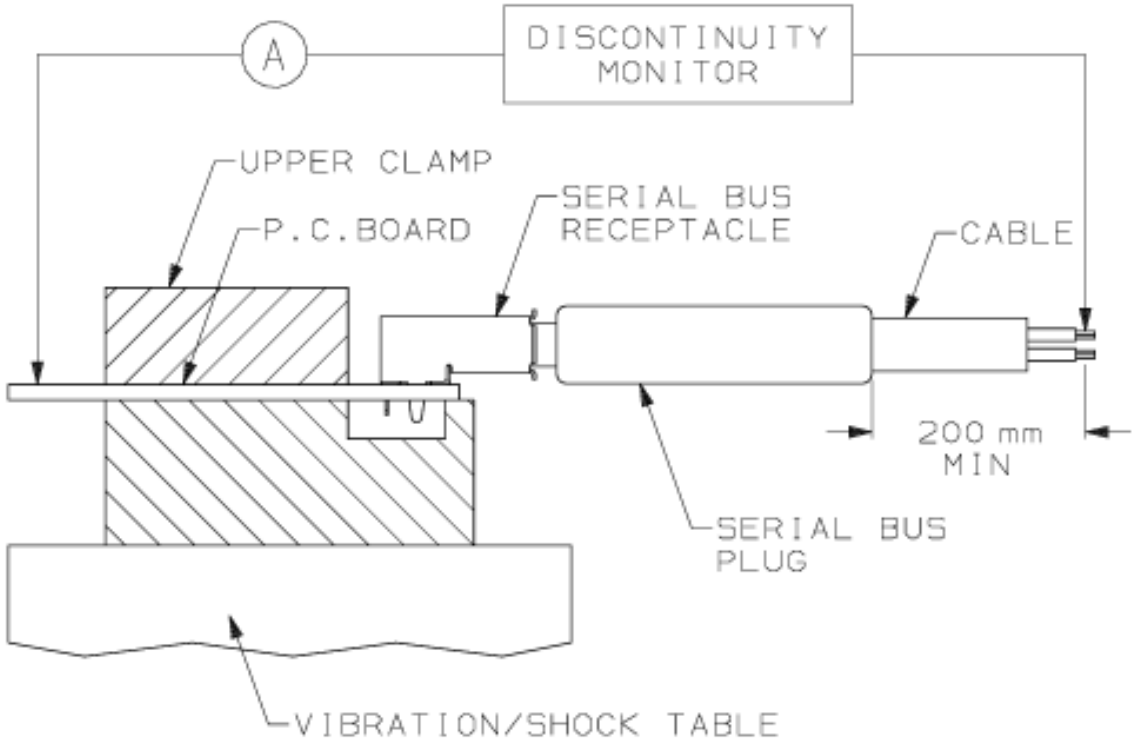


Figure 6-15. Vibration and Physical Shock Mounting Fixture

6.3.5.5 Additional Requirements

Flammability: Plastic material used in the construction of this item, shall be rated 94V-0, per UL-STD-94.

Marking: USB icon per Figure 6-13 on plug. Recommended that OEM's add an icon near the receptacle on end product where possible or practical.

Qualification: All suppliers when requested must be able to supply appropriate documentation to show conformance to the requirements of this chapter.

6.4 Cable Voltage Drop Requirements

The USB physical layer specification requires that the maximum power distribution voltage drop between two hubs or between hubs and functions should be 350 mV max. The table below lists the nominal lengths of power distribution cabling for each gauge of conductor. The following is a formula for the voltage drop to an unpowered hub of 350 mV.

$$V_{unpowered\_hub} = V_{switch} + 4 * V_{connector} + 2 * V_{cable}$$

Where:

$V_{switch} = I_{max}$ * (board resistance and FET resistance) = 100 mV (max. by definition)

$V_{connector} = I_{max}$ * 30 mΩ (connector resistance) = 15 mV

$V_{cable} = I_{max}$ * cable resistance

I_{max} = 500 mA

With the above information, V_{cable} = 95 mV, assuming two connectors are used in the cable assembly,

For a 95 mV drop using copper wire at 20 °C, Table 6-9 lists cable lengths with a current of 500 mA. Figure 6-16 shows the voltage drop distribution.

Table 6-9. Cable Lengths vs. Gauge

Gauge	Resistance	Length (Max.)
28	0.232 Ω/m	.81 m
26	0.145 Ω/m	1.31 m
24	0.091 Ω/m	2.08 m
22	0.057 Ω/m	3.33 m
20	0.036 Ω/m	5.00 m

Note: This table does not include additional temperature effects (approximately 10%).

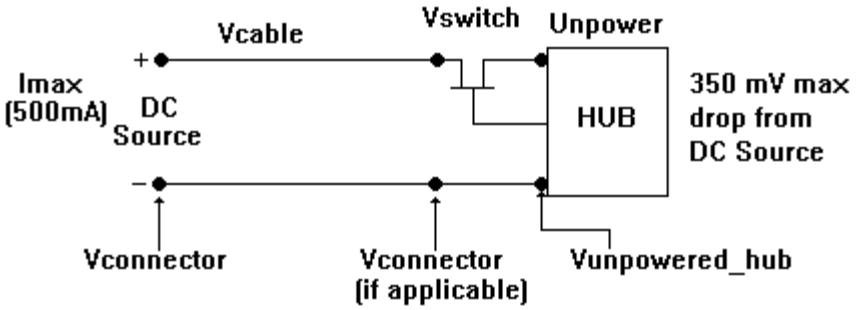


Figure 6-16. Cable and Connector Voltage Drop Distribution

It is recommended that each individual implementer verify proper DC voltage drop. If the implementer uses different materials than above, then it is responsible for proper DC voltage at the unpowered hub.

To meet the 5 meter maximum length requirement of this specification, a wire range of 20 AWG to 28 AWG is needed for the DC power distribution conductors.

Note: For typical functions that do not require 500 mA, smaller wire gauges can be used as appropriate per the voltage drop requirements.

6.5 Propagation Delay

If the cabling you have selected cannot meet the requirements of Section 6.3.1.2, then use Table 6-10 to limit the cable length for fully rated channels.

Table 6-10. Propagation Delay vs. Cable Length

Cable Propagation Delay Specification	Maximum Cable Length
9.0 ns/m	3.3 m
8.0 ns/m	3.7 m
7.0 ns/m	4.3 m
6.5 ns/m	4.6 m

Note: The implementation must use the shortest cable that meets the requirements of Section 6.3.1.2, Section 6.4, and Section 6.5.

6.6 Grounding

The shield must be terminated to the connector plug for completed assemblies. At the host end, the shield, DC power, and chassis ground should be bonded together. The complete bus should have only one DC ground point at the host end. All other devices should not connect the shield or DC return to chassis ground. This prevents circulating low frequency currents. However, AC coupling is permitted for EMI compliance. The coupling impedance must be less than 250 kΩ at 60 Hz and not greater than 15 Ω between 3 and 30 MHz. The dielectric voltage rating of the capacitor must be 250 Vac (rms).

6.7 Regulatory Information

Recommendation and guidelines for the installation of this cabling per applicable local regulations are the responsibility of the OEM. It is recommended that guidelines such as EIA CB8-1981[4] and ANSI/NFPA 70-1984 as well as local codes and regulations be followed.

Chapter 7
Electrical

This chapter describes the electrical specification for the USB. It contains signaling, power distribution, and physical layer specifications.

7.1 Signaling

The signaling specification for the USB is described in the following subsections.

7.1.1 USB Driver Characteristics

The USB uses a differential output driver to drive the USB data signal onto the USB cable. The static output swing of the driver in its low state must be below the V_{OL} of 0.3 V with a 1.5 kΩ load to 3.6 V and in its high state must be above the V_{OH} of 2.8 V with a 15 kΩ load to ground as listed in Table 7-4. The output swings between the differential high and low state must be well balanced to minimize signal skew. Slew rate control on the driver is required to minimize the radiated noise and cross talk. The driver's outputs must support three-state operation to achieve bi-directional half duplex operation. High impedance is also required to isolate the port from downstream devices that are being hot inserted or which are connected but powered down. The driver must tolerate a voltage on the signal pins of -0.5 V to 3.8 V with respect to local ground reference without damage. It must tolerate this voltage for 10.0 µs while the driver is active and driving, and tolerate the condition indefinitely when the driver is in its high impedance state.

7.1.1.1 Full Speed (12 Mbs) Driver Characteristics

A full speed USB connection is made through a shielded, twisted pair cable with a characteristic impedance (Z_0) of 90 Ω ±15% and a maximum length of 5 meters. The impedance of each of the drivers must be between 29 Ω and 44 Ω. The data line rise and fall times must be between 4 ns and 20 ns, smoothly rising or falling (monotonic), and be well matched to minimize RFI emissions and signal skew.

For a CMOS implementation, the driver impedance will typically be realized by a CMOS driver with an impedance significantly less than this resistance with a discrete series resistor making up the balance. Figure 7-1 shows an example of how a full speed driver might be done using two identical CMOS buffers which have an output impedance between 3 Ω and 15 Ω and two series resistors of 27 Ω. Figure 7-2 shows the full speed driver signal waveforms.

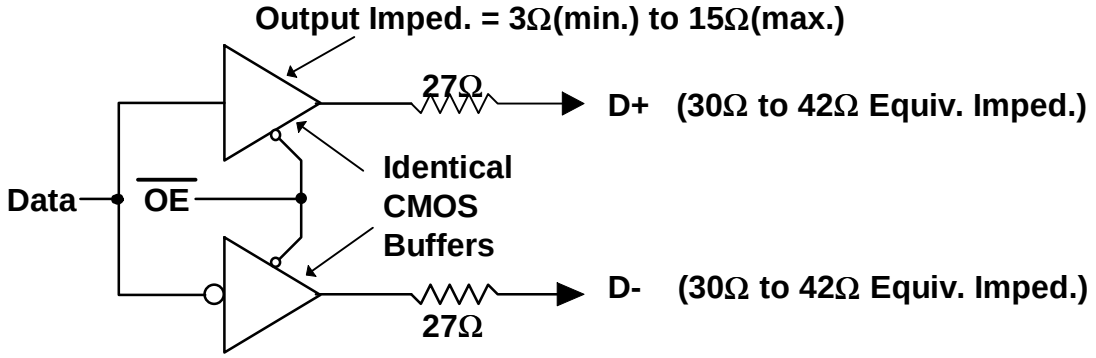


Figure 7-1. Example Full Speed CMOS Driver Circuit

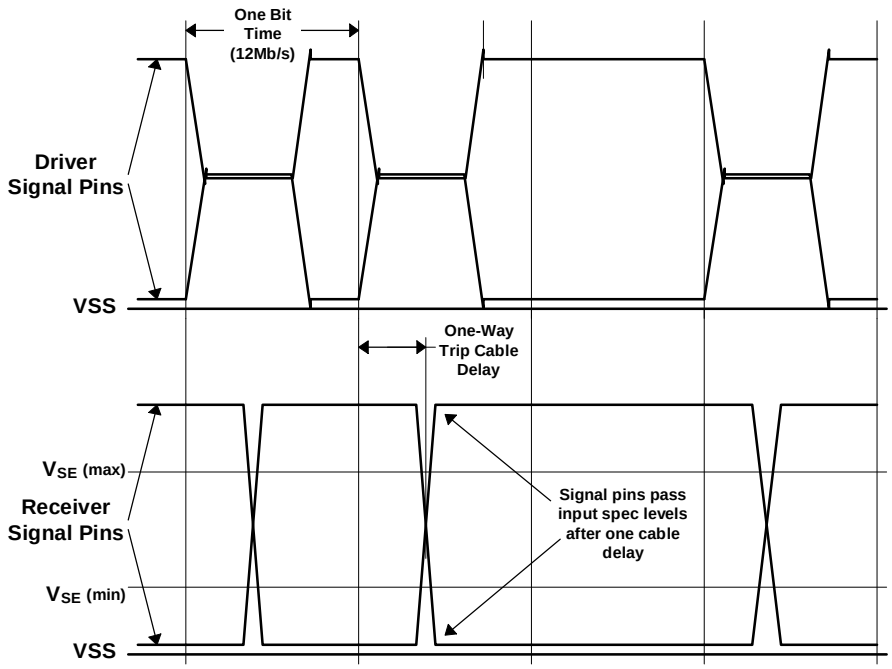


Figure 7-2. Full Speed Driver Signal Waveforms

7.1.1.2 Low Speed (1.5 Mbs) Driver Characteristics

A low speed USB connection is made through an unshielded, untwisted wire cable a maximum of 3 meters in length. The rise and fall time of the signals on this cable must be greater than 75 ns to keep RFI emissions under FCC class B limits, and less than 300 ns to limit timing delays and signaling skews and distortions. The driver must reach the specified static signal levels with smooth rise and fall times, and minimal reflections and ringing when driving the cable (see Figure 7-3). This cable and driver are used only on network segments between low speed devices and the ports to which they are connected.

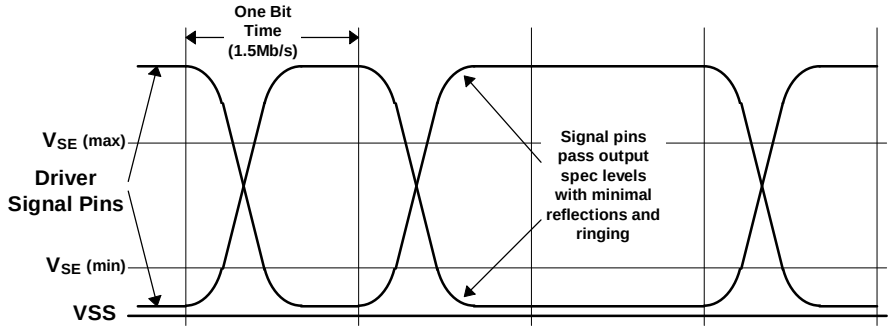


Figure 7-3. Low Speed Driver Signal Waveforms

7.1.1.3 Driver Usage

Full speed buffers are used on the upstream ports (towards the host) of all hubs and full speed functions. All devices with hubs must be full speed devices. A full speed driver will be used to send data at both and low speed data rates. However, the signaling always uses full speed signaling conventions (refer to Table 7-1) and edge rates. Running at low speed data rates does not change the driver's characteristics.

Low speed buffers are used on the upstream ports of low speed functions. The downstream ports of all hubs (including the host) are required to be capable of both driver characteristics, such that any type of

device can be plugged in to these ports (see Figure 7-5 and Figure 7-6). Low speed drivers only send at the low speed data rate using low speed signaling conventions (refer to Table 7-1) and edge rates.

7.1.2 Receiver Characteristics

A differential input receiver must be used to accept the USB data signal. The receiver must feature an input sensitivity of at least 200 mV when both differential data inputs are in the range of at least 0.8 V to 2.5 V with respect to its local ground reference. This is called the common mode input voltage range. Proper data reception is also required when the differential data lines are outside the common mode range, as shown in Figure 7-4. The receiver must tolerate static input voltages between -0.5 V to 3.8 V with respect to its local ground reference without damage. In addition to the differential receiver, there must be a single-ended receiver for each of the two data lines. The receivers must have a switching threshold between 0.8 V and 2.0 V (TTL inputs). It is recommended that the single-ended receiver have some hysteresis to reduce its sensitivity to noise.

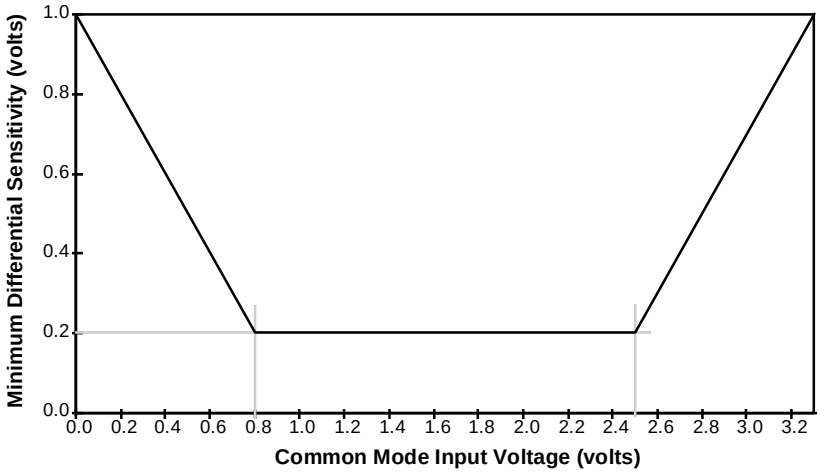


Figure 7-4. Differential Input Sensitivity Over Entire Common Mode Range

7.1.3 Signal Termination

The USB is terminated at the hub and function ends as shown in Figure 7-5 and Figure 7-6. Full speed and low speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable. Full speed devices are terminated as shown in Figure 7-5 with the pull-up on the D+ line. Low speed devices are terminated as shown in Figure 7-6 with the pull-up on the D- line.

The pull-up terminator is a 1.5 kΩ ±5% resistor tied to a voltage source between 3.0 V and 3.6 V referenced to the local ground. The pulldown terminators are resistors of 15 kΩ ±5% connected to their local ground.

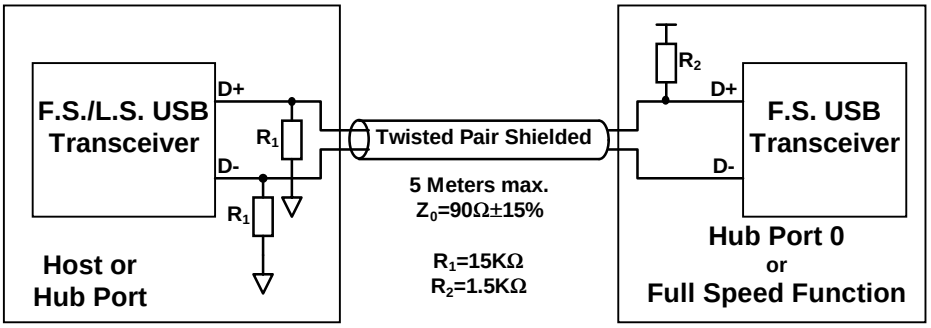


Figure 7-5. Full Speed Device Cable and Resistor Connections

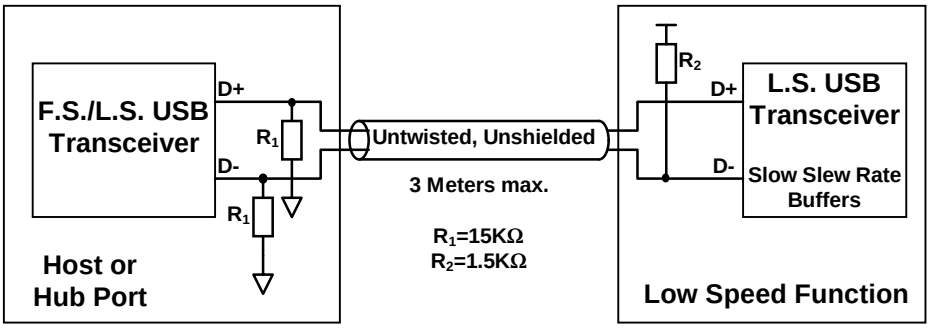


Figure 7-6. Low Speed Device Cable and Resistor Connections

7.1.4 Signaling Levels

Table 7-1 summarizes the USB signaling levels. The signaling levels are described below and in the following sections.

Table 7-1. Signaling Levels

Bus State	Signaling Levels	
	From Originating Driver	At Receiver
Differential "1"	(D+) - (D-) > 200 mV and D+ or D- > V_{SE} (min.)	
Differential "0"	(D+) - (D-) < -200 mV and D+ or D- > V_{SE} (min.)	
Data J State:		
Low Speed	Differential "0"	
Full Speed	Differential "1"	
Data K State:		
Low Speed	Differential "1"	
Full Speed	Differential "0"	
Idle State:		
Low Speed	Differential "0" and D- > V_{SE} (max.) and D+ < V_{SE} (min.)	
Full Speed	Differential "1" and D+ > V_{SE} (max.) and D- < V_{SE} (min.)	
Resume State:		
Low Speed	Differential "1" and D+ > V_{SE} (max.) and D- < V_{SE} (min.)	
Full Speed	Differential "0" and D- > V_{SE} (max.) and D+ < V_{SE} (min.)	
Start of Packet (SOP)	Data lines switch from Idle to K State	
End of Packet (EOP)	D+ and D- < V_{SE} (min) for 2 bit times[1] followed by an Idle for 1 bit time	D+ and D- < V_{SE}(min) for ≥ 1 bit time[2] followed by a J State
Disconnect (Upstream only)	(n.a.)	D+ and D- < V_{SE}(max) for ≥ 2.5 μs
Connect (Upstream only)	(n.a.)	D+ or D- > V_{SE}(max) for ≥ 2.5 μs
Reset (Downstream only)	D+ and D- < V_{SE} for ≥10 ms	D+ and D- < V_{SE} (min)for ≥ 2.5 μs (must be recognized within 5.5 μs)[3]

Note 1: The width of EOP is defined in bit times relative to the speed of transmission.

Note 2: The width of EOP is defined in bit times relative to the device type receiving the EOP.

Note 3: These times apply to an active device that is not in the suspend state.

The J and K data states are the two logical levels used to communicate differential data in the system. Differential signaling is measured from the point where the data line signals cross over. Differential data signaling is not concerned with the level at which the signals cross, as long as it is within the common mode range. When the bus is not in the differential signaling mode, the data lines have to be outside the VSE switching threshold range to be valid, as in the idle and resume states. Note that the idle and resume states are logically equivalent to the J and K states respectively.

Table 7-1 shows the J and K states for a full speed to be inverted from those of low speed. The sense of data, idle, and resume signaling is set by the type of device that is being attached to a port. If a full speed device is attached to a port, that segment of the USB network uses full speed signaling conventions (and fast rise and fall times) even if the data being sent across the data lines is at the low speed data rate. The low speed signaling conventions shown in Table 7-1 (plus slow rise and fall times) are only used between a low speed device and the port to which it is attached. The method of determining the device type, and, therefore, the signaling convention used, is described in the next section.

7.1.4.1 Connect and Disconnect Signaling

All ports on the downstream side of the host or a hub have pull-down resistors on both the D+ and D- lines. All devices have a pull-up resistor on one of the data lines on their upstream port. The type of device determines which data line has the pull-up resistor. Full speed devices have the pull-up on the D+ line (see Figure 7-5) and low speed devices have the pull-up on the D- line (see Figure 7-6). When a device is attached to hub or host but the data lines are not being driven, these resistors create a quiescent bias condition on the lines such that the data line with the pull-up is above 2.8 V and the other data line is near ground. This is called the idle state.

When no function is attached to the downstream port of the host or hub or the pull-up resistor on an attached device is not powered, the pull-down resistors will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port. This creates a state called a single-ended zero (SE0) on the downstream port. A disconnect condition is indicated if an SE0 persists on a downstream port for more than 2.5 μs (30 full speed bit times). Note that disconnect signaling applies only in an upstream direction (see Figure 7-7).

A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5 μs (30 full speed data bit times). The data line that is high when the port state changes from disconnected to connected sets the idle state for this bus segment and determines whether the connected device is a full speed device or a low speed device. All signaling levels given in Table 7-1 are set for this network segment (and this segment alone) once the idle state is determined. Figure 7-8 shows a full speed device connection sequence and Figure 7-9 shows a low speed device connection sequence.

All hub ports start out in an implied disconnected state after power is applied to that port. If a device is connected to the port, the port goes through the connect sequence described above to detect the device type and set the port signaling characteristics (refer to Section 11.2.3).

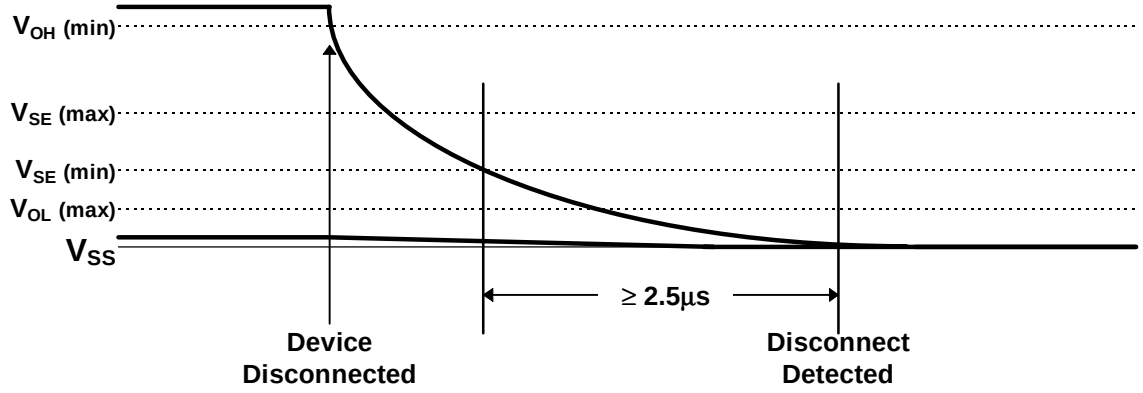


Figure 7-7. Disconnect Detection

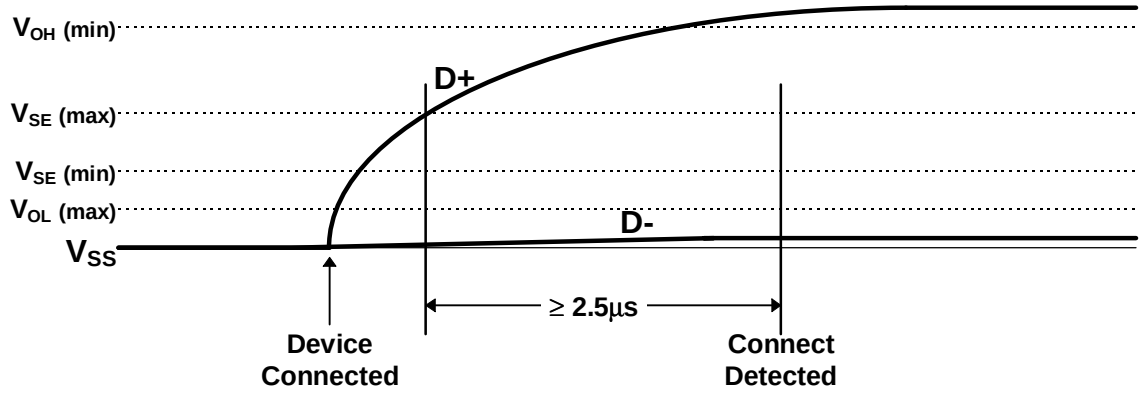


Figure 7-8. Full Speed Device Connect Detection

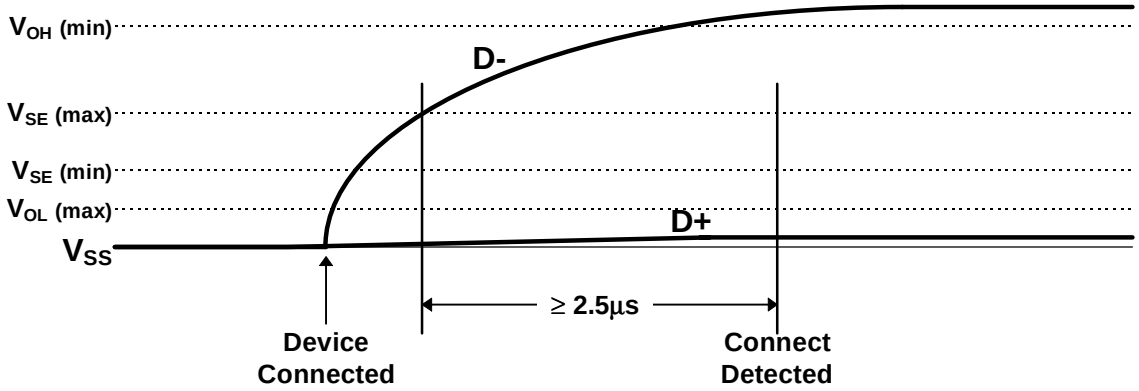


Figure 7-9. Low Speed Device Connect Detection

7.1.4.2 Data Signaling

Data transmission within a packet is done with differential signals. A differential 1 on the bus is represented by D+ being at least 200 mV more positive than D- as seen at the receiver, and a differential 0 is represented by D- being at least 200 mV more positive than D+ as seen at the receiver. The signal cross over point must be between 1.3 V and 2.0 V.

The start of a packet (SOP) is signaled by the originating port by driving the D+ and D- lines from the idle state to the opposite logic level (K state). This switch in levels represents the first bit of the Sync field. Hubs must limit the distortion of the width of the first bit after SOP when it is retransmitted to less than 5 ns. Distortion can be minimized by matching the nominal data delay through the hub with the output enable delay of the hub.

The single-ended 0 state is used to signal an end of packet (EOP). The single-ended 0 state is indicated by both D+ and D- being below 0.8 V. EOP will be signaled by driving D+ and D- to the single-ended 0 state for two bit times followed by driving the lines to the idle state for one bit time. The transition from the single-ended 0 to the idle state defines the end of the packet. The idle state is asserted for 1 bit time and then both the D+ and D- output drivers are placed in their high-impedance state. The bus termination resistors hold the bus in the idle state. Figure 7-10 shows the signaling for start and end of a packet.

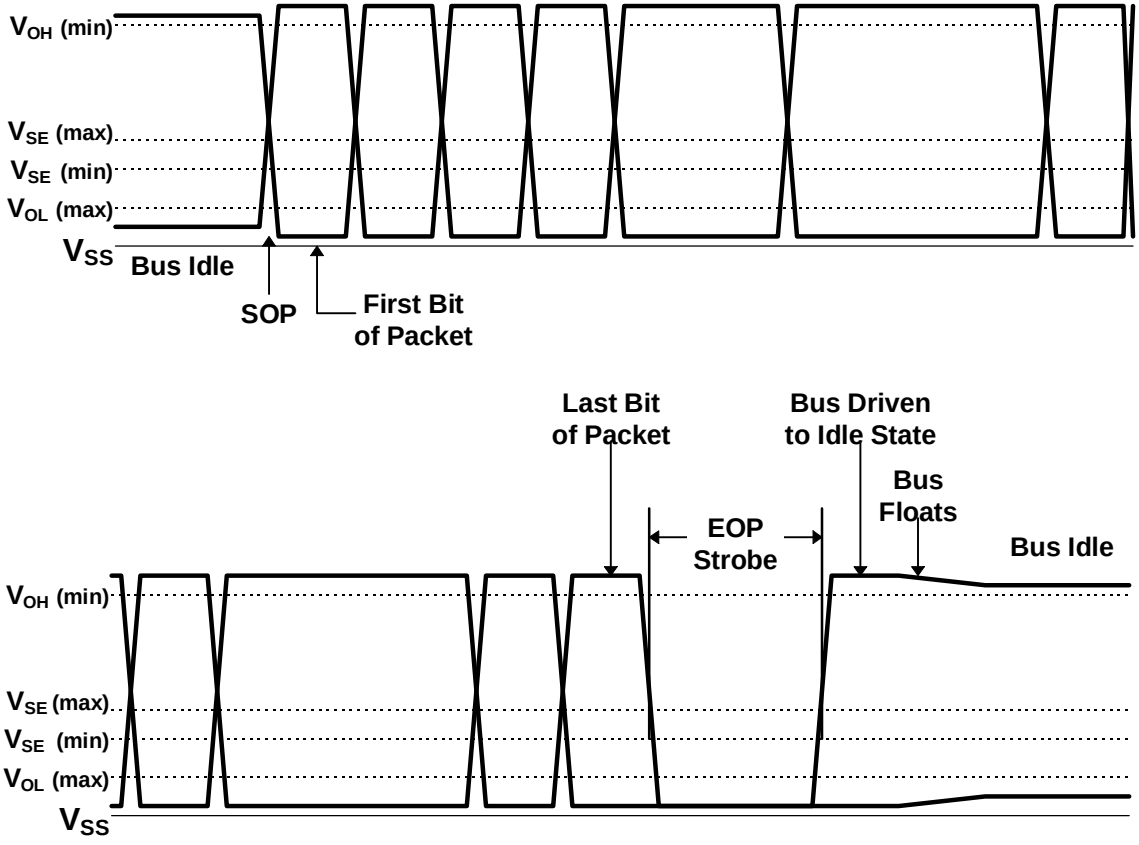


Figure 7-10. Packet Transaction Voltage Levels

7.1.4.3 Reset Signaling

A reset is signaled downstream from a hub port on the bus by the presence of an extended SE0 at the upstream port of a device. After the reset is removed, the device will be in the attached, but not yet addressed or configured state (refer to Section 9.1). Note that reset signaling applies only in the downstream direction.

The reset signal can be generated by host command on any hub or host controller port. The reset signal must be generated for a minimum of 10 ms. The port that generated the reset will be sent to the logically disconnected state at the end of the reset. If a device is connected to the port, the bus pull-up resistor will determine the device type (low or full speed) and the port will end up in the disabled state (refer to Section 11.2.3).

An active device (powered and not in the suspend state) seeing a single-ended zero on its upstream port for more than 2.5 μs may treat that signal as a reset, but must have interpreted the signaling as a reset within 5.5 μs. A device that recognizes a reset from a SE0 between 32 and 64 full speed bit times or between 4 and 8 low speed bit times satisfies these requirements. The reset signal propagates through all enabled ports of any hubs downstream of the signaling port, but does not propagate through any ports that are disabled. A bus-powered hub that receives a reset on its root port removes power from all its downstream ports. After the reset is removed, all devices that received the reset are set to their default USB address and are in the unconfigured state. All ports on a hub that received a reset are disabled.

Hubs must be able to establish connectivity, and all devices must be able to accept device address via a SET_ADDRESS command (refer to Section 9.4) no later than 10 ms after the reset is removed. Failure to establish connectivity or accept an address may cause the device not to be recognized by the USB enumerator. In the case of a hub, it may also cause any devices connected to that hub not to be recognized. All other requests for data or service, except SETUP packets (refer to Section 8.4.5.4), can be NAKed for a period up to 5.0 seconds after which the device is declared defective and is not recognized.

Reset can wake a device from the suspended mode. It is recommended that the device wait for its clocks to stabilize before accepting the reset to avoid spurious single-ended zero events from causing the device to reset. A device may take up to 10 ms to wake up from the suspended state.

7.1.4.4 Suspending

All devices must support the suspend mode. Devices can go into the suspended mode from any powered state. They go into the suspend state when they see a constant idle state on their bus lines for more than 3.0 ms. Any bus activity will keep a device out of the suspend state. The SOF packet (refer to Section 8.4.2) is guaranteed to occur once a frame to keep full speed devices awake during normal bus operation. Hubs that are not in the suspend state keep low speed devices awake by generating a low speed EOP on enabled ports that are attached to low speed devices (refer to Section 11.2.5.1). When a device is in the suspend state, it draws less than 500 μA from the bus. Even though suspended devices draw at most 500 μA from the bus, hub ports must be able to supply their maximum rated current to the downstream devices when the hub is in the suspended state. This is necessary to support remote wake-up as described in Section 7.2.3.

All devices can be awakened from the suspend state by switching the bus state to the resume state, by normal bus activity, or by signaling a reset. Some devices have the ability to be awakened by actions associated with their internal functions and then cause signaling on their upstream connection to wake or alert the rest of the system. This feature is called remote wake-up and is described in Section 7.1.4.5.

7.1.4.4.1 Global Suspend

Global suspend is used when no communication is desired anywhere on the bus and the whole network is placed in the suspend state. The host signals the start of suspend by ceasing all its transmissions (including the SOF token). As each device on the bus recognizes the lack of activity, and that the bus is in the idle state for the appropriate length of time, it goes into its suspend state. As hubs go into the suspended state, they cease to send the low speed EOP on any downstream low-speed configured ports.

7.1.4.4.2 Selective Suspend

The system software may want to conserve power by suspending only certain segments of the topology, while continuing regular operation on the remaining segments. Segments of the network can be selectively put into the suspend state by suspending the hub port to which that segment is attached. The suspended port will block activity to this segment of the bus and the attached devices will go into suspend after the appropriate delay as described above.

Any non-hub device can be suspended in this way. Any hub not involved in connecting the remaining devices to the host may also be suspended by disabling the port to which it is attached. Devices that are selectively suspended can still alert the system with a remote wake-up signaling, although the process is slightly different. A description of the suspend port state can be found in Section 11.2.3 and selective suspend is further described in Section 11.5.2.

7.1.4.5 Resume

Once a device is in the suspend state, its operation can be resumed by receiving non-idle signaling on the bus, or it can signal the system to resume operation if it has the remote wake-up capability. The resume signaling state is used by the host or a device with remote wake-up to awaken the system. Hubs play an important role in the propagation and generation of resume signaling. The following description is an outline of a general global resume sequence. A complete description of the resume sequence, the special cases caused by selective suspend, and the role of the hub is given in Section 11.5.

The host may signal resume anytime after it places the bus into the suspend mode. It must send the resume signaling for at least 20 ms and then end the resume signaling with a standard low speed EOP (two low speed bit times of SE0 followed by a transition to the idle state). The 20 ms of resume signaling insures that all devices in the network that are enabled to see the resume are awakened. The EOP tears down the connectivity established by the resume signaling and prepares the hubs for normal operation. After resuming the bus, the host must begin sending bus traffic (at least the SOF token) within 3 ms to keep the system from going back into the suspend state.

A device with remote wake-up capability must wait for at least 5 ms after the bus is in the idle state before sending the remote wake-up resume signaling. This allows the hubs to get into their suspend state and prepare for propagating resume signaling. The remote wake-up device must hold the resume signaling for at least 10 ms and no more than 15 ms. At the end of the resume signaling, the device puts its data lines into the high impedance state.

The hub upstream of the remote wake-up device will propagate the resume signaling to its root port and all of its enabled downstream ports, including the port which originally signaled the resume. The hub must begin this rebroadcast of the resume signaling within 50 μs of receiving the original resume. The resume signal will propagate in this manner upstream until it reaches the host (or a suspended hub port - refer to Section 11.5.2.1) which will reflect the resume downstream on its enabled ports, as in any other hub. Meanwhile, the hubs that transported the resume to the host wake up within 10 ms. After a 1 ms delay, they reverse the direction of connectivity in their root port from upstream to downstream and reflect the signal state on the root port onto their enabled downstream ports. This puts the host in control of the resume signaling which then proceeds as described above.

Port connects and disconnects can also cause the affected hub to send a resume signal and awaken the system. Refer to Section 11.5 for more details.

(Note that the host can wake up the entire bus by resetting it. This requires that the entire bus must be re-enumerated and reconfigured.)

Refer to Section 7.2.3 for a description of power control during suspend and resume.

7.1.5 Data Encoding/Decoding

The USB employs NRZI data encoding when transmitting packets. In NRZI encoding, a 1 is represented by no change in level and a 0 is represented by a change in level. Figure 7-11 shows a data stream and the NRZI equivalent and Figure 7-12 is a flow diagram for NRZI. The high level represents the J state on the data lines in this and subsequent figures showing NRZI encoding. A string of zeros causes the NRZI data to toggle each bit time. A string of ones causes long periods with no transitions in the data.

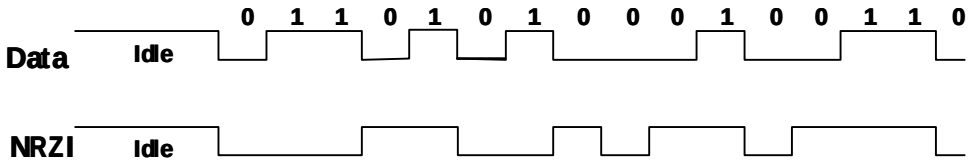


Figure 7-11. NRZI Data Encoding

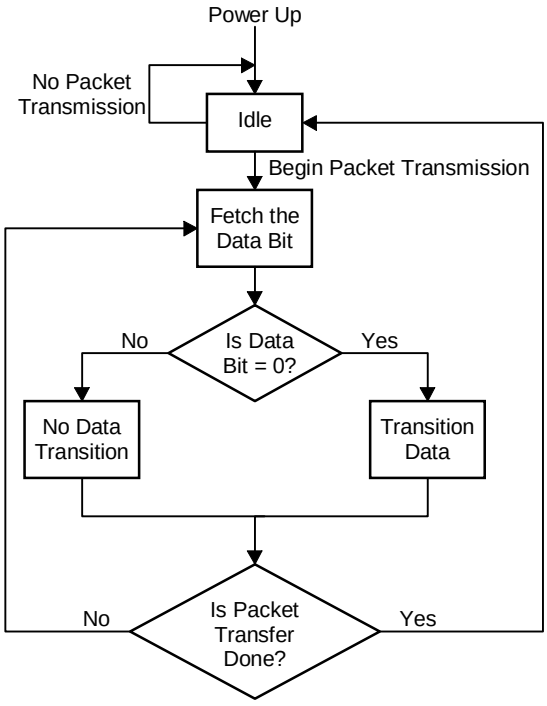


Figure 7-12. Flow Diagram for NRZI

7.1.6 Bit Stuffing

In order to ensure adequate signal transitions, bit stuffing is employed by the transmitting device when sending a packet on the USB (see Figure 7-13 and Figure 7-14). A 0 is inserted after every six consecutive 1's in the data stream before the data is NRZI encoded to force a transition in the NRZI data stream. This gives the receiver logic a data transition at least once every seven bit times to guarantee the data and clock lock. The receiver must decode the NRZI data, recognize the stuffed bits, and discard them. Bit stuffing is enabled beginning with the Sync Pattern and throughout the entire transmission. The data "one" that ends the Sync Pattern is counted as the first one in a sequence. Bit stuffing is always enforced, without exception. If required by the bit stuffing rules, a zero bit will be inserted even if it is the last bit before the end-of-packet (EOP) signal.

