The following chapter is excerpted from

USB Complete

Everything You Need to Develop Custom USB Peripherals

Third Edition

by Jan Axelson

For more information about the book, code samples, and links to other USB information and tools, visit Lakeview Research's *USB Central* page on the web:

www.Lvr.com/usb.htm

USB Complete Third Edition
ISBN 1-931448-02-7

1

USB Basics

What if you had the chance to design a peripheral interface from scratch? Your wish list would likely include these qualities:

- Easy to use, so there's no need to fiddle with configuration and setup details.
- Fast, so the interface doesn't become a communications bottleneck.
- Reliable, so that errors are rare, with automatic retries when errors occur.
- Versatile, so many kinds of peripherals can use the interface.
- Inexpensive, so manufacturers and users don't balk at the price.
- Power-conserving, to save energy and extend battery life in portable computers and devices.
- Supported by the Windows and other operating systems, so developers don't have to write low-level drivers to communicate with the peripherals.

The Universal Serial Bus (USB) has all of these qualities. USB was designed from the ground up to be an interface for communicating with many types of peripherals without the limits and frustrations of older interfaces.

Every recent PC and Macintosh computer includes USB ports that can connect to standard peripherals such as keyboards, mice, scanners, cameras, printers, and drives as well as custom hardware for just about any purpose.

This chapter introduces USB, including its advantages and limits, some history about the interface and recent enhancements to it, and a look at what's involved in designing and programming a device with a USB interface.

What USB Can Do

USB is a likely solution any time you want to use a computer to communicate with a device outside of the computer. The interface is suitable for mass-produced, standard peripheral types as well as small-volume designs, including one-of-a-kind projects.

To be successful, an interface has to please two audiences: the users who want to use the peripherals and the developers who design the hardware and write the code that communicates with the device. USB has features to please both.

Benefits for Users

From the user's perspective, the benefits of USB are ease of use, fast and reliable data transfers, flexibility, low cost, and power conservation. Table 1-1 compares USB with other popular interfaces.

Ease of Use

Ease of use was a major design goal for USB, and the result is an interface that's a pleasure to use for many reasons:

One interface for many devices. USB is versatile enough to be usable with a variety of peripheral types. Instead of having a different connector type and supporting hardware for each peripheral, one interface serves many.

Table 1-1: Comparison of popular computer interfaces. Where a standard doesn't specify a maximum, the table shows a typical maximum.

Interface	Format	Number of Devices (maximum)	Distance (maximum, feet)	Speed (maximum, bits/sec.)	Typical Use
USB	asynchronous serial	127	16 (up to 96 ft. with 5 hubs)	1.5M, 12M, 480M	Mouse, keyboard, drive, audio, printer, other standard and custom peripherals
Ethernet	serial	1024	1600	10G	General network communications
IEEE-1394b (FireWire 800)	serial	64	300	3.2G	Video, mass storage
IEEE-488 (GPIB)	parallel	15	60	8M	Instrumentation
IrDA	asynchronous serial infrared	2	6	16M	Printers, hand-held computers
I^2C	synchronous serial	40	18	3.4M	Microcontroller communications
Microwire	synchronous serial	8	10	2M	Microcontroller communications
MIDI	serial current loop	2 (more with flow-through mode)	50	31.5k	Music, show control
Parallel Printer Port	parallel	2 (8 with daisy-chain support)	10–30	8M	Printers, scanners, disk drives
RS-232 (EIA/TIA-232)	asynchronous serial	2	50-100	20k (115k with some hardware)	Modem, mouse, instrumentation
RS-485 (TIA/EIA-485)	asynchronous serial	32 unit loads (up to 256 devices with some hardware)	4000	10M	Data acquisition and control systems
SPI	synchronous serial	8	10	2.1M	Microcontroller communications

Automatic configuration. When a user connects a USB peripheral to a PC, Windows detects the peripheral and loads the appropriate software driver. The first time the peripheral connects, Windows may prompt the user to insert a disk with driver software, but other than that, installation is automatic. There's no need to restart the system before using the peripheral.

Easy to connect. With USB, there's no need to open the computer's enclosure to add an expansion card for each peripheral. A typical PC has four or more USB ports. You can expand the number of ports by adding hubs with additional ports.

