Figures

Tables

Chapter 1
Introduction

1.1 Motivation

The original motivation for the Universal Serial Bus (USB) came 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 USB 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 have made computers less confrontational and easier to reconfigure. However, from the end user's 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 more recent motivation for USB 2.0 stems from the fact that PCs have increasingly higher performance and are capable of processing vast amounts of data. At the same time, PC peripherals have added more performance and functionality. User applications such as digital imaging demand a high performance connection between the PC and these increasingly sophisticated peripherals. USB 2.0 addresses this need by adding a third transfer rate of 480 Mb/s to the 12 Mb/s and 1.5 Mb/s originally defined for USB. USB 2.0 is a natural evolution of USB, delivering the desired bandwidth increase while preserving the original motivations for USB and maintaining full compatibility with existing peripherals.

Thus, USB continues to be 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 USB. 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 interoperate in an open architecture. The specification is intended as an enhancement to the PC architecture, spanning portable, business desktop, 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

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. This specification can be used for developing new products and associated software.

1.4 USB Product Compliance

Adopters of the USB 2.0 specification have signed the USB 2.0 Adopters Agreement, which provides them access to a reciprocal royalty-free license from the Promoters and other Adopters to certain intellectual property contained in products that are compliant with the USB 2.0 specification. Adopters can demonstrate compliance with the specification through the testing program as defined by the USB Implementers Forum. Products that demonstrate compliance with the specification will be granted certain rights to use the USB Implementers Forum logo as defined in the logo license.

1.5 Document Organization

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

- Peripheral implementers should particularly read Chapters 5 through 11.
- USB Host Controller implementers should particularly read Chapters 5 through 8, 10, and 11.
- USB device driver implementers should particularly read Chapters 5, 9, and 10.

This document is complemented and referenced by the *Universal Serial Bus Device Class Specifications.* Device class specifications exist for a wide variety of devices. 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.

ACK	Handshake packet indicating a positive acknowledgment.
Active Device	A device that is powered and is not in the Suspend state.
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., there is no shared master clock). See also rate adaptation.
Asynchronous SRC	The incoming sample rate, Fs_i, and outgoing sample rate, Fs_o, of the SRC process are independent (i.e., there is no shared master clock). See also sample rate conversion.
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 (micro)frame.
Bandwidth	The amount of data transmitted per unit of time, typically bits per second (b/s) or bytes per second (B/s).
Big Endian	A method of storing data that places the most significant byte of multiple-byte values at a lower storage address. For example, a 16-bit integer stored in big endian format places the least significant byte at the higher address and the most significant byte at the lower address. See also 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.
b/s	Transmission rate expressed in bits per second.
B/s	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	One of the four USB transfer types. Bulk transfers are non-periodic, large bursty communication typically used for a transfer that can use any available bandwidth and can also be delayed until bandwidth is available. See also transfer type.
Bus Enumeration	Detecting and identifying USB devices.

Byte	A data element that is eight bits in size.
Capabilities	Those attributes of a USB device that are administrated by the host.
Characteristics	Those qualities of a USB device that are unchangeable; for example, the device class is a device characteristic.
Client	Software resident on the host that interacts with the USB System Software to arrange data transfer between a function and the host. The client is often the data provider and consumer for transferred data.
Configuring Software	Software resident on the host software that is responsible for configuring a USB device. This may be a system configurator or software specific to the device.
Control Endpoint	A pair of device endpoints with the same endpoint number that are used by a control pipe. Control endpoints transfer data in both directions and, therefore, use both endpoint directions of a device address and endpoint number combination. Thus, each control endpoint consumes two endpoint addresses.
Control Pipe	Same as a message pipe.
Control Transfer	One of the four USB transfer types. Control transfers support configuration/command/status type communications between client and function. See also transfer type.
CRC	See Cyclic Redundancy Check.
CTI	Computer Telephony Integration.
Cyclic Redundancy Check (CRC)	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 USB Specification and used by a USB device when it is first powered or reset. The default address is 00H.
Default Pipe	The message pipe created by the USB System Software to pass control and status information between the host and a USB device's endpoint zero.
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 USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical. When used as a non-specific reference, a USB device is either a hub or a function.
Device Address	A seven-bit value representing the address of a device on the USB. The device address is the default address (00H) when the USB device is first powered or the device is reset. Devices are assigned a unique device address by the USB System Software.

Device Endpoint	A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device. See also endpoint address.
Device Resources	Resources provided by USB devices, such as buffer space and endpoints. See also Host Resources and Universal Serial Bus Resources.
Device Software	Software that is responsible for using a USB device. This software may or may not also be responsible for configuring the device for use.
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 two words (i.e., four 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 Electrically Erasable Programmable Read Only Memory.
EEPROM	See Electrically Erasable Programmable Read Only Memory.
Electrically Erasable Programmable Read Only Memory (EEPROM)	Non-volatile rewritable memory storage technology.
End User	The user of a host.
Endpoint	See device endpoint.
Endpoint Address	The combination of an endpoint number and an endpoint direction on a USB device. Each endpoint address supports data transfer in one direction.
Endpoint Direction	The direction of data transfer on the USB. The direction can be either IN or OUT. IN refers to transfers to the host; OUT refers to transfers from the host.
Endpoint Number	A four-bit value between 0H and FH, inclusive, associated with an endpoint on a USB device.
Envelope detector	An electronic circuit inside a USB device that monitors the USB data lines and detects certain voltage related signal characteristics.
EOF	End-of-(micro)Frame.
EOP	End-of-Packet.
External Port	See port.
Eye pattern	A representation of USB signaling that provides minimum and maximum voltage levels as well as signal jitter.
False EOP	A spurious, usually noise-induced event that is interpreted by a packet receiver as an EOP.

Frame	A 1 millisecond time base established on full-/low-speed buses.
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.
Fs	See sample rate.
Full-duplex	Computer data transmission occurring in both directions simultaneously.
Full-speed	USB operation at 12 Mb/s. See also low-speed and high-speed.
Function	A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.
Handshake Packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NAK.
High-bandwidth endpoint	A high-speed device endpoint that transfers more than 1024 bytes and less than 3073 bytes per microframe.
High-speed	USB operation at 480 Mb/s. See also low-speed and full-speed.
Host	The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
Host Controller	The host's USB interface.
Host Controller Driver (HCD)	The USB software layer that abstracts the Host Controller hardware. The Host Controller Driver provides an SPI for interaction with a Host Controller. The 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 also Device Resources and Universal Serial Bus Resources.
Hub	A USB device that provides additional connections to the USB.
Hub Tier	One plus the number of USB links in a communication path between the host and a function. See Figure 4-1.
Interrupt Request (IRQ)	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 that caused the request.
Interrupt Transfer	One of the four USB transfer types. Interrupt transfer characteristics are small data, non-periodic, low-frequency, and bounded-latency. Interrupt transfers are typically used to handle service needs. See also transfer type.
I/O Request Packet	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.
IRP	See I/O Request Packet.
IRQ	See Interrupt Request.
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 that is sent by the host.
Isochronous Source Endpoint	An endpoint that is capable of producing an isochronous data stream and sending it to the host.
Isochronous Transfer	One of the four USB transfer types. Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device. See also transfer type.
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.
kb/s	Transmission rate expressed in kilobits per second.
kB/s	Transmission rate expressed in kilobytes per second.
Little Endian	Method of storing data that places the least significant byte of multiple-byte values at lower storage addresses. For example, a 16-bit integer stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address. See also big endian.
LOA	Loss of bus activity characterized by an SOP without a corresponding EOP.
Low-speed	USB operation at 1.5 Mb/s. See also full-speed and high-speed.
LSb	Least significant bit.
LSB	Least significant byte.
Mb/s	Transmission rate expressed in megabits per second.
MB/s	Transmission rate expressed in megabytes per second.
Message Pipe	A bi-directional pipe that transfers data using a request/data/status paradigm. The data has an imposed structure that allows requests to be reliably identified and communicated.
Microframe	A 125 microsecond time base established on high-speed buses.
MSb	Most significant bit.
MSB	Most significant byte.
NAK	Handshake packet indicating a negative acknowledgment.
Non Return to Zero Invert (NRZI)	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 USB 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 USB device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.

Packet ID (PID)	A field in a USB 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.
Phase	A token, data, or handshake packet. A transaction has three phases.
Phase Locked Loop (PLL)	A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
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). See also stream pipe and message pipe.
PLL	See Phase Locked Loop.
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 the USB, the point where a USB device is attached.
Power On Reset (POR)	Restoring a storage device, register, or memory to a predetermined state when power is applied.
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 USB 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 USB hub directly attached to the Host Controller. This hub (tier 1) is attached to the host.
Root Port	The downstream port on a Root 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 (Hz).