Data Encoding Sequence:

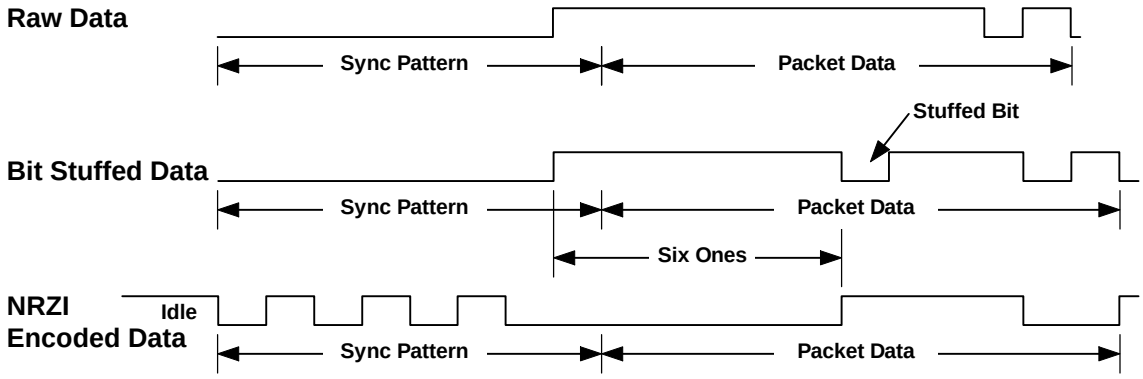


Figure 7-13. Bit Stuffing

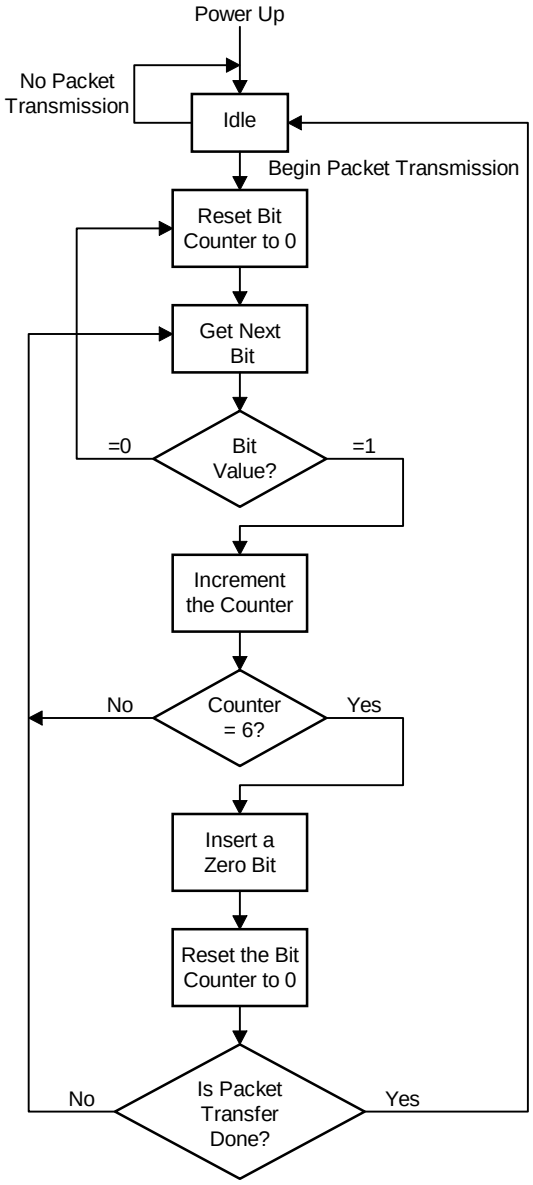


Figure 7-14. Flow Diagram for Bit Stuffing

7.1.7 Sync Pattern

The NRZI bit pattern shown in Figure 7-15 is used as a synchronization pattern and is prefixed to each packet. This pattern is equivalent to a data pattern of seven 0's followed by a 1 (0x80).

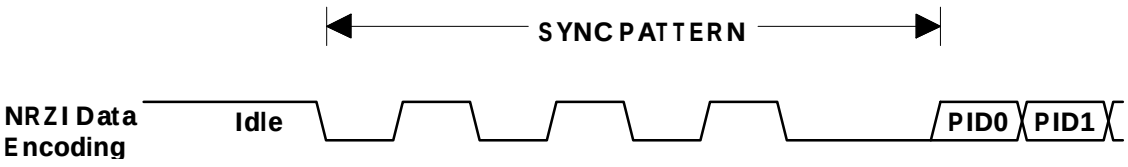


Figure 7-15. Sync Pattern

7.1.8 Initial Frame Interval and Frame Adjustability

The USB defines a frame interval to be 1.000 ms long. The frame interval is measured from the start of the Start of Frame (SOF) PID in one frame to the same point in the SOF token of the next frame. The permitted tolerance on the initial frame interval is ±0.05% (500 ppm). This tolerance includes inaccuracies from all sources: initial frequency accuracy, crystal capacitive loading, supply voltage on the oscillator, temperature, and aging.

The host controller must be able to adjust the frame interval. There are two possible components to the frame interval adjustability. If the host's data rate clock is not exactly 12.00 Mbs, then the initial ±0.05% frame interval accuracy can be met by changing the default number of bits per frame from the nominal of 12,000. (A host controller component, that has a range of possible clock source values, may have to make this initial frame count a programmable value. The range of these values is given in Section 7.1.9.) An additional adjustability of ±15 full speed bit times is required to allow the host to synchronize to an external time reference. The frame interval can be reprogrammed by no more than one full speed bit time each adjustment during normal bus operation.

Hubs and certain full speed functions need to track the frame interval. They also are required to have sufficient frame timing adjustability to compensate for its own frequency tolerance and track the host's ±15 full speed bit time adjustability range.

7.1.9 Data Signaling Rate

The full speed data rate is nominally 12 Mbs. The data rate tolerance for host, hub, and full speed functions is ±0.25% (2500 ppm). The accuracy of the host controller's data rate must be known to better than ±0.05% (500 ppm) in order to meet the frame interval accuracy. This tolerance includes inaccuracies from all sources: initial frequency accuracy, crystal capacitive loading, supply voltage on the oscillator, temperature, and aging.

The low speed data rate is nominally 1.5 Mbs. The permitted frequency tolerance for low speed functions is ±1.5% (15000 ppm). This tolerance includes inaccuracies from all sources: initial frequency accuracy, crystal capacitive loading, supply voltage on the oscillator, temperature, and aging. The jitter in the low speed data rate must be less than 10 ns. This tolerance allows the use of resonators in low cost, low speed devices

7.1.10 Data Signal Rise and Fall Time

The output rise time and fall time are measured between 10% and 90% of the signal (see Figure 7-16). Edge transition time for the rising and falling edges of full speed data signals is 4 ns (minimum) and 20 ns (maximum) measured with a capacitive load (C_L) of 50 pF. The rise and fall times must be well matched. The rise and fall time of low speed signals is 75 ns (minimum) into a load of 50 pF and 300 ns (maximum) into a capacitive load of 350 pF. In both cases, the rising and falling edges should be smoothly transitioning (monotonic) when driving their respective cables to avoid excessive EMI. (In full speed signaling, the driver and cable impedances are not matched, so reflections from the receiver end of the cable are expected.)

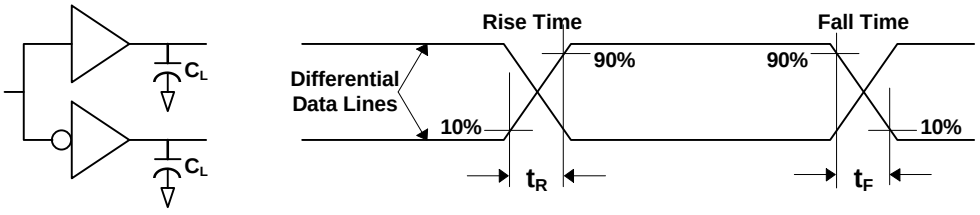


Figure 7-16. Data Signal Rise and Fall Time

7.1.11 Data Source Signaling

This section covers the timing characteristics of data produced and sent from a device (the data source). Section 7.1.12 covers the timing characteristics of data that is transmitted through the repeater section of a hub. In this section, T_{PERIOD} is defined as the actual period of the data rate which can have a range as defined in Section 7.1.9.

7.1.11.1 Data Source Jitter

The source of data can have some variation (jitter) in the timing of edges of the data transmitted. The time between any set of data transitions is N * T_{PERIOD} ± jitter time, where 'N' is the number of bits between the transitions. The data jitter is measured with the same capacitive load used for maximum rise and fall times and is measured at the crossover points of the data lines as shown in Figure 7-17.

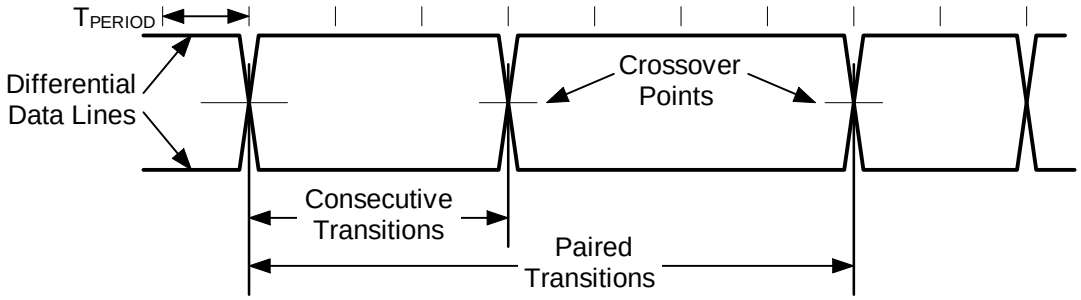


Figure 7-17. Data Jitter

For full speed transmissions, the jitter time for any consecutive differential data transitions must be within ±2.0 ns and within ±1.0 ns for any set of paired differential data transitions. For low speed transmissions, the jitter time for any consecutive differential data transitions must be within ±25 ns and within ±10 ns for any set of paired differential data transitions. These jitter numbers include timing variations due to differential buffer delay and rise/fall time mismatches, internal clock source jitter and to noise and other random effects.

7.1.11.2 EOP Width

The width of the SE0 in the 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 (see Figure 7-18).

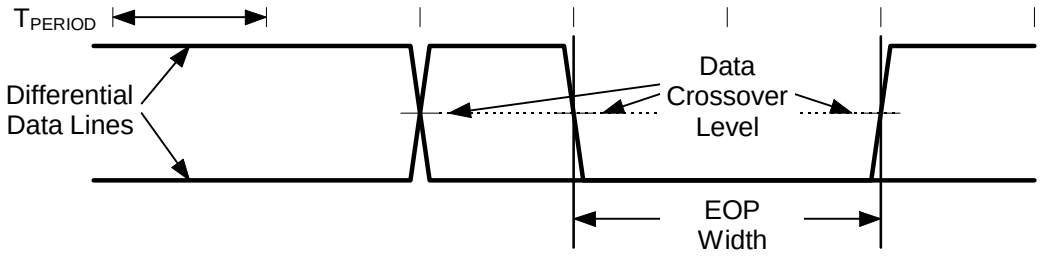


Figure 7-18. EOP Width Timing

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.

The full speed receiver must accept a 82 ns wide SE0 followed by a J transition as a valid EOP. An SE0 narrower than 40 ns or any SE0 not followed by a J transition must be rejected as an EOP. The receiver may reject or accept an EOP between 40 ns and 82 ns as dictated by implementation depending on sampling and synchronization. A low speed receiver must accept a 670 ns wide SE0 followed by a J transition as a valid EOP. An SE0 narrower than 330 ns or an SE0 not followed by a J transition must be rejected as an EOP. An EOP between 330 ns and 670 ns may be rejected or accepted as above. The host is an exception to this rule and always uses full speed EOP timing rules while receiving and either data rate. (This is necessary to support end-of-frame error recovery. Refer to Section 11.4.5.) Any SE0 that is 2.5 μs or wider is automatically a reset or a disconnect (depending on direction).

7.1.12 Hub Signaling Timings

The propagation of a full speed, differential data signal through a hub is shown in Figure 7-19. The downstream signaling is measured without a cable connected to the port and with a 50 pF capacitive load. The total delay through the cable and hub electronics must be a maximum of 70 ns. If the hub has a USB standard detachable cable, then the delay through hub electronics must be a maximum of 40 ns to allow for a worst case cable delay of 30 ns. The delay through this hub is measured in both upstream and downstream directions as shown in Figure 7-19B, from data line crossover at the input port to data line crossover at the output port.

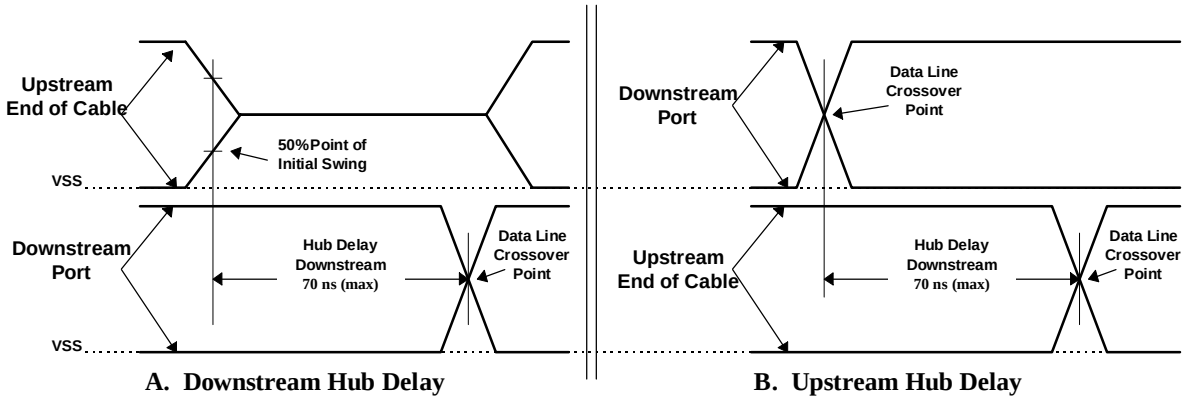


Figure 7-19. Hub Propagation Delay of Full Speed Differential Signals

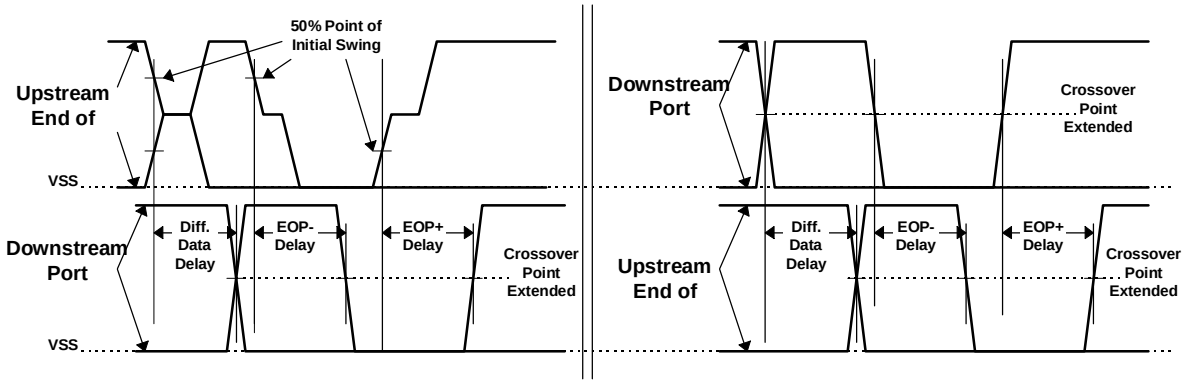
Low speed propagation delay for differential signals is measured in the same fashion as for full speed signaling. The maximum low speed hub delay is 300 ns. This allows for the slower low speed buffer propagation delay and rise and fall times.

When the hub acts as a repeater, it must reproduce the received signal accurately on its outputs. This means that for differential signals, the propagation delays of a J to K state transition must match closely to the delays of a K to J state transition. The maximum difference allowed between these two delays (as measured in Figure 7-19) for a hub plus cable is ±3.0 ns. Similarly, the difference in delay between any two J to K or K to J transitions through a hub should be less than ±1.0 ns.

An exception to this case is the skew that can be introduced in the SOP idle to K state transition (refer to Section 7.1.4.2). In this case, the delay to the opposite port includes the time to enable the output buffer. However, the delays should be closely matched to the normal hub delay and the maximum additional delay difference over a normal J to K transition is -3.0 ns to +5.0 ns. This limits the maximum distortion of the first bit in the packet. (Note: Because of this distortion of the SOP transition relative to the next K to J state transition, the first sync field bit should not be used to synchronize the receiver to the data stream.)

The EOP has to be propagated through a hub in the same way as the differential signaling. The propagation delay for sensing an SE0 must be no less than the greater of the J to K or K to J differential data delay (to avoid truncating the last data bit in a packet) but not more than 15 ns greater than the larger differential delay at full speed and 200 ns at low speed (to prevent creating a bit stuff error at the end of the packet). EOP delays are shown in Figure 7-20.

Since the sense levels for the SE0 state are not at the midpoint of the signal swing, the width of the single-ended 0 state will be changed as it passes through each hub. A hub may not change the length of a full speed single-ended 0 state by more than ±15 ns as measured by the difference of the EOP- and EOP+ delays (see Figure 7-20). A single-ended 0 from a low speed device has long rise and fall times and is subject to greater skew, but this conditions exists only on the cable from the low speed device to the port to which it is connected. Thereafter, the signaling uses full speed buffers and their faster rise and fall times. The single-ended 0 from the low speed device cannot be changed by more than ±300 ns as it passes through the hub to which the device is connected. This time allows for some signal conditioning in the low speed port to reduce its sensitivity to noise.



A. Downstream EOP Delay **B. Upstream EOP Delay**

Figure 7-20. Timing of EOP

7.1.13 Receiver Data Jitter

The data receivers for all types of devices must be able to properly decode the differential data in the presence of jitter. The more of the bit cell that any data edge can occupy and still be decoded, the more reliable the data transfer will be. Data receivers are required to decode differential data transitions that occur in a window plus and minus a nominal quarter bit cell from the nominal (centered) data edge position. (A simple 4X oversampling state machine DPLL can be built that satisfies these requirements.)

Jitter will be caused by the delay mismatches discussed above and by mismatches in the source and destination data rates (frequencies). The receive data jitter budgets for full and low speed are given in Table 7-2 and Table 7-3. These tables give the value and totals for each source of jitter for both consecutive (next) and paired transitions. Note that the jitter component related to the source or destination frequency tolerance has been allocated to the appropriate device (i.e., the source jitter includes bit shifts due to source frequency inaccuracy over the worst case data transition interval). The output driver jitter can be traded off against the device clock accuracy in a particular implementation as long as the jitter specification is met.

The low speed jitter budget table has an additional line in it because the jitter introduced by the hub to which the low speed device is attached is different from all the other devices in the data path. The remaining devices operate with full speed signaling conventions (though at low speed data rate).

Table 7-2. Full Speed Jitter Budget

HUBS	5	Full Speed Freq. Tol.	0.0025
MAX BITS/TRANS	7		

Jitter Source	Full Speed			
	Next Transition		Paired Transition	
	Each (ns)	Total (ns)	Each (ns)	Total (ns)
Source Driver Jitter	2.0	2.0	1.0	1.0
Source Freq. Tol. - Worst case	0.21/bit	1.5	0.21/bit	3.0
Source Jitter Total		**3.5**		**4.0**
Hub Jitter	3.0	15.0	1.0	5.0
Jitter Spec		**18.5**		**9.0**
Destination Freq. Tol.	0.21/bit	1.5	0.21/bit	3.0
Rcv Jitter Budget		**20.0**		**12.0**

Table 7-3. Low Speed Jitter Budget

HUBS	5	Full Speed Freq. Tol.	0.0025
MAX BITS/TRANS	7	Low Speed. Freq. Tol.	0.015

Jitter Source	Low Speed - Upstream			
	Next Transition		Paired Transition	
	Each (ns)	Total (ns)	Each (ns)	Total (ns)
Function Driver Jitter	25.0	25.0	10.0	10.0
Function Freq. Tol - Worst Case	10.0/bit	70.0	10.0/bit	140.0
Source (Function) Jitter Total		**95.0**		**150.0**
Hub w/ Low Speed Device Jitter	45.0	45.0	45.0	45.0
Remaining (Full Speed) Hubs' Jitter	3.0	12.0	1.0	4.0
Jitter Spec		**152.0**		**200.0**
Host Freq. Tol.	1.7/bit	12.0	1.7/bit	24.0
Host Rcv Jitter Budget		**164.0**		**225.0**

Table 7-3. Low Speed Jitter Budget (continued)

Jitter Source	Low Speed - Downstream			
	Next Transition		Paired Transition	
	Each (ns)	Total (ns)	Each (ns)	Total (ns)
Host Driver Jitter	2.0	2.0	1.0	1.0
Host Freq. Tol. - Worst Case	1.7/bit	12.0	1.7/bit	24.0
Source (Host) Jitter Total		**14.0**		**25.0**
Hub w/ Low Speed Device Jitter	45.0	45.0	15.0	15.0
Remaining (Full Speed) Hubs' Jitter	3.0	12.0	1.0	4.0
Jitter Spec		**75.0**		**45.0**
Function Freq. Tol.	10.0/bit	70.0	10.0/bit	140.0
Function Rcv Jitter Budget		**150.0**		**185.0**

Note: This table describes the host transmitting at low speed data rate using full speed signaling to a low speed device through the maximum number of hubs. When the host is directly connected to the low speed device, then it uses low speed data rate and low speed signaling, and the host has to meet the source jitter listed in the "Jitter Spec" row.

7.1.14 Cable Delay

Only one data transition is allowed on a USB cable at a time. A full speed signal edge has to transition, propagate to the far end of the cable, return, and settle within one full speed bit time. Therefore, the maximum allowed cable delay is 30 ns. Independent of cable velocity, the maximum cable length is 5.0 meters for full speed devices and 3.0 meters for low speed devices. The maximum one-way data delay on a cable is measured as shown in Figure 7-21.

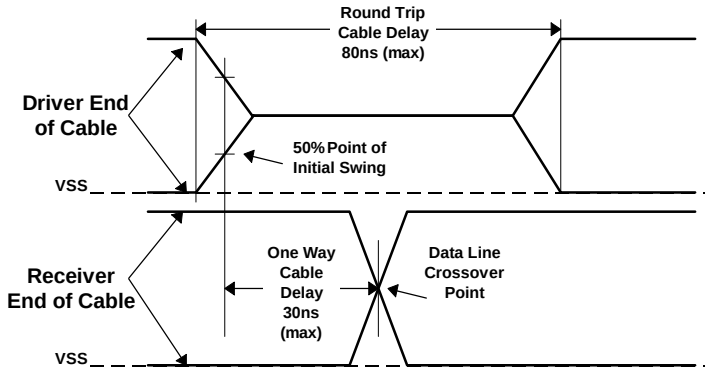


Figure 7-21. Cable Delay

7.1.15 Bus Turnaround Time/Interpacket Delay

A new device may not begin driving the bus until the previous device has completed the EOP sequence and has disabled its drivers. This is assured by not allowing the new device to drive the bus until it has

detected that the bus is in the J state after the SE0 in the EOP for at least two bit times. This minimum of two bit times applies to all devices, including back to back host packet transmissions.

If a function is expected to provide a response to a host transmission, the response must be seen on the upstream end of the cable within 7.5 bit times of the bus returning to the J state after the EOP as seen on the upstream end of its cable. The maximum bus turnaround time for a function or hub without an integrated cable is 6.5 bit times. This maximum bus turnaround time prevents a full speed receiving agent from timing out after 16 bit times on a response in a maximum depth topology (refer to Section 7.1.16). However, these timings apply to both full speed and low speed devices.

The maximum delay a host has to respond to a data packet sent by a function (if a handshake is required) is 7.5 bit times, measured at the host's port pins. There is no maximum delay between packets in separate transactions.

7.1.16 Maximum End to End Signal Delay

A device expecting a response to a transmission will invalidate the transaction if it does not see the start of packet (SOP) transition within the time-out period after the end of the transmission (after the SE0 to J state transition in the EOP). This can occur between an IN token and the following data packet or between a data packet and the handshake packet (refer to Chapter 8). The device expecting the response will not time out before 16 bit times and not after 18 bit times measured at the data pins of the device after the end of the EOP. The host will not begin transmission of the token for the next transaction before 18 bit times (measured at its data pins) after it sees the end of a packet with no response.

Figure 7-22 depicts the configuration of six signal hops (cables) that results in worst allowable case signal delay. The maximum propagation delay from the upstream end of a hub's cable to any downstream port connector is 70 ns.

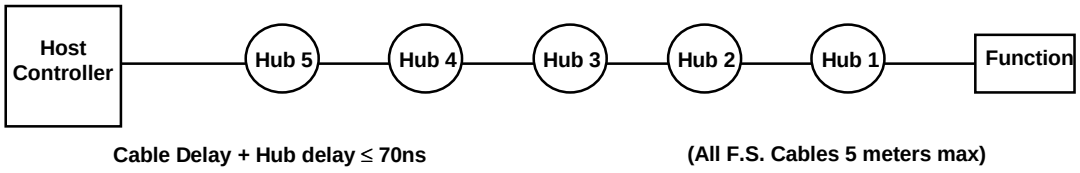


Figure 7-22. Worst Case End to End Signal Delay Model

7.2 Power Distribution

This section describes the USB power distribution specification.

7.2.1 Classes of Devices

The power sourcing and sinking requirements of different device classes can be simplified with the introduction of the concept of a unit load. A unit load is defined to be 100 mA.

USB supports a range of devices as categorized by their power consumption; these include:

- **Bus-powered hubs** - Draws all of the power to any internal functions and downstream ports from the USB connector power pins. May draw up to one load upon power up and a total of five loads, which is split between any embedded functions and external ports. External ports in a bus-powered hub can supply only one load per port regardless of the current drawn on the other ports of that hub. The hub must be able to supply this port current when the hub is in the active or suspended state.
- **Self-powered hubs** - Power for the internal functions and downstream ports does not come from USB, although the USB interface may draw up to one load from its upstream connection to allow the interface to function when the remainder of the hub is powered down. The hub must be able to supply up to five unit loads on all of its external downstream ports even when the hub is in the suspended state.
- **Low power, bus-powered functions** - All power to these devices comes from USB connector. They may draw no more than one unit load at any time.
- **High power, bus-powered functions** - All power to these devices comes from USB connector. They must draw no more than one unit load upon power up and may draw up to five unit loads after being configured.
- **Self-powered functions** - May draw up to one load from their upstream connection to allow the interface to function when the remainder of the hub is powered down. All other power comes from an external (to USB) source.

The hub on the host in a desktop computer is a self-powered hub. The same hub in a notebook computer can be defined to be either a self-powered or bus-powered hub.

All devices, whether they are bus-powered or self-powered, can only draw (sink) current from the bus. They must not supply current upstream to a host or hub port. On power up, all devices need to insure that their upstream port is not enabled, so that the device is able to receive the reset signaling, and that the maximum operating current drawn by a device is one unit load. If a device draws power from the bus, its internal supply derived from V_{bus} must be stable within 100 ms of V_{bus} reaching 4.4 V. All devices which are drawing power from the bus must be able enter the suspend state and reduce their current consumption from V_{bus} to less than 500 µA (refer to Section 7.1.4.4 and Section 9.2.5.1).

7.2.1.1 Bus-powered Hubs

The above requirements can be met for bus-powered hubs with a power control circuit such as that shown in Figure 7-23. Bus-powered hubs often contain at least one embedded function. Power is always available to the hub's controller, which permits host access to power management and other configuration registers during the enumeration process. An embedded function may require that its power be switched, so that upon power-up the entire device (hub and embedded functions) draws no more than one unit load. Power switching on any embedded function may be implemented either by removing its power or by shutting off the clock. Switching on the embedded function is not required if the aggregate power drawn by it and the hub controller is less than one unit load. The total current drawn by an bus-powered device is the sum of the current to the hub controller, any embedded function(s), and the downstream ports.

Figure 7-23 shows the partitioning of power based upon the maximum upstream current of five loads: one unit load for the hub controller and the embedded function, and one load for each of the downstream ports. The maximum number of downstream ports that can be supported is limited to four. If more ports are required, then the hub will need to be self-powered. If the embedded function(s) and hub controller draw more than one unit load, then the number of ports must be appropriately reduced. Power control to a bus-powered hub may require a regulator. If present, the regulator is always enabled to supply the hub controller. The regulator can also power the embedded functions(s). Inrush current limiting must also be incorporated into the regulator subsystem.

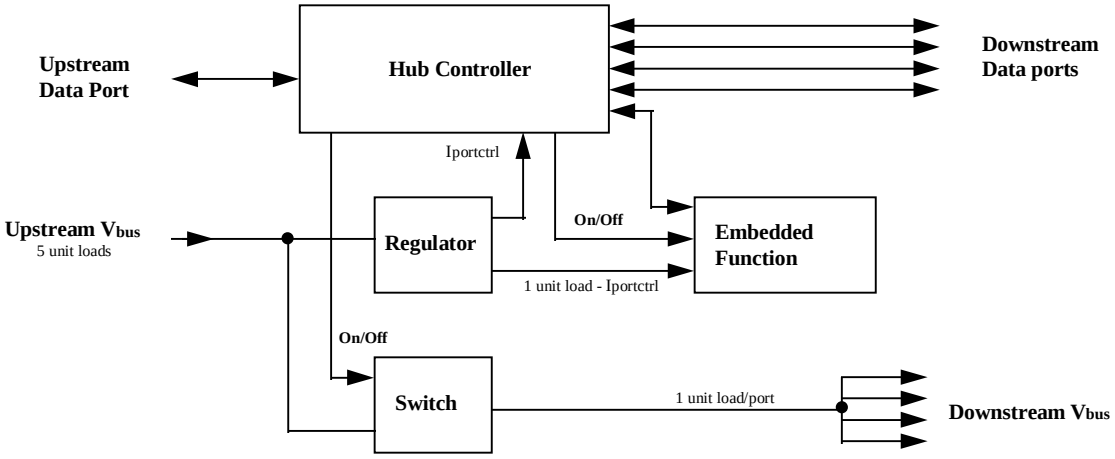


Figure 7-23. Compound Bus-powered Hub

Power to downstream ports must be switched. The hub controller supplies a software controlled on/off signal from the host, which is in the "off" state when the device is powered up or after reset signaling. When switched to the "on" state, the switch implements a soft turn-on function which prevents excessive transient current from being drawn from the upstream port. The voltage drop across the upstream cable, connectors, and switch in a bus-powered hub must not exceed 350 mV at maximum rated current.

7.2.1.2 Self-powered Hubs

Self-powered hubs have a local power supply that furnishes power to any embedded functions and to all downstream ports, as shown in Figure 7-24. Power for the hub controller, however, may be supplied from either the upstream port (a "hybrid" powered hub) or the local power supply. The advantage of supplying the hub controller from the upstream supply is that communication from the host is possible even if the device's power supply remains off. This makes it possible to differentiate between a disconnected and an unpowered device.

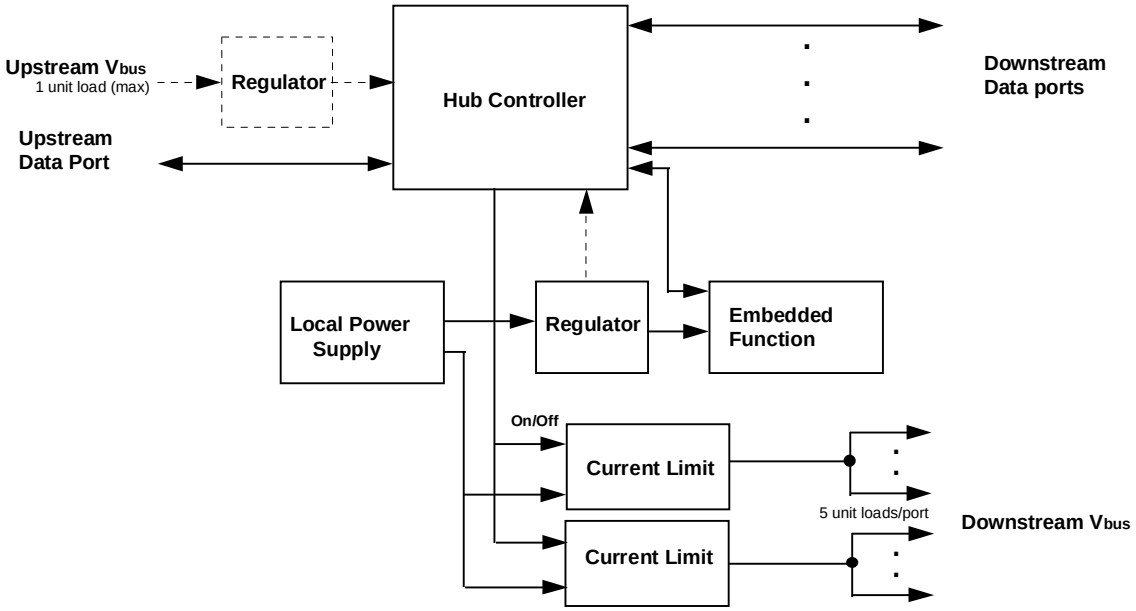


Figure 7-24. Compound Self-powered Hub

Power is provided to all downstream ports from the local power supply. The number of ports that can be supported is limited only by what the local supply can deliver and by safety concerns. A reasonable maximum is seven ports. Each port must be capable of supplying at least five unit loads. However, no single port can deliver more than 5.0 A in order to meet regulatory safety limits. Current limiting on the downstream ports may need to be partitioned into two or more port subgroups in order to deliver sufficient power to all ports without exceeding the current available through one port. If all seven ports were wired in parallel, the available current at a given port would be 7 * 500 mA = 3.5 A. This is very close to the safety limit. By implementing two current limit circuits, the maximum current that needs to be supplied in operation is reduced to 1.5 A to 2.0 A, which gives a comfortable margin to the safety limit.

7.2.1.2.1 Overcurrent Protection

The host and all self-powered hubs must implement overcurrent protection for safety reasons, and they must have a way to detect the overcurrent condition and report it to the USB software. Should the aggregate current drawn by a group of downstream ports exceed a preset value, the overcurrent protector removes power from all downstream ports and reports the condition through the hub to host controller. The preset value cannot exceed 5.0 A and should be sufficiently above the maximum allowable port current such that power up or dynamic attach transient currents do not trip the overcurrent protector. If an overcurrent condition occurs on any port, subsequent operation of the USB is not guaranteed, and once the condition is removed, it may be necessary to reinitialize the bus as would be done upon power-up. Overcurrent limiting methods can include poly fuses, standard fuses, or some type of solid state switch. The only requirements are that current be limited to five unit loads per port and that the host is notified of an overcurrent condition.

Current limiting should not occur even if illegal topologies are configured, due to the protection afforded by power switching in high power functions and bus-powered hubs. Instead, the overcurrent circuits are used to protect from catastrophic device failures, software errors that turn on devices when the current budget has been exceeded, and user actions such as shorting out the connector pins.

7.2.1.2.2 Power Supply Isolation

Figure 7-24 is based on the assumption that the local power supply shares a common ground with the upstream and downstream ports. Its V_{bus}, however, is isolated from the V_{bus} of the upstream port. There is an additional requirement that the chassis ground (if one exists) of the self-powered hub be DC isolated from the USB signal ground. Chassis ground connects to the ground of a 120 Vac power cable, and there is no guarantee that AC grounds from two different outlets are at the same potential. Failure to observe this precaution could result in large low frequency currents running through USB ground paths.

7.2.1.3 Low-power, Bus-powered Functions

A low power function is one that draws less than one unit load from the USB cable when fully operational. The regulator block must both limit inrush current and supply the necessary voltage for the proper signaling levels. Figure 7-25 shows a typical bus-powered low-power function, such as a mouse. Low power regulation can be integrated into the function silicon. For higher currents, in the range of 20 mA to 100 mA, an IC linear regulator may be used. Low power functions must be capable of operating with input V_{bus} voltages as low as 4.40 V measured at the plug end of the cable.

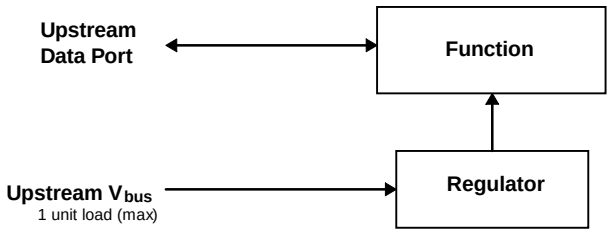


Figure 7-25. Low-power Function

7.2.1.4 High-power, Bus-powered Functions

A function is defined as being high powered if, when fully powered, it draws over one and a maximum of five unit loads from the USB cable. A high-power function requires staged switching of power. It must first come up in a reduced power state of less than one unit load. At bus enumeration time, its total power requirements are obtained and compared against the available power budget. If sufficient power exists in the power budget, the remainder of the function may be powered on. If insufficient power is available, the remainder of the function is not powered and a power limit warning message is sent to the client. A high-power function is shown in Figure 7-26. The function's electronics have been partitioned into two sections; the function controller contains the minimum amount of circuitry necessary to permit enumeration and power budgeting. The remainder of the function resides in the function block. High power functions must be capable of operating in their low power (one unit load) mode with an input voltage as low as 4.40 V, so that it will work even when plugged into an bus-powered hub. They must also be capable of operating at full power (up to five unit loads) with an input V_{bus} voltage of 4.75 V measured at the upstream plug end of the cable.

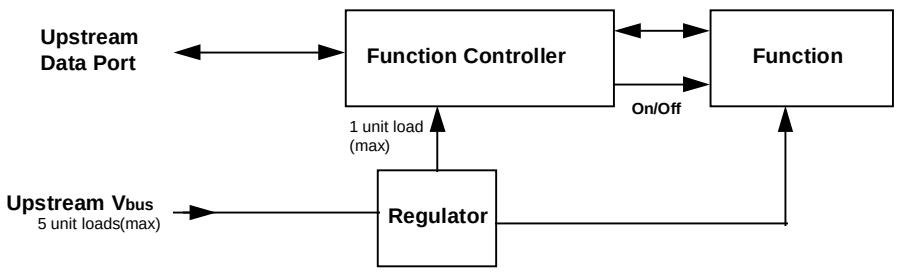


Figure 7-26. High-power, Bus-powered Function

7.2.1.5 Self-powered Functions

Figure 7-27 shows a self-powered function. The function controller is powered either from the upstream bus via a low power regulator or from the local power supply. The advantage of the former scheme is that it permits detection and enumeration of a self-powered function whose local power supply is turned off. When the function controller is externally powered, the maximum upstream power that it can draw is one unit load, and the regulator block must implement inrush current limiting. The amount of power that the function block may draw is limited only by the local power supply. Because the local power supply is not required to power any downstream bus ports, it does not need to implement current limiting, soft start, or power switching.

Self-powered functions must adhere to the same ground and V_{bus} isolation rules as self-powered hubs.

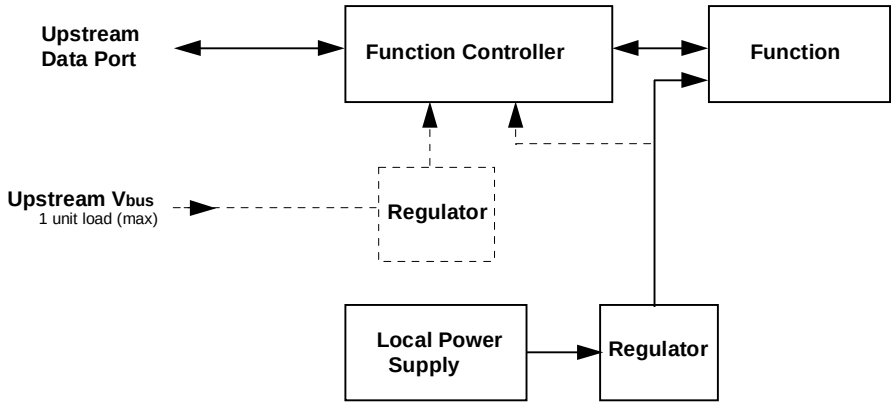


Figure 7-27. Self-powered Function

7.2.2 Voltage Drop Budget

The voltage drop budget is determined from:

- The voltage supplied by host or powered hub ports is 4.75 V to 5.25 V.
- Bus-powered hubs can have a maximum drop of 350 mV from their cable plug where they attach to a source of power to their output port connectors where they supply a source of power.
- All hubs and functions must be able to provide configuration information with as little as 4.40 V at the connector end of their upstream cables. Only low power functions need to be able to be fully operational with this minimum voltage.
- Functions drawing more than one unit load must operate with a 4.75 V minimum input voltage at the connector end of their upstream cables.