Easy cables. USB cable connectors are keyed so you can't plug them in wrong. A cable segment can be as long as 5 meters. With hubs, a peripheral can be as far as 30 meters from its host PC. USB connectors are small and compact in contrast to typical RS-232 and parallel connectors. To ensure reliable operation, the USB specification includes detailed requirements that all cables and connectors must meet.

Hot pluggable. You can connect and disconnect a USB peripheral whenever you want, whether or not the system and peripheral are powered, without damaging the PC or device. The operating system detects when a peripheral is attached and readies it for use.

No user settings. USB peripherals don't have user-selectable settings such as port addresses and interrupt-request (IRQ) lines so there are no jumpers to set or configuration utilities to run.

Frees hardware resources for other devices. Using USB for as many peripherals as possible frees up IRQ lines for the peripherals that require them. The PC dedicates a series of port addresses and one IRQ line to the USB host controller, but individual peripherals don't require additional resources or any PC programming that involves specifying port addresses or detecting hardware interrupts. In contrast, peripherals with other interfaces may require dedicated port addresses, an IRQ line, and an expansion slot.

No power supply required (sometimes). The USB interface includes power-supply and ground lines that provide a nominal +5V from the computer's or hub's power supply. A peripheral that requires up to 500 milliamperes can draw all of its power from the bus instead of having to provide a

power supply. In contrast, peripherals that use other interfaces may have to choose between including a power supply inside the device or using a bulky and inconvenient external supply.

Speed

USB supports three bus speeds: high speed at 480 Megabits/sec., full speed at 12 Megabits/sec., and low speed at 1.5 Megabits/sec. The USB host controllers in recent PCs support all three speeds.

The bus speeds describe the rate that information travels on the bus. In addition to data, the bus must carry status, control, and error-checking signals. Plus, all peripherals must share the bus. So the rate of data transfer that an individual peripheral can expect will be less than the bus speed. The theoretical maximum rate for a single data transfer is about 53 Megabytes/sec. at high speed, 1.2 Megabytes/sec. at full speed, and 800 bytes/sec. at low speed.

The USB 1.0 specification defined low and full speeds. Low speed was included for two reasons. Mice require flexible cables to make the devices easy to move around. Low-speed cables don't require twisted pairs or as much shielding and thus can be more flexible than full/high-speed cables. Also, low-speed devices can often be manufactured more cheaply. Full speed was intended to replace most other peripherals that used RS-232 (serial) and parallel ports. The data-transfer rates attainable at full speed are comparable to or better than the speeds attainable with earlier interfaces. High speed became an option with the release of version 2.0 of the USB specification.

Reliability

The reliability of USB is due to both the hardware and the protocols for data transfer. The hardware specifications for USB drivers, receivers, and cables ensure a quiet interface that eliminates most noise that could cause data errors. The USB protocol enables the detecting of errors in received data and notifying the sender so it can retransmit. The detecting, notifying, and retransmitting are done in hardware and don't require any programming or user intervention.

Low Cost

Even though USB is more complex than earlier interfaces, the components and cables are inexpensive. A device with a USB interface is likely to cost the same or less than an equivalent device with an older interface or a more recent interface such as IEEE-1394.

Low Power Consumption

Power-saving circuits and code can automatically power down USB peripherals when not in use yet keep them ready to respond when needed. The reduced power consumption saves money, is environmentally friendly, and for battery-powered devices, allows a longer time between recharges.

Wireless Communications

USB originated as a wired interface, but options now exist for wireless devices that use USB to communicate with PCs.

Benefits for Developers

Many of the user advantages described above also make things easier for developers. For example, USB's defined cable standards and automatic error checking mean that developers don't have to worry about specifying cable characteristics or providing error checking in software.

USB has other advantages that benefit developers. The developers include the hardware designers who select components and design the circuits in devices, the programmers who write the software embedded in the devices, and the programmers who write the PC software that communicates with the devices.

The benefits to developers result from the flexibility built into the USB protocol, the support in the controller chips and operating system, and the support available from the USB Implementers Forum.

Versatility

USB's four transfer types and three speeds make the interface feasible for many types of peripherals. There are transfer types suited for exchanging

large and small blocks of data, with and without time constraints. For data that can't tolerate delays, USB can guarantee bandwidth or a maximum time between transfers. These abilities are especially welcome under Windows, where accessing peripherals in real time is often a challenge. Although the operating system, device drivers, and application software can introduce unavoidable delays, USB makes it as easy as possible to achieve transfers that are close to real time.