Sample Rate Conversion (SRC)	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
Service	A procedure provided by a System Programming Interface (SPI).
Service Interval	The period between consecutive requests to a USB 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.
SOF	See Start-of-Frame.
SOP	Start-of-Packet.
SPI	See System Programming Interface.
Split transaction	A transaction type supported by host controllers and hubs. This transaction type allows full- and low-speed devices to be attached to hubs operating at high-speed.
SRC	See Sample Rate Conversion.
Stage	One part of the sequence composing a control transfer; stages include the Setup stage, the Data stage, and the Status stage.
Start-of-Frame (SOF)	The first transaction in each (micro)frame. An SOF allows endpoints to identify the start of the (micro)frame and synchronize internal endpoint clocks to the host.
Stream Pipe	A pipe that transfers data as a stream of samples with no defined USB 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 (SPI)	A defined interface to services provided by system software.
TDM	See Time Division Multiplexing.
TDR	See Time Domain Reflectometer.
Termination	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
Time Division Multiplexing (TDM)	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 Domain Reflectometer (TDR)	An instrument capable of measuring impedance characteristics of the USB signal lines.

Timeout	The detection of a lack of bus activity for some predetermined interval.
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.
Transaction translator	A functional component of a USB hub. The Transaction Translator responds to special high-speed transactions and translates them to full/low-speed transactions with full/low-speed devices attached on downstream facing ports.
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 standard transfer types are defined: control, interrupt, bulk, and isochronous.
Turn-around Time	The time a device needs to wait to begin transmitting a packet after a packet has been received to prevent collisions on the USB. 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 the USB.
Universal Serial Bus Driver (USBD)	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.
Universal Serial Bus Resources	Resources provided by the USB, such as bandwidth and power. See also Device Resources and Host Resources.
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.
USBD	See Universal Serial Bus Driver.
USB-IF	USB Implementers Forum, Inc. is a nonprofit corporation formed to facilitate the development of USB compliant products and promote the technology.
Virtual Device	A device that is represented by a software interface layer. An example of a virtual device is a hard disk with its associated device driver and client software that makes it able to reproduce an audio .WAV file.
Word	A data element that is two bytes (16 bits) in size.

Chapter 3
Background

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

3.1 Goals for the Universal Serial Bus

The USB is specified to be an industry-standard extension to the PC architecture with a focus on PC peripherals that enable consumer and business applications. The following criteria were applied in defining the architecture for the USB:

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 480 Mb/s
- Full support for real-time data for voice, audio, and video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehension of various PC configurations and form factors
- Provision of a standard interface capable of quick diffusion into product
- Enabling new classes of devices that augment the PC's capability
- Full backward compatibility of USB 2.0 for devices built to previous versions of the specification

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 USB. As can be seen, a 480 Mb/s bus comprehends the high-speed, full-speed, and low-speed data ranges. Typically, high-speed and full-speed data types may be isochronous, while low-speed data comes from interactive devices. The USB is primarily a PC bus but can be readily applied to other host-centric computing devices. The software architecture allows for future extension of the USB by providing support for multiple USB Host Controllers.

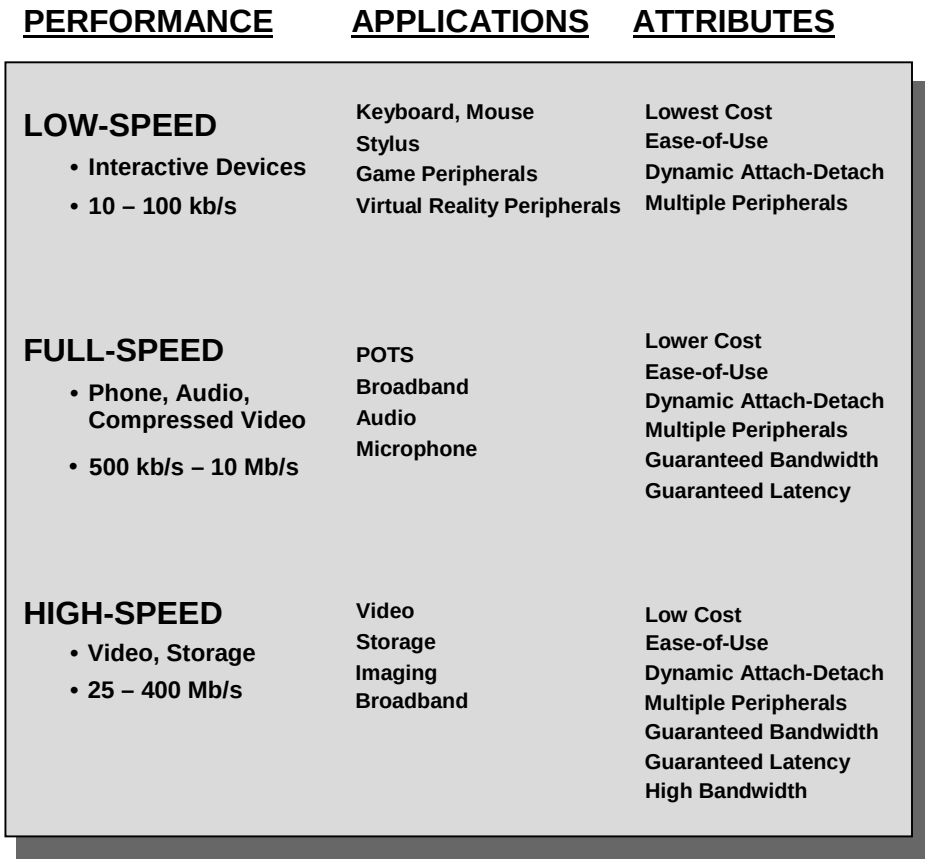

Figure 3-1. Application Space Taxonomy

3.3 Feature List

The USB 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 the following benefits:

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 kb/s to several hundred Mb/s
- Supports isochronous as well as asynchronous transfer types over the same set of wires
- Supports concurrent operation of many devices (multiple connections)
- Supports up to 127 physical devices
- Supports transfer of multiple data and message streams between the host and devices
- Allows compound devices (i.e., peripherals composed of many functions)
- Lower protocol overhead, resulting in high bus utilization

Isochronous bandwidth

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, video, etc.

Flexibility

- Supports a wide range of packet sizes, which allows a range of device buffering options
- Allows a wide range of device data rates by accommodating packet buffer size and latencies
- Flow control for buffer handling is built into the protocol

Robustness

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

Synergy with PC industry

- Protocol is simple to implement and integrate
- Consistent with the PC plug-and-play architecture
- Leverages existing operating system interfaces

Low-cost implementation

- Low-cost subchannel at 1.5 Mb/s
- Optimized for integration in peripheral and host hardware
- Suitable for development of low-cost peripherals
- Low-cost cables and connectors
- Uses commodity technologies

Upgrade path

- Architecture upgradeable to support multiple USB Host Controllers in a system

Chapter 4
Architectural Overview

This chapter presents an overview of the Universal Serial Bus (USB) architecture and key concepts. The 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.

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 the following:

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

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

4.1.1 Bus Topology

The USB 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.

Due to timing constraints allowed for hub and cable propagation times, the maximum number of tiers allowed is seven (including the root tier). Note that in seven tiers, five non-root hubs maximum can be supported in a communication path between the host and any device. A compound device (see Figure 4-1) occupies two tiers; therefore, it cannot be enabled if attached at tier level seven. Only functions can be enabled in tier seven.

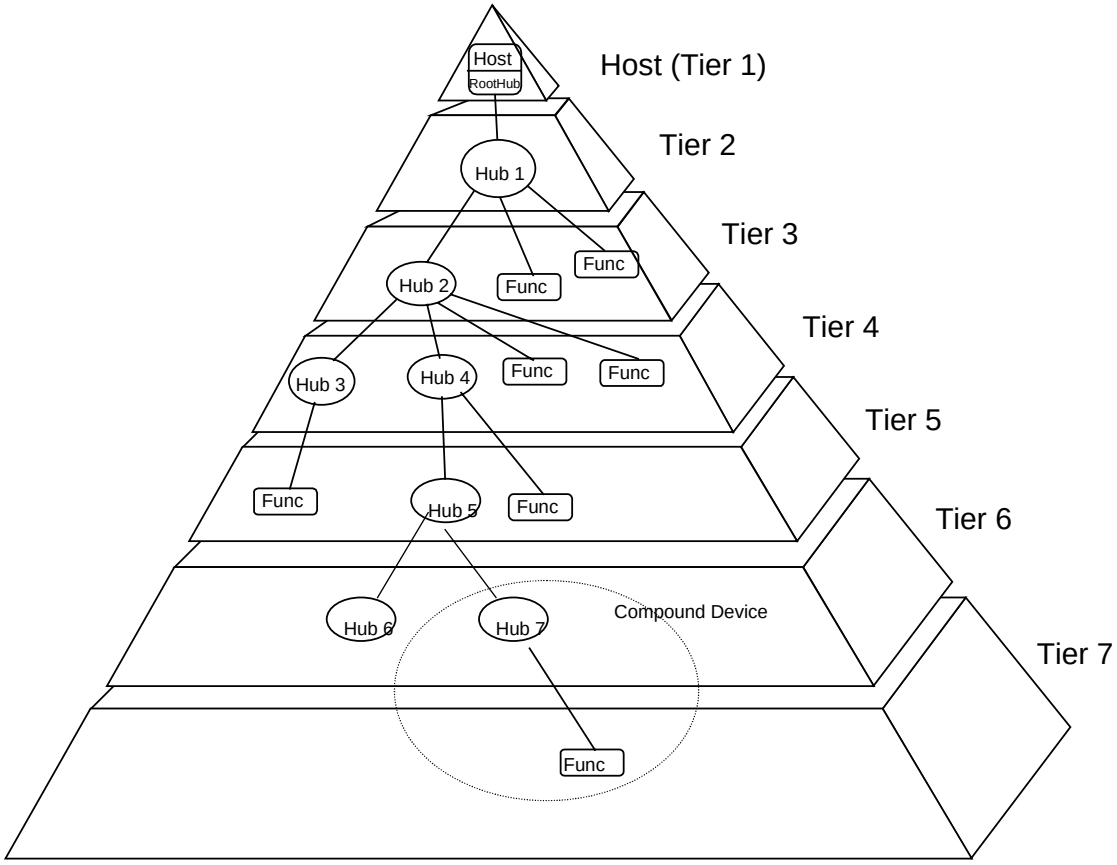

Figure 4-1. Bus Topology

4.1.1.1 USB Host

There is only one host in 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.