Figure 7-28 shows the minimum allowable voltages in a worst case topology consisting of an bus-powered hub driving an bus-powered function.

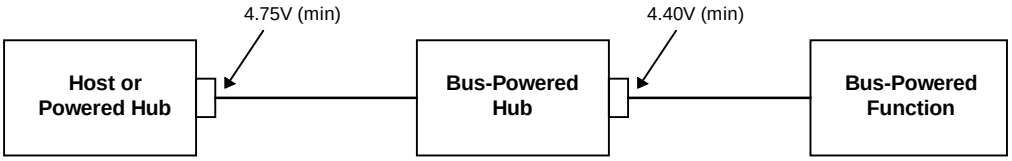


Figure 7-28. Worst Case Voltage Drop Topology

These requirements place stringent restrictions on cable and connector IR drops (refer to Section 6.4) and port switch drops. However, they should be achievable with existing technologies at a reasonable cost.

7.2.3 Power Control During Suspend/Resume

When a device is put into the suspend mode, it must draw less than 500 μA from the bus. This current includes power drawn through the bus pull-up and pull-down. For bus-powered hubs, this current does not include the current drawn by powered devices connected to downstream ports.

When a hub is in the suspend mode, it still must be able to provide the maximum current per port (one unit of current per port for bus-powered hubs and five units per port for self-powered hubs). This is necessary to support remote wake-up capable devices which will power-up while the remainder of the system is still suspended.

When devices wake-up, either by themselves (remote wake-up) or by seeing resume signaling, they must limit the inrush current on V_{bus}. The target maximum droop in the hub V_{bus} is 330 mV or about 10% of the nominal signal swing from the function. The device must have sufficient on-board bypass capacitance or a controlled power-on sequence such that the current drawn from the hub does not exceed the maximum current capability of the port at any time while the device is waking up.

7.2.4 Dynamic Attach and Detach

The act of plugging or unplugging a hub or function must not affect the functionality of another device on other segments of the network. Unplugging a function will stop the transaction between that function and the host. However, the hub to which this function was attached will recover from this condition and will alert the host that the port has been disconnected.

7.2.4.1 Inrush Current Limiting

When a function or hub is plugged into the network, it has a certain amount of on-board capacitance between V_{bus} and ground. Also, the regulator will supply current to its output bypass capacitance and to the function as soon as power is applied. As a result, if no measures are taken to prevent it, there can be a surge of current into the device sufficient to pull the V_{bus} on the hub below its minimum operating level. Inrush currents can also occur when a high power function is switched into its high power mode. This problem must be solved by limiting the inrush current and by providing sufficient capacitance in each hub to prevent the power supplied to the other ports from going out of tolerance. An additional motivation for limiting inrush current is to minimize contact arcing, thereby prolonging connector contact life.

The target maximum droop in the hub V_{bus} is 330 mV or about 10% of the nominal signal swing from the function. In order to meet this target, the following conditions must be met:

- The maximum load that can be placed at the downstream end of a cable is 10 μF in parallel with 44 Ω. The 10 μF capacitance represents any bypass capacitor directly connected across the V_{bus} lines in the function plus any capacitive effects visible through the regulator in the device. The 44 Ω resistor represents one unit current load generated by the device during connect.
- If more bypass capacitance is required in the device due to large swings in the load current, then the device must incorporate some form of surge current limiting for current in the cable such that it matches the characteristics of the above load.
- The hub port V_{bus} power lines must be bypassed with no less than a 120 μF tantalum capacitor (equivalent aluminum capacitor values are under test). Good standard bypass methods should be used to minimize inductances between the bypass capacitors and the connectors. The bypass capacitors themselves should have a low dissipation factor to allow decoupling at higher frequencies.

The upstream port of a hub is also required to meet the above requirements. Furthermore, a bus-powered hub must provide additional surge limiting in the form of a soft start circuit when it enables power to its downstream ports.

Signal pins are protected from excessive currents during dynamic attach by being recessed in the connector such that the power pins make contact first. This guarantees that the power rails to the downstream device are referenced before the signal pins make contact. Also, the hub port signal lines are disabled and in a high impedance state during connect, allowing no current to flow for standard signal levels.

7.2.4.2 Dynamic Detach

When a device is detached from the network with power flowing in the cable, the inductance of the cable will cause a large flyback voltage to occur on the open end of the device cable. This flyback voltage is not destructive. Proper bypass measures on the hub ports will suppress any coupled noise. The frequency range of this noise is inversely dependent on the length of the cable to a maximum of 60 MHz for a one meter cable. This will require some low capacitance, very low inductance bypass capacitors on each hub port connector. The flyback voltage and the noise it creates is also moderated by the bypass capacitance on the device end of the cable. Also, there must be some minimum capacitance on the device end of the cable to insure that the inductive flyback on the open end of the cable does not cause the voltage on the device end to reverse polarity. A minimum of 1.0 μF is recommended for bypass across V_{bus}.

Again, signal pins are protected from excessive voltages during dynamic detach by being recessed in the connector such that they break contact before the power pins.

7.3 Physical Layer

The physical layer specifications are described in the following subsections.

7.3.1 Environmental

The operating environment for USB is 0 °C to 70 °C ambient.

USB must meet the following regulatory requirements;

EMI:

FCC part 15 class B

EN55022:1994 (Based on CISPR-22:1993)

EN5082-1:1992 (Generic Immunity standard)

VCCI (Japan version of CISPR-22)

Safety:

UL, CSA

7.3.2 Bus Timing/Electrical Characteristics

Table 7-4. DC Electrical Characteristics

Parameter	Symbol	Conditions (Notes 1,2)	Min.	Max.	Unit
Supply Voltage:					
Powered (Host or Hub) Port	VBUS		4.75	5.25	V
Bus-powered Hub Port	VBUS		4.40	5.25	V
Supply Current:					
Powered Host/Hub Port (out)	ICCPRT		500		mA
Bus-powered Hub Port (out)	ICCUPT		100		mA
High Power Function (in)	ICCHPF			500	mA
Low Power Function (in)	ICCLPF			100	mA
Unconfig. Function/Hub (in)	ICCINIT			100	mA
Suspended Device	ICCS			500	µA
Leakage Current:					
Hi-Z State Data Line Leakage	ILO	0 V< V_{IN}<3.3 V	-10	+10	µA
Input Levels:					
Differential Input Sensitivity	VDI	|(D+)-(D-)|, and Figure 7-4	0.2		V
Differential Common Mode Range	VCM	Includes VDI range	0.8	2.5	V
Single Ended Receiver Threshold	VSE		0.8	2.0	V
Output Levels:					
Static Output Low	VOL	RL of 1.5 kΩ to 3.6 V		0.3	V
Static Output High	VOH	RL of 15 kΩ to GND	2.8	3.6	V
Capacitance:					
Transceiver Capacitance	CIN	Pin to GND		20	pF
Downstream Hub Port Bypass Capacitance	CHPB	Vbus to GND (tantalum)	120		µF
Root Port Bypass Capacitance	CRPB	Vbus to GND Note 9	1.0	10.0	µF
Terminations:					
Bus Pull-up Resistor on Root Port	RPU	(1.5 kΩ ± 5%)	1.425	1.575	kΩ
Bus Pull-down Resistor on Downstream Port	RPD	(15 kΩ ± 5%)	14.25	15.75	kΩ
Cable Impedance and Timing:					
Cable Impedance (Full Speed)	ZO	(45 Ω ±15%)	38.75	51.75	Ω
Cable Delay (one way)	TCBL	Figure 7-21		30	ns

Table 7-5. Full Speed Source Electrical Characteristics

Parameter	Symbol	Conditions (Notes 1,2,3)	Min.	Max.	Unit
Driver Characteristics:					
Transition Time:		Note 5, 6 and Figure 7-16			
Rise Time	T_R	C_L = 50 pF	4	20	ns
Fall Time	T_F	C_L = 50 pF	4	20	ns
Rise / Fall Time Matching	T_{RFM}	(T_R/T_F)	90	110	%
Output Signal Crossover Voltage	V_{CRS}		1.3	2.0	V
Driver Output Resistance	Z_{DRV}	Steady State Drive	28	43	Ω
Data Source Timings:					
Full Speed Data Rate	T_{DRATE}	Ave. Bit Rate (12 Mb/s ± 0.25%)	11.97	12.03	Mbs
Frame Interval	T_{FRAME}	1.0 ms ± 0.05%	0.9995	1.0005	ms
Source Differential Driver Jitter		Note 7, 8 and Figure 7-29			
To Next Transition	T_{DJ1}		-3.5	3.5	ns
For Paired Transitions	T_{DJ2}		-4.0	4.0	ns
Source EOP Width	T_{EOPT}	Note 8 and Figure 7-30	160	175	ns
Differential to EOP transition Skew	T_{DEOP}	Note 8 and Figure 7-30	-2	5	ns
Receiver Data Jitter Tolerance		Note 8 and Figure 7-31			
To Next Transition	T_{JR1}		-18.5	18.5	ns
For Paired Transitions	T_{JR2}		-9	9	ns
EOP Width at receiver		Note 8 and Figure 7-30			
Must reject as EOP	T_{EOPR1}		40		ns
Must accept as EOP	T_{EOPR2}		82		ns

Table 7-6. Low Speed Source Electrical Characteristics

Parameter	Symbol	Conditions (Notes 1,2,4)	Min.	Max.	Unit
Driver Characteristics:					
Transition Time:		Note 5, 6 and Figure 7-16			
Rise Time	T_R	C_L = 50 pF	75		ns
		C_L = 350 pF		300	ns
Fall Time	T_F	C_L = 50 pF	75		ns
		C_L = 350 pF		300	ns
Rise/Fall Time Matching	T_{RFM}	(T_R/T_F)	80	120	%
Output Signal Crossover Voltage	V_{CRS}		1.3	2.0	V
Data Source Timings:					
Low Speed Data Rate	T_{DRATE}	Ave. Bit Rate (1.5 Mb/s ± 1.5%)	1.4775	1.5225	Mbs
Source Differential Driver Jitter		Note 7, 8 and Figure 7-29			
Host (Downstream):					
To Next Transition	T_{DDJ1}		-75	75	ns
For Paired Transitions	T_{DDJ2}		-45	45	ns
Function (Upstream):					
To Next Transition	T_{UDJ1}		-95	95	ns
For Paired Transitions	T_{UDJ2}		-150	150	ns
Source EOP Width	T_{EOPT}	Note 8 and Figure 7-30	1.25	1.50	µs
Differential to EOP transition Skew	T_{DEOP}	Note 8 and Figure 7-30	-40	100	ns
Receiver Data Jitter Tolerance		Note 8 and Figure 7-31			
At Host (Upstream):					
To Next Transition	T_{UJR1}		-152	152	ns
For Paired Transitions	T_{UJR2}		-200	200	ns
At Function (Downstream):					
To Next Transition	T_{DJR1}		-75	75	ns
For Paired Transitions	T_{DJR2}		-45	45	ns
EOP Width at receiver		Note 8 and Figure 7-30			
Must reject as EOP	T_{EOPR1}		330		ns
Must accept	T_{EOPR2}		675		ns

Table 7-7. Hub/Repeater Electrical Characteristics

Parameter	Symbol	Conditions (Note 2)	Min.	Max.	Unit
Hub Characteristics (Full Speed):					
Driver Characteristics (Refer to Table 7-5)		Root port and Downstream ports configured as Full Speed			
Hub Differential Data Delay		Note 3, 7, 8 and Figure 7-32			
(with cable)	THDD1			70	ns
(without cable)	THDD2			40	ns
Hub Differential Driver Jitter (including cable)		Note 3, 7, 8 and Figure 7-32			
To Next Transition	THDJ1		-3	3	ns
For Paired Transitions	THDJ2		-1	1	ns
Data bit width distortion after SOP	TSOP	Note 3, 8 and Figure 7-32	-5	3	
Hub EOP Delay Relative to t_{HDD}	TEOPD	Note 3, 8 and Figure 7-33	0	15	ns
Hub EOP Output Width Skew	THESK	Note 3, 8 and Figure 7-33	-15	15	ns
Hub Timings (Low Speed):					
Driver Characteristics (Refer to Table 7-6)		Downstream ports configured as Low Speed			
Hub Differential Data Delay	TLHDD	Note 4, 7, 8 and Figure 7-32		300	ns
Hub Differential Driver Jitter (including cable)		Note 4, 7, 8 and Figure 7-32			
Downstream:					
To Next Transition	TLDHJ1		-45	45	ns
For Paired Transitions	TLDHJ2		-15	15	ns
Upstream:					
To Next Transition	TLUHJ1		-45	45	ns
For Paired Transitions	TLUHJ2		-45	45	ns
Data bit width distortion after SOP	TSOP	Note 4, 8 and Figure 7-32	-60	45	
Hub EOP Delay Relative to t_{HDD}	TLEOPD	Note 4, 8 and Figure 7-33	0	200	ns
Hub EOP Output Width Skew	TLHESK	Note 4, 8 and Figure 7-33	-300	+300	ns

Note 1: All voltages measured from the local ground potential, unless otherwise specified.

Note 2: All timings use a capacitive load (CL) to ground of 50 pF, unless otherwise specified.

Note 3: Full Speed timings have a 1.5 kΩ pull-up to 2.8 V on the D+ data line.

Note 4: Low Speed timings have a 1.5 kΩ pull-up to 2.8 V on the D- data line.

Note 5: Measured from 10% to 90% of the data signal.

Note 6: The rising and falling edges should be smoothly transitioning (monotonic).

Note 7: Timing difference between the differential data signals.

Note 8: Measured at crossover point of differential data signals.

Note 9: The maximum load specification is the maximum effective capacitive load allowed that meets the target hub VBUS droop of 330 mV.

7.3.3 Timing Waveforms

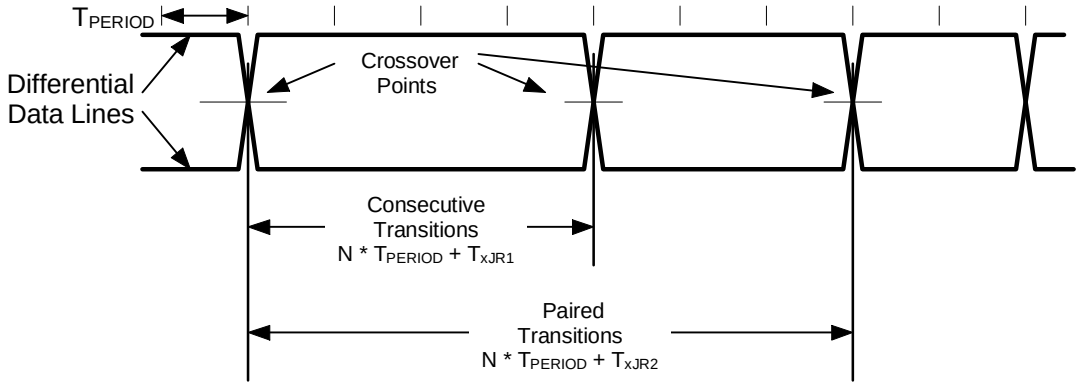


Figure 7-29. Differential Data Jitter

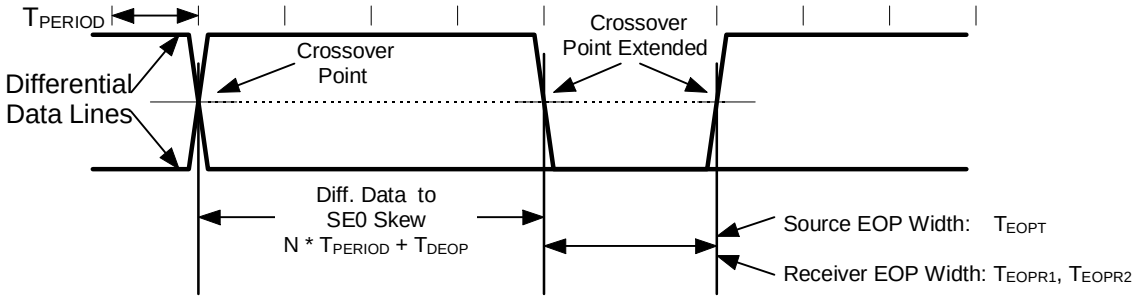


Figure 7-30. Differential to EOP Transition Skew and EOP Width

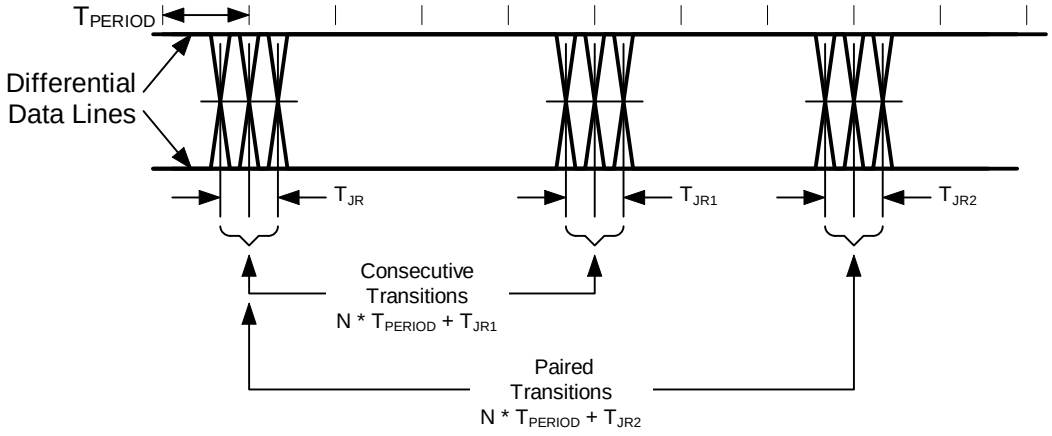
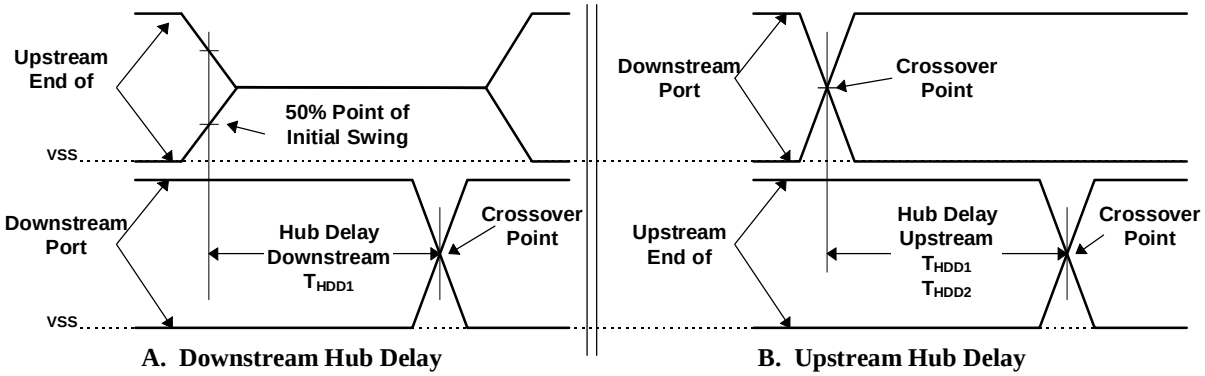


Figure 7-31. Receiver Jitter Tolerance

Refer to Section 7.1.11 for the definition of T_{PERIOD}



A. Downstream Hub Delay

B. Upstream Hub Delay

Hub Differential Jitter:

$T_{HDJ1} = T_{HDDx}(J) - T_{HDDx}(K)$ or $T_{HDDx}(K) - T_{HDDx}(J)$ Consecutive Transitions

$T_{HDJ2} = T_{HDDx}(J) - T_{HDDx}(J)$ or $T_{HDDx}(K) - T_{HDDx}(K)$ Paired Transitions

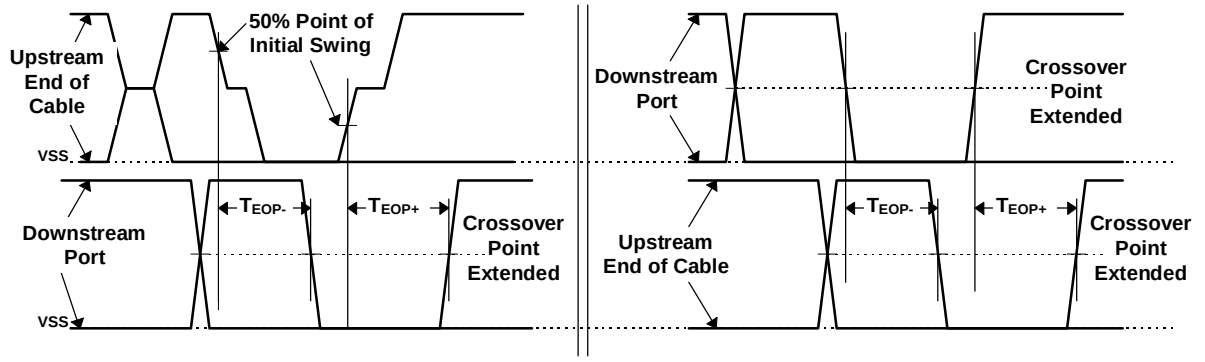
Bit after SOP Width Distortion: (Same as data jitter for SOP and next J transition.)

$T_{HDJ1} = T_{HDDx}(SOP) - T_{HDDx}(next\ J)$

Low Speed timings are determined in the same way for:

T_{LHDD}, T_{LDHJ1}, T_{LDJH2}, T_{LUHJ1}, T_{LUJH2}, and T_{LSOP} .

Figure 7-32. Hub Differential Delay, Differential Jitter, and SOP Distortion



A. Downstream EOP Delay

B. Upstream EOP Delay

EOP Delay

$T_{EOPD} = T_{EOP-}$

EOP Skew

$T_{HESK} = T_{EOP+} - T_{EOP-}$

Low Speed timings are determined in the same way for:

T_{LEOPD} and T_{LHESK}

Figure 7-33. Hub EOP Delay and EOP Skew

Chapter 8
Protocol Layer

This chapter presents a bottom-up view of the USB 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 are then covered. The chapter finishes with a discussion of retry synchronization, babble, and loss of bus activity recovery.

8.1 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.

8.2 SYNC Field

All packets begin with a 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 following packet diagrams (refer to Section 7.1.7). 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.

8.3 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) delimiter is described in Chapter 7.

8.3.1 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 in Figure 8-1. 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.

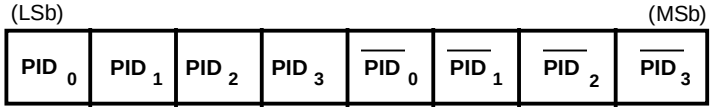


Figure 8-1. PID Format

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 in Table 8-1.

Table 8-1. PID Types

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 low speed devices.

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

8.3.2 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.

8.3.2.1 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. As shown in Figure 8-2, 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.

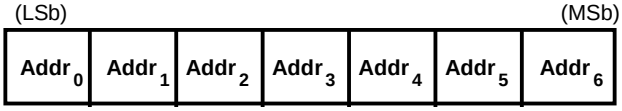


Figure 8-2. ADDR Field

8.3.2.2 Endpoint Field

An additional four-bit endpoint (ENDP) field, shown in Figure 8-3, 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.

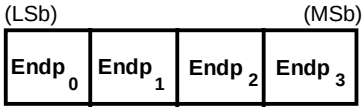


Figure 8-3. Endpoint Field

8.3.3 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.

8.3.4 Data Field

The data field may range from 0 to 1023 bytes and must be an integral numbers of bytes. Figure 8-4 shows the format for multiple bytes. Data bits within each byte are shifted out LSB first.

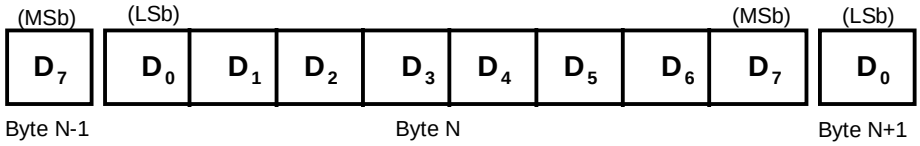


Figure 8-4. Data Field Format

Data packet size varies with the transfer type as described in Chapter 5.

8.3.5 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. For each data bit sent or received, the high order bit of the current remainder is XORed with the data bit and then the remainder is shifted left one bit and the low order bit set to 0. If the result of that XOR is 1, then the remainder is XORed with the generator polynomial.

When the last bit of the checked field is sent, the CRC in the generator is inverted and sent to the checker

MSB first. When the last bit of the CRC is received by the checker and no errors have occurred, the remainder will be equal to the polynomial residual.

Bit stuffing requirements must be met for the CRC, and this includes the need to insert a zero at the end of a CRC if the preceding six bits were all ones.

8.3.5.1 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:

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

The binary bit pattern that represents this polynomial is 00101. If all token bits are received without error, the five-bit residual at the receiver will be 01100.

8.3.5.2 Data CRCs

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

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

The binary bit pattern that represents this polynomial is 1000000000000101. If all data and CRC bits are received without error, the 16-bit residual will be 1000000000001101.

8.4 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.

8.4.1 Token Packets

Figure 8-5 shows the field formats for a token packet. 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.

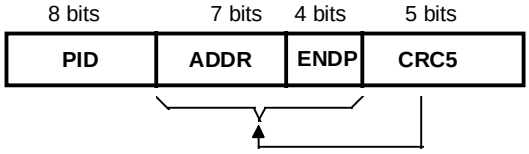


Figure 8-5. Token Format

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.

8.4.2 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 as illustrated in Figure 8-6.

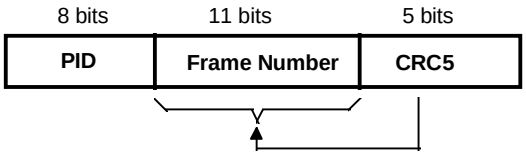


Figure 8-6. SOF Packet

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, it must comprehend both the PID and the time stamp.

8.4.3 Data Packets

A data packet consists of a PID, a data field, and a CRC as shown in Figure 8-7. 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 (refer to Section 8.6).

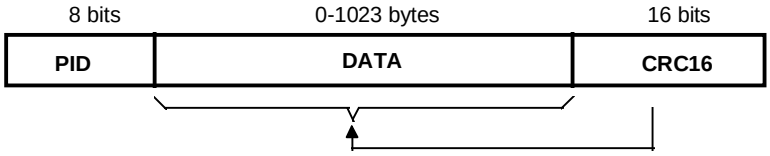


Figure 8-7. Data Packet Format

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.

8.4.4 Handshake Packets

Handshake packets, as shown in Figure 8-8, 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.

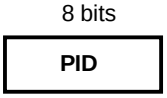


Figure 8-8. Handshake Packet

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 (refer to Section 8.6 for details). An ACK handshake is applicable only in transactions in 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. 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 (see Figure 8-9 and Figure 8-13). 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.

8.4.5 Handshake Responses

Transmitting and receiving functions must return handshakes based upon an order of precedence detailed in Table 8-2 through Table 8-4. Not all handshakes are allowed, depending on the transaction type and whether the handshake is being issued by a function or the host.

8.4.5.1 Function Response to IN Transactions

Table 8-2 shows the possible responses a function may make in response to an IN token. If the function is unable to send data, due to a stall or a flow control condition, it issues a STALL or NAK handshake, respectively. If the function is able to issue data, it does so. If the received token is corrupted, the function returns no response.

Table 8-2. Function Responses to IN Transactions

Token Received Corrupted	Function Tx Endpoint Stalled	Function Can Transmit Data	Action Taken
Yes	Don't care	Don't care	Return no response
No	Yes	Don't care	Issue STALL handshake
No	No	No	Issue NAK handshake
No	No	Yes	Issue data packet

8.4.5.2 Host Response to IN Transactions

Table 8-3 shows the host response to an IN transaction. The host is able to return only one type of handshake, an ACK. If the host receives a corrupted data packet, it discards the data and issues no response. If the host cannot accept data from a function, (due to problems such as internal buffer overrun) this condition is considered to be an error and the host returns no response. If the host is able to accept data and the data packet is received error free, the host accepts the data and issues an ACK handshake.

Table 8-3. Host Responses to IN Transactions

Data Packet Corrupted	Host Can Accept Data	Handshake Returned by Host
Yes	N/A	Discard data, return no response
No	No	Discard data, return no response
No	Yes	Accept data, issue ACK

8.4.5.3 Function Response to an OUT Transaction

Handshake responses for an OUT transaction are shown in Table 8-4. A function, upon receiving a data packet, may return any one of the three handshake types. If the data packet was corrupted, the function returns no handshake. If the data packet was received error free and the function's receiving endpoint is stalled, the function returns a STALL handshake. If the transaction is maintaining sequence bit synchronization and a mismatch is detected (refer to Section 8.6 for details), then the function returns ACK and discards the data. If the function can accept the data and has received the data error free, it returns an ACK handshake. If the function cannot accept the data packet due to flow control reasons, it returns a NAK.

Table 8-4. Function Responses to OUT Transactions in Order of Precedence

Data Packet Corrupted	Receiver Stalled	Sequence Bits Mismatch	Function Can Accept Data	Handshake Returned by Function
Yes	N/A	N/A	N/A	None
No	Yes	N/A	N/A	STALL
No	No	Yes	N/A	ACK
No	No	No	Yes	ACK
No	No	No	No	NAK

8.4.5.4 Function Response to a SETUP Transaction

Setup defines a special type of host to function data transaction which permits the host to initialize an endpoint's synchronization bits to those of the host. Upon receiving a Setup transaction, a function must accept the data. Setup transactions cannot be stalled or NAKed and the receiving function must accept the Setup transfer's data. If a non-control endpoint receives a SETUP PID, it must ignore the transaction and return no response.

8.5 Transaction Formats

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

8.5.1 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 as shown in Figure 8-9. 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.

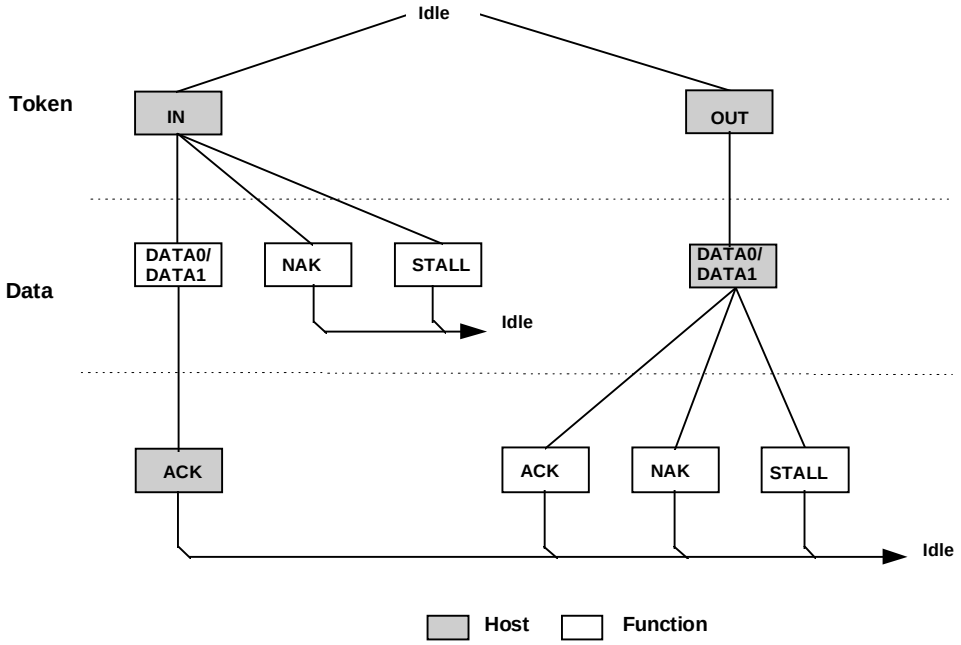


Figure 8-9. Bulk Transaction Format

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.

Figure 8-10 shows the sequence bit and data PID usage for bulk reads and writes. 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.

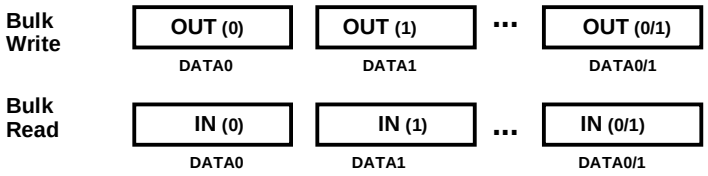


Figure 8-10. Bulk Reads and Writes

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 (refer to Section 8.6).

8.5.2 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. Figure 8-11 shows the Setup transaction format. 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.

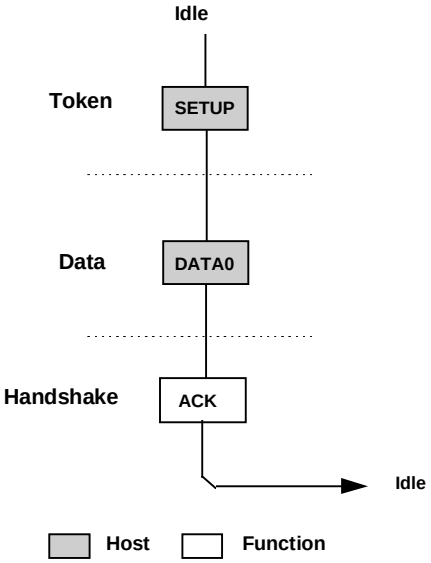


Figure 8-11. Control Setup Transaction

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. Figure 8-12 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.

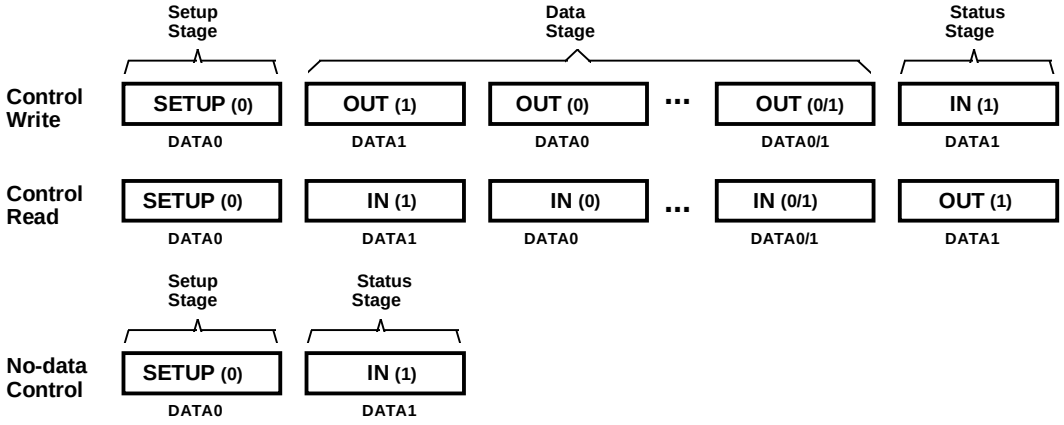


Figure 8-12. Control Read and Write Sequences

8.5.2.1 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. The following table summarizes the type of responses required for each. Control write transfers return status information on the data phase of the transfer. Control read transfers return status information on the handshake phase after the host has issued a zero length data packet during the previous data phase.

Table 8-5. Status Phase Responses

Status Response	Control Write Transfer (sent during data phase)	Control Read Transfer (send during handshake phase)
Function completes	0 length data packet	ACK handshake
Function has an error	STALL handshake	STALL handshake
Function is busy	NAK handshake	NAK handshake

For control reads, the host sends a zero length data packet to the control endpoint. The endpoint's handshake response indicates the completion status. NAK indicates that the function is still processing the command and that the host should continue the status phase. ACK indicates that the function has completed the command and is ready to accept a new command and STALL indicates that the function has an error that prevents it from completing the command.

For control writes, the function responds with either a handshake or a zero length data packet to indicate its status. A NAK indicates that the function is still processing the command and that the host should continue the status phase, return of a zero length packet indicates normal completion of the command, and STALL indicates that the function has an error that prevents it from completing the command. Control write transfers which return a zero length data packet during the data phase always cause the host to return an ACK handshake to the function.

If, during a Data or Status stage, a command endpoint is sent more data or is requested to return more data than was indicated in the Setup stage, it should return a STALL. If a control endpoint returns STALL during the Data stage, there will be no Status stage for that control transfer.

8.5.2.2 Error Handling on the Last Data Transaction

If the ACK handshake on an IN transaction is 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, the toggle retry mechanism cannot be used and an alternative scheme must be used.

The host that 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. A detailed analysis of retry policy is presented in Section 8.6.4.

8.5.3 Interrupt Transactions

Interrupt transactions consist solely of IN as shown in Figure 8-13. 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.

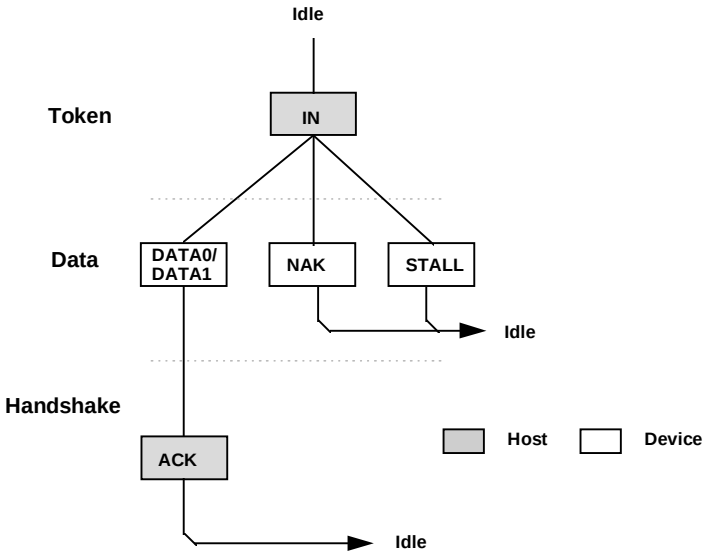


Figure 8-13. Interrupt Transaction Format

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 shown in Figure 8-10.

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.

8.5.4 Isochronous Transactions

ISO transactions have a token and data phase, but no handshake phase, as shown in Figure 8-14. 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.

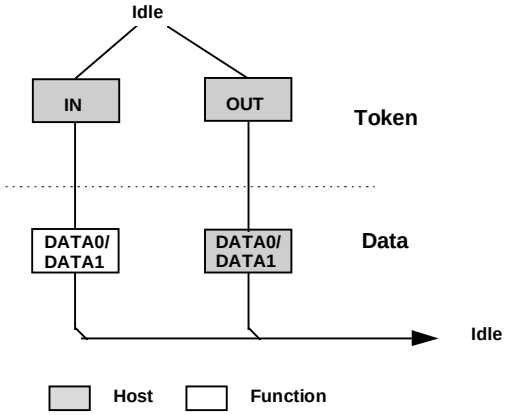


Figure 8-14. Isochronous Transaction Format

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

8.6 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. The synchronization mechanism used varies with the transaction type. Data toggle synchronization is not supported for ISO transfers.

8.6.1 Initialization via SETUP Token

Control transfers use the SETUP token for initializing host and function sequence bits. Figure 8-15 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 host's and function's sequence bits are equal to 1 at the end of the SETUP transaction.

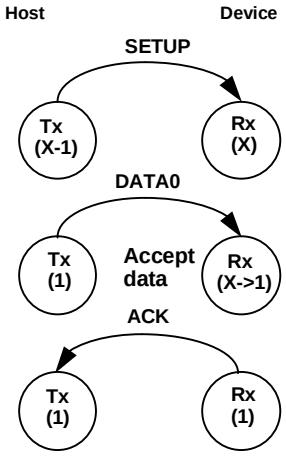


Figure 8-15. SETUP Initialization

8.6.2 Successful Data Transactions

Figure 8-16 shows the case where two successful transactions have occurred. For the data transmitter, this means that it toggles its sequence bit upon receipt of an ACK. The receiver toggles its sequence bit only if it receives a valid data packet and the packet's data PID matches the receiver's sequence bit.