Unlike other interfaces, USB doesn't assign specific functions to signal lines or make other assumptions about how the interface will be used. For example, the status and control lines on the PC's parallel port were defined with the intention of communicating with line printers. There are five input lines with assigned functions such as indicating a busy or paper-out condition. When developers began using the port for scanners and other peripherals that send large amounts of data to the PC, having just five inputs was a limitation. (Eventually the interface was enhanced to allow eight bits of input.) USB makes no such assumptions and is suitable for just about any peripheral type.

For communicating with common peripheral types such as printers, keyboards, and drives, USB has defined classes that specify device requirements and protocols. Developers can use the classes as a guide instead of having to reinvent everything from the ground up.

Operating System Support

Windows 98 was the first Windows operating system with reliable support for USB, and the editions that have followed, including Windows 2000, Windows Me, Windows XP, and Windows Server 2003, support USB as well. This book focuses on Windows programming for PCs, but other computers and operating systems also have USB support, including Apple Computer's Macintosh and the Linux operating system for PCs. Some real-time kernels also support USB.

A claim of operating-system support for USB can mean many things. At the most basic level, an operating system that supports USB must do three things:

- Detect when devices are attached to and removed from the system.
- Communicate with newly attached devices to find out how to exchange data with them.
- Provide a mechanism that enables software drivers to communicate with the computer's USB hardware and the applications that want to access USB peripherals.

At a higher level, operating system support may also mean the inclusion of class drivers that enable application programmers to access devices. If the operating system doesn't include a driver appropriate for a specific peripheral, the peripheral vendor must provide the driver.

With each new edition of Windows, Microsoft has added class drivers. Supported device types in recent Windows editions include human interface devices (keyboards, mice, game controllers), audio devices, modems, still-image and video cameras, scanners, printers, drives, and smart-card readers. Filter drivers can support device-specific features and abilities within a class. Applications use Application Programming Interface (API) functions or other operating-system components to communicate with device drivers.

For devices that aren't in supported classes, some vendors of USB peripheral controllers provide drivers that developers can use with the vendor's controllers.

USB device drivers use the Windows Driver Model (WDM), which defines an architecture for drivers that run under Windows 98 and later Windows editions. The aim is to enable developers to support multiple Windows editions with a single driver, though some devices may require different drivers for Windows 98/Windows Me and for Windows 2000/Windows XP. Because Windows includes low-level drivers that handle communications with the USB hardware, writing a USB device driver is typically easier than writing drivers for devices that use other interfaces.

Peripheral Support

On the peripheral side, each USB device's hardware must include a controller chip that manages the details of USB communications. Some controllers

are complete microcontrollers that include a CPU, program and data memory, and a USB interface. Other controllers must interface to an external CPU that communicates with the USB controller as needed.

The peripheral is responsible for responding to requests to send and receive data used in identifying and configuring the device and for reading and writing other data on the bus. In some controllers, some functions are microcoded in hardware and don't need to be programmed.

Many USB controllers are based on popular architectures such as Intel Corporation's 8051 or Microchip Technology's PICMicro®, with added circuits and machine codes to support USB communications. If you're already familiar with a chip architecture that has a USB-capable variant, you don't need to learn an entirely new architecture. Most peripheral manufacturers provide sample code for their chips. Using this code as a starting point can save much time.

USB Implementers Forum

With some interfaces, you're pretty much on your own when it comes to getting a design up and running. With USB, you have plenty of help via the USB Implementers Forum, Inc. (USB-IF) and its Web site (*www.usb.org*). The USB-IF is the non-profit corporation founded by the companies that developed the USB specification.

The USB-IF's mission is to support the advancement and adoption of USB technology. To that end, the USB-IF offers information, tools, and tests. The information includes the specification documents, white papers, FAQs, and a Web forum where developers can discuss USB-related topics. The tools provided by the USB-IF include software and hardware to help in developing and testing products. The support for testing includes developing compliance tests to verify proper operation and holding compliance workshops where developers can have their products tested and earn the rights for their devices to display the USB logo.

Beyond the Hype

All of USB's advantages mean that it's a good candidate for use on many peripherals. But a single interface can't handle every task.

Interface Limits

Every interface has limits that make the interface impractical for some applications. For USB, limits to be aware of include speed and distance, lack of support for peer-to-peer communications, no ability to broadcast, and lack of support in older hardware and operating systems.