4.1.1.2 USB Devices

USB devices are one of the following:

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

USB devices present a standard USB interface in terms of the following:

- Their comprehension of the USB protocol
- Their response to standard USB operations, such as configuration and reset
- Their standard capability descriptive information

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

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

The USB transfers signal and power over a four-wire cable, shown in Figure 4-2. The signaling occurs over two wires on each point-to-point segment.

There are three data rates:

- The USB high-speed signaling bit rate is 480 Mb/s.
- The USB full-speed signaling bit rate is 12 Mb/s.
- A limited capability low-speed signaling mode is also defined at 1.5 Mb/s.

USB 2.0 host controllers and hubs provide capabilities so that full-speed and low-speed data can be transmitted at high-speed between the host controller and the hub, but transmitted between the hub and the device at full-speed or low-speed. This capability minimizes the impact that full-speed and low-speed devices have upon the bandwidth available for high-speed devices.

The low-speed mode is defined to support a limited number of low-bandwidth devices, such as mice, because more general use would degrade 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.

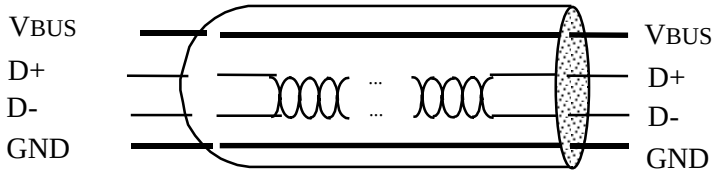

Figure 4-2. USB Cable

The cable also carries VBUS and GND wires on each segment to deliver power to devices. VBUS is nominally +5 V at the source. The 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 high/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 the USB System 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.

4.3.2 Power Management

A USB host may have a power management system that is independent of the USB. The USB System Software interacts with the host's power management system to handle system power events such as suspend or resume. Additionally, USB devices typically implement additional power management features that allow them to be power managed by system software.

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

4.4 Bus Protocol

The USB is a polled bus. The Host Controller initiates all data transfers.

Most 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.

Some bus transactions between host controllers and hubs involve the transmission of four packets. These types of transactions are used to manage the data transfers between the host and full-/low- speed devices.

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. Most pipes come into existence when a USB device is configured. One message pipe, the Default Control Pipe, 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 pipes. At the hardware level, this prevents buffers from underrun or overrun situations by using a NAK handshake to throttle the data rate. When NAKed, a transaction is retried 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 pipes. Thus, multiple stream 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 timeouts for lost or corrupted 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 packet 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 allows for error handling in hardware or software. Hardware error handling includes reporting and retry of failed transfers. A USB Host Controller will try a transmission that encounters errors up to three times before informing the client software of the failure. 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 time. Consequently, system software must accommodate dynamic changes in the physical bus topology.

4.6.1 Attachment of USB Devices

All USB devices attach to the USB through ports on specialized USB devices known as hubs. Hubs have status bits that are used to report the attachment or removal of a USB device on one of its ports. The host queries the hub to retrieve these bits. In the case of an attachment, the host enables the port and addresses the USB device through the device's control pipe at the default address.

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

If the attached 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 handled by host software that is appropriate for the function.

4.6.2 Removal of USB Devices

When a USB device has been removed from one of a hub's ports, the hub disables the port and provides an indication of device removal to the host. The removal indication is then handled by appropriate USB System Software. If the removed USB device is a hub, the USB System Software must handle the removal of both the hub and of all of the USB devices that were previously attached to the system through the hub.

4.6.3 Bus Enumeration

Bus enumeration is the activity that identifies and assigns unique addresses to devices attached to a bus. Because the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration is an on-going activity for the USB System Software. Additionally, bus enumeration for the USB also includes the detection and processing of removals.

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-directional or bi-directional pipes. USB data transfers take place between host software and a particular endpoint on a USB device. Such associations between the host software and a USB device endpoint are called pipes. In general, data movement though one pipe is independent from the data flow in any other pipe. A given USB device may have many pipes. As an example, a given USB device could have an endpoint that supports a pipe for transporting data to the USB device and another endpoint that supports a pipe for transporting data from the USB device.

The USB architecture comprehends four basic types of data transfers:

- Control Transfers: Used to configure a device at attach time and can be used for other device-specific purposes, including control of other pipes on the device.
- Bulk Data Transfers: Generated or consumed in relatively large and bursty quantities and have wide dynamic latitude in transmission constraints.
- Interrupt Data Transfers: Used for timely but reliable delivery of data, for example, characters or coordinates with human-perceptible echo or feedback response characteristics.
- Isochronous Data Transfers: Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers).

A pipe supports only one of the types of transfers described above for any given device configuration. The USB data flow model is described in more detail in Chapter 5.

4.7.1 Control Transfers

Control data is used by the USB System Software to configure devices when they are first attached. Other driver software can choose to use 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 invoking a limited number of retries in hardware. Also, the bandwidth taken up by bulk data can vary, depending on other bus activities.

4.7.3 Interrupt Transfers

A limited-latency transfer to or 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 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 that 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, drop-outs in the data stream will occur due to buffer or frame underruns or overruns. Even if data is delivered at the appropriate rate by USB hardware, delivery delays introduced by software 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 buffers. The goal for the USB architecture is to ensure that buffering-induced hardware delay is bounded to within a few milliseconds.

The 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. Further, different device bit rates, with a wide dynamic range, can be concurrently supported.

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, human interface, printer, imaging, or mass storage device. The hub device class indicates a specially designated USB device that 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 USB address that is assigned when the device is attached and enumerated. Each USB device additionally supports one or more pipes through which the host may communicate with the device. All USB devices must support a specially designated pipe at endpoint zero to which the USB device's USB control pipe will be attached. All USB devices support a common access mechanism for accessing information through this control pipe.

Associated with the control pipe at endpoint zero 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 capability. 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.

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 the USB. Figure 4-3 shows a typical hub. Hubs serve to simplify USB connectivity from the user's perspective and provide robustness at relatively low cost and complexity.

Hubs are wiring concentrators and enable the multiple attachment characteristics of the 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 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 attached to either high-, full- or low-speed devices.

A USB 2.0 hub consists of three portions: the Hub Controller, the Hub Repeater, and the Transaction Translator. The Hub 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 Host Controller provides the 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. The Transaction Translator provides the mechanisms that support full-/low-speed devices behind the hub, while transmitting all device data between the host and the hub at high-speed.