During each transaction, the receiver compares the transmitter sequence bit (encoded in the data packet PID as either DATA0 or DATA1) with its receiver sequence bit. If data cannot be accepted, the receiver must issue a NAK. If data can be accepted and the receiver's sequence bit matches the PID sequence bit, then data is accepted. Sequence bits may only change if a data packet is transmitted. Two-phase transactions in which there is no data packet leave the transmitter and receiver sequence bits unchanged.

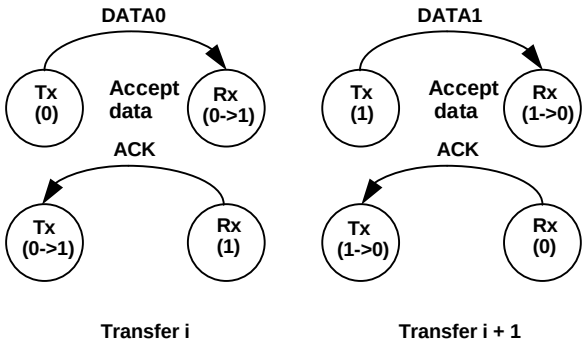


Figure 8-16. Consecutive Transactions

8.6.3 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 time out, depending on the circumstances, and the receiver will not toggle its sequence bit. Figure 8-17 shows the case 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.

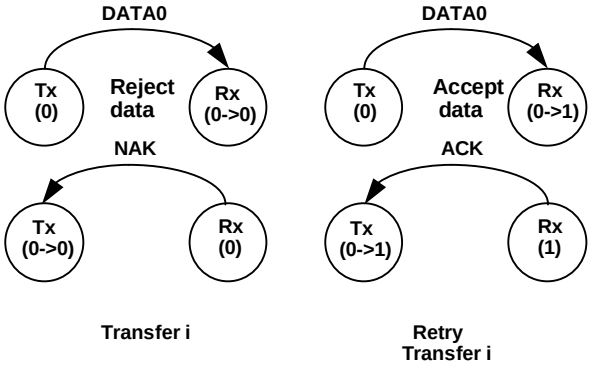


Figure 8-17. NAKed Transaction with Retry

8.6.4 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 as shown in Figure 8-18. Here the transmitter issues a valid data packet, which is successfully acquired by the receiver; however, the ACK handshake is corrupted.

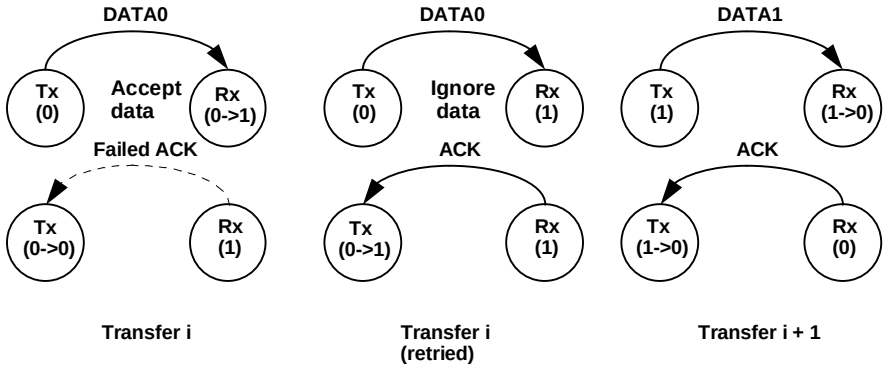


Figure 8-18. Corrupted ACK Handshake with Retry

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 the same length as 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.

8.6.5 Low Speed Transactions

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

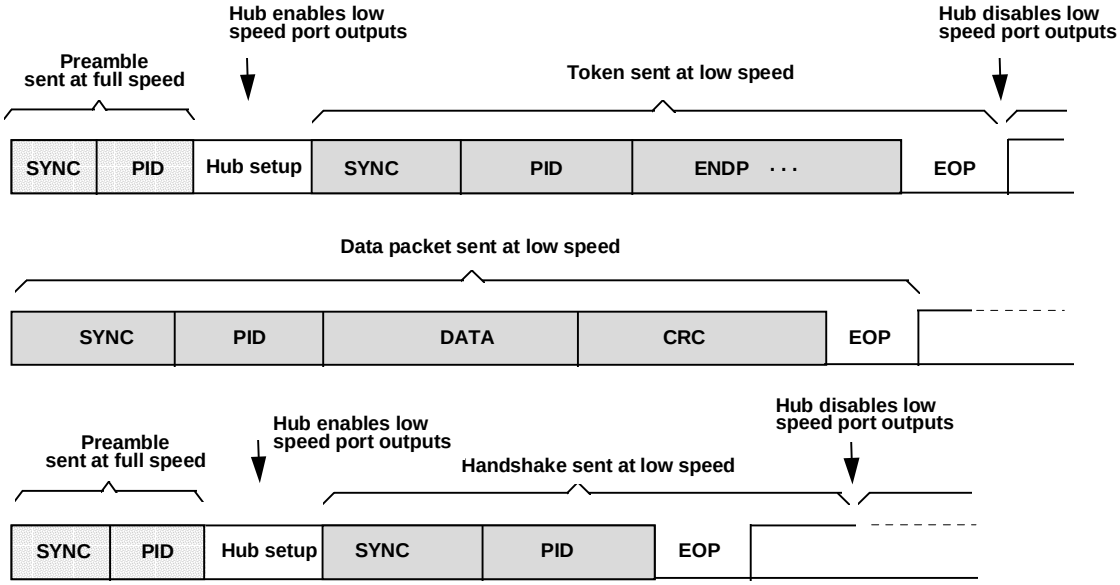


Figure 8-19. Low Speed Transaction

All downstream packets transmitted to low speed 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 four full speed bit times during which hubs must complete the process of configuring their repeater sections to accept low speed signaling. During this hub setup interval, hubs must drive their full speed and low speed 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.

Low speed signaling begins 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 low speed devices are connected. Hubs do not switch ports for upstream signaling; low speed ports remain enabled in the upstream direction for both low speed and full speed signaling.

Low speed and full speed transactions maintain a high degree of protocol commonality. However, low speed signaling does have certain limitations which include:

- Data payload limited to eight bytes, maximum.
- Low speed only supports interrupt and control types of transfers.
- The SOF packet is not received by low speed devices.

8.7 Error Detection and Recovery

USB permits 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.

8.7.1 Packet Error Categories

USB employs three error detection mechanisms: bit stuff violations, PID check bits, and CRCs. A bit stuff violation exists if a packet receiver detects seven or more consecutive bit times without a differential (J -> K or K -> J) transition, as detected on the physical D+ and D- lines, between the start and end of a packet. A PID error exists if the four PID check bits are not complements of their respective packet identifier bits. A CRC error exists if the computed checksum remainder at the end of a packet reception is not zero.

With the exception of the SOF token, any packet that is received corrupted causes the receiver to ignore it and discard any data or other field information that came with the packet. Table 8-6 lists error detection mechanisms, the types of packets to which they apply, and the appropriate packet receiver response.

Table 8-6. Packet Error Types

Field	Error	Action
PID	PID Check, Bit Stuff	Ignore packet
Address	Bit Stuff, Address CRC	Ignore token
Frame Number	Bit Stuff, Frame Number CRC	Ignore Frame Number field
Data	Bit Stuff, Data CRC	Discard data

8.7.2 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. The timer starts counting on the SE0 to IDLE transition of the EOP strobe and stops counting when the IDLE to K SOP transition is detected. 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 delay (refer to Section 7.1.14). 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.

As shown in Figure 8-20, 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.

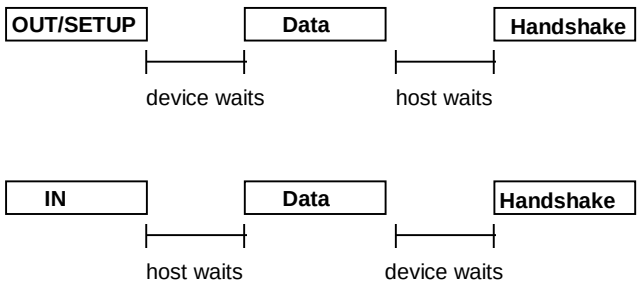


Figure 8-20. Bus Turnaround Timer Usage

8.7.3 False EOPs

False EOPs must be handled in a manner which guarantees that the packet currently in progress completes before the host or any other device attempts to transmit a new packet. If such an event were to occur, it would constitute a bus collision and have the ability to corrupt up to two consecutive transactions. Detection of false EOP relies upon the fact that a packet into which a false EOP has been inserted will appear as a truncated packet with a CRC failure. (The last 16 bits of the packet will have a very low probability of appearing to be a correct CRC.)

The host and devices handle false EOP situations differently. When a device sees a corrupted data packet, it issues no response and waits for the host to send the next token. This scheme guarantees that the device will not attempt to return a handshake while the host may still be transmitting a data packet. If a false EOP has occurred, the host data packet will eventually end, and the device will be able to detect the next token. If a device issues a data packet that gets corrupted with a false EOP, the host will ignore the packet and not issue the handshake. The device, expecting to see a handshake from the host, will time out.

If the host receives a corrupted data packet, it assumes that a false EOP may have occurred and waits for 16 bit times to see if there is any subsequent upstream traffic. If no bus transitions are detected within the 16 bit interval and the bus remains in the IDLE state, the host may issue the next token. Otherwise, the host waits for the device to finish sending the remainder of its packet. Waiting 16 bit times guarantees two conditions. The first condition is to make sure that the device has finished sending its packet. This is guaranteed by a time-out interval (with no bus transitions) greater than the worst case 6-bit time bit stuff interval. The second condition is that the transmitting device's bus turnaround timer must be guaranteed to expire. Note that the time-out interval is transaction speed sensitive. For full speed transactions, the host must wait 16 full speed bit times; for low speed transactions, it must wait 16 low speed bit times.

If the host receives a data packet with a valid CRC, it assumes that the packet is complete and need not delay in issuing the next token.

8.7.4 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 SOP followed by lack of bus activity (bus remains in J or K state) and no 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.

Chapter 9
USB Device Framework

A USB device may be divided into three layers. The bottom layer is a bus interface that transmits and receives packets. The middle layer handles routing data between the bus interface and various endpoints on the device. An endpoint is the ultimate consumer or provider of data. It may be thought of as a source or sink for data. The top layer is the functionality provided by the serial bus device; for instance, a mouse or ISDN interface.

This chapter describes the common attributes and operations of the middle layer of a USB device. These attributes and operations are used by the function-specific portions of the device to communicate through the bus interface and ultimately with the host.

9.1 USB Device States

A USB device has several possible states. Some of these states are visible to the USB and the host and others are internal to the USB device. This section describes those states.

9.1.1 Visible Device States

This section describes USB device states that are externally visible (see Figure 9-1). Note: USB devices perform a reset operation in response to a Reset request to the upstream port from the host. When reset signaling has completed, the USB device is reset.

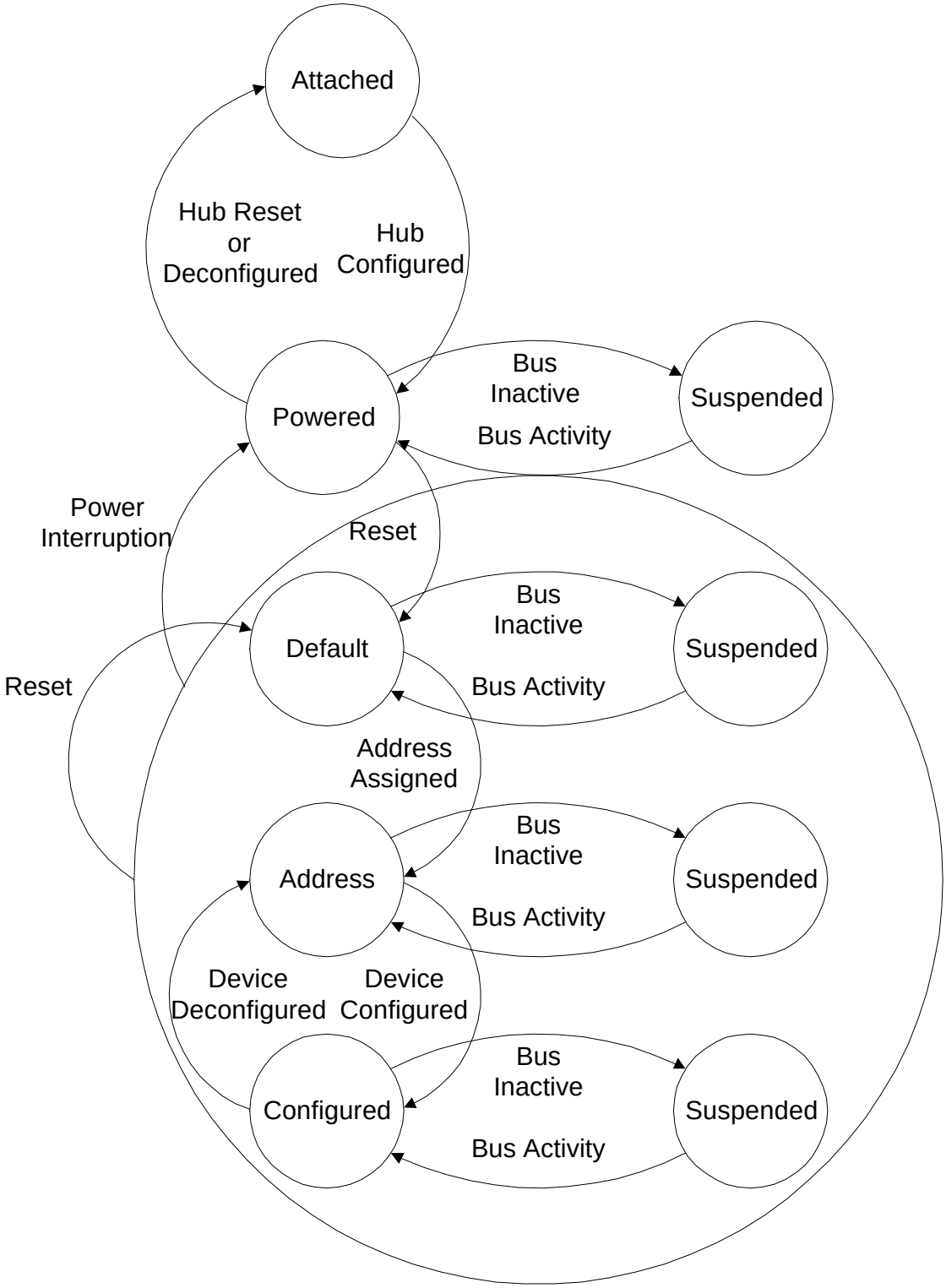


Figure 9-1. Device State Diagram

Table 9-1. Visible Device States

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to USB and powered, but has not been reset.
Yes	Yes	Yes	No	--	--	Device is attached to USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to USB, powered, has been reset, has unique address, is configured, and is not suspended. Host may now use the function provided by the device.
Yes	Yes	Yes	Yes	Yes	Yes	Device is, at minimum, attached to USB, has been reset, and is powered at the minimum suspend level. It may also have a unique address and be configured for use. However, since the device is suspended, the host may not use the device's function.

9.1.1.1 Attached

A USB device may be attached or detached from the USB. The state of a USB device when it is detached from the USB is not defined by this specification. This specification only addresses required operations and attributes once the device is attached.

9.1.1.2 Powered

USB devices may obtain power from an external source and/or from USB through the hub to which they are attached. Externally powered USB devices are termed self-powered. These devices may already be powered before they are attached to the USB. A device may support both self-powered and bus-powered configurations. Some device configurations support either power source. Other device configurations may only be available if the device is externally powered. Devices report their power source capability through the Configuration Descriptor. The current power source is reported as part of a device's status. Devices may change their power source at any time; e.g., from self- to USB-powered. If a configuration is capable of supporting both power modes, the power maximum reported for that configuration is the maximum the device will draw in either mode. The device must observe this maximum, regardless of its mode. If a configuration supports only one power mode and the power source of the device changes, the device will loose its current configuration and address and return to the attached state.

A hub port must be powered in order to detect port status changes, including attach and detach. Hubs do not provide any downstream power until they are configured, at which point they will provide power as allowed by their configuration and power source. A USB device must be able to be addressed within a specified time period from when power is initially applied (refer to Chapter 7). After an attachment to a port has been detected, the host shall enable the port, which will also reset the device attached to the port.

9.1.1.3 Default

After the device has been powered, it must not respond to any bus transactions until it has received a reset from the bus. After receiving a reset, the device is then addressable at the default address.

9.1.1.4 Address Assigned

All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device maintains its assigned address while suspended.

A USB device responds to requests on its default pipe whether the device is currently assigned a unique address or is using the default address.

9.1.1.5 Configured

Before the USB device's function may be used, the device must be configured. From the device's perspective, configuration involves writing a non-zero value to the device configuration register. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values. This includes setting the data toggle of any endpoint using data toggles to the value DATA0.

9.1.1.6 Suspended

In order to conserve power, USB devices automatically enter the Suspended state when the device has observed no bus traffic for a specified period (refer to Chapter 7). When suspended, the USB device maintains any internal status including its address and configuration.

All devices must suspend if bus activity has not been observed for the length of time specified in Chapter 7. Attached devices must be prepared to suspend at any time they are powered, whether they

have been assigned a non-default address or are configured. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled. This is referred to as selective suspend.

A USB device exits suspend mode when there is bus activity. A USB device may also request the host exit suspend mode or a selective suspend by using electrical signaling to indicate remote wakeup. The ability of a device to signal remote wakeup is optional. If a USB device is capable of remote wakeup signaling, the device must support the ability of the host to enable and disable this capability.

9.1.2 Bus Enumeration

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Chapter 11 for more information). At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, it's default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

9.2 Generic USB Device Operations

All USB devices support a common set of operations. This section describes those operations.

9.2.1 Dynamic Attachment and Removal

USB devices may be attached and removed at any time. The hub that provides the attachment point or port is responsible for reporting any change in the state of the port.

The host enables the hub port where the device is attached upon detection of an attachment, which also has the effect of resetting the device. A reset USB device has the following characteristics:

- Responds to the default USB address
- Is unconfigured
- Is not initially suspended

When a device is removed from a hub port, the host is notified of the removal. The host responds by disabling the hub port where the device was attached.

9.2.2 Address Assignment

When a USB device is attached, the host is responsible for assigning a unique address to the device after the device has been reset by the host and the hub port where the device is attached has been enabled.

9.2.3 Configuration

A USB device must be configured before its function may be used. The host is responsible for configuring a USB device. The host typically requests configuration information from the USB device to determine the device's capabilities.

As part of the configuration process, the host sets the device configuration and, where necessary, sets the maximum packet size for endpoints that require such limitation.

Within a single configuration, a device may support multiple interfaces. An interface is a related set of endpoints that present a single feature or function of the device to the host. The protocol used to communicate with this related set of endpoints and the purpose of each endpoint within the interface may be specified as part of a device class or vendor specific class definition.

In addition, an interface within a configuration may have alternate settings that redefine the number or characteristics of the associated endpoints. If this is the case, the device shall support the Get Interface and Set Interface requests to report or select a specific alternative setting for a specific interface.

Within each configuration, each interface descriptor contains fields that identify the interface number and the alternate setting. Interfaces are numbered from zero to one less than the number of concurrent interfaces supported by the configuration. Alternate settings range from zero to one less than the number of alternate settings for a specific interface. The default setting when a device is initially configured is alternate setting zero.

In support of adaptive device drivers that are capable of managing a related group of USB devices, the device and interface descriptors contain Class, SubClass, and Protocol fields. These fields are used to identify the function(s) provided by a USB device and the protocols used to communicate with the function(s) on the device. A Class code is assigned to a related class of devices that has been defined as a part of the USB specification. A class of devices is further subdivided into subclasses and within a class or subclass a protocol code defines how host software communicates with the device.

9.2.4 Data Transfer

Data may be transferred between a USB device endpoint and the host in one of four ways. Refer to Chapter 5 for the definition of the four types of transfers. Some endpoints may be capable of different types of data transfers. However, once configured, a USB device endpoint uses only one data transfer method.

9.2.5 Power Management

Power management on USB devices involves the issues described in the following sections.

9.2.5.1 Power Budgeting

For bus-powered devices, power is a limited resource. When a host detects the attachment of a bus-powered USB device, the host needs to evaluate the power requirements of the device. If USB device power requirements exceed available power, the device is not configured.

No USB device may require more than 100 mA when first attached. A configured bus-powered USB device attached to a self-powered hub may use up to 500 mA; however, some ports may not be able supply this much power and thus the device will not be usable.

All USB devices must support a suspended mode that requires less than 500 µA. A USB device automatically suspends when the bus is inactive, as previously described.

9.2.5.2 Remote Wakeup

Remote wakeup allows a suspended USB device to signal a host that may also be suspended. This notifies the host that it should resume from its suspended mode, if necessary, and service the external event that triggered the suspended USB device to signal the host. A USB device reports its ability to support remote wakeup in a configuration descriptor. If a device supports remote wakeup, it must also allow the capability to be enabled and disabled using the standard USB requests.

Remote wakeup is accomplished using electrical signaling described elsewhere in this document.

9.3 USB Device Requests

All USB devices respond to requests from the host on the device's default pipe. These requests are made using control transfers. The request and the request's parameters are sent to the device in the setup packet. The host is responsible for establishing the values passed in the following fields. Every setup packet has eight bytes, used as follows:

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bit-map	Characteristics of request D7 Data xfer direction 0 = Host to device 1 = Device to host D6..5 Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4..0 Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-2)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word sized field that varies according to request - typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a data phase

9.3.1 bmRequestType

This bit-mapped field identifies the characteristics of the specific request. In particular, this field identifies the direction of data transfer in the second phase of the control transfer. The state of the direction bit is ignored if the *wLength* field is zero, signifying there is no data phase.

The USB Specification defines a series of Standard requests that all devices must support. In addition, a device class may define additional requests. A device vendor may also define requests supported by the device.

Requests may be directed to the device, an interface on the device, or a specific endpoint on a device. This field also specifies the intended recipient of the request. When an interface or endpoint is specified, the *wIndex* field identifies the interface or endpoint.

9.3.2 bRequest

This field specifies the particular request. The *Type* bits in the *bmRequestType* field modify the meaning of this field. This specification only defines values for the *bRequest* field when the bits are reset to zero indicating a standard request (refer to Table 9-2).

9.3.3 wValue

The contents of this field vary according to the request. It is used to pass a parameter to the device specific to the request.

9.3.4 wIndex

The contents of this field vary according to the request. It is used to pass a parameter to the device specific to the request.

9.3.5 wLength

This field specifies the length of the data transferred during the second phase of the control transfer. The direction of data transfer (host to device or device to host) is indicated by the *Direction* bit of the *bRequestType* field. If this field is zero, there is no data transfer phase.

9.4 Standard Device Requests

This section describes the standard device requests defined for all USB devices (refer to Table 9-2).

USB devices must respond to standard device requests whether the device has been assigned a non-default address or the device is currently configured.

Table 9-2. Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Table 9-3. Standard Request Codes

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved for future use	2
SET_FEATURE	3
reserved for future use	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

Table 9-4. Descriptor Types

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5

Feature selectors are used when enabling or setting features, such as remote wakeup, specific to a device, interface or endpoint. The values for the feature selectors are given below.

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	Device	1
ENDPOINT_STALL	Endpoint	0

If an unsupported or invalid request is made to a USB device, the device responds by indicating a stall condition on the pipe used for the request. Control pipes, including the default pipe, must accept a setup transaction even if they are stalled. The ClearStall request is used to clear a stalled pipe. After the stall condition is cleared by the host, system software continues normal accesses to the control pipe. If for any reason, the device becomes unable to communicate via its default pipe due to an error condition, the device must be reset to clear the condition and restart the default pipe.

9.4.1 Clear Feature

This request is used to clear or disable a specific feature

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device, only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Section 9.4 for a definition of which feature selector values are defined for which recipients.

A ClearFeature request that references a feature that cannot be cleared or that does not exist will cause a stall.

9.4.2 Get Configuration

This request returns the current device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value

If the returned value is zero, the device is not configured.

9.4.3 Get Descriptor

This request returns the specified descriptor if the descriptor exists.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID (refer to Section 9.6.5)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-4). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to return. If the descriptor is longer than the *wLength* field, only the initial bytes of the descriptor are returned. If the descriptor is shorter than the *wLength* field, the device indicates the end of the control transfer by sending a short

packet when further data is requested. A short packet is defined as a packet shorter than the maximum payload size or a NULL data packet (refer to Chapter 5).

The standard request to a device supports three types of descriptors: DEVICE, CONFIGURATION, and STRING. A request for a configuration descriptor returns the configuration descriptor, all interface descriptors, and endpoint descriptors for all of the interfaces in a single request. The first interface descriptor immediately follows the configuration descriptor. The endpoint descriptors for the first interface follow the first interface descriptor. If there are additional interfaces, their interface descriptor and endpoint descriptors follow the first interface's endpoint descriptors.

All devices must provide a device descriptor and at least one configuration descriptor. If a device does not support a requested descriptor, it responds by stalling the pipe used for the request. A non-zero value as the first byte of a descriptor indicates the buffer contains a valid descriptor.

9.4.4 Get Interface

This request returns the selected alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Setting

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to determine the currently selected alternative setting.

9.4.5 Get Status

This request returns status for the specified recipient.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status

The Recipient bits of the *bRequestType* field specify the desired recipient. The data returned is the current status of the specified recipient.

A GetStatus request to a device returns the following information in little-endian order:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)						Remote Wakeup	Self Powered
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

The *Self Powered* field indicates whether the device is currently bus-powered or self-powered. If D0 is reset to zero, the device is bus-powered. If D0 is set to one, the device is self-powered. The *Self Powered* field may not be changed by the SetFeature or ClearFeature requests.

The *Remote Wakeup* field indicates whether the device is currently enabled to request remote wakeup. The default mode for devices that support remote wakeup is disabled. If D1 is reset to zero, the ability of the device to signal remote wakeup is disabled. If D1 is set to one, the ability of the device to signal remote wakeup is enabled. The *Remote Wakeup* field can be modified by the SetFeature and ClearFeature requests using the DEVICE_REMOTE_WAKEUP feature selector. This field is reset to zero when the device is reset.

A GetStatus request to an interface returns the following information in little-endian order:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

If the request is made to an endpoint, then the endpoint must be specified in the *wIndex* field. The upper byte of *xIndex* is reset to zero and the lower byte contains the endpoint number as follows:

D7	D6	D5	D4	D3	D2	D1	D0
Direction	Reserved (Reset to zero)			Endpoint Number			

For IN endpoints, D7 is set to one. For OUT endpoints, D7 is reset to zero.

A GetStatus request to an endpoint returns the following information:

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							Stall
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

If the endpoint is currently stalled, the *Stall* field is set to one. Otherwise the *Stall* field is reset to zero. The *Stall* field may be changed with the SetFeature and ClearFeature requests with the ENDPOINT_STALL feature selector. When set by the SetFeature request, the endpoint exhibits the same stall behavior as if the field had been set by a hardware condition. If the condition causing a stall has been removed, clearing the stall field results in the endpoint no longer returning a stall status. For this endpoints using a data toggle, clearing a stalled endpoint results in the data toggle being reinitialized to DATA0. This field is reset to zero after either the SetConfiguration or SetInterface request.

9.4.6 Set Address

This request sets the device address for all future device accesses.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None

The *wValue* field specifies the device address to use for all subsequent accesses.

As noted elsewhere, requests actually may result in up to three stages. In the first stage, the setup packet is sent to the device. In the optional second stage, data is transferred between the host and the device. In the final stage, status is transferred between the host and the device. The direction of data and status transfer depends on whether the host is sending data to the device or the device is sending data to the host. The status stage transfer is always in the opposite direction of the data stage. If there is no data stage, the status stage is from the device to the host.

Stages after the initial setup packet assume the same device address as the setup packet. The USB device does not change its device address until after the status stage of this request is completed successfully. Note that this is a difference between this request and all other requests. For all other requests, the operation indicated must be completed before the status stage.

9.4.7 Set Configuration

This request sets the device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None

The *wValue* field specifies the desired configuration. This value must be zero or match a configuration value from a configuration descriptor. If the value is zero, the device is placed in its unconfigured state.

9.4.8 Set Descriptor

This request is optional. If a device supports this request, existing descriptors may be updated or new descriptors may be added.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID (refer to Section 9.6.5)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-4). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to transfer from the host to the device.

9.4.9 Set Feature

This request is used to set or enable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device; only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Section 9.4 for a definition of which feature selector values are defined for which recipients. A SetFeature request that references a feature that cannot be set or that does not exist causes a stall.

9.4.10 Set Interface

This request allows the host to select an alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	Alternative Setting	Interface	Zero	None

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to select the desired alternate setting.

9.4.11 Synch Frame

This request is used to set and then report an endpoint's synchronization frame.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

When an endpoint supports isochronous transfers, the endpoint may also require per frame transfers to vary in size according to a specific pattern. The host and the endpoint must agree on which frame the repeating pattern begins. This request causes the endpoint to begin monitoring the SOF frame number to track the position of a given frame in its pattern. The number of the frame in which the pattern began is returned to the host. This frame number is the one conveyed to the endpoint by the last SOF prior to the first frame of the pattern.

This value is only used for isochronous data transfers using implicit pattern synchronization. If the endpoint is not isochronous or is not using this method, this request is not supported by the endpoint and returns a stall.

The starting frame is reset to zero by a device reset or by configuring the endpoint either via a SetConfiguration or a SetInterface request.

9.5 Descriptors

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type.

Using descriptors allows concise storage of the attributes of individual configurations because each configuration may reuse descriptors or portions of descriptors from other configurations that have the same characteristics. In this manner, the descriptors resemble individual data records in a relational database.

Where appropriate, descriptors contain references to string descriptors that provide displayable information describing a descriptor in human-readable form. The inclusion of string descriptors is optional. However, the reference fields within descriptors are mandatory. If a device does not support string descriptors, string reference fields must be reset to zero to indicate no string descriptor is available.

If a descriptor returns with a value in its length field that is less than defined by this specification, the descriptor is invalid and should be rejected by the host. If the descriptor returns with a value in its length field that is greater than defined by this specification, the extra bytes are ignored by the host, but the next descriptor is located using the length returned rather than the length expected.

Class and vendor specific descriptors may be returned in one of two ways. Class and vendor specific descriptors that are related to standard descriptors are returned in the same data buffer as the standard descriptor immediately following the related standard descriptor.

If, for example, a class or vendor specific descriptor is related to an interface descriptor, the related class or vendor specific descriptor is placed between the interface descriptor and the interface's endpoint descriptors in the buffer returned in response to a GET_CONFIGURATION_DESCRIPTOR request. The length of a standard descriptor is not increased to accommodate device class or vendor specific extensions. Class or vendor specific descriptors follow the same format as standard descriptors with the length and type fields as the first two bytes of the specific descriptor.

Class or vendor specific descriptors that are not related to a standard descriptor are returned using class or vendor specific requests.

9.6 Standard USB Descriptor Definitions

9.6.1 Device

A device descriptor describes general information about a USB device. It includes information that applies globally to the device and all of the device's configurations. A USB device has only one device descriptor.

All USB devices have an endpoint zero used by the default pipe. The maximum packet size of a device's endpoint zero is described in the device descriptor. Endpoints specific to a configuration and its interface(s) are described in the configuration descriptor. A configuration and its interface(s) do not include an endpoint descriptor for endpoint zero. Other than the maximum packet size, the characteristics of endpoint zero are defined by this specification and are the same for all USB devices.

The *bNumConfigurations* field identifies the number of configurations the device supports.

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	DEVICE Descriptor Type
2	*bcdUSB*	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e., 2.10 is 0x210). This field identifies the release of the USB Specification that the device and its descriptors are compliant with.
4	*bDeviceClass*	1	Class	Class code (assigned by USB). If this field is reset to 0, each interface within a configuration specifies its own class information and the various interfaces operate independently. If this field is set to a value between 1 and 0xFE, the device supports different class specifications on different interfaces and the interfaces may not operate independently. This value identifies the class definition used for the aggregate interfaces. (For example, a CD-ROM device with audio and digital data interfaces that require transport control to eject CDs or start them spinning.) If this field is set to 0xFF, the device class is vendor specific.

Offset	Field	Size	Value	Description
5	*bDeviceSubClass*	1	SubClass	Subclass code (assigned by USB). These codes are qualified by the value of the *bDeviceClass* field. If the *bDeviceClass* field is reset to 0, this field must also be reset to 0. If the *bDeviceClass* field is not set to 0xFF, all values are reserved for assignment by USB.
6	*bDeviceProtocol*	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the *bDeviceClass* and the *bDeviceSubClass* fields. If a device supports class-specific protocols on a device basis as opposed to an interface basis, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to 0, the device does not use class specific protocols on a device basis. However, it may use class specific protocols on an interface basis. If this field is set to 0xFF, the device uses a vendor specific protocol on a device basis.
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	*idVendor*	2	ID	Vendor ID (assigned by USB)
10	*idProduct*	2	ID	Product ID (assigned by the manufacturer)
12	*bcdDevice*	2	BCD	Device release number in binary-coded decimal
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer
15	*iProduct*	1	Index	Index of string descriptor describing product
16	*iSerialNumber*	1	Index	Index of string descriptor describing the device's serial number
17	*bNumConfigurations*	1	Number	Number of possible configurations

9.6.2 Configuration

The configuration descriptor describes information about a specific device configuration. The descriptor contains a *bConfigurationValue* field with a value that, when used as a parameter to the Set Configuration request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface may operate independently. For example, an ISDN device might be configured with two interfaces, each providing 64 kBs bi-directional channels that have separate data sources or sinks on the host. Another configuration might present the ISDN device as a single interface, bonding the two channels into one 128 kBs bi-directional channel.

When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned (refer to Section 9.4.2).

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has one or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Within an interface, an isochronous endpoint's maximum packet size may also be adjusted.

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	CONFIGURATION
2	*wTotalLength*	2	Number	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class or vendor specific) returned for this configuration.
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this configuration
5	*bConfigurationValue*	1	Number	Value to use as an argument to Set Configuration to select this configuration
6	*iConfiguration*	1	Index	Index of string descriptor describing this configuration
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7 Bus Powered D6 Self Powered D5 Remote Wakeup D4..0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source sets both D7 and D6. The actual power source at runtime may be determined using the Get Status device request. If a device configuration supports remote wakeup, D5 is set to 1.

Offset	Field	Size	Value	Description
8	*MaxPower*	1	mA	Maximum power consumption of USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA). Note: A device configuration reports whether the configuration is bus-powered or self-powered. Device status reports whether the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate that it is no longer self-powered. A device may not increase its power draw from the bus, when it loses its external power source, beyond the amount reported by its configuration. If a device can continue to operate when disconnected from its external power source, it continues to do so. If the device cannot continue to operate, it fails operations it can no longer support. Host software may determine the cause of the failure by checking the status and noting the loss of the device's power source.

9.6.3 Interface

This descriptor describes a specific interface provided by the associated configuration. A configuration provides one or more interfaces, each with its own endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoints for a particular interface immediately follow the interface descriptor in the data returned by the Get Configuration request. An interface descriptor is always returned as part of a configuration descriptor. It cannot be directly accessed with a Get or Set Descriptor request.

An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. The Set Interface request is used to select an alternate setting or to return to the default setting. The Get Interface request returns the selected alternate setting.

Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation. If a configuration has alternate settings for one or more of its interfaces, a separate interface descriptor and its associated endpoints are included for each setting.

If a device configuration supported a single interface with two alternate settings, the configuration descriptor would be followed by an interface descriptor with the *bInterfaceNumber* and *bAlternateSetting* fields set to zero and then the endpoint descriptors for that setting, followed by another interface descriptor and its associated endpoint descriptors. The second interface descriptor's *bInterfaceNumber* field would also be set to zero, but the *bAlternateSetting* field of the second interface descriptor would be set to one.

If an interface only uses endpoint zero, no endpoint descriptors follow the interface descriptor and the interface identifies a request interface that uses the default pipe attached to endpoint zero. In this case, the *bNumEndpoints* field shall be set to zero.

An interface descriptor never includes endpoint zero in the number of endpoints.

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	1	Number	Value used to select alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is 0, this interface only uses endpoint zero.
5	*bInterfaceClass*	1	Class	Class code (assigned by USB) If this field is reset to 0, the interface does not belong to any USB specified device class. If this field is set to 0xFF, the interface class is vendor specific. All other values are reserved for assignment by USB.
6	*bInterfaceSubClass*	1	SubClass	Subclass code (assigned by USB). These codes are qualified by the value of the *bInterfaceClass* field. If the *bInterfaceClass* field is reset to 0, this field must also be reset to 0. If the *bInterfaceClass* field is not set to 0xFF, all values are reserved for assignment by USB.

Offset	Field	Size	Value	Description
7	*bInterfaceProtocol*	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the *bInterfaceClass* and the *bInterfaceSubClass* fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to 0, the device does not use a class specific protocol on this interface. If this field is set to 0xFF, the device uses a vendor specific protocol for this interface.
8	*iInterface*	1	Index	Index of string descriptor describing this interface

9.6.4 Endpoint

Each endpoint used for an interface has its own descriptor. This descriptor contains the information required by the host to determine the bandwidth requirements of each endpoint. An endpoint descriptor is always returned as part of a configuration descriptor. It cannot be directly accessed with a Get or Set Descriptor request. There is never an endpoint descriptor for endpoint zero.

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: The endpoint number Bit 4..6: Reserved, reset to 0 Bit 7: Direction, ignored for control endpoints 0 OUT endpoint 1 IN endpoint

Offset	Field	Size	Value	Description
3	*bmAttributes*	1	Bit Map	This field describes the endpoint's attributes when it is configured using the *bConfigurationValue*. Bit 0 .. 1: Transfer Type 00 Control 01 Isochronous 10 Bulk 11 Interrupt All other bits are reserved
4	*wMaxPacketSize*	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For isochronous endpoints, this value is used to reserve the bus time in the schedule, required for the per frame data payloads. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. For interrupt, bulk, and control endpoints smaller data payloads may be sent, but will terminate the transfer and may or may not require intervention to restart. Refer to Chapter 5 for more information.
6	*bInterval*	1	Number	Interval for polling endpoint for data transfers. Expressed in milliseconds. This field is ignored for bulk and control endpoints. For isochronous endpoints this field must be set to 1. For interrupt endpoints, this field may range from 1 to 255.

9.6.5 String

String descriptors are optional. As noted previously, if a device does not support string descriptors, all references to string descriptors within device, configuration, and interface descriptors must be reset to zero.

String descriptors use UNICODE encodings as defined by *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volumes 1 and 2*, The Unicode Consortium, Addison-Wesley Publishing Company, Reading, Massachusetts. The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a sixteen-bit language ID (LANGID) defined by Microsoft for Windows as described in *Developing International Software for Windows 95 and Windows NT*, Nadine Kano, Microsoft Press, Redmond, Washington. String index 0 for

all languages returns an array of two-byte LANGID codes supported by the device. A USB device may omit all string descriptors.

The UNICODE string descriptor is not NULL terminated. The string length is computed by subtracting two from the value of the first byte of the descriptor.

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	STRING Descriptor Type
2	*bString*	N	Number	UNICODE encoded string

9.7 Device Class Definitions

All devices must support the above registers and descriptor definitions. Most devices provide additional registers and possibly, descriptors for device-specific extensions. In addition, devices may provide extended services which are common to a group of devices. In order to define a class of devices, the following information must be provided to completely define the appearance and behavior of the device class.

9.7.1 Descriptors

If the class requires any specific definition of the standard descriptors, the class definition must include those requirements as part of the class definition. In addition, if the class defines a standard extended set of descriptors, they must also be fully defined in the class definition. Any extended descriptor definitions should follow the approach used for standard descriptors; for example, all descriptors should begin with a length field.

9.7.2 Interface(s) and Endpoint Usage

When a class of devices is standardized, the interfaces used by the devices including how endpoints are used must be included in the device class definition. Devices may further extend a class definition with proprietary features as long as they meet the base definition of the class.

9.7.3 Requests

All of the requests specific to the class must be defined.

9.8 Device Communications

The USB communications model characterizes data and control traffic between the host and a given device across the USB interconnect. The host and the device are divided into the distinct layers shown in Figure 9-2.