Speed. USB is versatile, but it's not designed to do everything. USB's high speed makes it competitive with the IEEE-1394a (Firewire) interface's 400 Megabits/sec., but IEEE-1394b is faster still, at 3.2 Gigabits/sec.

Distance. USB was designed as a desktop-expansion bus with the expectation that peripherals would be relatively close at hand. A cable segment can be as long as 5 meters. Other interfaces, including RS-232, RS-485, IEEE-1394b, and Ethernet, allow much longer cables. You can increase the length of a USB link to as much as 30 meters by using cables that link five hubs and a device.

To extend the range beyond 30 meters, an option is to use a USB interface on the PC, then convert to RS-485 or another interface for the long-distance cabling and peripheral interface.

Peer-to-Peer Communications. Every USB communication is between a host computer and a peripheral. The host is a PC or other computer with host-controller hardware. The peripheral contains device-controller hardware. Hosts can't talk to each other directly, and peripherals can't talk to each other directly. Other interfaces, such as IEEE-1394, allow direct peripheral-to-peripheral communications.

USB provides a partial solution with USB On-The-Go. An On-The-Go device can function both as a peripheral and as a limited-capability host that can communicate with other devices. Two hosts can communicate with each other via a PC-to-PC network bridge cable, which contains two devices that each connect to a different PC and pass data between the PCs.

Broadcasting. USB provides no way to send a message simultaneously to multiple devices on the bus. The host must send the message to each device individually. If you must have broadcasting ability, use IEEE-1394 or Ethernet.

Legacy Hardware. Older ("legacy") computers and peripherals don't have USB ports. If you want to connect a legacy peripheral to a USB port, a solution is a converter that translates between USB and the older interface. Several sources have converters for use with peripherals with RS-232, RS-485, and Centronics-type parallel ports. But the converter solution is useful only for peripherals that use conventional protocols supported by the converter's device driver. For example, most parallel-port converters support communications only with printers. Converters that will work with most devices that have RS-232 interfaces are available, however.

If you want to use a USB peripheral with a PC that doesn't support USB, a solution is to add USB capabilities to the PC. To do so, you'll need to add USB host-controller hardware and install an operating system that supports USB. The hardware is available on expansion cards that plug into a PCI slot or on a replacement motherboard. The Windows edition must be Windows 98 or later. Hardware that doesn't meet Windows 98's minimum requirements will need upgrades that may cost more than a new system.

If upgrading the PC to support USB isn't feasible, you might think a converter would be available to translate a peripheral's USB interface to the PC's RS-232, parallel, or other interface. But a converter isn't normally an option when the PC has the legacy interface. Creating a converter that contains the host-controller hardware and code that normally resides in the PC would cost too much to be practical.

Even on new systems, users may occasionally run applications on older operating systems such as MS-DOS. But for the most part, the drivers that Windows applications use to communicate with USB devices are specific to Windows. Without a driver, there's no way to access a USB peripheral. Although it's possible to write a USB driver for DOS, few peripheral vendors provide one. An exception is the mouse and keyboard, where a system's BIOS may include support to ensure that the peripherals are usable any

time, including from within DOS, from the BIOS screens that you can view on bootup, and from Windows' Safe mode.

Of course, the problem of supporting legacy hardware and operating systems is diminishing as these systems are replaced.

Developer Challenges

From the developer's perspective, the main challenges to USB are the complexity of the programming and for small-scale developers, the need to obtain a Vendor ID.

Protocol Complexity. A USB peripheral is an intelligent device that knows how to respond to requests and other events on the bus. Chips vary in how much firmware support they require to perform USB communications. In most cases, to program a USB peripheral, you need to know a fair amount about the USB's protocols, or rules for exchanging data on the bus. On the PC side, the device driver insulates application programmers from having to know many of the details, but device-driver writers need to be familiar with USB protocols and the driver's responsibilities.

In contrast, some older interfaces can connect to very simple circuits with very basic protocols. For example, the PC's original parallel printer port is just a series of digital inputs and outputs. You can connect to basic input and output circuits such as relays, switches, and analog-to-digital converters, with no computer intelligence required on the peripheral side. The PC software can monitor and control the individual bits on the ports.