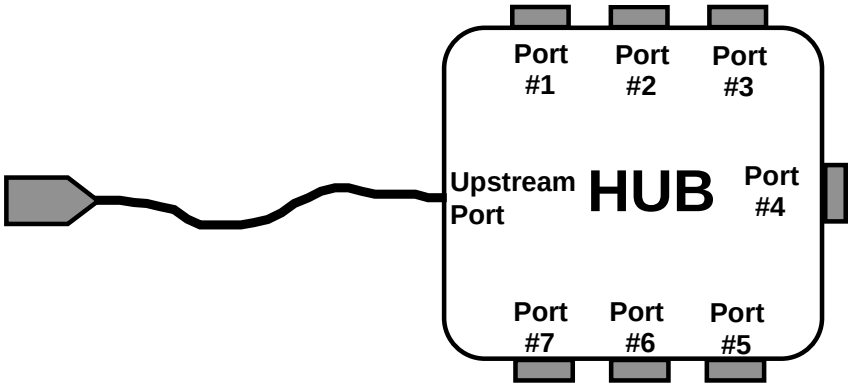

Figure 4-3. A Typical Hub

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

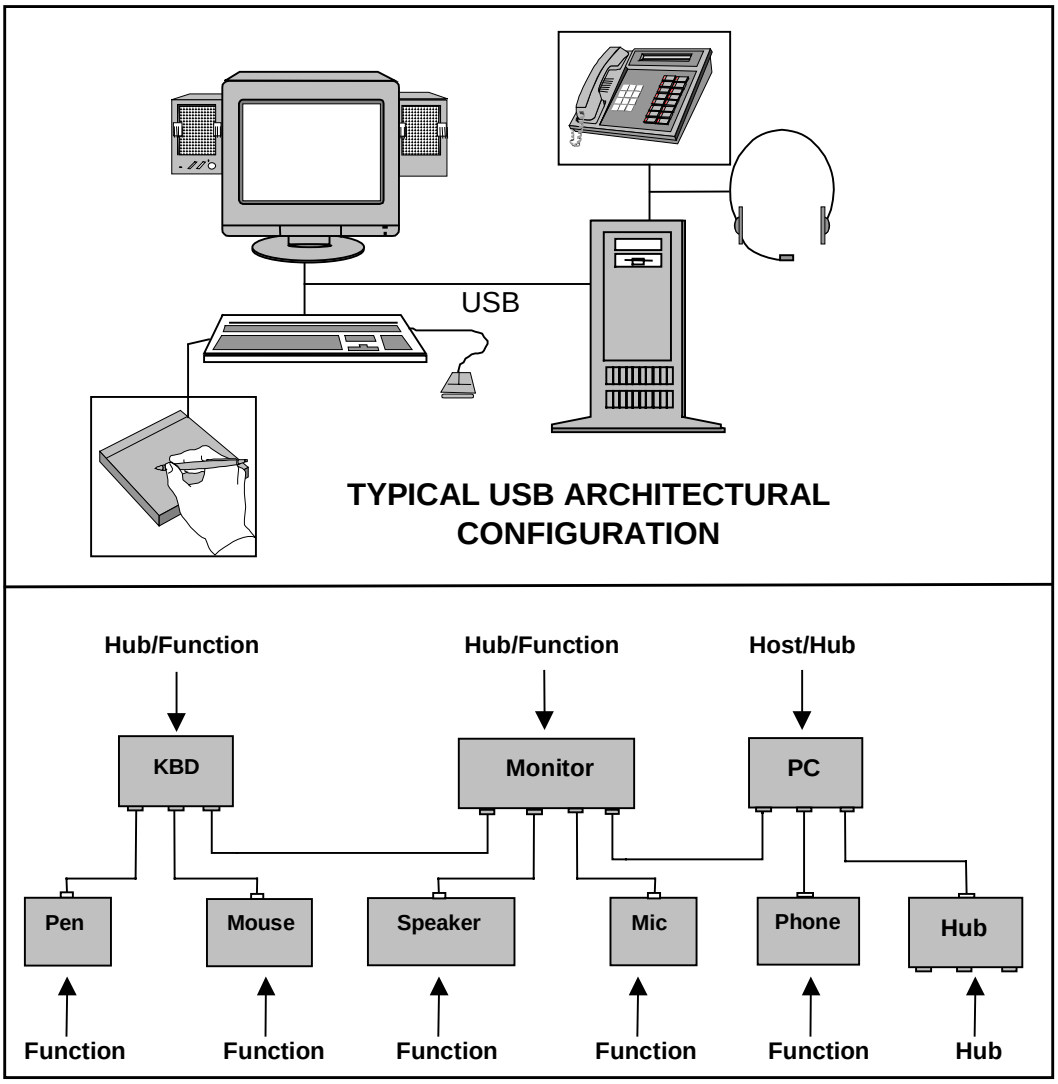

Figure 4-4. Hubs in a Desktop Computer Environment

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 non-removable 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 include the following:

- A human interface device such as a mouse, keyboard, tablet, or game controller
- An imaging device such as a scanner, printer, or camera
- A mass storage device such as a CD-ROM drive, floppy drive, or DVD drive

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 power to attached USB devices

The USB System Software on the host manages interactions between USB devices and host-based device software. There are five areas of interactions between the USB System Software and device software:

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

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. The information in this chapter affects all implementers. The information presented is at a level above the signaling and protocol definitions of the system. Consult Chapter 7 and Chapter 8 for more details about their respective parts of the USB system. This chapter provides framework information that is further expanded in Chapters 9 through 11. All implementers should read this chapter so they understand the 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 is 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. The 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.

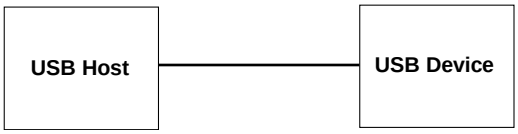

Figure 5-1. Simple USB Host/Device View

Figure 5-2 shows a deeper overview of the 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 the USB in a particular operating system. The USB System Software is 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.

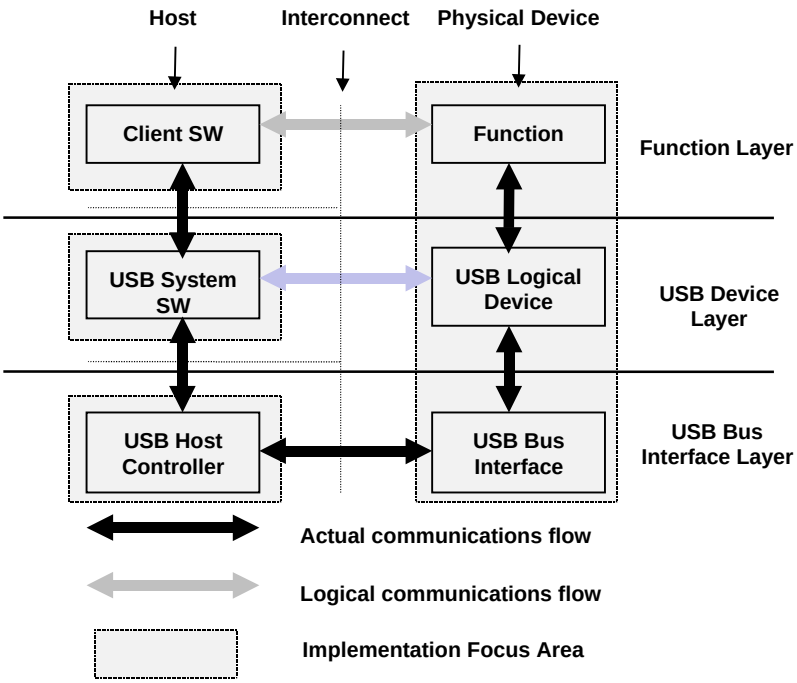

Figure 5-2. USB Implementation Areas

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 the 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 5.11 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 5.12 presents features of the USB specific to devices requiring isochronous data transfers. Device implementers for non-isochronous devices do not need to read Section 5.12.

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 is shown in Figure 5-3 and includes the following:

- USB Host Controller
- Aggregate USB System Software (USB Driver, Host Controller Driver, and host software)
- Client

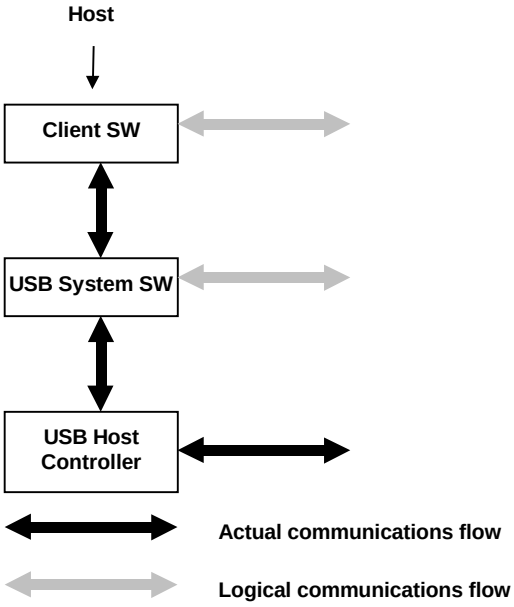

Figure 5-3. Host Composition

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 gains access to the bus only 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.

5.2.2 USB Devices

A USB physical device's logical composition is shown in Figure 5-4 and includes the following:

- 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.

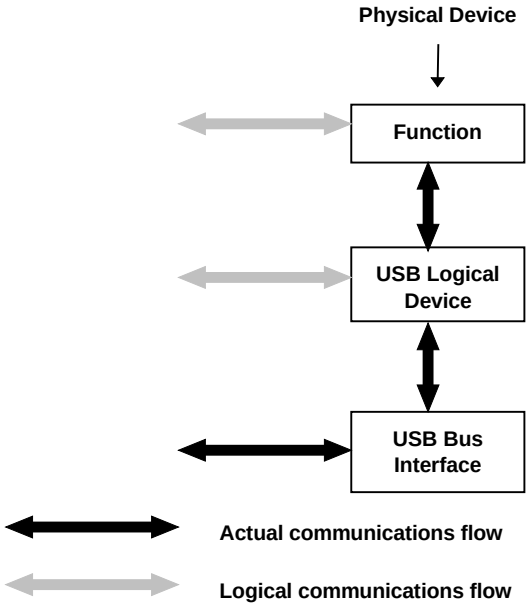

Figure 5-4. Physical Device Composition

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 that 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.