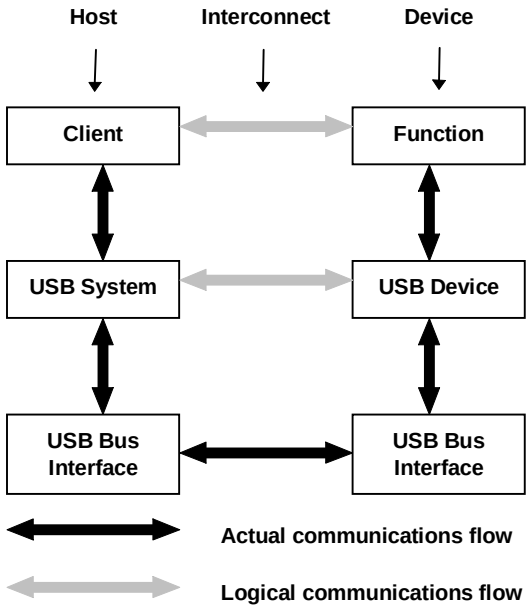


Figure 9-2. Interlayer Communications Model

The actual communication on the host, as indicated by vertical arrows, takes place via SPIs. The interlayer relationships on the device are implementation-specific. Between the host and device, all communications must ultimately occur on the physical USB wire. However, there are logical host-device interfaces between horizontal layers. Between client software, resident on the host, and the function provided by the device, communications are typified by a contract based on the needs of the application currently using the device and the capabilities provided by the device. This client-function interaction creates the requirements for all of the underlying layers and their interfaces.

This section describes the communications model from the point of view of the device and its layers. Figure 9-3 illustrates, based on the overall view introduced in Chapter 8, the device's view of its communication with the host.

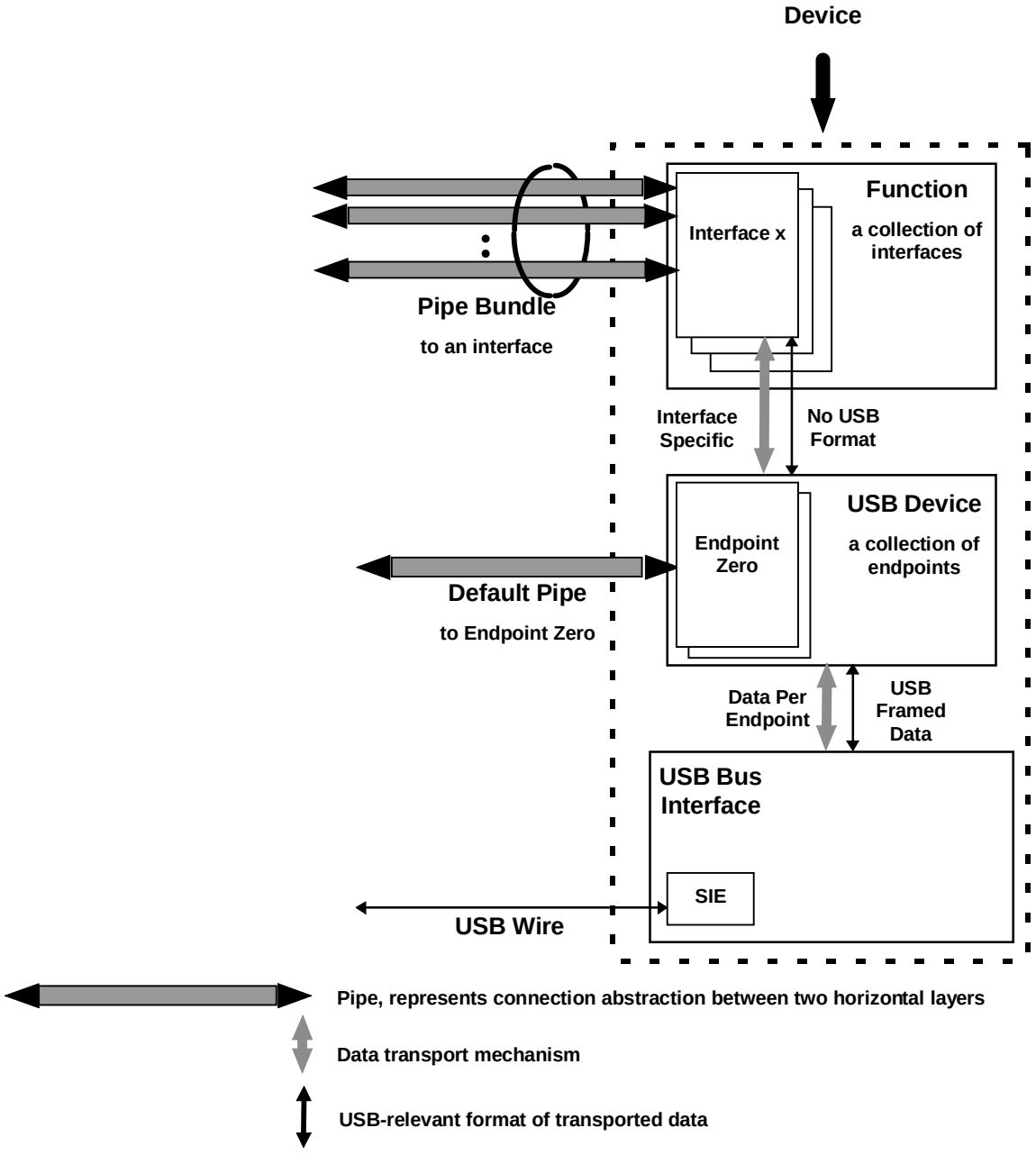


Figure 9-3. Device Communications

The USB bus interface handles interactions among the electrical and protocol layers (refer to Chapters 7 and 8). The USB device layer presents a uniform abstraction of the USB device to the host. It is this layer that is primarily described here. The function layer uses the capabilities provided by the USB device layer, combined into a given interface, to support the requirements of a host-based application.

A USB device acts as a collection of endpoints, each capable of supporting different types of pipes. Each pipe can support one of the following transfer types at a time:

- Control
- Isochronous
- Interrupt
- Bulk

These transfer types are described in more detail in Chapter 5. Each of the transfer types, however, requires the associated endpoint to behave in a certain fashion. A given endpoint may support a variety of transfer types. However, once a pipe is associated with an endpoint, the endpoint only uses a single transfer type. In this section, when discussing the behavior of an endpoint for a given transfer type, it is assumed that the endpoint has been associated with a pipe supporting that specific transfer type. The basic communication mechanisms used by endpoints are:

- Pipe mode
- Start of Frame (SOF) synchronization
- Handshakes
- Data toggles

The mode of a pipe indicates whether the data flow across the pipe is stream or message mode. Devices may use the SOF as generated by the host to synchronize their internal clocks. Devices may use handshakes and data toggles to implement error and flow control.

Traffic between a client and a function may require a certain transport rate. The client, USB and the function all will be using, at best, slightly different clock rates. To ensure that all of the required data can be delivered with minimum buffering required, the various clocks must be synchronized. Refer to Chapter 5 for a discussion of the synchronization options. Additionally, in order to support the just-in-time delivery capability implied by clock synchronization, the size of the data packets transmitted between the host and the device will be normalized such that variations in size over time are minimized. To support data flows in which the loss of data is acceptable as long as the loss can be accurately communicated, sample headers may be used by the host and the device to communicate the expected transmission volumes. Refer to Chapter 5 for the definition of sample headers.

These basic communication mechanisms are described, from the device's point of view, in greater detail below. Each of the different transfer types uses these basic communication mechanisms in different ways.

9.8.1 Basic Communication Mechanisms

This section describes in more detail the basic communication mechanisms as supported by the USB Device layer.

9.8.1.1 Pipe Mode

A pipe supports either stream or message mode transfers. In stream mode, the data flow is considered to be a unidirectional serial stream of samples. In message mode, data is delivered as a related set of bytes. Message mode pipes are always considered to be capable of being bi-directional.

A stream mode pipe is always unidirectional. When in the stream mode, the endpoint expects to receive a token either requesting the endpoint to send data or alerting it that data will be sent to it. The amount of data sent will always be equal to or less than the current MaxPacketSize for the endpoint.

Message transfers begin with a command from the host to the device. The device may respond to the command with data, the host may follow the command with data for the device, or the command may

require no data to be transmitted in which case a NULL data packet will be sent. In message mode, an endpoint must keep track of where it is in the phase sequence defined by the mode. An endpoint expects the first transaction of a message sequence to be a setup for the subsequent communication. After setup is received, an endpoint usually expects to receive a token requesting the endpoint to send data (IN token) or alerting it that data will be sent to the endpoint (OUT Token). The endpoint will know what the direction of the subsequent transactions will be, based on the setup command that started the series of transactions. Some setup commands do not require subsequent transactions to or from an endpoint. Setup transactions are always eight bytes or less. The subsequent transactions will always be of a size equal to or less than the current MaxPacketSize for the endpoint.

9.8.1.2 Synchronization

The host provides a special SOF token to the bus at regularly timed intervals. The interval between SOF's, within error tolerances (refer to Chapter 7), is 1 ms. Endpoints may use the receipt of this token to synchronize their associated clock to the USB clock. This enables endpoints to match their rate of data consumption or production to the host's rate.

Not all endpoints require SOF synchronization. Some endpoints requiring synchronization have clocks which cannot be synchronized to the 1 ms bus clock provided by USB. Such devices have two choices. They can attempt to have the entire USB synchronize to them or such devices may periodically adjust their transfer rate as they compensate for the difference between the USB clock and their own clock.

USB provides for a maximum of one client per USB instance to adjust the host's SOF generation. This client performs the adjustment based on feedback provided by an associated device. The rate of SOF token generation remains 1 ms, however. Refer to Chapter 10 for a complete discussion of this adjustment mechanism. If an endpoint has not been configured to adjust the USB clock using the SOF handshake, or if the endpoint is not capable of so adjusting the clock, then the endpoint must continually adjust its data flow.

Therefore, as noted above, there are three possible types of synchronization interaction for an endpoint with regard to SOF. The endpoint may:

1. Synchronize its clock exactly to the existing USB clock.
2. Adjust the bus clock.
3. Synchronize with the host by adjusting its data flow.

It is important to note that an endpoint requiring synchronization, that cannot implement the type of synchronization described in (1) but can implement the type described in (2), must also implement the type described in (3). This is because such an endpoint cannot be guaranteed that it will be chosen as the endpoint to adjust the bus clock. Only one device on the entire USB will be used to adjust the SOF.

9.8.1.3 Handshakes

Endpoints use the handshake phase of USB transactions to communicate error and data flow needs to the host. Endpoints may also receive handshakes from the host to communicate error conditions. The types of handshakes used by the endpoint vary according to the transfer type supported. These handshakes are described in detail in Chapter 8.

9.8.1.4 Data Toggles

When an error or flow control situation occurs, some pipes are allowed to skip the frame in which the condition occurred and transfer the data scheduled for that frame during a subsequent frame. In some cases, it is possible that the receiver of the data had indicated to the transmitter that the data had been successfully received, but that the transmitter believes, due to a bus error, that the data was not received successfully. The transmitter will then retransmit the same data. The receiver needs some mechanism to

understand that the data it is now receiving is a retransmission of data it has already received and not new data.

USB provides this information by using data toggles which are the PIDs for the data phase of transactions. Depending on the transfer type, the endpoint needs to understand data toggles and generate or process data PIDs accordingly. Refer to Chapter 8 for a more complete discussion of data toggles.

Table 9-5 summarizes the USB communication mechanisms.

Table 9-5. USB Communications Mechanisms

	Control	Isochronous	Interrupt	Bulk
Pipe Mode	Message	Stream	Stream	Stream
Synchronization	None	Bus, external, or software	None	None
Handshakes	Yes	Not used	Yes	Yes
Data Toggles	Yes	Ignored	Yes	Yes
Required Buffering	Minimum of eight bytes	Twice frame traffic	Single transaction	Single transaction
Error and Status Handling	Guaranteed delivery reports fatal errors only	Reports missing or corrupt data - no retries	Guaranteed delivery reports fatal errors only	Guaranteed delivery reports fatal errors only

Chapter 10
USB Host: Hardware and Software

The USB interconnect supports data traffic between a host and a USB device. This chapter describes the host interfaces necessary to facilitate USB communication between a software client, resident on the host, and a function implemented on a device.

10.1 Overview of the USB Host

The basic flow and interrelationships of the USB communications model are shown in Figure 10-1.

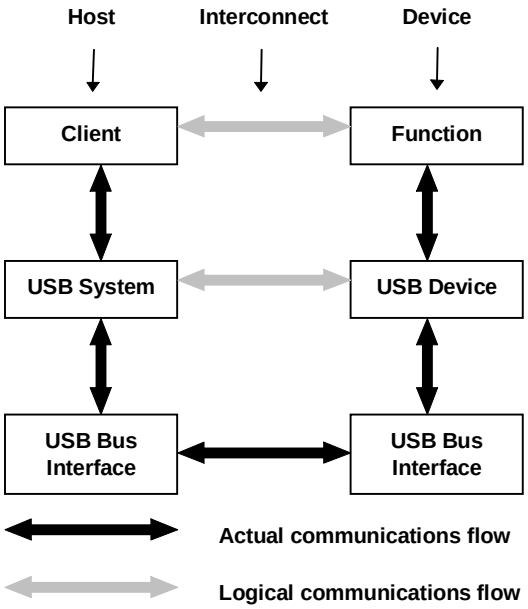


Figure 10-1. Interlayer Communications Model

The host and the device are divided into the distinct layers depicted in Figure 10-1. The actual communication on the host is indicated by vertical arrows. The corresponding interfaces on the device are implementation-specific. All communications between the host and device ultimately occur on the physical USB wire. However, there are logical host-device interfaces between each horizontal layer. These communications, between client software resident on the host and the function provided by the device, are typified by a contract based on the needs of the application currently using the device and the capabilities provided by the device.

This client-function interaction creates the requirements for all of the underlying layers and their interfaces.

This chapter describes this model from the point-of-view of the host and its layers. Figure 10-2 describes, based on the overall view introduced in Chapter 5, the host's view of its communication with the device.

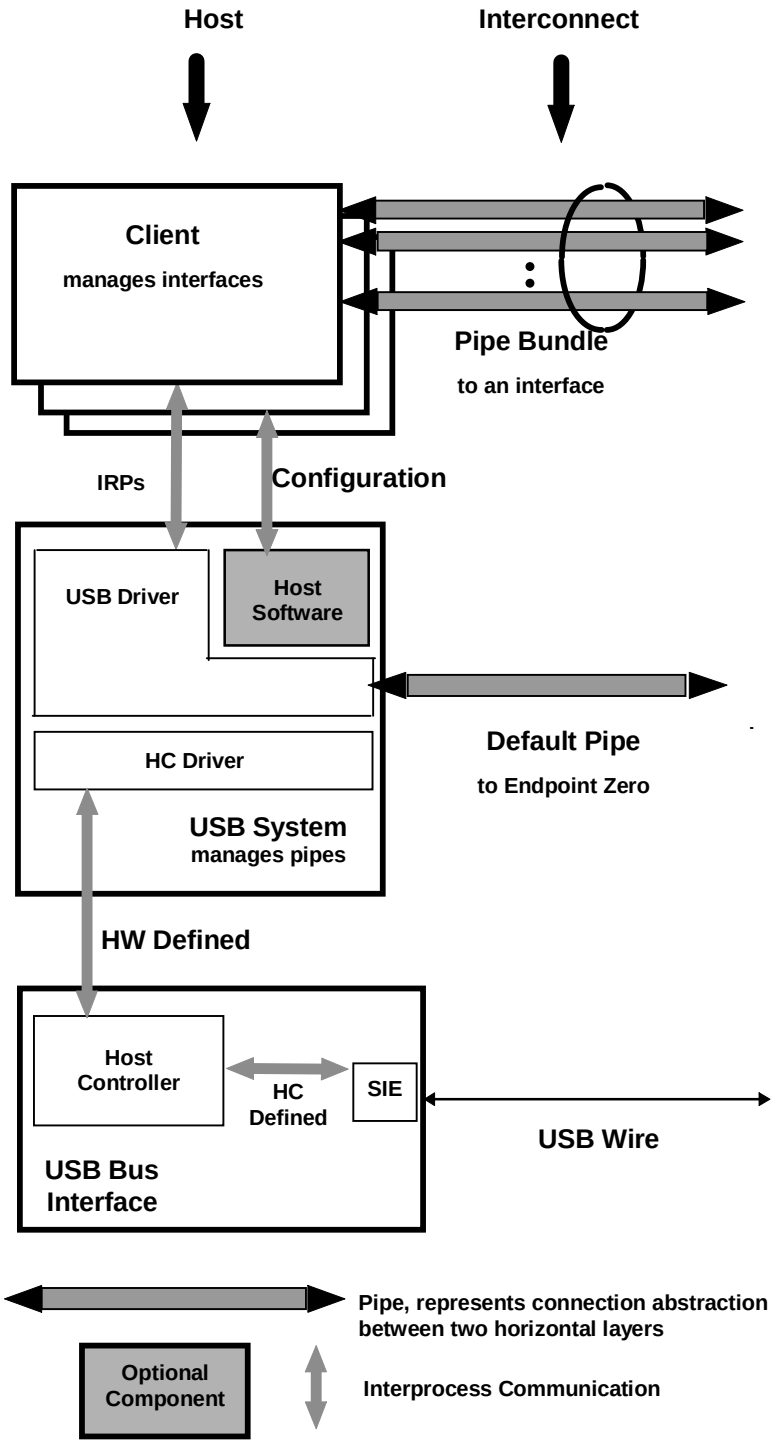


Figure 10-2. Host Communications

There is only one host for each USB. The major layers of a host are:

- USB bus interface
- USB system
- Client

The USB bus interface handles interactions for the electrical and protocol layers (refer to Chapter 7 and Chapter 8). From the interconnect point-of-view, a similar USB bus interface is provided by both the USB device and the host, as exemplified by the Serial Interface Engine (SIE). On the host, however, the USB bus interface has additional responsibilities due to the unique role of the host on the USB and is implemented as the host controller. The host controller also has an integrated, or root, hub providing attachment points to the host.

The USB system uses the host controller to manage data transfers between the host and the USB device. The interface between the USB system and the host controller is dependent on the hardware definition of the host controller. The USB system, in concert with the host controller, performs the translation between the client's view of data transfers and the USB transactions appearing on the interconnect. This includes the addition of any USB feature support such as protocol wrappers. The USB system is also responsible for managing USB resources; e.g., bandwidth, such that client access to USB is possible.

The USB system has three basic components:

- Host Controller Driver
- USB Driver
- Host Software

The Host Controller Driver (HCD) exists to more easily map the various possible host controller implementations into the USB system, such that a client can interact with its device without knowing to which host controller the device is connected. The USB Driver (USBD) provides the basic host interface (USBDI) for clients to USB devices. The interface between the HCD and the USBD is known as the Host Controller Driver Interface (HCDI). This interface is never available directly to clients and thus is not defined by the USB specification. A particular HCDI is, however, defined by each operating system that supports various host controller implementations.

The USBD provides data transfer mechanisms in the form of I/O Request Packets (IRPs), which consist of a request to transport data across a specific pipe. In addition to providing data transfer mechanisms, the USBD is responsible for presenting to its clients an abstraction of a USB device which can be manipulated for configuration and state management. As part of this abstraction, the USBD owns the default pipe (see Chapter 5 and Chapter 9) through which all USB devices are accessed for the purposes of standard USB control. This default pipe represents a logical communication between the USBD and the abstraction of a USB device as shown in Figure 10-2.

In some operating systems, additional non-USB host software is available which provides configuration and loading mechanisms to device drivers. In such operating systems, the device driver shall use the provided interfaces instead of directly accessing the USBDI mechanisms.

The client layer describes all of those software entities which are responsible for directly interacting with their peripherals. When each device is independently attached to the system, these clients might interact directly with the peripheral hardware. The shared characteristics of USB place a USB software stack between the client and its device; that is, a client cannot directly access the device's hardware.

Overall, the host layers provide the following capabilities:

- Detecting the attachment and removal of USB devices
- Managing USB standard control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Controlling the electrical interface between the host controller and USB devices, including the provision of a limited amount of power

The following sections describe these responsibilities and the requirements placed on the USBDI in greater detail. The actual interfaces used for a specific combination of host platform and operating system are described in the appropriate Operating System Environment Guide.

All hubs provide a status change pipe (see Chapter 11) on which status changes for hubs and their ports are reported. This includes a notification of when a USB device is attached to or removed from one of their ports. A USBD client generically known as the hub driver receives these notifications as owner of the hub's status change pipe. For device attachments, the hub driver then initiates the device configuration process. In some systems, this hub driver is a part of the host software provided by the operating system for managing devices.

10.1.2 Control Mechanisms

Control information may be passed between the host and a USB device using in-band or out-of-band signaling. In-band signaling mixes control information with data in a pipe outside the awareness of the host. Out-of-band signaling places control information in a separate pipe.

There is a message pipe called the default pipe for each attached USB device. This logical association between a host and a USB device is used for USB standard control flow such as device enumeration and configuration. The default pipe provides a standard interface to all USB devices. The default pipe may also be used for device-specific communications, as mediated by the USBD which owns the default pipes of all of the USB devices.

A particular USB device may allow the use of additional message pipes to transfer device-specific control information. These pipes use the same communications protocol as the default pipe, but the information transferred is specific to the USB device and is not standardized by the Universal Serial Bus Specification.

The USBD supports the sharing of the default pipe, which it owns and uses, with its clients. It also provides access to any other control pipes associated with the device.

10.1.3 Data Flow

The host controller is responsible for transferring streams of data between the host and USB devices. These data transfers are treated as a continuous stream of bytes. USB supports four basic types of data transfers:

- Control transfers
- Isochronous transfers
- Interrupt transfers
- Bulk transfers

For additional information on transfer types, refer to Chapter 5.

Each device presents one or more interfaces which a client may use to communicate with it. Each interface is composed of zero or more pipes which individually transfer data between the client and a particular endpoint on the device. The USBD establishes interfaces and pipes at the explicit request of host software. The host controller provides service based on parameters provided by the host software when the configuration request is made.

A pipe has several characteristics based on the delivery requirements of the data to be transferred. Examples of these characteristics are: the rate at which data needs to be transferred, whether data is provided at a steady rate or sporadically, how long data may be delayed before delivery, and whether the loss of data being transferred is catastrophic.

A USB device endpoint describes the characteristics required for a specific pipe. Endpoints are described as part of a USB device's characterization information. For additional details, refer to Chapter 9.

10.1.4 Collecting Status and Activity Statistics

As a common communicant for all control and data transfers between the host and USB devices, the USB system and the host controller is well positioned to track status and activity information. Such information is provided upon request to host software allowing that software to manage status and activity information.

10.1.5 Electrical Interface Considerations

The host provides power to USB devices attached to the root hub. The amount of power provided by a port is specified in the discussion of hubs in Chapter 11.

10.2 Host Controller Requirements

In all implementations, host controllers perform the same basic duties with regard to the USB and its attached devices. These basic duties are described below.

The host controller has requirements from both the host and the USB. The following is a brief overview of the functionality provided. Each capability is discussed in detail in subsequent subsections.

State Handling	As a component of the host, the host controller reports and manages its states.
Serializer/Deserializer	For data transmitted from the host, the host controller converts protocol and data information from its native format to a bit stream transmitted on USB. For data being received into the host, the reverse operation is performed.
Frame Generation	The host controller produces SOF tokens at a period of 1 ms.
Data Processing	The host controller processes requests for data transmission to and from the host.
Protocol Engine	The host controller supports the protocol specified by USB.
Transmission Error Handling	All host controllers exhibit the same behavior when detecting and reacting to the defined error categories.

The following subsections present a more detailed discussion of the required capabilities of the host controller.

10.2.1 State Handling

As a normal component in the host, the host controller has a series of states which the USB system manages. Additionally, the host controller has two areas of USB-relevant state:

- Root hub
- State change propagation

The root hub presents to the hub driver the same standard states as other USB devices. The host controller supports these states and their transitions for the hub. For detailed discussions of USB states, including their interrelations and transitions, refer to Chapter 9.

The overall state of the host controller is inextricably linked with that of the root hub and of the overall USB. Any host controller state changes which are visible to attached devices must be reflected in corresponding device state changes such that the resulting host controller and device states are consistent.

USB devices request a wakeup through the use of resume signaling (refer to Chapter 11), which causes hubs to tear down connectivity, and devices to return to their configured state. The host controller itself may cause a resume event through the same signaling method. The host controller must notify the rest of the host of a resume event through a mechanism or mechanisms specific to that system's implementation.

10.2.2 Serializer/Deserializer

The actual transmission of data across the physical USB takes places as a serial bit stream. A Serial Interface Engine (SIE), whether implemented as part of the host or a USB device, handles the serialization and deserialization of USB transmissions. On the host, this SIE is part of the host controller.

10.2.3 Frame Generation

It is the host controller's responsibility to partition USB time into 1 ms quantities called "frames." Frames are created by the host controller through issuing Start of Frame (SOF) tokens at 1.00 ms intervals as shown in Figure 10-3. The SOF token is the first transmission in the frame period. After issuing a SOF token, the host controller is free to transmit other transactions for the remainder of the frame period. When the host controller is in its normal operating state, SOF tokens must be continuously generated at the 1 ms periodic rate, regardless of the other bus activity or lack thereof. If the host controller enters a state where it is not providing power on the bus, it must not generate SOFs. Also, if the host controller is not generating SOFs, it may enter a power-reduced state.

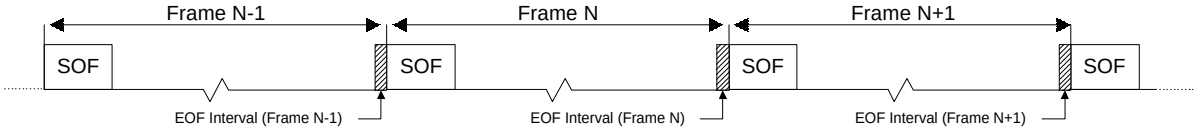


Figure 10-3. Frame Creation

The SOF token holds the highest priority access to the bus. Babble circuitry in hubs electrically isolates any active transmitters during the End of Frame (EOF) interval, providing an idle bus for the SOF transmission.

- The host controller must allow the length of the USB frame to be adjusted by ±1 bit time (refer to Section 10.5.3.2.2). The host controller maintains the current frame number which may be read by the USB system. The current frame number is used to uniquely identify one frame from another.
- Incremented at the end of every frame period.
- Valid through the subsequent frame.

The host transmits the lower 11-bits of the current frame number in each SOF token transmission. When requested from the host controller, the current frame number is the frame number in existence at the time the request was fulfilled. The current frame number as returned by the host (host controller or HCD) is at least 32 bits, although the host controller itself is not required to maintain more than 11 bits.

The host controller shall cease transmission during the EOF interval. When the EOF interval begins, any transactions scheduled specifically for the frame which has just passed are retired. If the host controller is executing a transaction at the time the EOF interval is encountered, the host controller terminates the transaction.

10.2.4 Data Processing

The host controller is responsible for receiving data from the USB system and sending it to the USB and for receiving data from the USB and sending it to the USB system. The particular format used for the data communications between the USB system and the host controller is implementation specific, within the rules for transfer behavior described in Chapter 5.

10.2.5 Protocol Engine

The host controller manages the USB protocol level interface. It inserts the appropriate protocol information for outgoing transmissions. It also strips and interprets, as appropriate, the incoming protocol information.

10.2.6 Transmission Error Handling

The host controller must be capable of detecting the following transmission error conditions, which are defined from the host's point-of-view:

- Time-out conditions after a host-transmitted token or packet. These errors occur when the addressed endpoint is unresponsive or when the structure of the transmission is so badly damaged that the targeted endpoint does not recognize it.
- Data errors resulting in missing or invalid transmissions.
  - The host controller sends or receives a packet shorter than that required for the transmission; for example, a transmission extending beyond EOF or a lack of resources available to the host controller.
  - An invalid CRC field on a received data packet.
- Protocol errors.
  - An invalid handshake PID. For example, a malformed or inappropriate handshake.
  - A false EOP.
  - A bit stuffing error.

For each bulk, command, and interrupt transaction, the host must maintain an error count tally. Errors result from the types of conditions described above, not as a result of an endpoint NAKing a request. This value reflects the number of times the transaction has encountered a transmission error. If the error count tally for a given transaction reaches three, the host retires the transfer. When a transfer is retired due to excessive errors, the last error type will be indicated. Isochronous transactions are attempted only once, regardless of outcome, and, therefore, no error count is maintained for this type.

10.3 Overview of Software Mechanisms

The HCD and the USBD present software interfaces based on different levels of abstraction. They are, however, expected to operate together in a specified manner to satisfy the overall requirements of the USB system (see Figure 10-2). The requirements for the USB software stack are expressed primarily as requirements on the USBDI. The division of duties between the USBD and the HCD is not defined. However, the one requirement of the HCDI which must be met is that it support, in the specified operating system context, multiple definitions of host controllers.

The HCD provides an abstraction of the host controller and an abstraction of the host controller's view of data transfer across USB. The USBD provides an abstraction of the USB device and of the data transfers between the client of the USBD and the function on the USB device. Overall, the USB software stack acts as a facilitator for transmitting data between the client and the function and as a control point for the USB-specific interfaces of the USB device. As part of facilitating data transfer, the USB software provides buffer management capabilities and allows the synchronization of the data transmittal to the needs of the client and the function.

The specific requirements on the USBDI are described later in this chapter. The exact functions which fulfill these requirements are described in the relevant Operating System Environment Guide for the HCDI and the USBDI. The procedures involved in accomplishing data transfers via the USBDI are described below.

10.3.1 Device Configuration

Different operating system environments perform device configuration using different software components and different sequences of events. A specific operating system method is not assumed by the USB system. However, there are some basic requirements that must be fulfilled by any USB system implementation. In some operating systems, these requirements are met by existing host software. In others, the USB system provides the capabilities.

The USB system assumes a specialized client of the USBD, called a hub driver, which acts as a clearing house for addition of devices to and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USBD clients, in an operating system specific manner, to recognize and configure the device. This model, shown in Figure 10-4 is the basis of the following discussion.

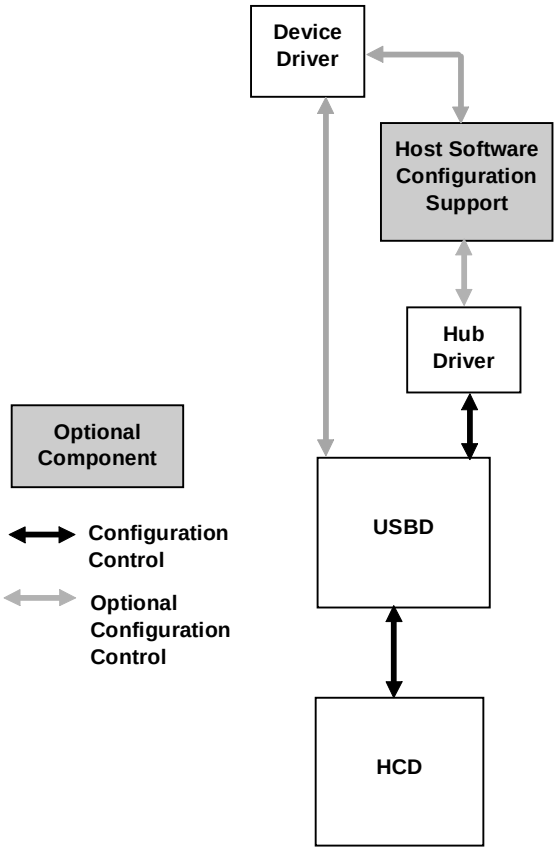


Figure 10-4. Configuration Interactions

When a device is attached, the hub driver receives a notification from the hub detecting the change. The hub driver, using the information provided by the hub, requests a device identifier from the USBD. The USBD in turn sets up the default pipe for that device and returns a device identifier to the hub driver.

The device is now ready to be configured for use. For each device, there are three types of configuration which must be complete before that device is ready for use:

1. Device configuration. This includes setting up all of the device's USB parameters and allocating all USB host resources which are visible to the device. This is accomplished by setting the configuration value on the device. A limited set of configuration changes, such as alternate setting, are allowed without totally reconfiguring the device. Once the device is configured, it is, from its point of view, ready for use.
2. USB configuration. In order to actually create a USBD pipe ready for use by a client, additional USB information, not visible to the device, must be specified by the client. This information, known as the Policy for the pipe, describes how the client will use the pipe. This includes such items as the maximum amount of data the client will transfer with one IRP, the maximum service interval the client will use, the client's notification identification, and so on.
3. Function configuration. Once configuration types 1 and 2 have been accomplished, the pipe is completely ready for use from the USB's point of view. However, additional vendor- or class-specific setup may be required before the pipe can actually be used by the client. This configuration is a private matter between the device and the client and is not standardized by the USBD.

The following paragraphs describe the device and USB configuration requirements.

The actual device configuration is performed by the responsible configuring software. Depending on the particular operating system implementation, the software responsible for configuration can include:

- The hub driver
- Other host software
- A device driver

The configuring software first reads the device descriptor, then requests the description for each possible configuration. It may use the information provided to load a particular client, such as a device driver, which initially interacts with the device. The configuring software, perhaps with input from that device driver, chooses a configuration for the device. Setting the device configuration sets up all of the endpoints on the device and returns a collection of interfaces to be used for data transfer by USBD clients. Each interface is a collection of pipes owned by a single client.

This initial configuration uses the default settings for interfaces and the default bandwidth for each endpoint. A USBD implementation may additionally allow the client to specify alternate interfaces when selecting the initial configuration. The USB system will verify that the resources required for the support of the endpoint are available and, if so, will allocate the bandwidth required. Refer to Section 10.3.2 for a discussion of resource management.

The device is now configured, but the created pipes are not yet ready for use. The USB configuration is accomplished when the client initializes each pipe by setting a policy to specify how it will interact with the pipe. Among the information specified is the client's maximum service interval and notification information. Among the actions taken by the USB system, as a result of setting the policy, is determining the amount of buffer working space required beyond the data buffer space provided by the client. The size of the buffers required is based upon the usage chosen by the client and upon the per transfer needs of the USB system.

The client receives notifications when IRPs complete, successfully or due to errors. The client may also wake up independently of USB notification to check the status of pending IRPs.

The client may also choose to make configuration modifications such as enabling an alternate setting for an interface or changing the bandwidth allocated to a particular pipe. In order to perform these changes, the interface or pipe, respectively, must be idle.

10.3.2 Resource Management

Whenever a pipe is setup by the USBD for a given endpoint, the USB system must determine if it can support the pipe. The USB system makes this determination based on the requirements stated in the endpoint descriptor. One of the endpoint requirements which must be supported in order to create a pipe for an endpoint is the bandwidth necessary for that endpoint's transfers. There are two stages to check for available bandwidth. First the maximum execution time for a transaction is calculated. Then, the frame schedule is consulted to determine if the indicated transaction will fit.

The allocation of the guaranteed bandwidth for isochronous and interrupt pipes, and the determination of whether a particular control or bulk transaction will fit into a given frame, can be determined by a software heuristic in the USB system. If the actual transaction execution time in the host controller exceeds the heuristically determined value, the host controller is responsible for ensuring that frame integrity is maintained (refer to Section 10.2.3). The following discussion describes the requirements on the USB system heuristic.

In order to determine if bandwidth can be allocated, or if a transaction can be fit into a particular frame, the maximum transaction execution time must be calculated. The calculation of the maximum transaction execution time requires that the following information be provided: (note that some of this information may be provided by an agent other than the client)

- Number of data bytes (MaxPacketSize) to be transmitted.
- Transfer type.
- Depth in the topology. If less precision is allowed, the maximum topology depth may be assumed.

This calculation must include the bit transmission time, the signal propagation delay through the topology, and any implementation specific delays such as preparation or recovery time required by the host controller itself. Refer to Chapter 5 for examples of formulas that can be used for such calculations.

10.3.3 Data Transfers

The basis for all client-function communication is the interface: a bundle of related pipes associated with a particular USB device.

A given interface is managed by exactly one client on the host. The client initializes each pipe of an interface by setting the policy for that pipe. This includes the maximum amount of data to be transmitted per IRP and the maximum service interval for the pipe. A service interval is stated in milliseconds and describes the interval over which an IRP's data will be transmitted for an isochronous pipe. It describes the polling interval for an interrupt pipe. The client is notified when a specified request is completed. The client manages the size of each IRP such that its duty cycle and latency constraints are maintained. Additional policy information includes the notification information for the client.

The client provides the buffer space required to hold the transmitted data. The USB system uses the policy to determine the additional working space it will require .

The client views its data as a contiguous serial stream, which it manages in a similar manner to those streams provided over other types of bus technologies. Internally, the USB system may, depending on its own policy and any host controller constraints, break the client request down into smaller requests to be sent across the USB. However, two requirements must be met whenever the USB system chooses to undertake such division:

- The division of the data stream into smaller chunks is not visible to the client.
- USB samples are not split across bus transactions. Refer to Chapter 9 for a definition of USB sample and its relationship to the pipe's natural sample size.

When a client wishes to transfer data, it will send an IRP to the USBD. Depending on the direction of data transfer, a full or empty data buffer will be provided. When the request is complete (successfully or due to an error condition), the IRP and its status is returned to the client. Where relevant, this status is also provided on a per transaction basis.

10.3.4 Common Data Definitions

In order to allow the client to receive request results as directly as possible from its device, it is desirable to minimize the amount of processing and copying required between the device and the client. To facilitate this, some control aspects of the IRP are standardized such that the information provided by the client may be directly used by different layers in the stack. The particular format for this data is dependent on the actualization of the USBDI in the operating system. Some data elements may in fact not be directly visible to the client at all, but are generated as a result of the client request.

The following data elements define the relevant information for a request:

- Identification of the pipe associated with the request. Identifying this pipe also describes information such as transfer type for this request.
- Notification identification for the particular client.
- Location and length of data buffer which is to be transmitted or received.
- Completion status for the request. Both the summary status, and, as required, detailed per-transaction status must be provided.
- Location and length of working space. This is implementation dependent.

The actual mechanisms used to communicate requests to the USBD are operating system specific. However, beyond the requirements stated above for what request-related information must be available, there are also requirements on how requests will be processed. The basic requirements are described in Chapter 5. Additionally, the USBD provides a mechanism to designate a group of isochronous IRPs for which the transmission of the first transaction of each IRP will occur in the same frame. The USBD also provides a mechanism for designating an uninterruptable set of vendor- or class-specific requests to a default pipe. No other requests to that default pipe, including standard, class, or vendor request may be inserted in the execution flow for such an uninterruptable set. If any request in this set fails, the entire set is retired.

10.4 Host Controller Driver

The Host Controller Driver (HCD) is an abstraction of host controller hardware and the host controller's view of data transmission over the USB. The HCDI meets the following requirements:

- Provides an abstraction of the host controller hardware.
- Provides an abstraction for data transfers by the host controller across the USB interconnect.
- Provides an abstraction for the allocation (and de-allocation) of host controller resources, to support guaranteed service to USB devices.
- Presents the root hub and its behavior according to the hub class definition. This includes supporting the root hub such that the hub driver interacts with the root hub exactly as it would for any hub. In particular, even though a root hub can be implemented in a combination of hardware and software, the root hub responds initially to the default device address (from a client perspective), returns descriptor information, supports having its device address set, and supports the other hub class requests. However, bus transactions may or may not need to be generated to accomplish this behavior given the close integration possible between the host controller and the root hub.

The HCD provides a software interface (HCDI) which implements the required abstractions. The function of the HCD is to provide an abstraction which hides the details of the host controller hardware. Below the host controller hardware is the physical USB and all the attached USB devices.

The HCD is the lowest tier in the USB software stack. The HCD has only one client: the Universal Serial Bus Driver (USBD). The USBD maps requests from many clients to the appropriate HCD. A given HCD may manage many host controllers.

The HCDI is not directly accessible from a client. Therefore, the specific interface requirements for the HCDI are not discussed here.

10.5 Universal Serial Bus Driver

A Universal Serial Bus Driver (USBD) provides a collection of mechanisms that operating system components, typically device drivers, use to access USB devices. The only access to USB devices is that provided by the USBD. The USBD implementations are operating system specific. The mechanisms provided by the USBD are implemented using as appropriate and augmenting as necessary the mechanisms provided by the operating system environment in which it runs. The following discussion centers on the basic capabilities required for all USBD implementations. For specifics of the USBD operation within a specific environment, see the relevant Operating System Environment Guide for the USBD. A single instance of the USBD directs accesses to one or more HCDs that in turn connect to one or more USB host controllers. If allowed, how USBD instancing is managed is dependent upon the operating system environment. However, from the client's point-of-view, the USBD with which it communicates manages all of the attached USB devices.

10.5.1 Overview

Clients of USBD direct commands to devices or move streams of data to or from pipes. The USBD presents two groups of software mechanisms to clients: command mechanisms and pipe mechanisms.

Command mechanisms allow clients to configure and control USBD operation as well as to configure and generically control a USB device. In particular, command mechanisms provide all access to the device's default pipe.

Pipe mechanisms allow a USBD client to manage device specific data and control transfers. Pipe mechanisms do not allow a client to directly address the device's default pipe.

Figure 10-5 presents an overview of the USBD structure.