With USB, applications can't just read and write to port addresses, and devices can't just present a series of inputs and outputs to read and write to directly. To access a USB device, applications must communicate with a class or device driver that in turn communicates with the lower-level USB drivers that manage communications on the bus. The device must implement the protocols that enable the PC to detect, identify, and communicate with the device.

Evolving Support in the Operating System. The class drivers included with Windows enable applications to communicate with many devices. Often, you can design your device to use one of the provided drivers. If not,

you may be able to use or adapt a driver provided by the controller-chip vendor. If you need to provide your own driver, there are toolkits that make the job of writing USB drivers easier.

Fees. The USB-IF provides the USB specification, related documents, software for compliance testing, and much more, all for free on its Web site. Anyone can develop USB software without paying a licensing fee.

However, anyone who distributes a device with a USB interface must obtain the rights to use a Vendor ID. At this writing, the administrative fee for obtaining a Vendor ID from the USB-IF is $1500. If you join the USB-IF (currently $2500/year), a Vendor ID is included along with other benefits such as admittance to compliance workshops. The Vendor ID and a Product ID assigned by the vendor are embedded in each device to identify the device to the operating system.

The fee is no problem for developers of high-volume products but can be an impediment to developers who expect to sell small quantities of inexpensive devices. With a few controllers that use the chip vendor's driver and require no vendor programming for the USB interface, peripheral developers can use the chip manufacturer's Vendor ID and a Product ID that the chip manufacturer assigns to the peripheral developer.

Evolution of an Interface

The main reason that new interfaces don't come around very often is that existing interfaces have the irresistible pull of all of the peripherals that users don't want to scrap. Using an existing interface also saves the time and expense of designing a new interface. This is why the designers of the original IBM PC chose compatibility with the existing Centronics parallel interface and the RS-232 serial-port interface—to speed up the design process and enable users to connect to printers and modems already on the market. These interfaces proved serviceable for close to two decades. But as computers became more powerful and the number and kinds of peripherals increased, the older interfaces became a bottleneck of slow communications with limited options for expansion.

A break with tradition is justified when the desire for enhancements is greater than the inconvenience and expense of change. This is the situation that prompted the development of USB.

The copyright on the USB 2.0 specification is assigned jointly to seven corporations, all heavily involved with PC hardware or software: Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. The USB-IF's Web site has the USB 2.0 specification, related documents, and other information for developers and end users.

Original USB

Version 1.0 of the USB specification was released in January 1996. Version 1.1 is dated September 1998. USB 1.1 added one new transfer type (interrupt OUT). In this book, *USB 1.x* refers to USB 1.0 and 1.1. April 2000 saw the release of USB 2.0 which added the option to use high speed. Engineering Change Notices (ECNs) contain revisions and additions to the specification, including defining a new mini-B connector, specifying a way for devices to use bus pull-up and pull-down resistors with looser tolerances, and defining a new descriptor type (the Interface Association Descriptor).

USB capability first became available on PCs with the release of Windows 95's OEM Service Release 2, available only to vendors installing Windows 95 on the PCs they sold. The USB support in these versions was limited and buggy, and there weren't a lot of USB peripherals available, so use of USB was limited in this era.

Things improved with the release of Windows 98 in June 1998. By this time, many more vendors had USB peripherals available, and USB began to take hold as a popular interface. Windows 98 Second Edition (SE) fixed some bugs and further enhanced the USB support. The original version of Windows 98 is called Windows 98 Gold, to distinguish it from Windows 98 SE.

This book concentrates on PCs running Windows 98 and later Windows editions. Windows NT4 preceded Windows 98 and doesn't support USB. Windows 2000, Windows Me, Windows XP, and Windows Server 2003 all support USB.

In this book, the term *PC* includes all of the various computers that share the common ancestor of the original IBM PC. The expression *Windows 98 and later* means Windows 98, Windows 98 SE, Windows 2000, Windows Me, Windows XP, and Windows Server 2003, and is also likely to apply to any Windows editions that follow. A USB-capable PC is assumed to be using Windows 98 or later. A host computer is any computer that can communicate with USB peripherals.

USB 2.0

As USB 1.x gained in popularity, it became clear that a faster bus speed would be useful. Investigation showed that a bus speed forty times faster than full speed could remain backwards-compatible with the low- and full-speed interfaces. Version 2.0's support for a bus speed of 480 Megabits/sec. makes USB much more attractive for peripherals such as printers, scanners, disk drives, and video.