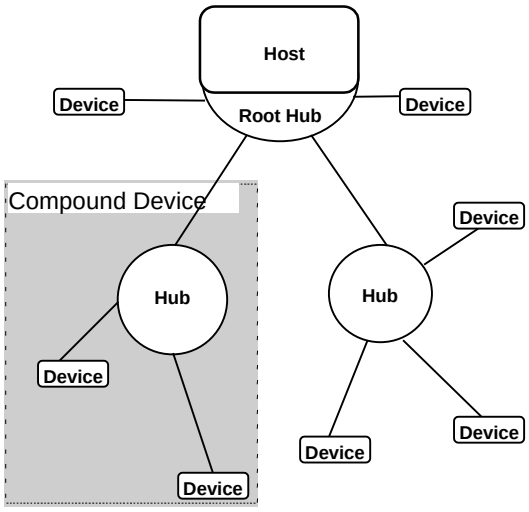

Figure 5-5. USB Physical Bus Topology

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. The hub and each function attached to the hub within the compound device is assigned its own device address. A device that has multiple interfaces controlled independently of each other is referred to as a composite device. A composite device has only a single device address. 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.

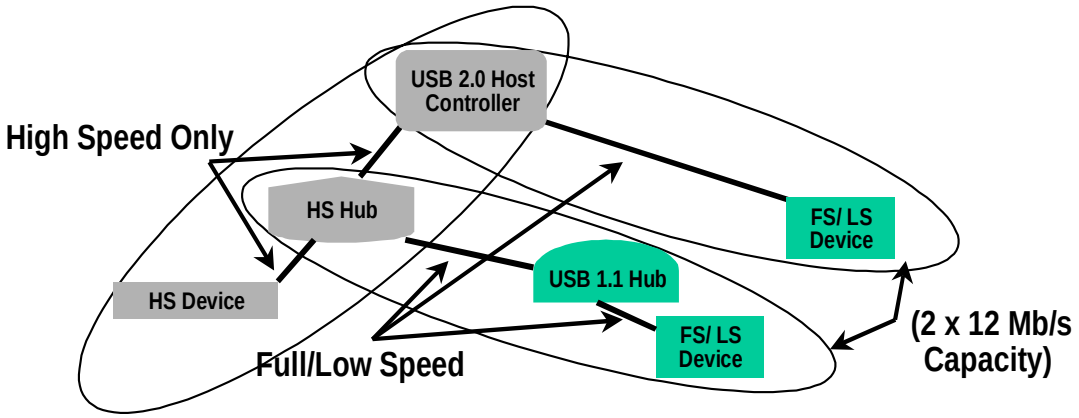

Figure 5-6. Multiple Full-speed Buses in a High-speed System

The hub plays a special role in a high-speed system. The hub isolates the full-/low-speed signaling environment from the high-speed signaling environment. Figure 5-6 shows a hub operating in high speed supporting a high-speed attached device. The hub also allows USB1.1 hubs to attach and operate at full-/low-speed along with other full-/low-speed only devices. The host controller also directly supports attaching full-/low-speed only devices. Chapter 11 describes the details of how the hub accomplishes the isolation of the two signaling environments.

Each high-speed operating hub essentially adds one (or more) additional full-/low-speed buses; i.e., each hub supports additional (optionally multiple) 12 Mb/s of USB full-/low-speed bandwidth. This allows more full-/low-speed buses to be attached without requiring additional host controllers in a system. Even though there can be several 12 Mb/s full-/low-speed buses, there are only at most 127 USB devices attached to any single host controller.

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 5-7 that corresponds to the physical topology shown in Figure 5-5. Hubs are logical devices also but are not shown in Figure 5-7 to simplify the picture. Even though most host/logical device activities use this logical perspective, the host maintains an awareness of the 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.

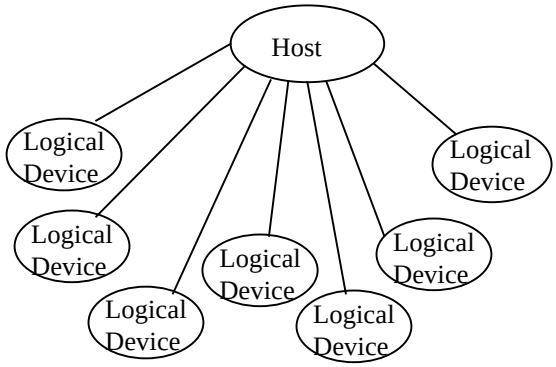

Figure 5-7. USB Logical Bus Topology

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 the USB. This allows the designer of the device and client software to focus on the hardware/software interaction design details. Figure 5-8 illustrates a device designer's perspective of the relationships of client software and USB functions with respect to the USB logical topology of Figure 5-7.

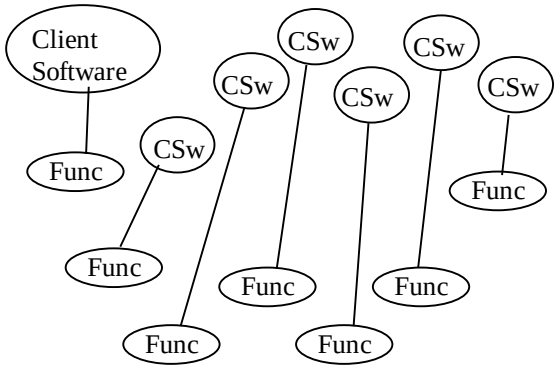

Figure 5-8. Client Software-to-function Relationships

5.3 USB Communication Flow

The 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. The 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.

Figure 5-9 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 through 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 the USB System Software and the client software. This interface provides clients with convenient functions for manipulating USB devices.

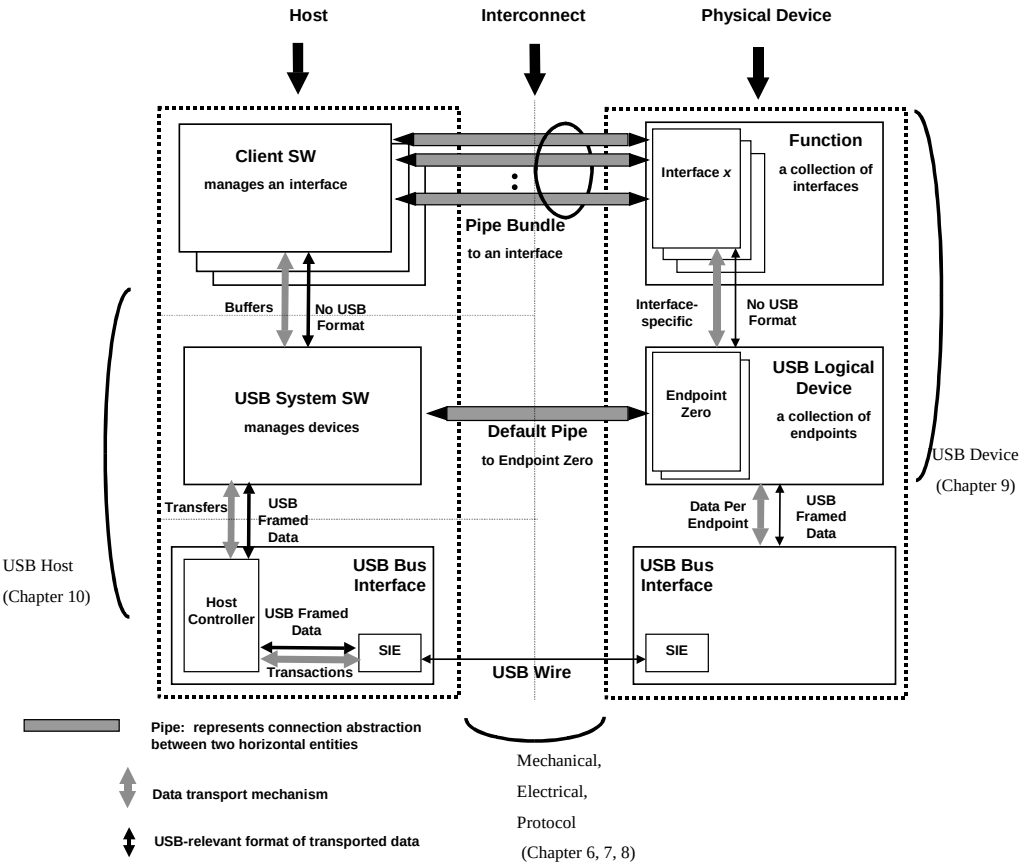

Figure 5-9. USB Host/Device Detailed View

A USB logical device appears to the USB system as a collection of endpoints. Endpoints are grouped into endpoint sets that implement an interface. Interfaces are views to the function. The USB System Software manages the device using the Default Control Pipe. 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 5-10 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.