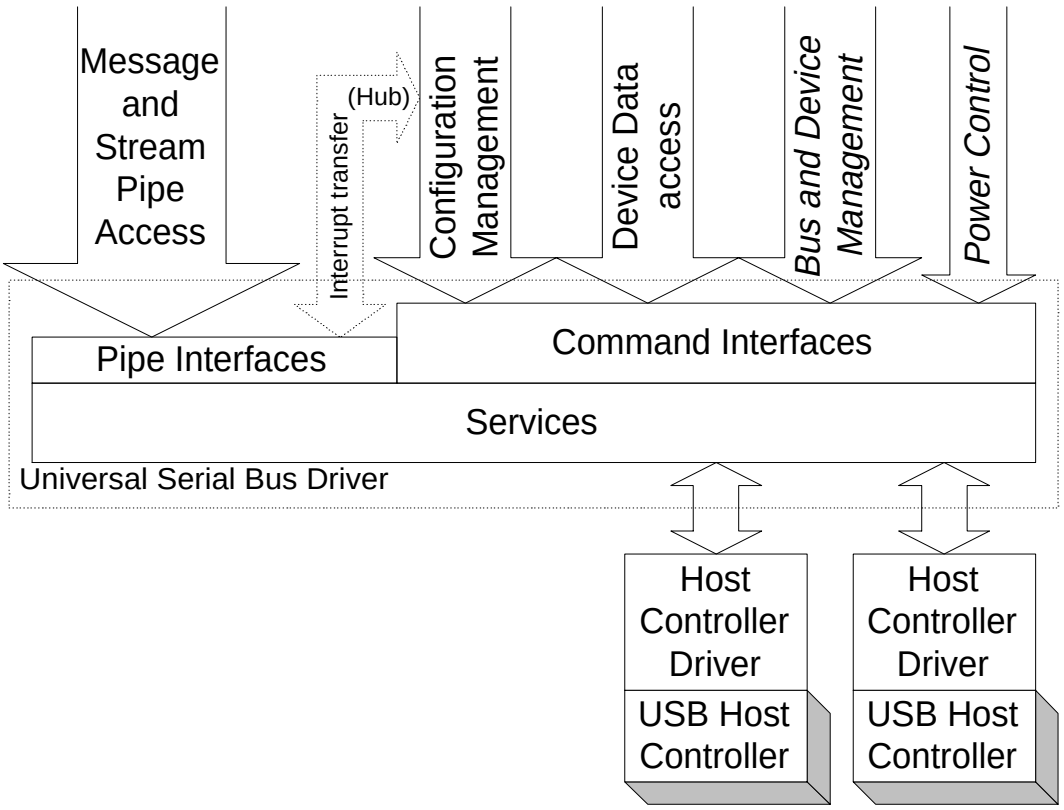


Figure 10-5. Universal Serial Bus Driver Structure

10.5.1.1 Initialization

Specific USBD initialization is operating system dependent. When a particular USB managed by USBD is initialized, the management information for that USB is also created. Part of this management information is the default address device and its default pipe.

Whenever a device is attached to a USB, it responds to a special address known as the default address (refer to Chapter 9) until its unique address is assigned by the hub driver. In order for the USB system to interact with the new device, the default address device for that particular USB and that default address device's default pipe must always be available to the hub driver whenever a device is attached. The default address device and its default pipe are created by the USB system when a new USB is initialized.

10.5.1.2 USBD Pipe Usage

Pipes are the method by which a device endpoint is associated with a host software entity. Pipes are owned by exactly one such entity on the host. Although the basic concept of a pipe is the same no matter who the owner, some distinction of capabilities provided to the USBD client occurs between two groups of pipes:

- Default pipes, which are owned and managed by the USBD
- All other pipes, which are owned and managed by clients of the USBD

Default pipes are never directly accessed by clients, although they are often used to fulfill some part of client requests relayed via command mechanisms.

10.5.1.2.1 Default Pipes

The USBD is responsible for allocating and managing appropriate buffering to support transfers on the default pipe which are not directly visible to the client such as setting a device address. For those transfers which are directly visible to the client, such as sending vendor and class commands or reading a device descriptor, the client must provide the required buffering.

10.5.1.2.2 Client Pipes

Any pipe not owned and managed by the USBD can be owned and managed by a USBD client. From the USBD viewpoint, the pipe is owned by a single client. In fact, a cooperative group of clients can manage the pipe provided they behave as a single coordinated entity when using the pipe.

The client is responsible for providing the amount of buffering it needs to service the data transfer rate of the pipe within a service interval attainable by the client. Additional buffering requirements for working space are specified by the USB system.

10.5.1.3 Service Capabilities

The USBD provides services in the following categories:

- Configuration via command mechanisms
- Transfer services via both command and pipe mechanisms
- Event notification
- Status reporting and error recovery

10.5.2 USBD Command Mechanism Requirements

USBD command mechanisms allow a client generic access to a USB device. Generally these commands allow the client to make read or write accesses to one of potentially several device data and control spaces. The client provides as little as a device identifier and the relevant data or empty buffer pointer.

USBD command transfers do not require that the USB device be configured. Many of the device configuration facilities provided by the USBD are command transfers.

Following are the specific requirements on the command mechanisms provided.

10.5.2.1 Interface State Control

USBD clients must be able to set a specified interface to any settable pipe state. Setting an interface state results in all of the pipes in that interface moving to that state. Additionally, all of the pipes in an interface may be reset or aborted.

10.5.2.2 Pipe State Control

USBD pipe state has two components:

- Host status
- Reflected endpoint status

Whenever the pipe status is reported, the value for both components will be identified. The pipe status reflected from the endpoint is the result of the endpoint being in a particular state. The USBD client manages the pipe state as reported by the USBD. For any pipe state reflected from the endpoint, the client must also interact with the endpoint to change the state.

A USBD pipe is in exactly one of the following states:

- Active. The pipe's policy has been set and the pipe is able to transmit data. The client can query as to whether any IRPs are outstanding for a particular pipe. Pipes for which there are no outstanding IRPs are still considered to be in the active state as long as they are able to accept new IRPs.
- Stalled. An error has occurred on the pipe. This state may also be a reflection of the corresponding *stalled* endpoint on the device.
- Idle. The pipe will not accept further IRPs. The endpoint must also be in the *idle* state.

USBD clients must be able to set a specified pipe to the idle or active state from any of the states described above. Clients also must be able to set an endpoint to either the active or idle state. Additionally, clients manipulate pipe state in the following ways:

- Aborting a pipe. All of the IRPs scheduled for a pipe are retired immediately and returned to the client with a status indicating they have been aborted. Neither the host state nor the reflected endpoint state of the pipe is affected.
- Resetting a pipe. The pipe's IRPs are aborted. The host state is moved to active. If the reflected endpoint state needs to be changed, that must be commanded explicitly by the USBD client.

10.5.2.3 Getting Descriptors

The USBDI must provide a mechanism to retrieve standard device, configuration and string descriptors, as well as any class- or vendor-specific descriptors.

10.5.2.4 Getting Current Configuration Settings

The USBDI must provide a facility to return, for any specified device, the current configuration descriptor. If the device is not configured, no configuration descriptor is returned. This action is equivalent to returning the configuration descriptor for the current configuration by requesting the specific configuration descriptor. It does not, however, require the client to know the identifier for the current configuration. This will return all of the configuration information, including:

- All of the configuration descriptor information as stored on the device including all of the alternate settings for all of the interfaces
- Indicators for which of the alternate settings for interfaces are active
- Pipe handles for endpoints in the active alternate settings for interfaces
- Actual MaxPacketSizes for endpoints in the active alternate settings for interfaces

Additionally, for any specified pipe, the USBDI must provide a facility to return the *MaxPacketSize* which is currently being used by the pipe.

10.5.2.5 Adding Devices

The USBDI must provide a mechanism for the hub driver to inform USBD of the addition of a new device to a specified USB and to retrieve the USBD ID of the new USB device. The USBD tasks include assigning the device address and preparing the device's default pipe for use.

10.5.2.6 Removing Devices

The USBDI must provide a facility for the hub driver to inform the USBD that a specific device has been removed.

10.5.2.7 Managing Status

The USBDI must provide a mechanism for obtaining and clearing device-based status, on a device, interface, or pipe basis.

10.5.2.8 Sending Class Commands

This USBDI mechanism is used by a client, typically a class specific or adaptive driver, to send one or more class specific commands to a device.

10.5.2.9 Sending Vendor Commands

This USBDI mechanism is used by a client to send one or more vendor specific commands to a device.

10.5.2.10 Establishing Alternate Settings

The USBDI must provide a mechanism to change the alternate setting for a specified interface. As a result, the pipe handles for the previous setting are released and new pipe handles for the interface are returned. For this request to succeed, the interface must be idle; i.e., all pipes in the interface must be in the idle state.

10.5.2.11 Establishing a Configuration

Configuring software requests a configuration by passing a buffer containing a configuration descriptor to the USBD. The USBD requests resources for the endpoints in the configuration, and if all resource requests succeed, the USBD sets the device configuration and returns interface handles with corresponding pipe handles for all of the active endpoints. The default values are used for all alternate settings for interfaces and for the *MaxPacketSize* for endpoints. These default values may be subsequently modified.

Note: the specific interface implementing configuration may require specific alternate settings to be identified.

10.5.2.12 Setting Descriptors

For devices supporting this behavior, the USBDI allows existing descriptors to be updated or new descriptors to be added.

10.5.2.13 Establishing the Maximum Packet Size for a Pipe

The USBDI must provide a mechanism to modify a pipe's transfer characteristics. The USBD adjusts requested resources and sets the current maximum data payload size per bus transaction for the specified pipe. This service may only apply to a pipe which has been created by establishing a configuration and which is currently idle.

10.5.3 USBD Pipe Mechanisms

This part of the USBDI offers clients the highest-speed, lowest overhead data transfer services possible. Higher performance is achieved by shifting some pipe management responsibilities from the USBD to the client. As a result, the pipe mechanisms are implemented at a more primitive level than the data transfer services provided by the USBD command mechanisms. Pipe mechanisms do not allow access to a device's default pipe.

USBD pipe transfers are available only after both the device and USB configuration have completed successfully. At the time the device is configured, the USBD requests the resources required to support all device pipes in the configuration. Clients are allowed to modify the configuration, constrained by whether the specified interface or pipe is idle.

Clients provide full buffers to outgoing pipes and retrieve transfer status information following the completion of a request. The transfer status returned for an outgoing pipe allows the client to determine the success or failure of the transfer.

Clients provide empty buffers to incoming pipes and retrieve the filled buffers and transfer status information from incoming pipes following the completion of a request. The transfer status returned for an incoming pipe allows a client to determine the amount and the quality of the data received.

10.5.3.1 Supported Pipe Types

The four types of pipes supported, based on the four transfer types, are described below.

10.5.3.1.1 Isochronous Data Transfers

Each buffer queued for an isochronous pipe is required to be viewable as a stream of samples. As with all pipe transfers, the client establishes a policy for using this isochronous pipe, including the relevant service interval for this client. Lost or missing bytes, which are detected on input, and transmission problems, which are noted on output, are indicated to the client.

The client queues a first buffer, starting the pipe streaming service. To maintain the continuous streaming transfer model used in all isochronous transfers, the client queues an additional buffer before the current buffer is retired.

The USBD is required to be able to provide a sample stream view of the client's data stream. In other words, using the client's specified method of synchronization, the precise packetization of the data is hidden from the client. Additionally, a given transaction is always contained completely within some client data buffer.

For an output pipe, the client provides a buffer of data. The USBD allocates the data across the frames for the service period using the client's chosen method of synchronization.

For an input pipe, the client must provide an empty buffer large enough to hold the maximum number of bytes the client's device will deliver in the service period. The USB system strips USB defined packaging information from the stream such that bytes are contiguous in the client's buffer. Where missing or invalid bytes are indicated, the USBD leaves the space which the bytes would have occupied in place in the buffer and identifies the error. One of the consequences of using no synchronization method, is that this reserved space is assumed to be the maximum packet size. The buffer-retired notification occurs when the IRP completes. Note that the input buffer need not be full when returned to the client.

The USBD may optionally provide additional views of isochronous data streams. The USBD is also required to be able to provide a packet stream view of the client's data stream.

10.5.3.1.2 Interrupt Transfers

Interrupt transfers originate in a USB device and are delivered to a client of the USB Driver.

The client queues a buffer large enough to hold the interrupt transfer data (typically a single USB transaction). When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.3 Bulk Transfers

Bulk transfers may originate either from the device or the client. No periodicity or guaranteed latency is assumed. When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.4 Control Transfers

All message pipes transfer data in both directions. In all cases, the client outputs a setup stage to the device endpoint. The optional data stage may be either an input or an output and the final status is always logically presented to the host. For details of the defined message protocol, refer to Chapter 8.

The client prepares a buffer specifying the command phase and any optional data or empty buffer space. The client receives a buffer-retired notification when all phases of the control transfer are complete, or an error notification, if the transfer is aborted due to transmission error.

10.5.3.2 USBD Pipe Mechanism Requirements

The following pipe mechanisms are provided.

10.5.3.2.1 Aborting IRPs

The USBDI must allow IRPs for a particular pipes to be aborted.

10.5.3.2.2 Adjusting the Start-Of-Frame

The USBDI must allow a master client to change the number of bit times in a USB frame. A client wishing to adjust the SOF must already have received master client status from the USBD (refer to Section 10.5.3.2.5). Invoking this change more frequently than once every 6 ms has undefined results.

10.5.3.2.3 Managing Pipe Policy

The USBDI must allow a client to set and clear the policy for an individual pipe or for an entire interface. Any IRPs made by the client prior to successfully setting a policy are rejected by the USBD.

10.5.3.2.4 Queuing IRPs

The USBDI must allow clients to queue IRPs for a given pipe. When IRPs are returned to the client, the request status is also returned. A mechanism is provided by the USBD to identify a group of isochronous IRPs whose first transactions will all occur in the same frame.

10.5.3.2.5 Being a Master Client

The USBDI must allow a client to request becoming a master client for a given USB and to release this capability when it is no longer required. Only master clients may adjust the SOF for a given USB.

A client requesting master status identifies itself with an interface handle for the device from which it is mastering.

10.5.4 Managing the USB via the USBD Mechanisms

Using the provided USBD mechanisms, the following general capabilities are supported by any USB system.

10.5.4.1 Configuration Services

Configuration Services operate on a per device basis. The USBD performs device configuration at the direction of the configuring software. A hub driver has a special role in device management and provides at least the following capabilities:

- Device attach/detach recognition, driven by an interrupt pipe owned by the hub driver
- Device reset, accomplished by the hub driver by resetting the hub port upstream of the device
- Directs the USBD to perform device address assignment

The USBDI additionally provides the following configuration facilities, which may be used by the hub driver or other configuring software available on the host:

- Device identification and access to configuration information (via access to descriptors on the device)
- Device configuration via command mechanisms

When the hub driver informs the USBD of a device attachment, the USBD establishes the default pipe for the new device.

10.5.4.1.1 Configuration Management

Configuration Management services are provided primarily as a set of specific interface commands which generate USB transactions on the default pipe. The notable exception is the use of an additional interrupt pipe that delivers hub status directly to the hub driver.

Every hub initiates an interrupt transfer when there is a change in the state of one of the hub ports. Generally, the port state change will be the connection or removal of a downstream USB device. (Refer to Chapter 11 for more information.)

10.5.4.1.2 Initial Device Configuration

The device configuration process begins when a hub reports via its status change pipe the connection of a new USB device.

Configuration Management services allow configuring software to select a USB device configuration from the set of configurations listed in the device. The USBD verifies that the data transfer rates given for all endpoints in the configuration do not exceed the capabilities of the USB with the current schedule before setting the device configuration.

10.5.4.1.3 Modifying a Device Configuration

Configuration Management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if the data transfer rates given for all end points in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.

Configuration Management services allow configuring software to return a USB device to a not configured state.

10.5.4.1.4 Device Removal

Error recovery and/or device removal processing begins when a hub reports via its status change pipe that communication with a USB device has ceased.

10.5.4.2 Bus and Device Management

Bus and Device Management services allow a client to become the master client on a USB, and as the master client, to adjust the number of bit times in a frame on that bus. A master client can explicitly release master status, or the client's master status will be automatically released when the device containing the referenced interface is reset or detached. There can be at most one master client on a USB. The new master client will be awarded a special master handle to be used when adjusting the SOF.

A master client may add or subtract one bit time to the current USB frame. The client must reference an interface identifying the USB being adjusted and allowing the client's master status to be validated. Adjusting SOF more frequently than once every 6 ms has undefined results.

10.5.4.3 Power Control

The USB system will provide power management in a system-specific manner. No USB requirements beyond those described in Chapter 9 are placed on the USB system.

No standard commands are provided for use with all USB devices. Individual devices may choose to offer a power control interface via vendor specific control(s). Device classes may define class-specific power control capabilities.

All USB devices must however support the Powered Down state (refer to Chapter 9). Basic control of the power provided to a device is exercised via control of the hub port to which the device is attached.

10.5.4.4 Event Notifications

USBD clients receive several kinds of event notifications through a number of sources:

- Completion of an action initiated by a client.
- Interrupt transfers over stream pipes can deliver notice of device events directly to USBD clients. For example, hubs use an interrupt pipe to deliver events corresponding to changes in hub status.
- Event data can be embedded by devices in streams.
- Standard device interface commands, device class commands, vendor specific commands, and even general control transfers over message pipes can all be used to poll devices for event conditions.

10.5.4.5 Status Reporting and Error Recovery Services

The command and pipe mechanisms both provide status reporting on individual requests as they are invoked and completed.

Additionally, USB device status is available to USBD clients using the command mechanisms.

The USBD provides clients with pipe error recovery mechanisms by allowing pipes to be reset or aborted.

10.6 Operating System Environment Guides

As noted previously, the actual interfaces between USB software and host software are specific to the host platform and operating system. A companion specification is required for each combination of platform and operating system with USB support. These specifications describe the specific interfaces used to integrate USB into the host. Each operating system provider for USB software identifies a compatible Universal USB Specification revision.

Chapter 11
Hub Specification

This chapter describes the architectural requirements for the USB hub. It contains a description of the two principal sub-blocks: the hub repeater and the hub controller. The chapter also describes the hubs operation for error recovery, reset, and suspend/resume. The second half of the chapter defines hub request behavior and hub descriptors.

The hub specification supplies sufficient information to permit an implementer to design a USB hub which conforms to the USB specification.

11.1 Overview

Hubs provide the electrical interface between USB devices and the host and are directly responsible for supporting many of the attributes that make USB user friendly and hide its complexity from the user. Listed below are the major aspects of USB functionality that hubs must support:

- Connectivity behavior
- Power management
- Device connect/disconnect detection
- Bus fault detection and recovery
- Full/Low speed device support

A hub consists of two components, the hub repeater and the hub controller. The repeater is responsible for connectivity setup and tear-down. It also supports exception handling such as bus fault detection and recovery and connect/disconnect detect. The hub controller provides the mechanism for host to hub communication. Hub specific status and control commands permit the host to configure a hub and to monitor and control its individual downstream ports.

11.2 Device Characteristics

11.2.1 Hub Architecture

Figure 11-1 shows a hub and the locations of its root and downstream ports. A hub consists of a repeater section and a hub controller section. The repeater is responsible for managing connectivity on a per packet basis, while the hub controller provides status and control and permits host access to the hub.

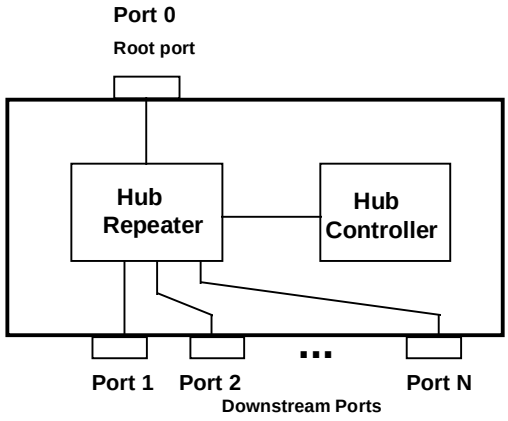


Figure 11-1. Hub Architecture

11.2.2 Hub Connectivity

Hubs display differing connectivity behavior depending on whether they are propagating packet traffic or resume signaling, or are in the idle state.

11.2.2.1 Packet Signaling Connectivity

The hub repeater contains one port that must always connect in the upstream direction (referred to as the root port) and one or more downstream ports. Upstream connectivity is defined as being towards the host, and downstream connectivity is defined as being towards a device. Figure 11-2 shows the packet signaling connectivity behavior for hubs in the upstream and downstream directions. A hub also has an idle state during which the hub makes no connectivity. When in the idle state all of the hub's ports are in the receive mode waiting for the start of the next packet.

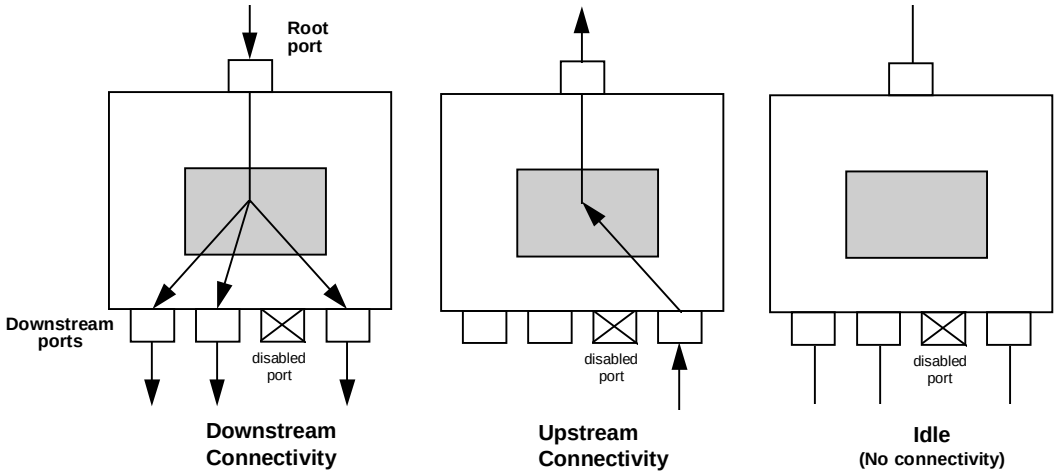


Figure 11-2. Hub Connectivity

If a downstream hub port is enabled (i.e., in a state where it can propagate signaling through the hub) and the hub detects an SOP on that port, connectivity is established in an upstream direction to the root port of that hub, but not to any other downstream ports. This means that when a device or a hub transmits a packet upstream, only those hubs in line between the transmitting device and the host will see the packet. When SOP on an upstream port is detected, all other downstream ports are locked. This guarantees that

hub connectivity will not be modified until the next EOP is detected or until the hub times out at the end of the frame.

In the downstream direction, hubs operate in a broadcast mode. When a hub detects an SOP on its root port, it establishes connectivity to all enabled downstream ports. If a port is not enabled, it does not receive any packet traffic activity from the root port and does not propagate packet signaling downstream.

11.2.2.2 Resume Connectivity

Hubs exhibit differing connectivity behaviors for upstream and downstream directed resume signaling. A hub which is in the suspend state reflects resume signaling from its root port to all of its enabled downstream ports. Figure 11-3 illustrates hub upstream and downstream resume connectivity.

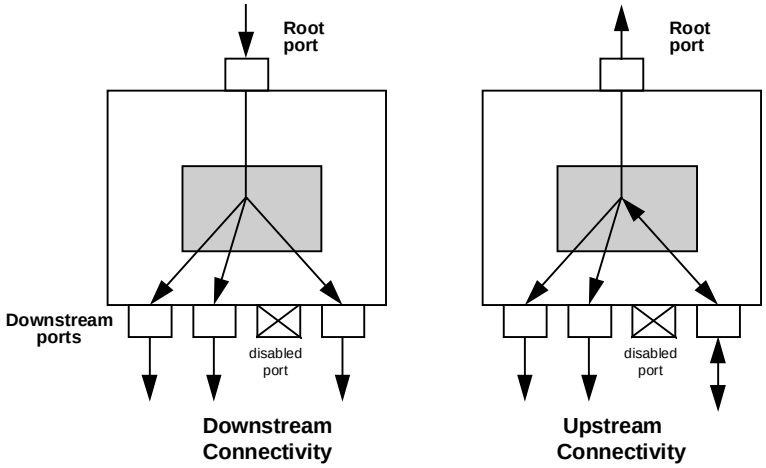


Figure 11-3. Resume Connectivity

If a hub is in the suspend state and detects resume signaling from a downstream port, the hub reflects that signaling upstream to its root and also to all of its enabled downstream ports, including the port which initiated the resume sequence. Resume signaling is not reflected to disabled ports. A detailed discussion of resume connectivity appears in Section 11.5

11.2.3 Hub Port States

A hub downstream port may be in one of four or five possible states and must comprehend such features as connect/disconnect detect, port enable/disable, suspend/resume, reset, and, optionally, power switching. Hubs must support independent port state machines on a per downstream port basis. Figure 11-4 illustrates the states for a non-power switched port, and Figure 11-5 shows the states for a port with power switching. Hub port states apply only to downstream ports.

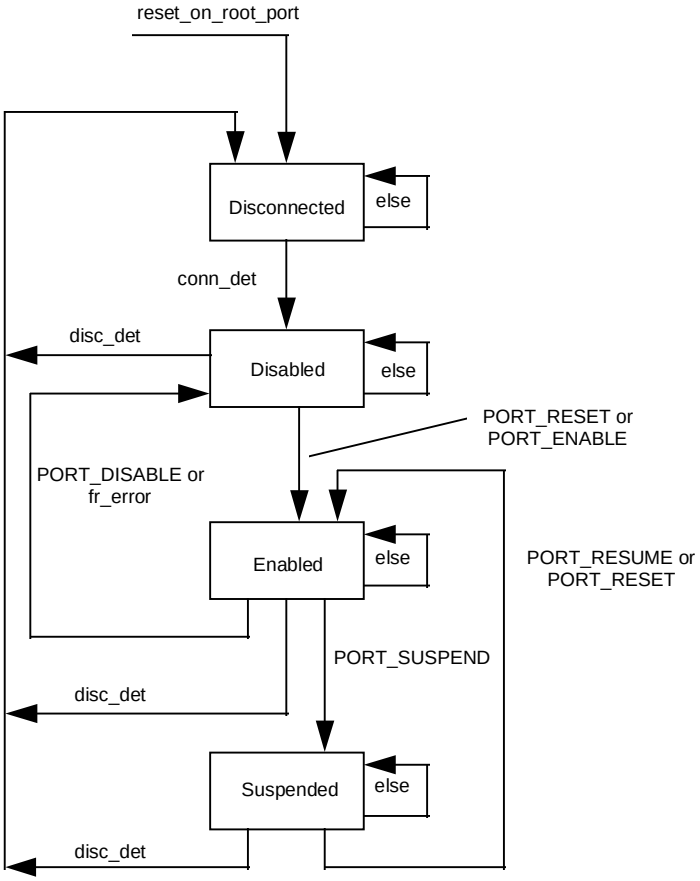


Figure 11-4. Non-power Switched Hub Port States

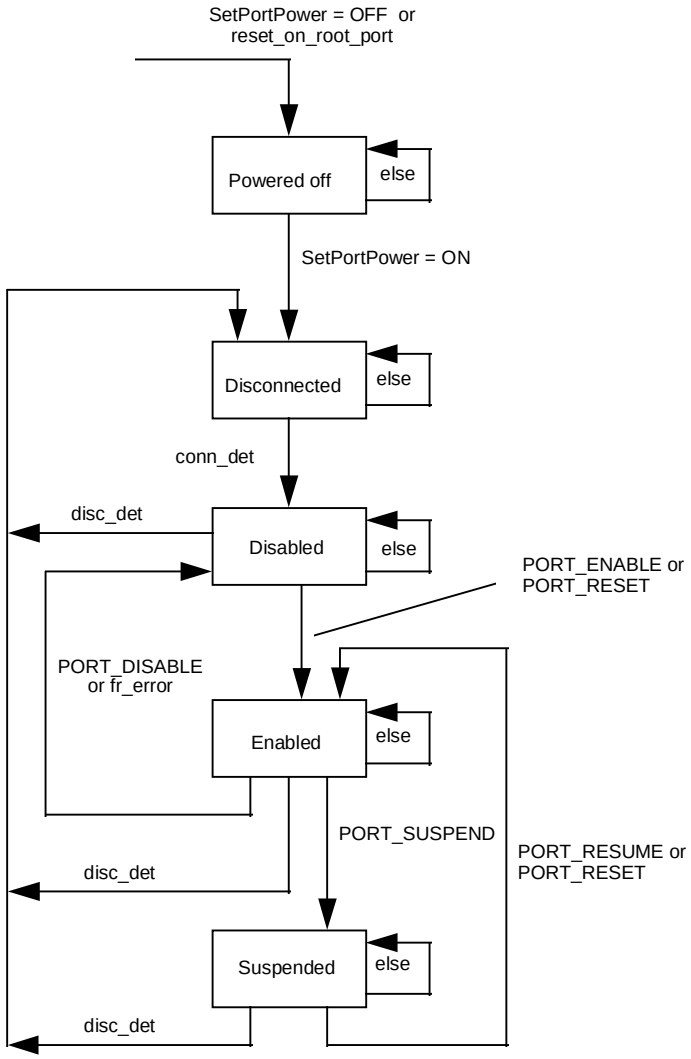


Figure 11-5. Power Switched Hub Port States

Each of the states is described below.

Powered off: This state is only supported for ports that have power switching as shown in Figure 11-5. A port transitions to its powered off state when the hub receives a ClearPortFeature(PORT_POWER) request, or when the hub detects root port reset signaling. A hub's downstream ports also transition to the powered off state when power is initially applied to a hub. A powered off port supplies no power downstream, and its signal outputs buffers are placed in the Hi-Z state. A powered off port ignores all upstream directed bus activity on that port.

Disconnected: A downstream port of a hub supporting power switching transitions from the powered off to the disconnected state when power is applied to the port via a SetPortFeature(PORT_POWER) request. If a hub does not support power switching, it transitions to the disconnected state upon power up or upon receipt of a root port reset. In the disconnected state, a port cannot propagate any signaling in either the upstream or downstream direction. However, the port can detect a connect event (conn_det), which sets a status field in the hub controller and causes the port state to transition to the disabled state.

Conn_det is asserted if a downstream port is in the disconnect state and detects 2.5 µs of continuous non SE0 signaling on the bus. When conn_det is first asserted, the idle bus state can be driven from either a low or full speed device, and may be either a DIFF0 or a DIFF1. The hub automatically determines the

device type (low or full speed) by determining whether D+ or D- is pulled high. Device speed determination is completed before the hub transitions to the disabled state.

Disabled: A port transitions to the disabled state from the disconnected state when it detects a connect event (conn_det). This requires that the port first be power switched on if the hub supports power switching. A port in the disabled state can only propagate downstream directed signaling arising from a SetPortFeature(RESET) request; at all other times, the port's output buffers are in the Hi-Z state. In the upstream direction, a port in the disabled state does not propagate any signaling through the hub to the root port when the hub is awake. However, certain events, such as disconnect will cause upstream resume signaling to be propagated to the root port if the hub is in the suspend state. A disconnect event (disc_det) will cause the port to return to the disconnect state and will set a status field in the hub controller. Disc_det is asserted whenever the port detects 2.5 μs of continuous SE0 when the port is not propagating downstream traffic. Before a disconnect event can be timed the hub suspended hub must first wake up.

Enabled: A port transitions to the enabled state upon receipt of a SetPortFeature(PORT_ENABLE) or a SetPortFeature(PORT_RESET) request. In the enabled state the port propagates all downstream signaling, full speed packet traffic and reset signaling; low speed packet traffic is propagated downstream when preceded by the preamble PID. In the enabled state, the port propagates all upstream signaling including full speed and low speed packets and resume signaling. A port transitions to the disabled state if it receives a ClearPortFeature(PORT_ENABLE) request or if a frame error (fr_error) occurs. The ClearPortFeature(PORT_ENABLE) request may be issued at any time by the host, and the hub must respond by immediately placing the port in the disabled state. An enabled port will transition to the disconnected state if a disconnect detect occurs.

Suspended: The hub selectively suspends all devices downstream of a port when it receives a SetPortFeature(PORT_SUSPEND) request. This request must not cause the now suspended port to stop propagating in the middle of a transaction; i.e., the current transaction must complete before the port enters the suspend state. A port displays different suspend connectivity behavior depending on whether the hub is awake or is itself suspended. If the hub is awake, no upstream or downstream connectivity can propagate through the port. However, if the hub is suspended, an idle to resume or an idle to SE0 transition on the port is reflected onto all other non-disabled ports as an idle to resume transition. This behavior makes it possible to suspend multiple hubs in series and still have a device at the bottom be able to wake up the entire bus.

If a hub is suspended and bus activity occurs on a suspended port, the hub first wakes up. The termination of a resume request to the port causes a status field to be set in the hub. In response, the host polls the hub to read the status field change and determine which on which port the resume occurred. The hub port transitions to the enabled state when the resume is complete. Details of the hub port signaling for selective resume are described in Section 11.5.2**.**

A disconnect event to a suspended hub causes the hub state to transition to the disconnected state and sets the hub controller status field to indicate that a disconnect has occurred. It is not possible to place a disconnected port directly into the suspend mode, since the port never exits the disconnected state.

Table 11-1 summarizes hub port behavior in different port states for different types of signaling. Hub behavior during resume signaling when the hub itself is in the suspend state constitutes a special case, and is discussed in Section 11.5.2.1.

Table 11-1. Port Behavior vs. Signaling

Signaling\State	Powered-off	Disconnected	Disabled	Enabled	Suspended
Reset on root port (hub with power switching)	Stay in **Powered-off**	Go to **Powered off**	Go to **Powered off**	Go to **Powered off**	Go to **Powered off**
Reset on root port (hub without power switching)	N.A.	Go to **Disconnected**	Go to **Disconnected**	Go to **Disconnected**	Go to **Disconnected**
ClearPortFeature(PORT_POWER**)**(hub with power switching)	Stay in **powered-off**	Go to **Powered off**	Go to **Powered off**	Go to **Powered off**	Go to **Powered off**
SetPortFeature(PORT_POWER**)**(hub with power switching)	Go to **Disconnected**	N.A.	N.A.	N.A.	N.A.
SetPortFeature(PORT_RESET**) request**	Stay in **Powered-off**	Go to **Enabled**	Go to **Enabled**	Stay in **Enabled**	Go to **Enabled**
SetPortFeature(PORT_ENABLE**)** request	Ignore	Ignore	Go to **Enabled**	Stay in **Enabled**	Ignore
ClearPortFeature(PORT_ENABLE**)** request	Ignore	Ignore	Stay at **Disabled**	Go to **Disabled**	Ignore
Downstream packet traffic (hub awake)	Do not propagate	Do not propagate	Do not propagate	Propagate traffic	Do not propagate
Upstream packet traffic (hub awake)	Do not propagate	Do not propagate	Do not propagate	Propagate traffic	Set status field, do not propagate
SetPortFeature(PORT_SUSPEND**)**request	Ignore	Ignore	Ignore	Go to **Suspend**	Ignore
ClearPortFeature(PORT_SUSPEND**)** request	Ignore	Ignore	Ignore	Ignore	Go to **Resume**
Disconnect detect	Ignore	Ignore	Go to **Disconnected**	Go to **Disconnected**	Go to **Disconnected**
Connect Event	Ignore	Go to **Disabled**	N.A.	N.A.	N.A.

11.2.3.1 Device Detach Detection

A hub is able to detect a detach event by means of a continuous SE0 persisting for at least 2.5 µs detected at a downstream port. In response to a detach event, the hub places the port into the Disconnected state and floats its output buffers to a Hi-Z. Device detach can only be detected while there is no downstream traffic on the bus. If power is removed from a port, it must respond by registering a detach event and placing the port in the Powered Off state. This guarantees if a device is detached and a new one added that the attach event will be acknowledged.

11.2.4 Bus State Evaluation

Bus state evaluation is done at the end of the frame and is able to discriminate between the SE0, the differential 1 and 0 bus states. When no device is connected to a downstream hub port, its pull-down resistors pull both D+ and D- below $V_{SE(min)}$.

Connect/Disconnect detect can only be performed after V_{bus} is applied to the downstream port. (This requirement only affects hubs whose downstream ports are power switched.) When a device is connected, the bus state changes from the disconnected to the attach detect state. Low speed devices pull up D- to an SE1 and leave D+ at SE0. Full speed devices pull up D+ to an SE1 and leave D- at SE0. Each downstream hub port must be capable of detecting and differentiating between low speed and full speed device connections once a device is connected. The differential J and K states are undefined until a device is attached and the device's speed has been ascertained.

When a connect or disconnect occurs, it must be reflected in the hub status by the end of the frame in which the event occurred unless the hub is in the reset or suspend modes. A hub in the suspend mode is awakened by a connect or disconnect event and must be capable of reporting the event upon completion of resume. Upon coming out of reset, a hub must detect which downstream ports have devices connected to them. Connect and disconnect changes are reported on a per-port basis.

11.2.5 Full vs. Low Speed Behavior

Hubs must differentiate between full speed and low speed devices when a device is connected to the bus or at power-up. Hubs detect whether a device is full or low speed when the hub port transitions from its disconnected to its disabled state. Devices attached to a hub are determined to be either full speed or low speed by detecting which data line (D- or D+) is pulled high. Low speed devices pull D- high, and full speed devices pull D+ high. Full speed signaling must not be transmitted to low speed devices. Doing so could cause low speed devices to mistakenly respond to full speed signaling and create a bus conflict. Communication between the host and the hub controller are always done using full speed signaling.

If a device is detected to be low speed, the hub port's output buffers are configured to operate at the slow slew rate (75-300 ns), and the port will not propagate downstream directed traffic unless it is prefaced with a preamble PID. Low speed signaling immediately follows the PID and is propagated to both low and full speed devices. Full speed devices will never misinterpret low speed traffic because no low speed data pattern can generate a valid full speed PID. When low speed signaling is enabled, a hub continues to propagate downstream signaling to all enabled ports until a downstream EOP is detected, at which time the output drivers for the low speed ports are turned off and will not be turned on again until the hub receives another PRE PID. If a port is disabled, no signaling is propagated to the port. Hubs must enable their low speed port drivers within four full speed bit times of having received the last bit of the PRE PID, and at that time they must drive a low speed J onto the bus.

If a downstream port is enabled, it propagates upstream directed bus signaling independently of whether the port was configured as low speed or full speed. Hubs implement slew rate selectable output buffers only in the downstream direction on their downstream ports; in the upstream direction, they transparently propagate both low and high speed traffic using fast (4-20 ns) edge rates. Low speed devices do not append a preamble onto their upstream traffic.

While propagating low speed traffic upstream, hubs must be able to respond to EOPs that are either two low speed bit times or two full speed bit times wide. The former will occur under normal operation in which the low speed device generates the EOP. The latter occurs if a downstream hub encounters an end of frame babble or LOA condition and generates an EOP upstream in response (refer to Section 11.4.5).

11.2.5.1 Low Speed Keep-Alive

All hub ports to which low speed devices are connected must generate a low speed keep-alive strobe, generated at the beginning of the frame, which consists of two low speed bit times of SE0 followed by at least 0.5 bit times of J state. Low speed devices use the strobe to prevent themselves from going into

suspend in the absence of low speed bus traffic. The hub repeater generates the keep-alive from its internal SOF counter, and the keep-alive strobe must start no sooner than the second EOF point and must complete no later than the EOP of the token packet.

The low speed keep alive strobe must be generated once per frame, and it must track the SOF token such that the following rules are adhered to.

1. When going into suspend, the keep-alive must not go away before the last SOF.
2. A hub is allowed to synthesize no more than three keep alive strobes after receipt of the last SOF.
3. Keep alive must occur no later than one frame after resumption of SOF.

11.2.6 Hub State Operation

The hub state operation is shown in Figure 11-6. Upon coming out of reset or power-up, a hub starts in the WFSOP state. The hub waits for a start of a packet (SOP) to be detected on its root port or any of its enabled downstream ports. If an SOP is detected, the hub establishes connectivity originating from the port on which the SOP occurred and transitions to its WFEOP state. It remains in this state until an end of packet (EOP) is encountered or until the end of frame (EOF) occurs. Under normal circumstances, and when not near the end of a frame, a hub repeater will transition back and forth between WFSOP and WFEOP.

A hub in the idle (WFSOP) state responds to the end of frame (EOF1) point by transitioning to the WFSOF state. If a hub is in the WFSOP state at EOF1, it transitions to the WFSOF state. Transitions from WFSOP and WFEOP to WFSOF are not errors, but simply indicate that the hub is nearing the end of its frame and cannot establish connectivity until the start of the next frame.