An external USB 2.0 hub must support all three speeds. Other USB 2.0 devices can support low, full, or high speed or a combination. USB 2.0 is backwards compatible with USB 1.1. In other words, USB 2.0 peripherals can use the same connectors and cables as 1.x peripherals, and a USB 2.0 peripheral works when connected to a PC that supports USB 1.x or 2.0. To use high speed, a high-speed-capable device must connect under a 2.0 host computer, and all hubs between the host computer and the device must be 2.0 hubs. Version 2.0 hosts and hubs can also communicate with 1.x peripherals. A 2.0-compliant hub with a slower device attached converts between speeds as needed. This ability increases the complexity of 2.0 hubs but conserves bus bandwidth without requiring different hubs for different speeds.

When USB 2.0 devices first became available, there was confusion among users about whether all USB 2.0 devices supported high speed. In an attempt to reduce the confusion, the USB-IF released naming and packaging recommendations that emphasize speed and compatibility rather than USB version numbers. The recommendations say that a product that supports high speed should be labeled a "Hi-Speed USB" product, and messages on the packaging might include "Fully compatible with Original

USB" and "Compatible with the USB 2.0 Specification." A product that supports low or full speed only is a "USB" product, and the recommended messages on packaging are "Compatible with the USB 2.0 Specification" and "Works with USB and Hi-Speed USB systems, peripherals and cables." Manufacturers should avoid references to low or full speed on consumer packaging.

USB On-The-Go

As USB became the interface of choice for all kinds of peripherals, developers began to ask for a way to connect their peripherals directly to each other and to other USB peripherals. For example, a user might want to attach a printer directly to a camera or connect two drives together to exchange files. The On-The-Go (OTG) Supplement to the USB 2.0 Specification released in 2001 defines a limited-capability host function that devices can implement to enable communicating with peripherals.

Wireless USB

An enhancement under development for USB is a Wireless USB specification to enable wireless communications with devices at up to 480 Megabits/sec. The specification should be available in 2005.

USB versus IEEE-1394

Another popular interface choice for new peripherals is IEEE-1394. Apple Computer's implementation of the interface is called Firewire. Generally, IEEE-1394 can be faster and more flexible than USB but is more expensive to implement. With USB, a single host controls communications with many devices. The host handles most of the complexity, so the devices' electronics can be relatively simple and inexpensive. IEEE-1394 devices can communicate with each other directly, and a single communication can be directed to multiple receivers. The result is a more flexible interface, but the devices' electronics are more complex and expensive.

IEEE-1394 is best suited for applications that require extremely fast communications or broadcasting to multiple receivers. USB is best suited for

common peripherals such as keyboards, printers, and scanners, as well as low- to moderate-speed and cost-sensitive applications. For many devices, such as drives, either interface works well, and in fact some devices include both interfaces.

USB versus Ethernet

For some applications, the choice is between USB and Ethernet. Ethernet's advantages include the ability to use very long cables, broadcasting ability, and support for Internet protocols in PCs and Ethernet-capable development systems. Like IEEE-1394, however, the hardware required to support Ethernet is more complex and expensive than typical USB peripheral hardware. USB is also more versatile with four transfer types and a variety of defined classes for different purposes.

Bus Components

The physical components of the Universal Serial Bus consist of the circuits, connectors, and cables between a host and one or more devices.

The host is a PC or other computer that contains a USB host controller and a root hub. These components work together to enable the operating system to communicate with the devices on the bus. The host controller formats data for transmitting on the bus and translates received data to a format that operating-system components can understand. The host controller also performs other functions related to managing communications on the bus. The root hub has one or more connectors for attaching devices. The root hub, in combination with the host controller, detects attached and removed devices, carries out requests from the host controller, and passes data between devices and the host controller.

The devices are the peripherals and additional hubs that connect to the bus. A hub has one or more ports for connecting devices. Each device must contain circuits and code that know how to communicate with the host. The USB specification defines the cables and connectors that connect devices to hubs.