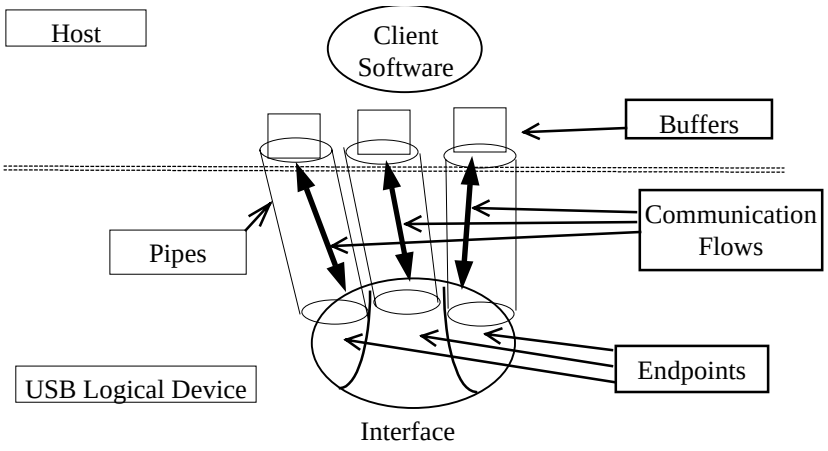

Figure 5-10. USB Communication Flow

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 the 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 independent endpoints. Each logical device has a unique address assigned by the system at device attachment time. Each endpoint on a device is given at design time a unique device-determined identifier called the endpoint number. Each endpoint has a device-determined direction of data flow. The combination of the device address, endpoint number, and direction allows each endpoint to be uniquely referenced. Each endpoint is a simplex connection that supports data flow in one direction: either input (from device to host) or output (from host to device).

An endpoint has characteristics that determine the type of transfer service required between the endpoint and the client software. An endpoint describes itself by:

- Bus access frequency/latency requirement
- Bandwidth requirement
- Endpoint number
- 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)
- The direction in which data is transferred between the endpoint and the host

Endpoints other than those with endpoint number zero are in an unknown state before being configured and may not be accessed by the host before being configured.

5.3.1.1 Endpoint Zero Requirements

All USB devices are required to implement a default control method that uses both the input and output endpoints with endpoint number zero. The USB System Software uses this default control method to initialize and generically manipulate the logical device (e.g., to configure the logical device) as the Default Control Pipe (see Section 5.3.2). The Default Control Pipe provides access to the device's configuration information and allows generic USB status and control access. The Default Control Pipe supports control transfers as defined in Section 5.5. The endpoints with endpoint number zero are always accessible once a device is attached, powered, and has received a bus reset.

A USB device that is capable of operating at high-speed must have a minimum level of support for operating at full-speed. When the device is attached to a hub operating in full-speed, the device must:

- Be able to reset successfully at full-speed
- Respond successfully to standard requests: set_address, set_configuration, get_descriptor for device and configuration descriptors, and return appropriate information

The high-speed device may or may not be able to support its intended functionality when operating at full-speed.

5.3.1.2 Non-endpoint Zero Requirements

Functions can have additional endpoints as required for their implementation. Low-speed functions are limited to two optional endpoints beyond the two required to implement the Default Control Pipe. Full-speed devices can have additional endpoints only limited by the protocol definition (i.e., a maximum of 15 additional input endpoints and 15 additional output endpoints).

Endpoints other than those for the Default Control Pipe cannot be used until the device is 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 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

The 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 the USB.

Additionally, pipes have the following 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).

The pipe that consists of the two endpoints with endpoint number zero is called the Default Control Pipe. This pipe is always available once a device is powered and has received a bus reset. Other pipes come into existence when a USB device is configured. The Default Control Pipe is used by the USB System Software to determine device identification and configuration requirements and to configure the device. The Default Control Pipe can also be used by device-specific software after the device is configured. The USB System

Software retains “ownership” of the Default Control 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 condition that causes it to send a STALL to the host (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 halt or error condition via a USBD call. An appropriate status informs the software client of the specific IRP result for error versus halt (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 of the maximum packet size until the last data payload that contains the remainder of the overall IRP. See the description of each transfer type 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 in-band 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.

Because 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 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 uni-directional in their communication flow.

Data flowing through a stream pipe is expected to interact with what the USB believes is a single client. The 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, which are explained in later sections.

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. In order to accommodate the request/data/status paradigm, message pipes impose a structure on the communication flow that 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 Default Control Pipe is always a message pipe.

The USB System Software ensures that multiple requests are not sent to a message pipe concurrently. A device is required to service only a single message request at a time per message pipe. Multiple software clients on the host can make requests via the Default Control Pipe, but they are sent to the device in a first-in, first-out order. A device 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).

A message pipe will not normally be sent the next message from the host until the current message's processing at the device has been completed. However, there are error conditions whereby a message transfer can be aborted by the host and the message pipe can be sent a new message transfer prematurely (from the device's 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). The USB does not allow a message pipe to be associated with different endpoint numbers for each direction.

Message pipes support the control transfer type, which is explained in Section 5.5.

5.3.3 Frames and Microframes

USB establishes a 1 millisecond time base called a frame on a full-/low-speed bus and a 125 µs time base called a microframe on a high-speed bus. A (micro)frame can contain several transactions. Each transfer type defines what transactions are allowed within a (micro)frame for an endpoint. Isochronous and interrupt endpoints are given opportunities to the bus every N (micro)frames. The values of N and other details about isochronous and interrupt transfers are described in Sections 5.6 and 5.7.

5.4 Transfer Types

The 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 the USB allows device-specific structured data to be transported within the USB-defined message data payload. The 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, the 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 the following:

- Data format imposed by the USB
- Direction of communication flow
- Packet size constraints

- Bus access constraints
- Latency constraints
- Required data sequences
- Error handling

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.

The 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: Low-frequency, bounded-latency communication.
- Bulk Transfers: Non-periodic, large-packet bursty communication, typically used for data that can use any available bandwidth and can 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 8.3.4. Chapter 8 also describes details of the protocol that are affected by use of each particular transfer type.

5.4.1 Table Calculation Examples

The following sections describe each of the USB transfer types. In these sections, there are tables that illustrate the maximum number of transactions that can be expected to be contained in a (micro)frame. These tables can be used to determine the maximum performance behavior possible for a specific transfer type. Actual performance may vary with specific system implementation details.

Each table shows:

- The protocol overhead required for the specific transfer type (and speed)
- For some sample data payload sizes:
    - The maximum sustained bandwidth possible for this case
    - The percentage of a (micro)frame that each transaction requires
    - The maximum number of transactions in a (micro)frame for the specific case
    - The remaining bytes in a (micro)frame that would not be required for the specific case
    - The total number of data bytes transported in a single (micro)frame for the specific case

A transaction of a particular transfer type typically requires multiple packets. The protocol overhead for each transaction includes:

- A SYNC field for each packet: either 8 bits (full-/low-speed) or 32 bits (high-speed)
- A PID byte for each packet: includes PID and PID invert (check) bits
- An EOP for each packet: 3 bits (full-/low-speed) or 8 bits (high-speed)
- In a token packet, the endpoint number, device address, and CRC5 fields (16 bits total)

- In a data packet, CRC16 fields (16 bits total)
- In a data packet, any data field (8 bits per byte)
- For transaction with multiple packets, the inter packet gap or bus turnaround time required.

For these calculations, there is assumed to be no bit-stuffing required.

Using the low speed interrupt OUT as an example, there are 5 packets in the transaction:

- A PRE special packet
- A token packet
- A PRE special packet
- A data packet
- A handshake packet

There is one bus turnaround between the data and handshake packets. The protocol overhead is therefore:

5 SYNC, 5 PID, Endpoint + CRC5, CRC16, 5 EOPs and interpacket delay (one bus turnaround, 1 delay between packets, and 2 hub setup times).

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 8.5.3 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 the Default Control Pipe as a message pipe. This pipe is used by the USB System Software. The Default Control Pipe provides access to the USB device's configuration, status, and control information. A function can, but is not required to, 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 carried only through message pipes. Consequently, data flows using control transfers must adhere to USB data structure definitions as described in Section 5.5.1.

The USB system 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. The 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 Control Transfer 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 a USB-defined structure except when carrying vendor-specific information. The Status transaction also has a USB-defined structure. Specific control transfer Setup/Data definitions are described in Section 8.5.3 and Chapter 9.

5.5.2 Control Transfer Direction

Control transfers are supported via bi-directional communication flow over message pipes. As a consequence, when a control pipe is configured, it uses both the input and output endpoint with the specified endpoint number.

5.5.3 Control Transfer 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. The allowable maximum control transfer data payload sizes for full-speed devices is 8, 16, 32, or 64 bytes; for high-speed devices, it is 64 bytes and for low-speed devices, it is 8 bytes. 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 information that is required by the protocol. A Setup packet is always eight bytes. A control pipe (including the Default Control Pipe) always uses its *wMaxPacketSize* value for data payloads.

An endpoint reports in its configuration information the value for its maximum data payload size. The 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, only 8-byte maximum data payload sizes for low-speed control endpoints, and only 64-byte maximum data payload size for high-speed control endpoints. No Host Controller is required to support larger or smaller maximum data payload sizes.

In order to determine the maximum packet size for the Default Control Pipe, the USB System Software reads the device descriptor. The host will read the first eight bytes of the device descriptor. The device always responds with at least these initial bytes in a single packet. After the host reads the initial part of the device descriptor, it is guaranteed to have read this default pipe's *wMaxPacketSize* field (byte 7 of the device descriptor). It will then allow the correct size for all subsequent transactions. For all other control endpoints, the maximum data payload size is known after configuration so that the USB System Software can ensure 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 *wMaxPacketSize* (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.