WFEOF2 is a special state which is entered only when a babble or loss of bus activity (LOA) is detected near the end of a frame and upstream connectivity is established. If a hub repeater is still in the WFEOP state (i.e., it has not received an EOP) when the EOF1 point is encountered, it transitions to the WFEOF2 state. It will remain there until its EOF2 point or an upstream EOP occurs, at which time it transitions to WFSOF and awaits the next Downstream SOP (DSOP), which will normally be the SOP associated with the SOF packet, and indicates the start of the next frame. When a DSOP occurs, the hub returns to the WFEOP state and waits for the end of the packet. If the end of a frame occurs and a hub still is maintaining downstream connectivity, the hub does not transition out of the WFEOP state; instead it waits for the next downstream EOP (usually the EOP of the SOF token), which will cause the hub to transition to the WFSOP state.

If a hub is still in its WFEOF2 state when EOF2 occurs, the port that established upstream connectivity must be disabled, regardless of the bus state. If, when EOF2 occurs, a hub is in the WFSOF state, and is driving upstream connectivity, and its bus is in the idle state, then the previously connected port's state must remain unchanged. If the hub is in the WFSOF state at EOF2 and driving upstream connectivity, but the bus is not in the idle state, the port must be disabled.

A hub will transition to the WFSOF state upon coming out of resume. Doing so guarantees that traffic cannot propagate upstream (and possibly lock up the bus) until the hub's frame timer has detected at least one SOF. The details of the interaction between the frame timer and the hub repeater state machine are described in Section 11.5.1.1.

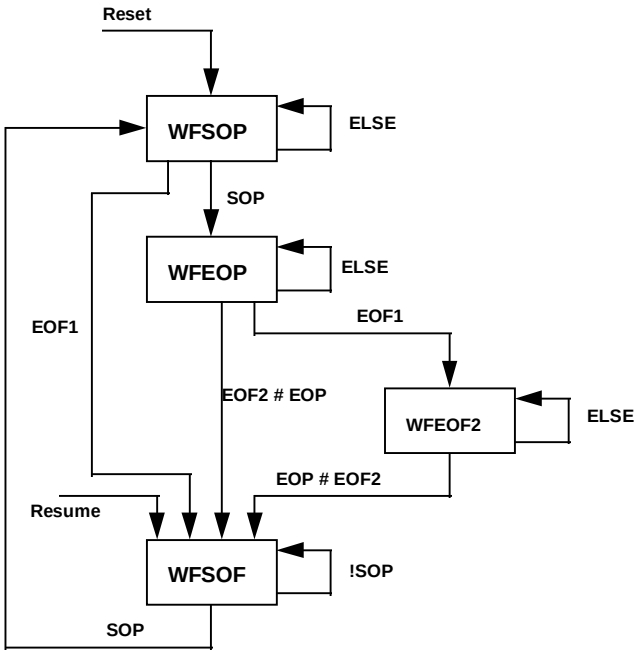


Figure 11-6. Hub Repeater States

The hub repeater maintains state across each packet that is detected and repeated by the hub. The repeater state machine does not need to track more than a single packet and need not, for example, track across multiple packets in a transaction. Figure 11-7 shows how hub states change in the course of a normal packet transmission.

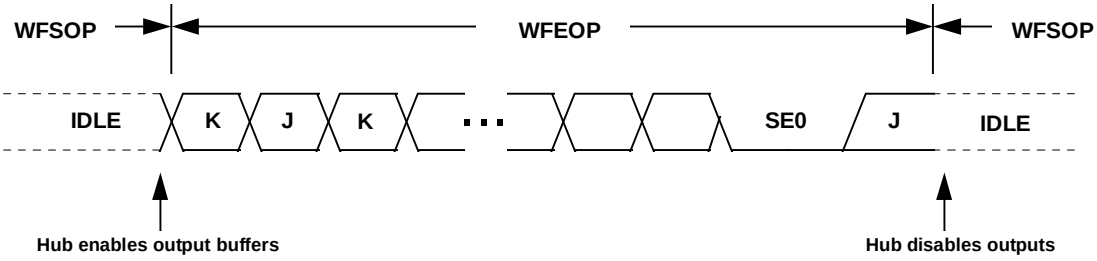


Figure 11-7. Hub States Across a Packet

11.2.6.1 Hub Repeater States

For upstream connections, a hub repeater transitions between four states: wait for start of packet (WFSOP), wait for end of packet (WFEOP), wait for EOF2 point (WFEOF2), and wait for start of frame (WFSOF). The EOF1 and EOF2 points are described in Section 11.4.5.1. The four states are described below. For downstream connections, the hub transitions between WFSOP, WFEOP, and WFSOF.

11.2.6.1.1 Wait for Start of Packet

The Wait for Start of Packet (WFSOP) is the state a hub occupies when there is no packet currently being propagated to or through the hub. Hubs transition to their WFSOP state upon coming out of reset. In the WFSOP state, all of a hub's ports are in the high impedance state, and all of its enabled ports are in the receive mode with their output buffers in the Hi-Z state. If the root port or any enabled downstream port detects an SOP, the hub establishes connectivity and transitions to the Wait for End of Packet state.

11.2.6.1.2 Wait for End of Packet

During the Wait for End of Packet (WFEOP) state, the hub has established its connectivity and is receiving packet traffic on one of its ports. The hub transparently propagates the traffic in either the upstream or downstream direction. Connectivity is maintained until the hub transitions out of this state. A hub transitions out of the WFEOP state when it detects an EOP or if it encounters an end of frame (EOF1) point (refer to Section 11.4.4). Detection of EOP causes the hub to transition back to WFSOP and is the normal sequence. If EOF1 is detected, the hub transitions to the WFEOF2 state.

11.2.6.1.3 Wait for EOF2 Point

The WFEOF2 state is entered only when the hub detects its EOF1 point and is still waiting for an EOP from a downstream port. This condition is potentially indicative of babble or loss of bus activity. A hub repeater remains in the WFEOF2 state until an EOP is detected or until its EOF2 point occurs.

11.2.6.1.4 Wait for Start of Frame

A hub repeater enters the Wait for Start of Frame (WFSOF) state either when EOF1 is detected and the hub is in the WFSOP state (normal end of frame behavior) or when the hub is in the WFEOF2 state and an EOP or EOF2 point is detected (babble/LOA) behavior.

11.3 Hub I/O Buffer Requirements

All hub ports must be able to detect and generate the J, K, and SE0 bus signaling states. This requires that hub ports be able to independently drive and monitor their D+ and D- outputs. Each hub port must have single ended receivers on the D+ and D- lines as well as a differential receiver. Details on voltage levels and drive requirements appear in Chapter 7. Figure 11-8 shows I/O circuitry for a typical hub port.

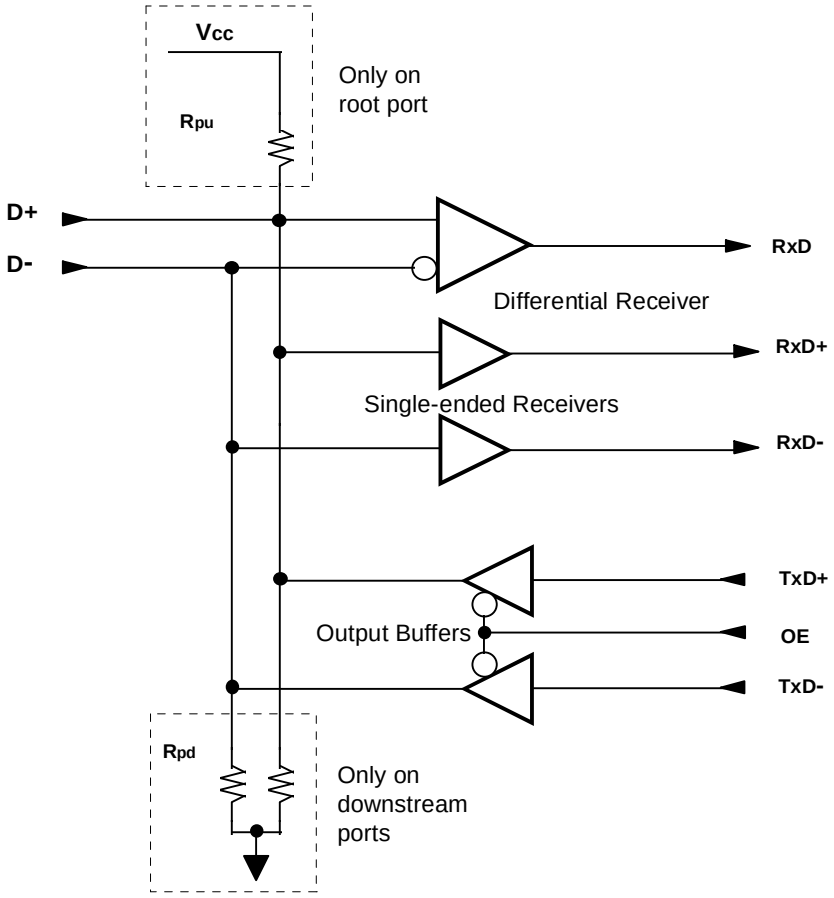


Figure 11-8. Hub Port I/O Driver and Receiver

Table 11-2 defines the hub I/O section's input and output signals.

Table 11-2. Hub I/O Section Signals

Signal Name	Direction	Description
D+, D-	I/O	External USB data lines
RxD	O	Received differential data
RxD+	O	Received single-ended value on D+ line
RxD-	O	Received single-ended value on D- line
TxD+	I	Transmitted data value
TxD-	I	Transmitted data value
OE	I	Output enable/disable on output buffers

D+ and D- are the I/O lines that connect to the USB physical medium. When placed in the Hi-Z state, they are pulled to near the ground or Vcc rails by resistors on the hub and device. RxD is the differential received data. RxD+ and RxD- are the received single ended data. TxD+ and TxD- are used to send differential data and single ended reset and EOP signaling. OE disables the output drivers.

11.3.1 Pull-up and Pull-down Resistors

Hubs, and the devices to which they connect, use a combination of pull-up and pull-down resistors to control D+ and D- in the absence of their being actively driven. These resistors establish voltage levels used to signal connect and disconnect and also maintain the data lines at their idle values when not being actively driven. Each hub downstream port requires a pull down (R_{pd}) on each data line; the hub root port requires a pull-up (R_{pu}) on its D+ line. Values for R_{pu} and R_{pd} appear in Chapter 7.

11.3.2 Edge Rate Control

Downstream hub ports must support both low speed and full speed edge rates. Full speed signaling specifies a rise/fall time of 4-20 ns. Low speed rise/fall times must be within a 75-300 ns range. Edge rate on a downstream port must be selectable, based upon whether a downstream device was detected as being full speed or low speed. The hub root port always uses full speed signaling, and its output buffers always operate with full speed edge rates.

11.4 Hub Fault Recovery Mechanisms

Hubs are the key USB component for establishing connectivity between the host and other devices. It is vital that any connectivity faults, especially those that might result in a deadlock, be detected and prevented from occurring. Hubs need to handle connectivity faults only when they are in the repeater mode. Hubs must also be able to detect and recover from lost or corrupted packets which are addressed to the hub controller. Since the hub controller is, in fact, another USB device, it must adhere to the same time-out rules as other USB devices, as described in Chapter 8.

11.4.1 Hub Controller Fault Recovery

The hub controller must be able to respond to and recover from corrupted and missing packet transmissions. These include lost or corrupted token, data, and handshake packets. The following table describes the possible field level errors which the hub controller can detect and its responses.

Table 11-3. Packet Error Types

Field	Error	Action
PID	PID check, bit stuff	Ignore packet
Address	Bit stuff, address CRC	Ignore token
Data	Bit stuff, data CRC	Discard data

11.4.2 False EOP

Hub handling of false EOP differs depending on whether the hub is operating as a repeater or is being accessed as a device. A hub operating as a repeater transparently propagates signaling, and cannot differentiate between a “good” EOP and a “false” EOP. If any EOP occurs, the hub tears down connectivity and waits for the next SOP. If the packet transmitter continues sending, the hub re-establishes connectivity on the next J to K transition. From a hub’s point of view, a false EOP makes a single packet look like two separate packets. The hub does not participate in false EOP error detection or recovery process when operating in the repeater mode.

When a hub is accessed as an USB device, the hub controller detects and recovers from false EOP the same as any other USB device, as described in Section 8.7.3.

11.4.3 Repeater Fault Recovery

Hubs must be able to detect and recover from conditions that leave them waiting indefinitely for an end of packet or that leaves the bus in a state other than the idle state at the end of a frame, i.e., an SOP without an EOP. There are two such hub fault conditions: loss of activity and babble. Loss of activity (LOA) is characterized by detection of a start of packet (SOP) followed by lack of bus activity (bus is stuck at the idle or K state) and no end of packet (EOP) by frame’s end. Babble is characterized by SOP followed by the presence of bus activity continuing past the end of a frame. Hubs have no notion of allocated bandwidth and must rely upon the frame timer to detect LOA and babble conditions. The recovery mechanism requires that hubs track the host’s frame timing and recover before the beginning of the next frame.

Hub fault recovery operates only in the upstream direction. The host is responsible for detecting and recovering from its own downstream directed errors. Hub repeater fault recovery must meet the following requirements:

- Devices driving illegal states at the end of a frame must be isolated from the bus by disabling the downstream hub port to which the device is connected.
- Hubs must return the bus to the idle state before the start of the next frame if the connectivity was previously established in an upstream direction.

Under non-fault conditions, these requirements are met by virtue of a hub receiving an EOP with every packet and having no bus traffic occur past the end of a frame. Before describing how hubs implement fault recovery, the hub frame will be described.

11.4.4 Hub Frame Timer

Each hub has a frame timer whose timing is derived from the host-generated SOF token and tracks the host SOF packet in both phase and period. The frame timer is reset each time an SOF is detected and is responsible for generating End of Frame (EOF) points. The hub frame timer must track the host SOF and be capable of remaining synchronized to the host SOF for the loss of up to two consecutive SOF tokens. All hubs must have an EOF timer, and it is used to identify two distinct points in time: a point (EOF1) at which time the hub must terminate its own upstream connectivity, and a point (EOF2) by which time the bus must see upstream traffic terminated. The delay between EOF1 and EOF2 corresponds to the timing skews between the host and the hub plus time required for EOP to occur and is illustrated in Figure 11-9.

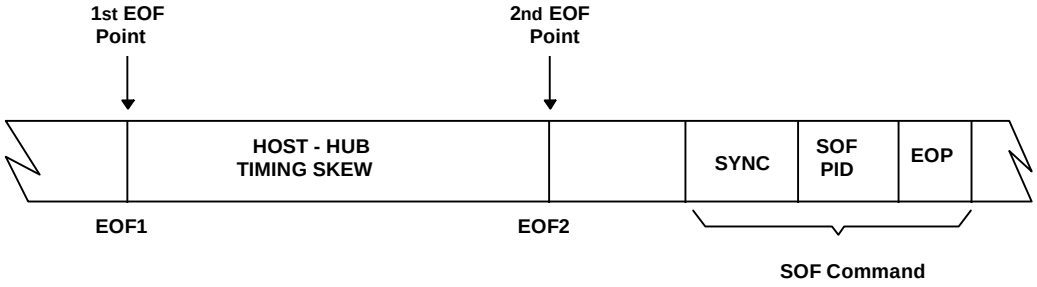


Figure 11-9. Host-Hub EOF Skew

The frame timer must to synchronize to the host's frame timing for worst case tolerances and offsets between the host and hub. The offsets have to accommodate the hub oscillator tolerance and purposely introduced deviations from 1.000 ms frame timing which occur when the host locks to an external source. The above parameters require a minimum frame length (FL_{min}) of 11,985 and a maximum frame length (FL_{max}) of 12,015.

11.4.4.1 Frame Timer Synchronization

The frame timer is clocked from the hub's clock source and is synchronized via SOF packets to the host's frame time. When a hub is first connected to the host the hub's frame timer is unsynchronized with respect to the host. When the first SOF token is detected the frame timer resets and starts counting once per 12 MHz clock cycle; it continues to count up until the next SOF token is detected. At this time the frame timer is reset and the previous count value is stored as the previous frame length (PFL) count. This value is updated for each frame in which an SOF is detected and represents, in hub clock counts, the host's frame time. The frame timer is considered locked to the host when the difference between the PFL count and the current count at the end of frame is less than 2 and the PFL lies between FL_{min} and FL_{max}.

Should an SOF token fail to be received, the hub uses the previous frame length count as a best approximation to the current frame length. The frame timer is guaranteed to remain synchronized to the

host for the loss of up to two consecutive SOF tokens. Hubs must be able to synchronize to the host within 3 frames after detecting the first SOF packet (assuming no missing SOF packets). When a hub comes out of resume its frame timer is not synchronized with that of the host. During this time, when the hub is in the active state, it must be prevented from establishing upstream connectivity until the hub frame timer is synchronized to the host.

The hub's error recovery, as described in Section 11.4.5, is operational whenever the hub frame timer is locked. If the frame timer is not locked the timer uses FLmin as the default frame length.

11.4.5 Hub Behavior Near EOF

Hub behavior near the end of frame is diagrammed in Figure 11-10. There are two end of frame timing markers, EOF1 and EOF2, corresponding to the first and second EOF points. All hubs receiving upstream traffic, upon detection of EOF1, transmit an EOP upstream for two full speed bit times, drive the bus to the idle state for one bit time, and then float the bus. Starting with EOF1, hubs are not permitted to re-establish upstream connectivity until the end of the next downstream packet which is usually the next SOF token.

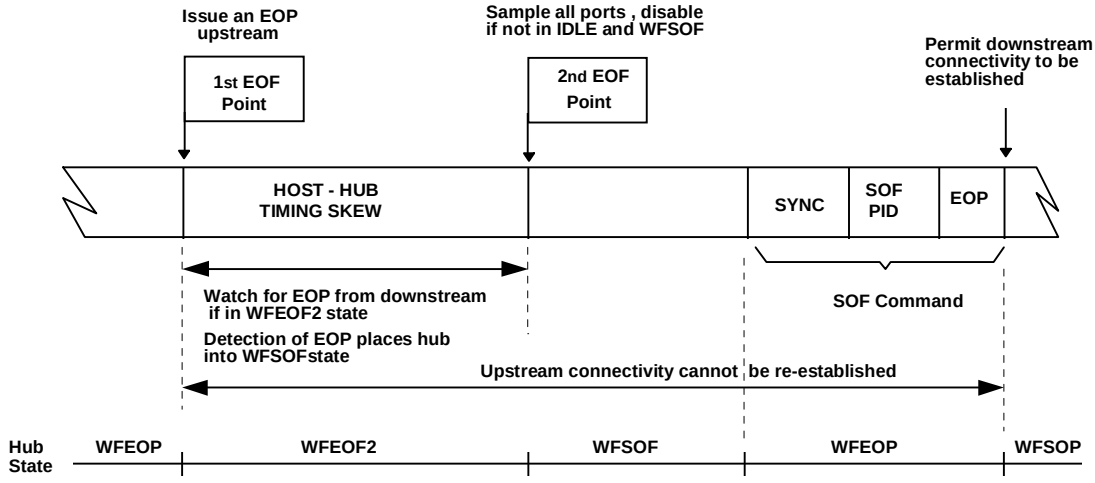


Figure 11-10. Hub Behavior Near End of Frame

Hub behavior at the end of frame is summarized below:

1. Hub recovery behavior is applicable only for upstream traffic. If a hub encounters the end of frame and downstream connectivity is established, it maintains connectivity and waits for the next downstream EOP, at which time it tears down connectivity and returns to the WFSOP state.
2. At the EOF1 point, all hubs will transmit an upstream EOP, followed by an idle state, and then float the bus, unless connectivity is already established in the downstream direction. The EOP must not truncate or lengthen any upstream directed EOP already in progress.
3. Hubs will not allow further connectivity to be established in the upstream direction after EOF1.
4. Hubs that were in the WFEOP state at EOF1 must watch for EOPs on the downstream port on which connectivity was established. They must monitor the bus from EOF1 to EOF2.
5. If an EOP from downstream is detected by a hub in the WFEOF2 state in the EOF1 to EOF2 window, the hub will transition to the WFSOF state and should see an idle on its port.
6. At EOF2, all ports will be sampled for their state, and a port will be disabled if it is not in the proper state (refer to Table 11-4). At EOF2, hubs still in WFEOF2 transition to the WFSOF state. Connectivity is still not allowed from downstream until after a packet is received from the host.

Table 11-4. Hub Behavior at EOF2

	Not Idle State	Idle State
WFSOF	Disable port	Do not disable port
WFEOF2	Disable port	Disable port

11.4.5.1 Skew Requirements

The host and hubs, while all synchronized to the host's SOF, are subject to certain skews which dictate the length of time between the EOF points, host behavior near EOF, and the next SOF. Figure 11-11 illustrates critical end of frame timing points.

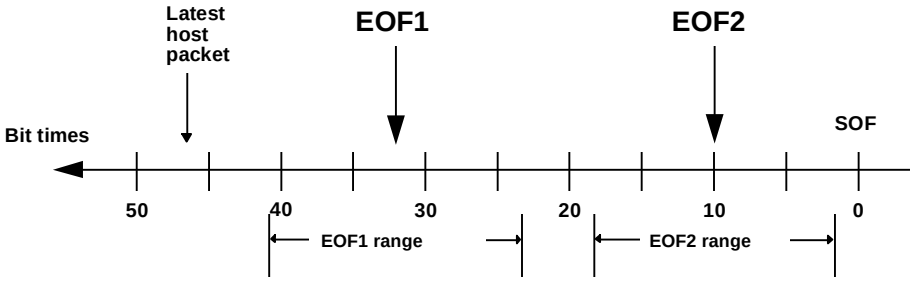


Figure 11-11. EOF Timing Points

11.4.5.2 Host-Hub Skew

The timing skew between the host's SOF point and the hub's SOF timer is minimized by the requirement that the hub track the host. Sources of skew include the fact that hubs may miss SOFs, and that the host frame counter can be adjusted to track an external master clock. The 12 MHz clock is the only clock actually specified, so this is the best granularity available by specification. Assuming a fixed host SOF timing and that two consecutive SOFs can be missed, the maximum cumulative host-hub skew without host timing wander is ±3 clocks. Assuming that the host clock may be adjusted by up to one bit time per frame, then the host can walk away from the hub by 1 + 2 + 3 = 6 clocks. The maximum host-hub skew is the sum of these two components or ±9 clocks.

The second EOF point must be sufficiently separated from the SOF point to permit hubs to recover and be ready to receive the SOF token from the host. A hub must finish sending its EOP before a hub to which it is attached reaches its second EOF point. This means that all hub EOF2 points must occur at least one bit time before the host issues SOF. All hub EOF2 points must lie within a ±9 bit time window; therefore, EOP must lie outside this window and complete at least 2 x 9 + 1 = 19 bit times before host SOF.

The next step is calculating how long it takes to generate EOP and how far back from SOF it must occur. Transmitting EOP requires four bit times. Therefore, a hub must start sending its EOP no later than 19 + 4 = 23 bit times before SOP. For a hub to be sure that it starts no later than the 23rd bit time, it must start 9 bit times before that or at bit time 32, which is the value of the first EOF point. The earliest that a hub might start sending EOP is 9 bit times before the first EOF point or at bit time 41.

A hub must not see a packet from the host start after the hub reaches its first EOF point. This could be as early as 41 bit times before SOF. Hub propagation delay must also be figured into the delay budget. The per-hub delay is approximately one bit time; so for a worst case topology of six hubs away from the host, there will be an additional 6 bit times of delay. Therefore, the host's EOF point for transmit is 41 + 6 = 47 bit times from SOF, relative to the host's SOF timer. If the host is still transmitting at bit 47 and not able to complete before SOF, it must force an error via a bit stuffing violation (at least eight 1's),

followed by an EOP. If the host is still receiving a packet or an EOP at bit 41, it should treat the packet as being in error. Table 11-5 summarizes hub and host EOF timing points.

Table 11-5. Hub and Host EOF Timing Points

Description	Number of Bits From Start of SOF	Notes
EOF1	32	End of frame point #1
EOF2	10	End of frame point #2
Host invalidates full speed Tx packet	47	Latest that host may start a full speed packet
Host invalidates low speed Tx packet	184	Latest that host may start a low speed packet (rounded up to the nearest low speed bit time)
Host invalidates Rx packet	41	Host treats any packet still being received at bit time 41 as bad

11.5 Suspend and Resume

Hubs must support suspend and resume both as a USB device and also in terms of propagating the suspend and resume signaling. Hubs support both global and selective suspend and resume. Selective suspend and resume are implemented via per port enable/disable. Global suspend is implemented by the host through the hub's root port. Global resume may be initiated either from the host or from a hub's downstream port.

11.5.1 Global Suspend and Resume

Global suspend is initiated by the host shutting off downstream traffic to the entire bus. A hub enters the suspend state if it detects a continuous idle state on its root port for at least 3.0 ms. When placed into the suspend state, a hub puts its repeater into the WFSOP state, floats all of its output drivers, maintains static values of all its control and status bits, and preserves the current state information for each of its downstream ports. A suspended hub has its clocks turned off, so it has no concept of time and can only respond to bus transitions.

Hub resume may be initiated by any bus transition on a hub's root port or on a downstream port in the enabled state. Hub resume may also be initiated by the connect/disconnect of a device on a downstream port in the disconnected, disabled, or suspended states. If a transition occurs on an enabled downstream port then the hub immediately reflects an idle to a resume bus transition to that port, all other enabled downstream ports, and its root port. Figure 11-12 shows the timing relationships during a resume sequence in which a device initiates a wake up to a hub.

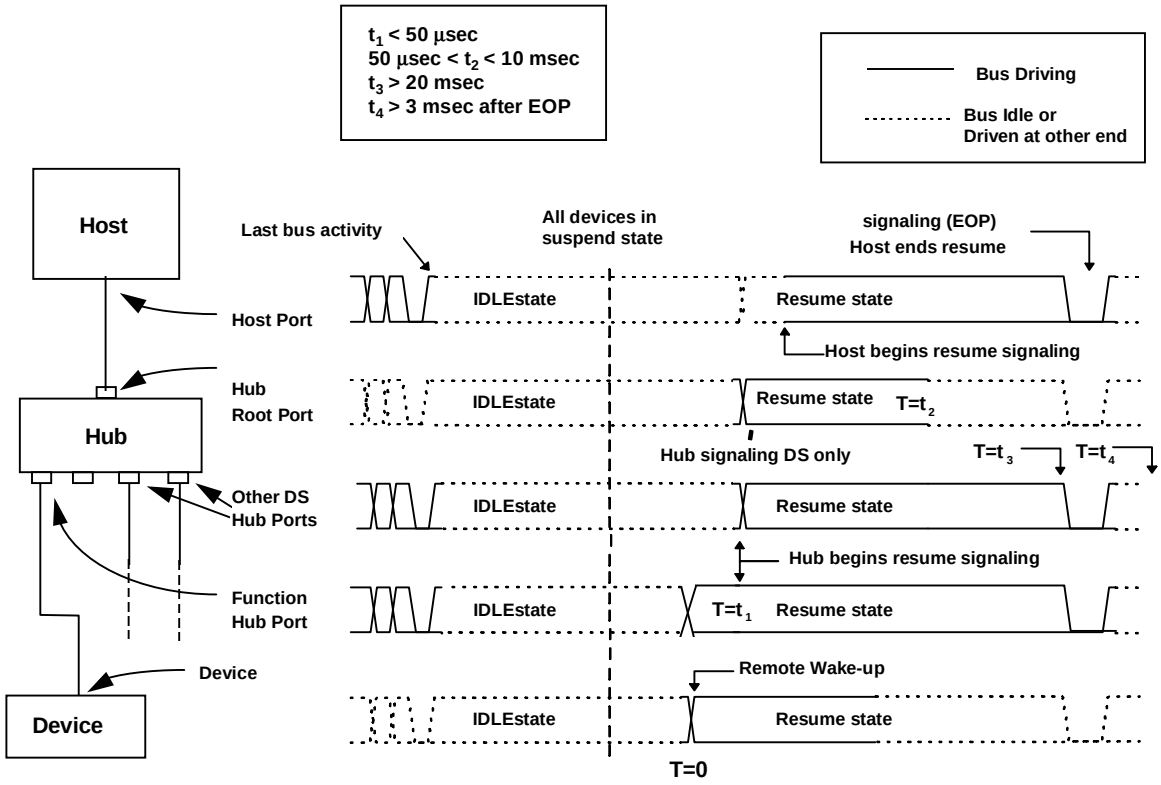


Figure 11-12. Resume Signaling

There are four time parameters that hubs must conform to, as shown in Figure 11-12. T=0 represents the time when the resume signaling arrives at the port. t_1 is the time by which a hub must respond by driving resume onto the upstream port and all enabled downstream ports. t_2 is the time at which the hub must stop driving an upstream initiated resume to its root port and reflect the bus state now being driven to its root port onto its downstream ports. The interval, t_3 is the time that a hub must generate downstream K signaling to the device that initiated the resume. t_4 is the time that the hub must wait after detecting the downstream low speed EOP until it can set its interrupt bit, indicating that a resume has occurred after a selective suspend.

When a device drives an idle to resume transition upstream to the hub the hub responds by driving the resume (K) state onto the bus to the root port and to all enabled downstream ports including the port which initiated the resume. (It is acceptable to drive both ends of a bus segment to the same state.) After driving the resume state, the hub begins the process of returning to a fully awake state (e.g., restart clocks). When the hub is awake, it will reverse the connectivity so that the K state on the hub's root port is sustaining the K on its downstream ports. (This scenario assumes that the hub immediately above the one in question has received resume signaling and is now driving it downstream.) A hub may not reverse the connectivity any faster than 50 µs, nor slower than 10 ms after receiving a resume from a downstream. The t_2 parameter also implies that a hub must be fully awake no later than 10 ms after receiving a resume request.

The resume signal propagates upstream until it reaches the host. The host reflects the resume signaling downstream for at least 20 ms, which guarantees that all devices will have time to wake up and detect the downstream resume signaling. The host terminates the resume sequence by driving an EOP for two low speed bit times. The EOP is interpreted as a valid end of packet, causes all hubs to tear down their connectivity and informs all devices on the bus that the resume sequence has completed. The device that initiated the resume must wait until it detects EOP to determine that the resume sequence has completed.

Hubs must be able to propagate downstream traffic immediately after the end of resume to prevent downstream devices from re-entering the suspend state. The hub controller must be able to receive packet traffic no later than 10 ms after the end of resume. Note: a remote wakeup device may not start a resume sequence until 5 ms after the last bus activity. This allows the hub to go into the suspend state so that it will resume all ports and not just the full speed ones. Table 11-6 summarizes the behavior of hubs in response to host initiated, downstream signaling.

Table 11-6. Suspended Hub Behavior During Global Resume

Port Status and Signaling Type	Downstream Port Response
Port enabled, resume (K) received	Signal resume downstream
Port disabled, resume (K) received	Do nothing
Port suspended, resume (K) received	Signal resume downstream

11.5.1.1 Resume and Hub Frame Timer

When a hub transitions from the suspended to the awake state, its frame timer is not operational and the hub's EOF timing recovery circuitry will not work until the first SOF has been received. To prevent a babbling downstream device on a recently resumed bus segment from locking up the bus it is necessary to prevent hubs from propagating upstream traffic while the hub is in the awake state until the hub's frame timer has started. This is achieved by making the hub repeater state machine transition to its WFSOF state upon coming out of resume. While the hub repeater is in the WFSOF state all upstream traffic is ignored. The repeater state machine remains in the WFSOF state until an SOF is detected, at which time it transitions to the WFEOP state. Downstream traffic is unaffected by the status of the frame timer. To prevent needless time-outs it is recommended that the host not send any packets addressed to devices on a just resumed bus segment until the host has issued at least one SOF token to the recently resumed hub or hubs. This condition is guaranteed by virtue of the fact that a hub does not report a resume detect to the host until 3.0 ms after the resume sequence completes.

11.5.2 Selective Suspend and Resume

Selective suspend and resume provide a means for placing a single device or a bus segment into a low power state. Selective suspend relies on the ability of a hub to selectively suspend individual ports via a SetPortFeature(PORT_SUSPEND) request, which places a port on a hub (referred to as the disabling hub) into the suspend state (see Figure 11-4 and Figure 11-5 for state diagrams). In the suspend state a port is prevented from propagating any bus activity (except the port reset request) downstream, and the port can only reflect upstream bus state changes via the hub's status bits; i.e., an awake hub cannot propagate upstream traffic from a suspended port to its root port. The hub must also insure that the port accessed via the port suspend request is not suspended in the middle of a packet transaction. In response to a port suspend, all devices downstream of the targeted port go into suspend after failing to see bus activity for 3.0 ms while the bus is in the idle state The port suspend request is only understood by an awake hub. If the host wishes to send the request to a suspended hub, it must first wake the hub and then issue the desired request.

11.5.2.1 Selective Resume to an Awake Hub

Figure 11-13 shows the signaling between a device and an awake hub (Hub Y). If a hub port is selectively suspended and the hub is in the awake state, then the hub must prevent any bus activity on the suspended port from being reflected to other enabled ports on the hub, as these ports may be transporting bus traffic. In addition, the resume on Port B must not propagate to other suspended ports (such as Port C). The solid line represents the extent of the resume signaling, which stops at Port B of Hub Y. Resume signaling may occur either from a host request to directly re-enable the hub port (downstream resume) or from a device on the suspended bus segment (upstream resume).

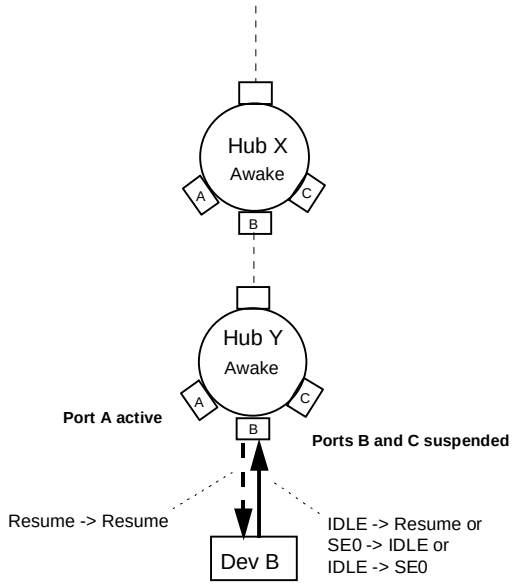


Figure 11-13. Selective Resume Signaling on an Awake Hub

Upstream resume may be initiated by an idle to resume (device issues resume), an idle to SE0 (device disconnect) or an SE0 to idle (device attach) bus transition on a suspended port. Hub Y, Port B must reflect a resume state within 50 µs of receipt of resume to inform the remote wakeup device (Device B) that its resume has been detected. If the transition on Port B was not idle to resume (J to K), Port B does not reflect any signaling downstream but continues to maintain its output buffers in the Hi-Z state. This latter case covers connect and disconnect events.

The receipt of resume signaling by Port B of Hub Y causes the following sequence of events to occur.

1. Hub Y, Port B reflects the resume signaling downstream within 50 µs of resume detection.
2. Hub Y maintains resume signaling downstream for at least 20 ms.
3. At the end of 20 ms, the hub terminates the resume with a low speed EOP (Downstream traffic starts flowing through the port at this time.)
4. 3.0 ms after the end of EOP, an interrupt in the hub controller is set.

The hub is responsible for maintaining all the resume timing parameters. The host need not keep track of any time and can poll the hub via a GetPortStatus request to determine that a resume event has occurred. Since the hub process selective resumes autonomously, the host will not receive direct signaling in the event of a selective resume that does not originate from the host. It is necessary for the host to poll downstream hubs (as part of its status polling) to determine that a selective resume event has completed. The 3.0 ms delay between the end of resume signaling and the setting of the interrupt bit guarantees that no packet traffic with a newly resumed device will occur until newly resumed devices have had time to

synchronize their frame timers with SOF; however, downstream devices will see traffic and SOF tokens not specifically addressed to them. When a resume is initiated by a downstream device, the host polls for the end of selective resume as part of its status polling The hub must also insure that the port is not re-enabled in the middle of a packet transaction.

Downstream selective resume may also be initiated via a ClearPortFeature(PORT_SUSPEND) request; This request causes the disabled port to drive resume signaling onto the bus for at least 20 ms, followed by a low speed EOP. As with a device initiated resume, the interrupt bit in the hub controller, signaling the end of resume, must not be asserted until 3.0 ms after the end of the low speed EOP. The host must poll the hub interrupt to determine when the resume has completed. Table 11-7 summarizes the behavior of an awake hub in the presence of resume signaling originating from downstream.

Table 11-7. Awake Hub Behavior During Resume

Port Status and Signaling Type	Signaled Port Response	Adjacent Enabled Port Response	Adjacent Disabled Port Response	Adjacent Suspended Port Response
Port suspended, resume (K) received	Reflect K downstream on signaled port. Initiate port wake-up. Set status bits and interrupt.	Do nothing	Do nothing	Do nothing
Port enabled, disabled or suspended and disconnect received	Set port disconnect, disable and change status bits. Set interrupt.	Do nothing	Do nothing	Do nothing
Port disabled and connect received	Set port connect, and change status bits. Set interrupt.	Do nothing	Do nothing	Do nothing

11.5.2.2 Selective Resume to a Suspended Hub

It is possible for the host to suspend a hub port and then suspend the entire hub. In this case, the hub's connectivity changes once the hub enters the suspended state. Figure 11-14 illustrates device initiated connectivity for suspended ports on a suspended hub. Device B may issue connect, disconnect, or resume signaling to Hub Y, Port B. In response, Hub Y must convert that signaling into a (J to K) transition, regardless of what signaling device B sends upstream. The hub drives the resume state to the root port and to other enabled downstream ports (Port A). Hub Y must drive Port B to the resume state if Device B drove the bus to the resume state. If Port B detects a connect or disconnect event, the hub does not drive a resume downstream to the signaling port, but maintains the port's output buffers in the Hi-Z state.

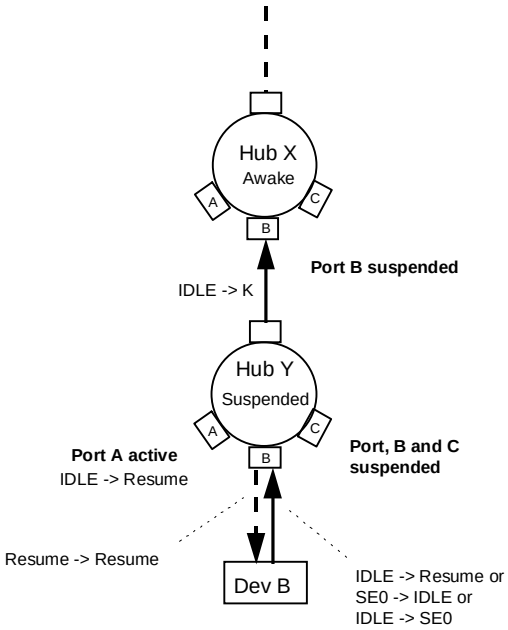


Figure 11-14. Device-Initiated Resume for Suspended Hub

Table 11-8. Suspended Hub Behavior During Resume

Port Status and Signaling Type	Signaled Port Response	Adjacent Enabled Port Response	Adjacent Disabled Port Response	Adjacent Suspended Port Response
Port enabled, resume (K) received	Reflect resume upstream and downstream. Do not set status bits	Signal resume downstream	Do nothing	Do nothing
Port disabled, resume (K) received	Do nothing	Do nothing	Do nothing	Do nothing
Port suspended, resume (K) received	Reflect K upstream on root port and downstream on signaled port	Signal resume downstream	Do nothing	Do nothing
Port enabled, disabled or suspended and disconnect received	Reflect resume upstream. Start hub wake-up. Update port connect and change status bits, set interrupt	Signal resume downstream	Do nothing	Do nothing
Port disabled and connect received	Reflect resume upstream. Start hub wake-up. Set port connect, and change status bits, set interrupt	Signal resume downstream	Do nothing	Do nothing

Host initiated signaling for suspended hubs is shown in **Figure 11-15**. Waking up a device at the bottom of a string of suspended hubs is a multi step procedure which is described below.

1. The host takes Port B in Hub X out of suspend by issuing a port resume request to the hub. In response to the resume request, Hub X initiates the resume signaling by driving at least 20 ms of K signaling followed by a low speed EOP out its Port B. Hub Y sees the resume signaling and starts its wake up process. The EOP indicates that the resume sequence is completed. 3.0 ms after the EOP is issued by Hub X, the resume complete status bit in Hub X's controller is set indicating that the resume sequence is completed. At the end of this stage Hub X, port B is in the enabled state and Hub Y is awake.

2. The next step is taking Port B on Hub Y out of the suspended state to the enabled state. The procedure is identical to that described in the previous step except that the request is issued to Hub Y, Port B instead of Hub X. At the end of this step, Port B on Hub Y is enabled and device B has received resume signaling and is awake. At this time the host may communicate with Device B. This resume policy permits the bus to be sequentially suspended while permitting any device on the suspended segment to awaken the bus. It also permits nested hubs to be awakened one tier at a time.