The Data stage of a control transfer from an endpoint to the host is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data specified during the Setup stage
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

When a Data stage is complete, the Host Controller advances to the Status stage instead of continuing on with another data transaction. If the Host Controller does not advance to the Status stage when the Data stage is complete, the endpoint halts the pipe as was outlined in Section 5.3.2. If a larger-than-expected data payload is received from the endpoint, the IRP for the control transfer will be aborted/retired.

The Data stage of a control transfer from the host to an endpoint is complete when all of the data has been transferred. If the endpoint receives a larger-than-expected data payload from the host, it halts the pipe.

5.5.4 Control Transfer Bus Access Constraints

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

An endpoint has no way to indicate a desired bus access frequency for a control pipe. The 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.

The USB requires that part of each (micro)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 for full-/low-speed endpoints or less than 20% of a microframe for high-speed endpoints, 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 (micro)frame; i.e., it is not required to be retried in the same (micro)frame.
- If there are more control transfers than reserved time, but there is additional (micro)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 for the available (micro)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 bus time.

High-speed control endpoints must support the PING flow control protocol for OUT transactions. The details of this protocol are described in Section 8.5.1.

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

The USB System Software can, at its discretion, vary the rate of control transfers to a particular endpoint. An endpoint and its client software cannot assume a specific rate of service for control transfers. A control endpoint may see zero or more transfers in a single (micro)frame. 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 (micro)frame timing limit the maximum number of successful control transfers within a (micro)frame for any USB system. For full-/low-speed buses, the number of successful control transfers per frame is limited to less than 29 full-speed eight-byte data payloads or less than four low-speed eight-byte data payloads. For high-speed buses, the number of control transfers is limited to less than 32 high-speed 64-byte data payloads per microframe.

Table 5-1 lists information about different-sized low-speed packets and the maximum number of packets possible in a frame. The table does not include the overhead associated with bit stuffing.

Table 5-1. Low-speed Control Transfer Limits

	Protocol Overhead (63 bytes)		(15 SYNC bytes, 15 PID bytes, 6 Endpoint + CRC bytes, 6 CRC bytes, 8 Setup data bytes, and a 13-byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	3000	26%	3	40	3
	2	6000	27%	3	37	6
	4	12000	28%	3	31	12
	8	24000	30%	3	19	24
Max		187500				187

For all speeds, because a control transfer is composed of several packets, the packets can be spread over several (micro)frames to spread the bus time required across several (micro)frames.

The 10% frame reservation for full-/low-speed non-periodic 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. Because the USB system 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 noncontiguous frames. A Host Controller, for various implementation reasons, may not be able to provide the theoretical maximum number of control transfers per frame.

For high-speed endpoints, the 20% microframe reservation for non-periodic transfers means that all high speed control transfers are contending for nominally six control transfers per microframe. High-speed control transfers contend for microframe time along with split-transactions (see Sections 11.15-11.21 for more information about split transactions) for full- and low-speed control transfers. Both full-speed and low-speed control transfers contend for the same available frame time. However, high-speed control transfers for some endpoints can occur simultaneously with full- and low-speed control transfers for other endpoints. Low-speed control transfers simply take longer to transfer.

Table 5-2 lists information about different-sized full-speed control transfers and the maximum number of transfers possible in a frame. This table was generated assuming that there is one Data stage transaction and that the Data stage has a zero-length status phase. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload sizes. The table does not include the overhead associated with bit stuffing.

Table 5-2. Full-speed Control Transfer Limits

	Protocol Overhead (45 bytes)		(9 SYNC bytes, 9 PID bytes, 6 Endpoint + CRC bytes, 6 CRC bytes, 8 Setup data bytes, and a 7-byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	32000	3%	32	23	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

Table 5-3 lists information about different-sized high-speed control transfers and the maximum number of transfers possible in a microframe. This table was generated assuming that there is one Data stage transaction and that the Data stage has a zero-length status stage. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload size. The table does not include the overhead associated with bit stuffing.

Table 5-3. High-speed Control Transfer Limits

	Protocol Overhead (173 bytes)		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (9x4 SYNC bytes, 9 PID bytes, 6 EP/ADDR+CRC,6 CRC16, 8 Setup data, 9x(1+11) byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ Microframe Useful Data**
	1	344000	2%	43	18	43
	2	672000	2%	42	150	84
	4	1344000	2%	42	66	168
	8	2624000	2%	41	79	328
	16	4992000	3%	39	129	624
	32	9216000	3%	36	120	1152
	64	15872000	3%	31	153	1984
Max		60000000				7500

5.5.5 Control Transfer 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, each control transaction and the next control transfer will be moved over the bus at some Host Controller implementation-defined time.

The endpoint can be busy for a device-specific time 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 halt 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. For the Default Control Pipe, a device reset will ultimately be required to clear the halt or error condition if the next Setup PID is not accepted.

The USB provides robust error detection and 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
- Guaranteed constant data rate through the pipe as long as data is provided to the pipe
- 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 5.12 presents more detail on special considerations for handling isochronous data on the USB.

5.6.1 Isochronous Transfer Data Format

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

5.6.2 Isochronous Transfer Direction

An isochronous pipe is a stream pipe and is, therefore, always uni-directional. 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 Isochronous Transfer Packet Size Constraints

An endpoint in a given configuration for an isochronous pipe specifies the maximum size data payload that it can transmit or receive. The USB System Software uses this information during configuration to ensure that there is sufficient bus time to accommodate this maximum data payload in each (micro)frame. If there is sufficient bus time for the maximum data payload, the configuration is established; if not, the configuration is not established.

The USB limits the maximum data payload size to 1,023 bytes for each full-speed isochronous endpoint. High-speed endpoints are allowed up to 1024-byte data payloads. A high speed, high bandwidth endpoint specifies whether it requires two or three transactions per microframe. Table 5-4 lists information about different-sized full-speed isochronous transactions and the maximum number of transactions possible in a frame. The table is shaded to indicate that a full-speed isochronous endpoint (with a non-zero *wMaxpacket* size) must not be part of a default interface setting. The table does not include the overhead associated with bit stuffing.

Table 5-4. Full-speed Isochronous Transaction Limits

	Protocol Overhead (9 bytes)	(2 SYNC bytes, 2 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 1-byte interpacket delay)				
	Data Payload	**Max Bandwidth(bytes/ second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame 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

Table 5-5 lists information about different-sized high-speed isochronous transactions and the maximum number of transactions possible in a microframe. The table is shaded to indicate that a high-speed isochronous endpoint must not be part of a default interface setting. The table does not include the overhead associated with bit stuffing.

Any given transaction for an 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. The 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 packet size seen by the receiver. However, these errors can be detected by either CRC on the data or by knowledge the receiver has about the expected size for any transaction.

Table 5-5. High-speed Isochronous Transaction Limits

	Protocol Overhead		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (2x4 SYNC bytes, 2 PID bytes, 2 EP/ADDR+addr+CRC5, 2 CRC16, and a 2x(1+11)) byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ MicroFrame Useful Data**
	1	1536000	1%	192	12	192
	2	2992000	1%	187	20	374
	4	5696000	1%	178	24	712
	8	10432000	1%	163	2	1304
	16	17664000	1%	138	48	2208
	32	27392000	1%	107	10	3424
	64	37376000	1%	73	54	4672
	128	46080000	2%	45	30	5760
	256	51200000	4%	25	150	6400
	512	53248000	7%	13	350	6656
	1024	57344000	14%	7	66	7168
	2048	49152000	28%	3	1242	6144
	3072	49152000	41%	2	1280	6144
Max		60000000				7500

All device default interface settings must not include any isochronous endpoints with non-zero data payload sizes (specified via *wMaxPacketSize* in the endpoint descriptor). Alternate interface settings may specify non-zero data payload sizes for isochronous endpoints. If the isochronous endpoints have a large data payload size, it is recommended that additional alternate configurations or interface settings be used to specify a range of data payload sizes. This increases the chance that the device can be used successfully in combination with other USB devices.

5.6.4 Isochronous Transfer Bus Access Constraints

Isochronous transfers can only be used by full-speed and high-speed devices.

The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers for full-speed endpoints. High-speed endpoints can allocate at most 80% of a microframe for periodic transfers.

An isochronous endpoint must specify its required bus access period. Full-/high-speed endpoints must specify a desired period as ($2^{bInterval-1}$) x F, where bInterval is in the range one to (and including) 16 and F is 125 µs for high-speed and 1ms for full-speed. This allows full-/high-speed isochronous transfers to have rates slower than one transaction per (micro)frame. However, an isochronous endpoint must be prepared to handle poll rates faster than the one specified. A host must not issue more than 1 transaction in a (micro)frame for an isochronous endpoint unless the endpoint is high-speed, high-bandwidth (see below). An isochronous IN endpoint must return a zero-length packet whenever data is requested at a faster interval than the specified interval and data is not available.