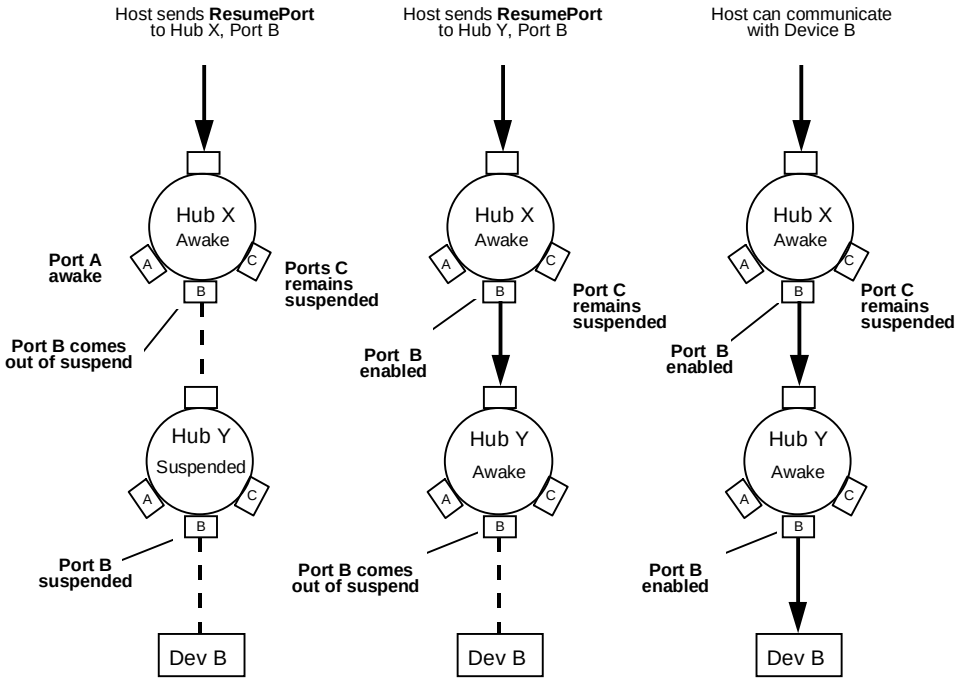


Figure 11-15. Host-initiated Resume Through Suspended Hubs

11.6 USB Hub Reset Behavior

A USB hub must be able to generate a per port reset via a host request and accept reset signaling on their root port. The following sections describe hub reset behavior and its interactions with resume, attach detect, and power-on.

11.6.1 Hub Receiving Reset on Root Port

Reset signaling to a hub is defined only in the downstream direction which is at the hubs' root port. An awake hub may start its reset sequence if it detects 2.5 µs or more of continuous SE0 signaling and must complete its reset sequence no later than 5.5 µs of continuous SE0. The 2.5 µs lower limit is set by a

need to prevent low speed EOP strobes (which are up to 1.3 μs in length) from being interpreted as reset. A suspended hub must interpret the start of reset edge as a resume signaling event and begin its wake-up sequence. The hub must be awake and have completed reset no later than 10 ms the completion of reset signaling

After completion of reset a hub controller is in the following state:

- Hub controller default address is 0
- Hub control bits set to default values
- Hub repeater is in the WFSOP state
- All downstream ports in Powered Off state (power-switched hubs)
- All downstream ports in Disconnected state (non power-switched hubs)

If a bus contains hubs with power-switched ports, the host reset is not guaranteed to propagate all the way downstream. The host has to guarantee that each tier is reset when it goes through the enumeration process, and the enumeration reset is done on a tier by tier basis. (However, the powered off devices are effectively reset, if they are off long enough, and self-powered devices/hubs below them reset themselves and their downstream ports.)

11.6.2 Per Port Reset

A hub can exercise per-port resets via the SetPortFeature(PORT_RESET) request. This request specifies a downstream port number. In response to a SetPortFeature(PORT_RESET) request, the hub drives an SE0 onto its downstream port for at least 10 ms, returns the bus to the idle (J) state, and the places the port into the enabled state. SetPortFeature(RESET) is an atomic operation; the 10 ms delay between start and end of reset is controlled by the hub. The hub must be able to return to the host the status of the reset request; i.e., whether the reset has completed, so that the host does not have to keep track of elapsed time during the reset operation. The port reset request can be issued to a port in any state; however, no downstream signaling is generated if the reset is issued to a port in the powered off or disconnected states.

Device attach detection requires that the port in question be power-switched on (if power switching is an option). When a device is attached, a hub can detect an attach via an SE0 to DIFF1 or DIFF0 bus transition. This requires that disabled ports not be driven by the hub while attach detection is being performed. This should not be a problem, as the port will have been disabled and its output drivers floated by detection of the previous detach event. The host can determine the device's speed by examining whether D+ or D- is pulled high.

Before a port to which a device has been connected can be enabled we must be assured that the device has been reset. Since it is not possible to rely on loss of Vbus, caused by a disconnect event, to reset the device, a port must be reset before being enabled. This is performed via an atomic SetPortFeature(PORT_RESET) request which issues SE0 reset signaling and then enables the port. USB devices, including hubs, must be able to respond to a host access no later than 10 ms after reset is de-asserted.

11.6.3 Power Bringup and Reset Delays

Since USB components may be hot plugged, and hubs may implement power switching, it is necessary to comprehend the delays between power switching and/or device attach and when the device's internal power has stabilized.

Figure 11-16 shows the case where a device is connected to a hub whose port is power switched on. There are two delays that need to be taken into consideration. $\Delta t1$ is the amount of time required for the hub port switch to operate. $\Delta t2$ is the time required for the device's internal power rail to stabilize. If a device were plugged into an non-switched or already switched on port, only $\Delta t2$ would need to be considered. $\Delta t1$ is a function of the type of hub port switch, and this parameter may be read via a hub controller command. $\Delta t2$ must be less than 100 ms for all hub and device implementations. It is necessary to specify a worst-case upper limit on $\Delta t2$, since it is device specific and cannot be reported until after the power-on and reset sequences are completed.

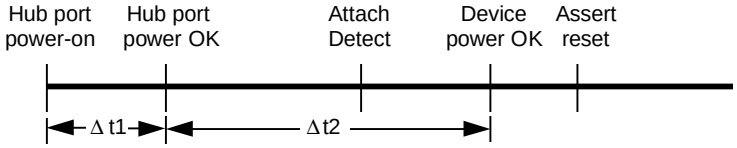


Figure 11-16. Power-on timing

As Figure 11-16 shows, it is possible to detect a device attach before its internal power has stabilized. One must guarantee a minimum of 10 ms during which a device's internal power is stable and reset is asserted. Therefore, reset cannot be asserted immediately after device attach but must instead wait for $\Delta t2$ to elapse. If the hub implements power switching, $\Delta t1 + \Delta t2$ must elapse before reset is applied.

USB devices must power on in such a manner that they do not drive D+ or D- (except with the pull-up resistor) during the reset process. This is required so the upstream hub can drive reset downstream and be assured that the downstream device will see the reset signaling.

11.7 Hub Power Distribution Requirements

Hubs can supply a specified amount of power to downstream components and are responsible for reporting their power distribution capabilities to the host during enumeration. USB requirements stipulate that generalized legal bus topologies be supported while at the same time preventing power-up of illegal topologies. An illegal power topology is one that violates the power contract established during enumeration.

Hubs may be either locally powered or bus powered, or a combination of the two. For example, a hub may derive power for its SIE and root port pull-up resistors from the bus while obtaining power for its downstream ports from a local power supply. A hub can only supply power in a downstream direction, and must never drive power upstream. A complete discussion of hub power distribution appears in Section 7.2.

Bus powered hubs must have port power switching for its downstream ports and are required to power off all downstream ports when the hub comes out of power-up, or when it receives a reset on its root port. Ports may also be switched on and off under host software control. An implementation may provide power switching on a per port basis or have a single switch for all the ports. Port reset requests do not affect the status of the power switching for a port. A hub port must be powered on in order to perform connect detection from the downstream direction.

11.7.1 Overcurrent Indication

For reasons of safety, all locally powered hubs must implement current limiting on their downstream ports. Under no conditions may more than 25 VA be drawn from any USB hub port. (The actual overcurrent trip point may be lower than this figure). If an overcurrent condition occurs, even if it is only momentary, it must be reported to the hub controller. This is done via an overcurrent state that is reflected through hub requests. The overcurrent detect state is entered on overcurrent detect and cleared by a host request or upon reset. Detection of overcurrent must disable all affected ports. If the overcurrent condition has caused a permanent disconnect of power (such as a blown fuse), the hub must report it upon coming out of reset or power-up.

Overcurrent protection may be implemented over all downstream ports in aggregate, or on a per port basis. The ports may optionally be split into two or more subgroups, each with its own overcurrent protection circuit.

11.8 Hub Endpoint Organization

The Hub Class defines one additional endpoint beyond Endpoint 0, which is required for all devices: the Status Change endpoint. The host system receives port and hub status change notifications through the Status Change endpoint. The Status Change endpoint supports interrupt transfers. If the hub has not detected changes on any of its ports, nor any hub status changes, the hub returns a NAK to requests on the Status Change endpoint. When the hub detects any status change, the hub responds with data describing the entity that changed. Host software driving the hub is responsible for examining the data transferred to determine which entity changed. Hubs are logically organized as shown in Figure 11-17.

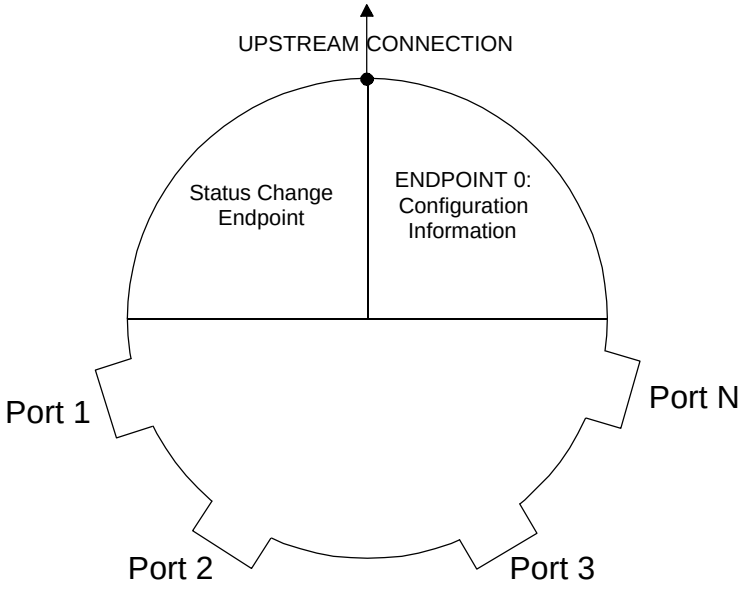


Figure 11-17. Example Hub Organization

11.8.1 Hub Information Architecture and Operation

Hub Descriptors and Hub/Port Status and Control are accessible through the default pipe. When a hub detects a change on a port or when the hub changes its own state, the Status Change endpoint transfers data to the host in the form specified in Section 11.8.3.

USB hubs detect changes in port states. Devices attached to the ports on a hub can cause various hardware events. In addition, host system software can cause changes to a hub's state by sending commands to the hub. Since there are two sources of changes to the hub, USB hubs report change information for each of the hardware-caused events. The hub continues to report a status change when polled until that particular event has been successfully acknowledged by the host. Using this reporting mechanism, system software determines what changes occurred since the last event reported by the hub. This approach makes it possible to minimize the device state information that system software must carry.

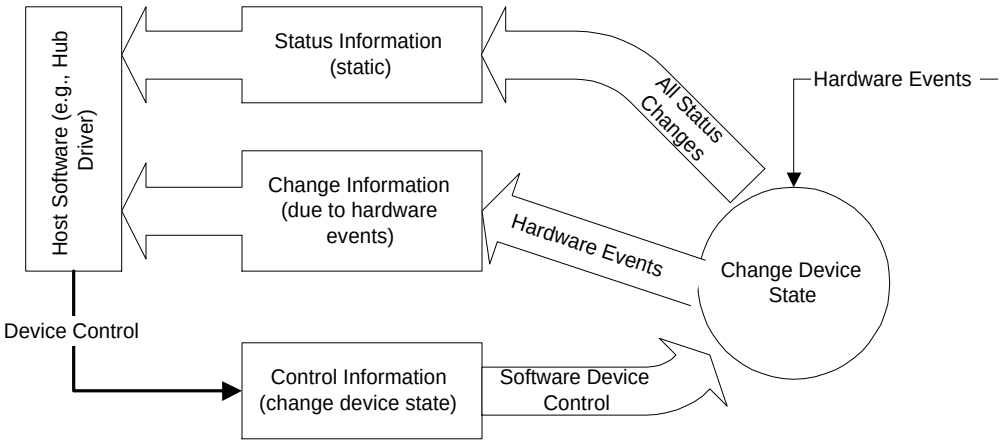


Figure 11-18. Relationship of Status, Status Change, and Control Information to Device States

Host software uses the interrupt pipe associated with the Status Change endpoint to detect changes in hub and port status.

11.8.2 Port Change Information Processing

Hubs report a port's status through port commands on a per-port basis. Host software acknowledges a port change by clearing the change state corresponding to the status change reported by the hub. The acknowledgment clears the change state for that port so future data transfers to the Status Change endpoint do not report the previous event. This allows the process to repeat for further changes (see Figure 11-19.)

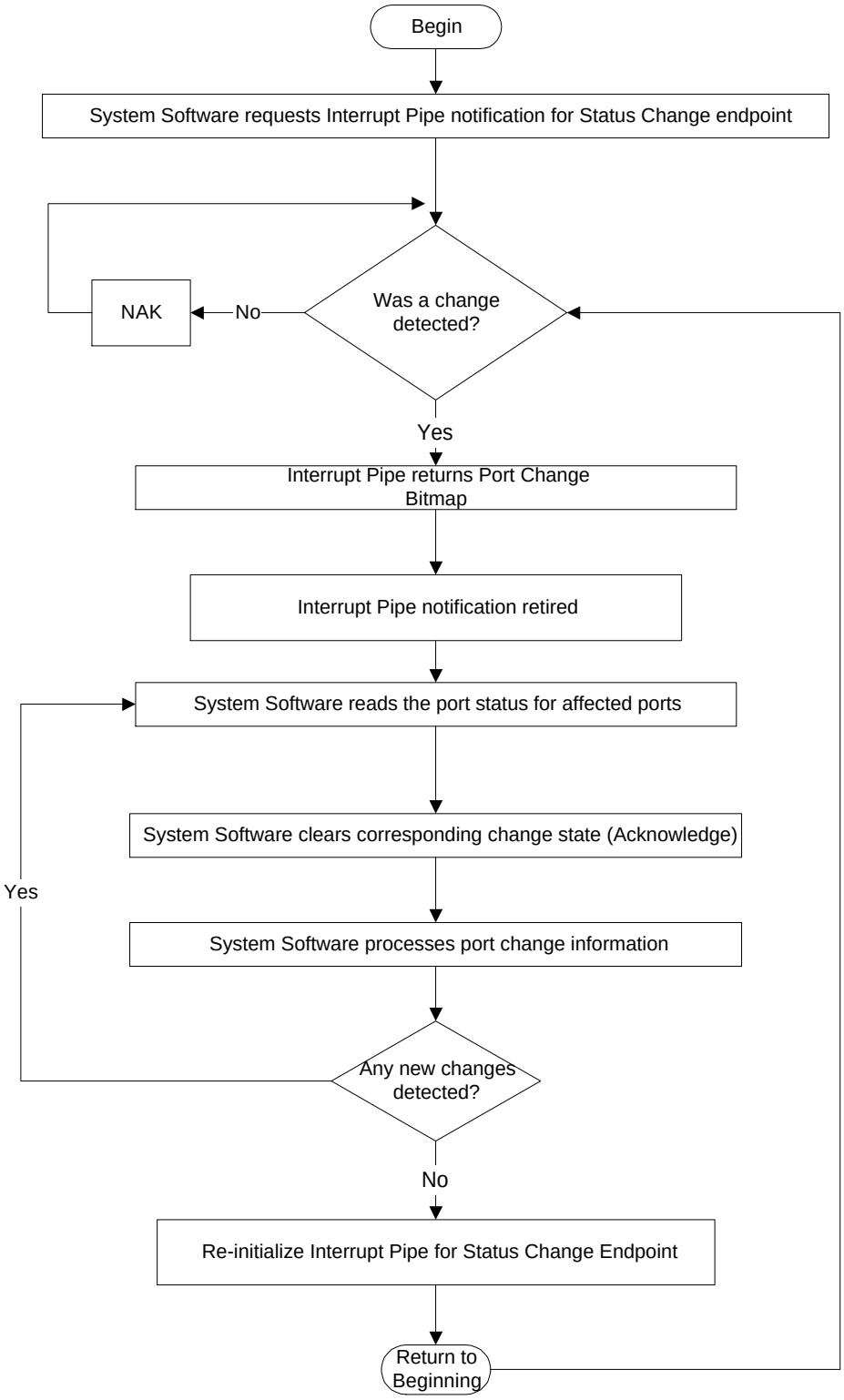


Figure 11-19. Port Status Handling Method

11.8.3 Hub and Port Status Change Bitmap

The Hub and Port Status Change Bitmap, shown in Figure 11-20, indicates whether the hub or a port has experienced a status change. This bitmap also indicates which port(s) have had a change in status. The hub returns this value on the Status Change endpoint. Hubs report this value in byte-increments. That is, if a hub has six ports, it returns a byte quantity and reports a zero in the invalid port number field locations. System software is aware of the number of ports on a hub (this is reported in the hub descriptor) and decodes the Hub and Port Status Change Bitmap accordingly. The hub reports any changes in hub status on bit 0 of the Hub and Port Status Change Bitmap.

The Hub and Port Status Change Bitmap size varies from a minimum size of one byte. Hubs only report as many bits as there are ports on the hub, subject to the byte-granularity requirement (i.e., round up to the nearest byte).

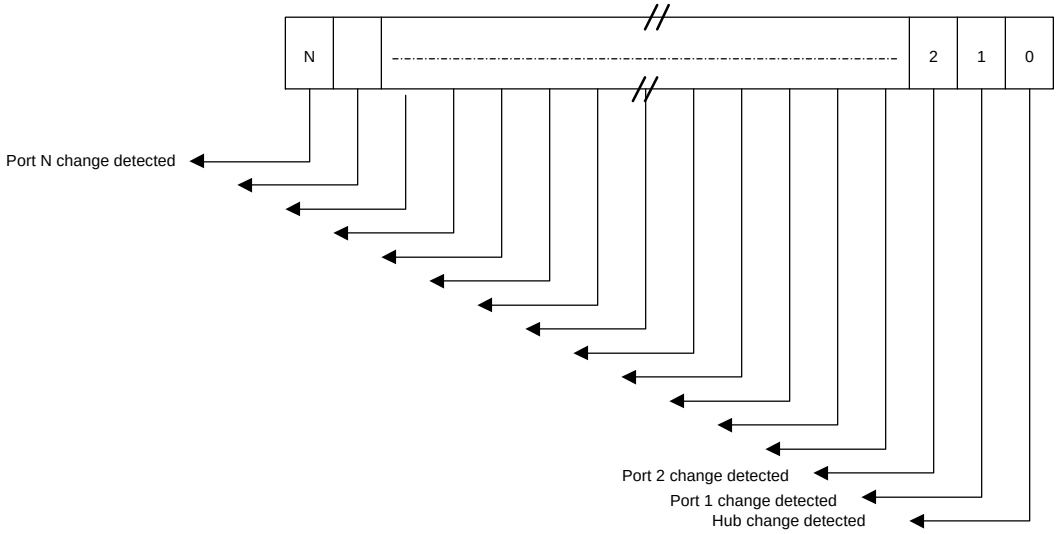


Figure 11-20. Hub and Port Status Change Bitmap

Any time the Status Change endpoint is polled by the host controller and any of the Status Changed bits are non-zero, the Hub and Port Status Change Bitmap is returned. Hubs sample the change at the End of Frame (EOF2) in preparation for a potential data transfer in the subsequent USB frame. If a change was detected, then data will be transferred through the Status Change endpoint in the subsequent USB frame. Figure 11-21 shows the sampling mechanism for hub and port change bits.

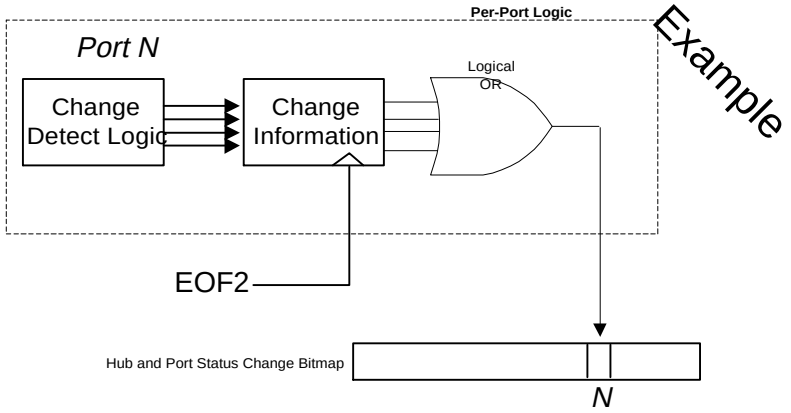


Figure 11-21. Example Hub and Port Change Bit Sampling

11.9 Hub Configuration

Hubs are configured through the standard USB device configuration commands. An unconfigured hub behaves like all other unconfigured devices with respect to power requirements and addressability. Unconfigured hubs do not turn power onto the downstream ports. Configuring a hub enables the Status Change endpoint. System software issues commands to the hub to switch port power on and off at appropriate times.

System software examines hub descriptor information before configuration to determine the hub's characteristics. By examining the hub's characteristics, system software ensures that illegal power topologies are not allowed by not powering on the hub's ports if doing so would violate the USB power topology.

11.10 Hub Port Power Control

Hubs allow port power to be controlled by the host system. As described previously, hubs support per-port or gang-mode power switching on the downstream ports. Switching port power is done via hub commands, defined below. Hubs with per-port power switching may also allow gang-mode power switching by specifying a certain value in a request field in the port power control request (refer to the SetPortFeature(PORT_POWER) request definition in Section 11.12.2.9).

Hubs may wish to mask the gang-mode power control for certain ports. This allows a hub to independently control the power switching for certain ports, regardless of the general port switching characteristics. For example, consider a hub with gang-mode port power switching that has a permanently attached device on a port (an "embedded port"). If the Port Power Control Mask field for the embedded port indicates that gang-mode power switching is masked, any hub commands that control the ports in gang-mode will not affect the embedded port.

11.11 Descriptors

Hub descriptors are derived from the general USB device framework. Hub descriptors define a hub device and the ports on that hub. The host accesses hub descriptors through the hub's default pipe.

The USB Specification (refer to Chapter 9) defines the following descriptors:

- Device
- Configuration
- Interface
- Endpoint
- String (optional)

The hub class defines additional descriptors (refer to Section 11.11.2). In addition, vendor-specific descriptors are allowed in the USB device framework. Hubs support standard USB device commands as defined in Chapter 9.

11.11.1 Standard Descriptors

The hub class pre-defines certain fields in standard USB descriptors. Other fields are either implementation-dependent or not applicable to this class.

Note: For the descriptors and fields shown below, the bits in a field are organized in a little-endian fashion; that is, bit location 0 is the least significant bit and bit location 8 is the most significant bit of a byte value.

Device Descriptor

bDeviceClass	=	HubClass
bDeviceSubClass	=	HubSubClass
wMaxPacketSize0	=	8 bytes

Interface Descriptor

bNumEndpoints	=	1
bInterface	=	1

Configuration Descriptor

MaxPower	=	The maximum amount of bus power this hub will consume in this configuration.

Endpoint Descriptor (for Status Change Endpoint)

bEndpointAddress	=	Implementation dependent
wMaxPacketSize	=	Implementation dependent
bmAttributes	=	Direction = In, Transfer Type = Interrupt (0b00000111)
bInterval	=	0xFF (Maximum allowable interval)

The hub class driver retrieves a device configuration from host system software using the GetDescriptor device request. The first endpoint descriptor returned by GetDescriptor request is, by specification, the Status Change endpoint descriptor. Hubs may define additional endpoints beyond the minimum required by this class definition. However, hubs conforming to this class standard always return the Status Change endpoint as the first endpoint descriptor in the standard interface.

11.11.2 Class-specific Descriptors

11.11.2.1 Hub Descriptor

Table 11-9. Hub Descriptor

Offset	Field	Size	Description
0	*bDescLength*	1	Number of bytes in this descriptor, including this byte.
1	*bDescriptorType*	1	Descriptor Type
2	*bNbrPorts*	1	Number of downstream ports that this hub supports.
3	*wHubCharacteristics*	2	D1..D0: Power Switching Mode 00 - Ganged power switching (all ports' power at once) 01 - Individual port power switching 1X - No power switching (ports always powered on when hub is on and off when hub is off). D2: Identifies a Compound Device 0 - Hub is not part of a compound device 1 - Hub is part of a compound device D4..D3: Over-current Protection Mode 00 - Global Over-current Protection. The hub reports over-current as a summation of all ports' current draw, without a breakdown of individual port over-current status. 01 - Individual Port Over-current Protection. The hub reports over-current on a per-port basis. Each port has an over-current indicator. 1X - No Over-Current Protection. This option is only allowed for bus-powered hubs that do not implement over-current protection. D15..D5: Reserved

Offset	Field	Size	Description
5	*bPwrOn2PwrGood*	1	Time (in 2 ms intervals) from the time power on sequence begins on a port until power is good on that port. System software uses this value to determine how long to wait before accessing a powered-on port.
6	*bHubContrCurrent*	1	Maximum current requirements of the hub controller electronics in mA.
7	*DeviceRemovable*	Variable depend-ing on number of ports on hub	Indicates if a port has a removable device attached. If a non-removable device is attached to a port, that port will never receive an insertion change notification. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, the field representing the port characteristics returns “0”. Bit definition: 0 - Device is removable 1 - Device is not removable (permanently attached) This is a bitmap corresponding to the individual ports on the hub: Bit 0: Reserved for future use Bit 1: Port 1 Bit 2: Port 2 Etc. Bit *n*: Port *n* (implementation dependent, up to a maximum of 255 ports).

Offset	Field	Size	Description
Variable	*PortPwrCtrlMask*	Variable depending on number of ports on hub	Indicates if a port is not affected by a gang-mode power control request. Ports that have this field set always require a manual SetPortFeature(PORT_POWER) request to control the port's power state. Bit definition: 0 - Port does not mask the gang-mode power control capability. 1 - Port is not affected by gang-mode power commands. Manual commands must be sent to this port to turn power on and off. This is a bitmap corresponding to the individual ports on the hub: Bit 0: Reserved for future use. Bit 1: Port 1 Bit 2: Port 2 Etc. Bit *n*: Port *n* (implementation dependent, up to a maximum of 255 ports).

11.12 Requests

11.12.1 Standard Requests

Table 11-10. Hub Responses to Standard Device Requests

bRequest	Hub Response
CLEAR_FEATURE	Standard
GET_CONFIGURATION	Standard
GET_DESCRIPTOR	Standard
GET_INTERFACE	Optional. Hubs only required to support one interface
GET_STATUS	Standard
SET_ADDRESS	Standard
SET_CONFIGURATION	Standard
SET_DESCRIPTOR	Optional
SET_FEATURE	Standard
SET_INTERFACE	Optional. Hubs only required to support one interface
SYNCH_FRAME	Optional. Hubs are not required to have isochronous endpoints.

11.12.2 Class-specific Requests

The hub class defines requests to which all hubs must respond.

Table 11-11. Hub Class Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
ClearHubFeature	00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None
ClearPortFeature	00100011B	CLEAR_ FEATURE	Feature Selector	Port	Zero	None
GetBusState	10100011B	GET_ STATE	Zero	Port	One	Per Port Bus State
GetHubDescriptor	10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
GetHubStatus	10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Indicators
GetPortStatus	10100011B	GET_ STATUS	Zero	Port	Four	Port Status and Change Indicators
SetHubDescriptor	00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
SetHubFeature	00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None
SetPortFeature	00100011B	SET_ FEATURE	Feature Selector	Port	Zero	None

Table 11-12. Hub Class Request Codes

bRequest	Value
GET_ STATUS	0
CLEAR_ FEATURE	1
GET_STATE	2
SET_ FEATURE	3
reserved for future use	*4-5*
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7

The following are the valid feature selectors for the hub class. See GetHubStatus and GetPortStatus for a description of the features.

Table 11-13. Hub Class Feature Selectors

	Recipient	Value
C_HUB_LOCAL_POWER	Hub	0
C_HUB_OVER_CURRENT	Hub	1
PORT_CONNECTION	Port	0
PORT_ENABLE	Port	1
PORT_SUSPEND	Port	2
PORT_OVER_CURRENT	Port	3
PORT_RESET	Port	4
PORT_POWER	Port	8
PORT_LOW_SPEED	Port	9
C_PORT_CONNECTION	Port	16
C_PORT_ENABLE	Port	17
C_PORT_SUSPEND	Port	18
C_PORT_OVER_CURRENT	Port	19
C_PORT_RESET	Port	20

11.12.2.1 Clear Hub Feature

This request resets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None

Clearing a feature disables that feature; refer to Table 11-13 for the feature selector definitions. If the feature selector is associated with a change indicator, clearing that indicator acknowledges the change. Both C_HUB_LOCAL_POWER and C_HUB_OVER_CURRENT may be acknowledged using this request.

11.12.2.2 Clear Port Feature

This request resets a value reported in the port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100011B	CLEAR_ FEATURE	Feature Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero.

Clearing a feature disables that feature; refer to Table 11-13 for the feature selector definitions. If the feature selector is associated with a change indicator, clearing that indicator acknowledges the change. Changes in connection, enable, suspend, reset, and over-current status are acknowledged using this request.

Clearing the PORT_SUSPEND feature causes a host-initiated resume on the specified port. Clearing the PORT_ENABLE feature causes the port to be disabled. Clearing the PORT_POWER feature causes the port to be powered off, subject to the constraints due to the hub's method of power switching. If a hub uses gang power switching, all ports must be requested to power off before any of the ports actually power off.

11.12.2.3 Get Bus State

This is an optional per-port diagnostic request which reads the bus state value, as sampled at the last EOF2.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_ STATE	Zero	Port	One	Per Port Bus State

The port number must be a valid port number for that hub, greater than zero.

Hubs may implement an optional diagnostic aid to facilitate system debug. Hubs implement this aid through this optional request. This diagnostic feature provides a glimpse of the USB bus state as sampled at the last EOF2 sample point.

Hubs that implement this diagnostic feature should store the bus state at each EOF2 state, in preparation for a potential request in the following USB frame.

The data returned is bit-mapped in the following manner. The value of the D- signal is returned in the field in bit 0. The value of the D+ signal is returned in the field in bit 1. Bits 2-7 are reserved for future use and are reset to zero.

Hubs that do not support this request respond with a stall.

11.12.2.4 Get Hub Descriptor

This request returns the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The GetDescriptor request for the hub class descriptor follows the same usage model as that of the standard GetDescriptor request (refer to Chapter 9). The standard hub descriptor is denoted by descriptor type zero. All hubs are required to implement one hub descriptor, with descriptor index zero.

11.12.2.5 Get Hub Status

This request returns the current hub status and the states that have changed since the previous acknowledgment.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Indicators

The first word of data contains *wHubStatus* (refer to Table 11-14). The second word of data contains *wHubChange* (refer to Table 11-15).

The fields returned are organized in such a way to allow system software to determine which states have changed. The bit locations in the *wHubStatus* and *wHubChange* fields correspond in a one-to-one fashion where applicable.

Local power and overcurrent changes are acknowledged using the ClearHubFeature request.

Table 11-14. Hub Status Field, *wHubStatus*

BIT	DESCRIPTION
0	**Local Power Status:** This is the state of the local power supply. This field only applies to self-powered hubs whose USB Interface Engine (SIE) is bus-powered or hubs that support either self-powered or bus-powered configurations. This field is returned as a result of a change to the hub's power source. This field reports whether local power is good. This field allows system software to determine the reason for the removal of power to devices attached to this hub or to react to changes to the local power supply state. If the hub does not support this feature, this field is RESERVED and follows the definition of the RESERVED bits below. This field reports the power status for the SIE and the remainder of the hub. 0 = Local power supply good 1 = Local power supply lost (inactive)
1	**Over-Current Indicator:** This field only applies to hubs that report over-current conditions on a global hub basis (as reported in the **Hub Descriptor**). If the hub does not report over-current on a global hub basis, this field is RESERVED and follows the definition of the RESERVED bits below. This field indicates that the sum of all the ports' current has exceeded the specified maximum and power to all the ports has been shut off. For more details on Over-Current protection, see Section 7.2.1.2.1. This field indicates an over-current condition due to the sum of all ports' current consumption. 0 = All power operations normal 1 = An over-current condition exists on a hub-wide basis
2-15	**Reserved** These bits return 0 when read.

Table 11-15. Hub Change Field, *wHubChange*

BIT	DESCRIPTION
0	**Local Power Status Change:** (C_HUB_LOCAL_POWER) This corresponds to Local Power Status, Bit 0 above. This field only applies to locally-powered (i.e., self-powered) hubs whose USB Interface Engine (SIE) is bus-powered, or hubs that support either self-powered or bus-powered configurations. This field is returned as a result of a change to the hub's power source. If the hub does not support this feature, then this field is RESERVED and follows the definition of the RESERVED bits below. This field reports whether a change has occurred to the local power status. 0 = No change has occurred on Local Power Status 1 = Local Power Status has changed
1	**Over-Current Indicator Change**: (C_HUB_OVER_CURRENT) This corresponds to Over-Current Indicator, Bit 1 above. This field only applies to hubs that report over-current conditions on a global hub basis (as reported in the Hub Descriptor). If the hub does not report over-current on a global hub basis, this field is RESERVED and follows the definition of the RESERVED bits below. This field reports whether a change has occurred to the Over-Current Indicator. This field is only set if an over-current condition has occurred (i.e., acknowledgment of this change by system software will not cause another change to be reported). 0 = No change has occurred on the Over-Current Indicator 1 = Over-Current Indicator has changed (i.e., over-current condition has occurred).
2-15	**Reserved** These bits return 0 when read.

11.12.2.6 Get Port Status

This request returns the current port status and the states that have changed since the previous acknowledgment.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_STATUS	Zero	Port	Four	Port Status and Change Indicators

The port number must be a valid port number for that hub, greater than zero.

The first word of data contains *wPortStatus* (refer to Table 11-16). The second word of data contains *wPortChange* (refer to Table 11-17).

The fields returned are organized in such a way to allow system software to determine which states have changed. The bit locations in the *wPortStatus* and *wPortChange* fields correspond in a one-to-one fashion where applicable.

Table 11-16. Port Status Field, *wPortStatus*

BIT	DESCRIPTION
0	**Current Connect Status:** (PORT_CONNECTION) This field reflects whether or not a device is currently connected to this port. This value reflects the current state of the port, and may not correspond directly to the event that caused the Insertion Status Change (Bit 0 in below) to be set. 0 = No device is present on this port 1 = A device is present on this port **NOTE: This field is always 1 for ports that have non-removable devices attached.**
1	**Port Enabled/Disabled:** (PORT_ENABLE) Ports can be enabled by host software only. Ports can be disabled by either a fault condition (disconnect event or other fault condition, including an over-current indication) or by host software. 0 = Port is disabled 1 = Port is enabled
2	**Suspend:** (PORT_SUSPEND) This field indicates whether or not the device on this port is suspended. Setting this field causes the device to suspend by not propagating bus traffic downstream. Resetting this field causes the device to resume. Bus traffic cannot be resumed in the middle of a bus transaction If the device itself is signaling a resume, this field will be cleared by the hub. 0 = Not suspended 1 = Suspended
3	**Over-Current Indicator:** (PORT_OVER_CURRENT) This field only applies to hubs that report over-current conditions on a per-port basis (as reported in the Hub Descriptor). If the hub does not report over-current on a per-port hub basis, this field is RESERVED and follows the definition of the RESERVED bits below. This field indicates that the device attached to this port has drawn current that exceeds the specified maximum and this port's power has been shut off. Port power shutdown is also reflected in the Port Power field above. For more details, see Section 7.2.1.2.1. This field indicates an over-current condition due to the device attached to this port. 0 = All power operations normal for this port. 1 = An over-current condition exists on this port. Power has been shut off to this port.
4	**Reset**: (PORT_RESET) This field is set when the host wishes to reset the attached device. It remains set until the reset signaling is turned off by the hub and the reset status change field is set. 0 = Reset signaling not asserted 1 = Reset signaling asserted
5-7	**Reserved** These bits return a "0" when read.
8	**Port Power:** (PORT_POWER) This field reflects a port's power state. Since hubs can implement different methods of port power switching, the meaning of this field varies depending on the type of power switching used. The device descriptor reports the type of power switching implemented by the hub. Hubs do not provide any power to their ports until they are in the configured state. 0 = This port is powered OFF 1 = This port is powered ON **NOTE: Hubs that do not support power switching always return a 1 in this field.**
9	**Low Speed Device Attached:** (PORT_LOW_SPEED) This is only relevant if a device is attached. 0 = Full Speed device attached to this port 1 = Low speed device attached to this port
10-15	**Reserved** These bits return 0 when read.

Table 11-17. Port Change Field, *wPortChange*

BIT	DESCRIPTION
0	**Connect Status Change:** (C_PORT_CONNECTION) Indicates a change has occurred in the port's Current Connect Status. The hub device sets this field for any changes to the port device connect status, even if system software has not cleared a connect status change.[1] 0 = No change has occurred on Current Connect Status 1 = Current Connect Status has changed **NOTE: For ports that have non-removable devices attached, this field is set only after a RESET condition to indicate to system software that a device is present on this port.**
1	**Port Enable/Disable Change:** (C_PORT_ENABLE) This field is only activated when a change in the port's enable/disable status was detected due to hardware changes. This field is not set if system software caused a port enable/disable change. 0 = No change has occurred on Port Enabled/Disabled status 1 = Port Enabled/Disabled status has changed
2	**Suspend Change:** (C_PORT_SUSPEND) This field indicates a change in the host-visible power state of the attached device. It indicates the device has transitioned out of the suspend state. Going into the suspend state will not set this field. The Suspend Change field is only set when the entire resume process has completed. That is, the hub has ceased signaling resume on this port and 3 ms have passed to allow the device to resynch to SOF. 0 = No change 1 = Resume complete
3	**Over-Current Indicator Change:** (C_PORT_OVER_CURRENT) This field only applies to hubs that report over-current conditions on a per-port basis (as reported in the Hub Descriptor). If the hub does not report over-current on a per-port hub basis, then this field is RESERVED and follows the definition of the RESERVED bits below. This field reports whether a change has occurred to the port Over-Current Indicator. 0 = No change has occurred on Over-Current Indicator 1 = Over-Current Indicator has changed
4	**Reset Change:** (C_PORT_RESET) This field is set when reset processing on this port is complete. As a reset of completing reset processing, the enabled status of the port is also set and the suspend change field reset. 0 = No change 1 = Reset Complete
5-15	**Reserved** These bits return 0 when read.

[1] If, for example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be "setting" an already-set bit (i.e., the bit will remain set). However, the hub will transfer the change bit only once when the host controller requests a data transfer to the Status Change endpoint. System software is responsible for determining state change history in such a case.

11.12.2.7 Set Descriptor

This request overwrites the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The SetDescriptor request for the hub class descriptor follows the same usage model as that of the standard SetDescriptor request (refer to Chapter 9). The standard hub descriptor is denoted by descriptor type zero. All hubs are required to implement one hub descriptor, with descriptor index zero.

This request is optional. This request writes data to a class-specific descriptor. The host provides the data that is to be transferred to the hub during the data transfer phase of the control transaction. This request writes the entire hub descriptor at once.

Hubs must buffer all the bytes received from this request to ensure that the entire descriptor has been successfully transmitted from the host. Upon successful completion of the bus transfer, the hub updates the contents of the specified descriptor.

Hubs that do not support this request respond with a stall.

11.12.2.8 Set Hub Feature

This request sets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None

Setting a feature enables that feature; refer to Table 11-13 for the feature selector definitions. Change indicators may not be acknowledged using this request.

11.12.2.9 Set Port Feature

This request sets a value reported in the port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	SET_ FEATURE	Feature Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero.

Setting a feature enables that feature; see for the feature selector definitions. Change indicators may not be acknowledged using this request.

Setting the PORT_SUSPEND feature causes bus traffic to cease on that port and, consequently, the device to suspend. Setting the reset feature (PORT_RESET) causes the hub to signal reset on that port. When the reset signaling is complete, the hub sets the C_PORT_RESET change indicator and immediately enables the port. Refer to Section 11.6.2 for a complete discussion of host initiated reset behavior.