A high-speed endpoint can move up to 3072 bytes per microframe (or 192 Mb/s). A high-speed isochronous endpoint that requires more than 1024 bytes per period is called a high-bandwidth endpoint. A high-bandwidth endpoint uses multiple transactions per microframe. A high-bandwidth endpoint must specify a period of 1x125 µs (i.e., a *bInterval* value of 1). See Section 5.9 for more information about the details of multiple transactions per microframe for high-bandwidth high-speed endpoints.

Errors on the bus or delays in operating system scheduling of client software can result in no packet being transferred for a (micro)frame. An error indication should be returned as status to the client software in such a case. A device can also detect this situation by tracking SOF tokens and noticing a disturbance in the specified bus access period pattern.

The bus frequency and (micro)frame timing limit the maximum number of successful isochronous transactions within a (micro)frame for any USB system to less than 151 full-speed one-byte data payloads and less than 193 high-speed one-byte data payloads. A Host Controller, for various implementation reasons, may not be able to provide the theoretical maximum number of isochronous transactions per (micro)frame.

5.6.5 Isochronous Transfer 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 by assuming transfers normally succeed. Isochronous receivers can determine whether they missed data during a (micro)frame. Also, a receiver can determine how much data was lost. Section 5.12 describes these USB mechanisms in more detail.

An endpoint for isochronous transfers never halts because there is no handshake to report a halt condition. Errors are reported as status associated with the IRP for an isochronous transfer, but the isochronous pipe is not halted in an error case. If an error is detected, the host continues to process the data associated with the next (micro)frame of the transfer. Only limited error detection is possible because 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 send or receive 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 Interrupt Transfer Data Format

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

5.7.2 Interrupt Transfer Direction

An interrupt pipe is a stream pipe and is therefore always uni-directional. An endpoint description identifies whether a given interrupt pipe's communication flow is into or out of the host.

5.7.3 Interrupt Transfer Packet Size Constraints

An endpoint for an interrupt pipe specifies the maximum size data payload that it will transmit or receive. The maximum allowable interrupt data payload size is 64 bytes or less for full-speed. High-speed endpoints are allowed maximum data payload sizes up to 1024 bytes. A high speed, high bandwidth endpoint specifies whether it requires two or three transactions per microframe. Low-speed devices are limited to eight 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. The 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 support maximum data payload sizes from 0 to 64 bytes for full-speed interrupt endpoints, from 0 to 8 bytes for low-speed interrupt endpoints, and from 0 to 1024 bytes for high-speed interrupt endpoints. See Section 5.9 for more information about the details of multiple transactions per microframe for high bandwidth high-speed endpoints. No Host Controller is required to support larger maximum data payload sizes.

The USB System Software determines the maximum data payload size that will be used for an interrupt pipe during device configuration. This size remains constant for the lifetime of a device's configuration. The USB System 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. 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 *wMaxPacketSize* value. A device can move data via an interrupt pipe that is larger than *wMaxPacketSize*. 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 *wMaxPacketSize* with some remainder. The endpoint must transfer each transaction except the last as *wMaxPacketSize* 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. An interrupt transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

When an interrupt transfer is complete, the Host Controller retires the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, the interrupt IRP will be aborted/retired and the pipe will stall future IRPs until the condition is corrected and acknowledged.

All high-speed device default interface settings must not include any interrupt endpoints with a data payload size (specified via *wMaxPacketSize* in the endpoint descriptor) greater than 64 bytes. Alternate interface settings may specify larger data payload sizes for interrupt endpoints. If the interrupt endpoints have a large data payload size, it is recommended that additional configurations or alternate interface settings be used to specify a range of data payload sizes. This increases the chances that the device can be used successfully in combination with other USB devices.

5.7.4 Interrupt Transfer Bus Access Constraints

Interrupt transfers can be used by low-speed, full-speed, and high-speed devices. High-speed endpoints can be allocated at most 80% of a microframe for periodic transfers. The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) full-/low-speed transfers.

The bus frequency and (micro)frame timing limit the maximum number of successful interrupt transactions within a (micro)frame for any USB system to less than 108 full-speed one-byte data payloads, or less than 10 low-speed one-byte data payloads, or to less than 134 high-speed one-byte data payloads. A Host Controller, for various implementation reasons, may not be able to provide the above maximum number of interrupt transactions per (micro)frame.

Table 5-6 lists information about different low-speed interrupt transactions and the maximum number of transactions possible in a frame. Table 5-7 lists similar information for full-speed interrupt transactions. Table 5-8 lists similar information for high-speed interrupt transactions. The shaded portion of Table 5-8 indicates the data payload sizes of a high-speed interrupt endpoint that must not be part of a default interface setting. The tables do not include the overhead associated with bit stuffing.

Table 5-6. Low-speed Interrupt Transaction Limits

	Protocol Overhead (19 bytes)	(5 SYNC bytes, 5 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 5-byte interpacket delay)				
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	9000	11%	9	7	9
	2	16000	11%	8	19	16
	4	32000	12%	8	3	32
	8	48000	14%	6	25	48
Max		187500				187

Table 5-7. Full-speed Interrupt Transaction Limits

	Protocol Overhead (13 bytes)	(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)				
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame 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

Table 5-8. High-speed Interrupt Transaction Limits

	Protocol Overhead		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (3x4 SYNC bytes, 3 PID bytes, 2 EP/ADDR+CRC bytes, 2 CRC16 and a 3x(1+11) byte interpacket delay(EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ Microframe Useful Data**
	1	1064000	1%	133	52	133
	2	2096000	1%	131	33	262
	4	4064000	1%	127	7	508
	8	7616000	1%	119	3	952
	16	13440000	1%	105	45	1680
	32	22016000	1%	86	18	2752
	64	32256000	2%	63	3	4032
	128	40960000	2%	40	180	5120
	256	49152000	4%	24	36	6144
	512	53248000	8%	13	129	6656
	1024	49152000	14%	6	1026	6144
	2048	49152000	28%	3	1191	6144
	3072	49152000	42%	2	1246	6144
Max		60000000				7500

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 specifying only 10 ms to 255 ms. High-speed endpoints can specify a desired period ($2^{bInterval-1}$)x125 µs, where *bInterval* is in the range 1 to (including) 16. The USB System 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 the USB (125 µs microframe or 1 ms frame). The client software and device can depend only on the fact that the host will ensure that the time duration between 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. 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 available, its data will be transferred at the next allocated period.

A high-speed endpoint can move up to 3072 bytes per microframe (or 192 Mb/s). A high-speed interrupt endpoint that requires more than 1024 bytes per period is called a high-bandwidth endpoint. A high-bandwidth endpoint uses multiple transactions per microframe. A high-bandwidth endpoint must specify a period of 1x125 µs (i.e., a *bInterval* value of 1). See Section 5.9 for more information about the details of multiple transactions per microframe for high-bandwidth high-speed endpoints.

Interrupt transfers are moved over the USB by accessing an interrupt endpoint every specified period. For input interrupt endpoints, 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 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 Interrupt Transfer 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 halt 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 halt condition is achieved via software intervention through a separate control pipe. This recovery will reset the data toggle bit to DATA0 for the endpoint on both the host and the device. 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 use any available bandwidth. 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 errors on the bus
- Guaranteed delivery of data but no guarantee 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; for a USB with little bandwidth available, bulk transfers may trickle out over a relatively long period of time.

5.8.1 Bulk Transfer Data Format

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

5.8.2 Bulk Transfer 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 Bulk Transfer 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. The USB defines the allowable maximum bulk data payload sizes to be only 8, 16, 32, or 64 bytes for full-speed endpoints and 512 bytes for high-speed endpoints. A low-speed device must not have bulk endpoints. 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. The 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 packet sizes for full-speed bulk endpoints and 512 bytes for high-speed bulk endpoints. No Host Controller is required to support larger or smaller maximum packet sizes.

During configuration, the 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 *wMaxPacketSize* value. 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. A bulk transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

When a bulk transfer is complete, the Host Controller retires the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, all pending bulk IRPs for that endpoint will be aborted/retired.

5.8.4 Bulk Transfer Bus Access Constraints

Only full-speed and high-speed devices can use bulk transfers.

An endpoint has no way to indicate a desired bus access frequency for a bulk pipe. The 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 time guaranteed to be available for bulk transfers as there is for control transfers. Bulk transfers are moved over the bus only on a bandwidth-available basis. If there is bus time that is not being used for other purposes, bulk transfers will be moved over the bus. 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 USB System Software at its discretion can vary the bus time made available for bulk transfers to a particular endpoint. 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.

High-speed bulk OUT endpoints must support the PING flow control protocol. The details of this protocol are described in Section 8.5.1.

The bus frequency and (micro)frame timing limit the maximum number of successful bulk transactions within a (micro)frame for any USB system to less than 72 full-speed eight-byte data payloads or less than 14 high-speed 512-byte data payloads. Table 5-9 lists information about different-sized full-speed bulk transactions and the maximum number of transactions possible in a frame. The table does not include the overhead associated with bit stuffing. Table 5-10 lists similar information for high-speed bulk transactions.

Table 5-9. Full-speed Bulk Transaction Limits

	Protocol Overhead (13 bytes)		(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame 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