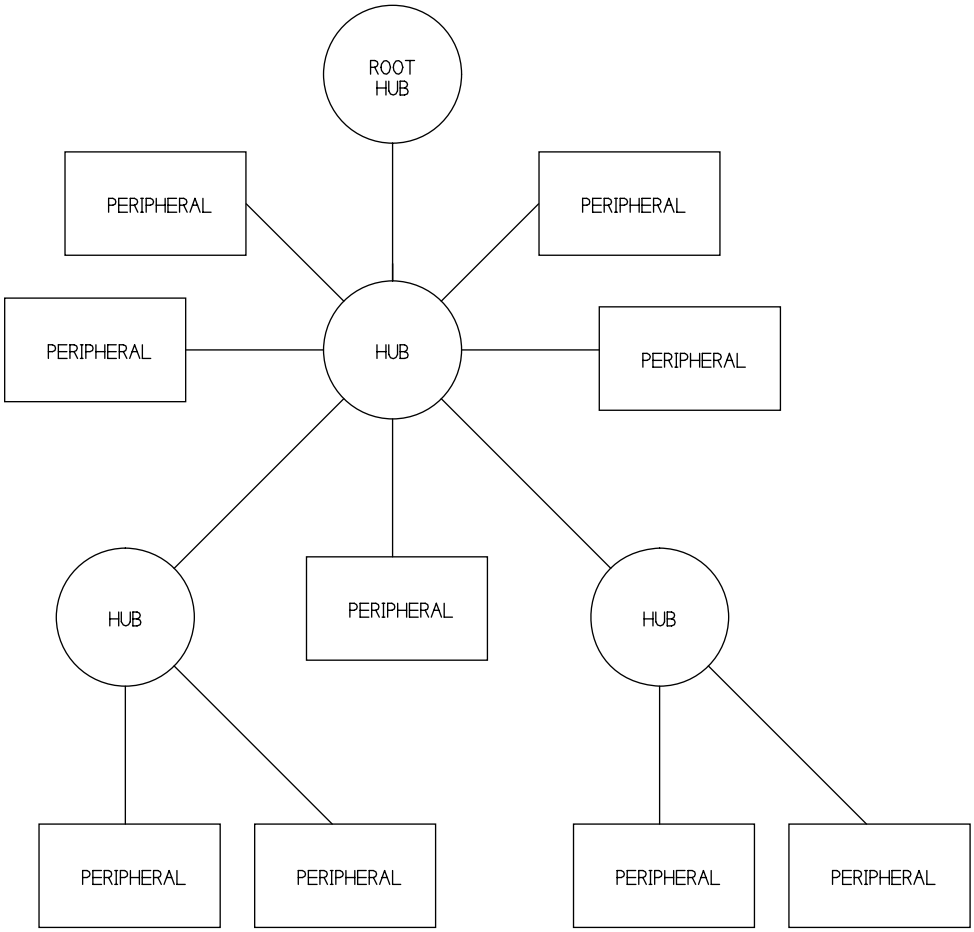

Figure 1-1: USB uses a tiered star topology, where each hub is the center of a star that can connect to peripherals or additional hubs.

Topology

The topology, or arrangement of connections, on the bus is a tiered star (Figure 1-1). At the center of each star is a hub. Each point on a star is a device that connects to a port on a hub. The number of points on each star can vary, with a typical hub having two, four, or seven ports. When there are

multiple hubs in series, you can think of them as connecting in a tier, or series, one above the next.

The tiered star describes only the physical connections. In programming, all that matters is the logical connection. To communicate, the host and device don't need to know or care how many hubs the communication passes through.

Only one device at a time can communicate with a host controller. To increase the available bandwidth for USB devices, a PC can have multiple host controllers.

Figure 1-2 shows a few possible configurations for a PC with two USB connectors. Some devices are compound devices that contain both a peripheral and a hub. You can cascade up to five external hubs in series, up to a total of 127 peripherals and hubs including the root hub. However, it may be impractical to have this many devices communicating with a single host controller.

In some cases, especially with compound devices where the hubs are hidden inside the peripherals, the peripherals may appear to be using a daisy-chain type of connection, where each new peripheral hooks to the last one in a chain. But the USB's topology is more flexible and complicated than a daisy chain. Each peripheral connects to a hub that manages communications with the host, and the peripherals and hubs aren't limited to connecting in a single chain.

Defining Terms

In the universe of USB, several everyday words have specific meanings. Along with *host*, defined earlier as the computer that controls the interface, three other such terms are *function, hub,* and *device*. It's also important to understand the concept of a USB port and how it differs from other ports such as RS-232.

Function

The USB specification defines a function as a device that provides a capability to the host. Examples of functions are a mouse, a set of speakers, or a

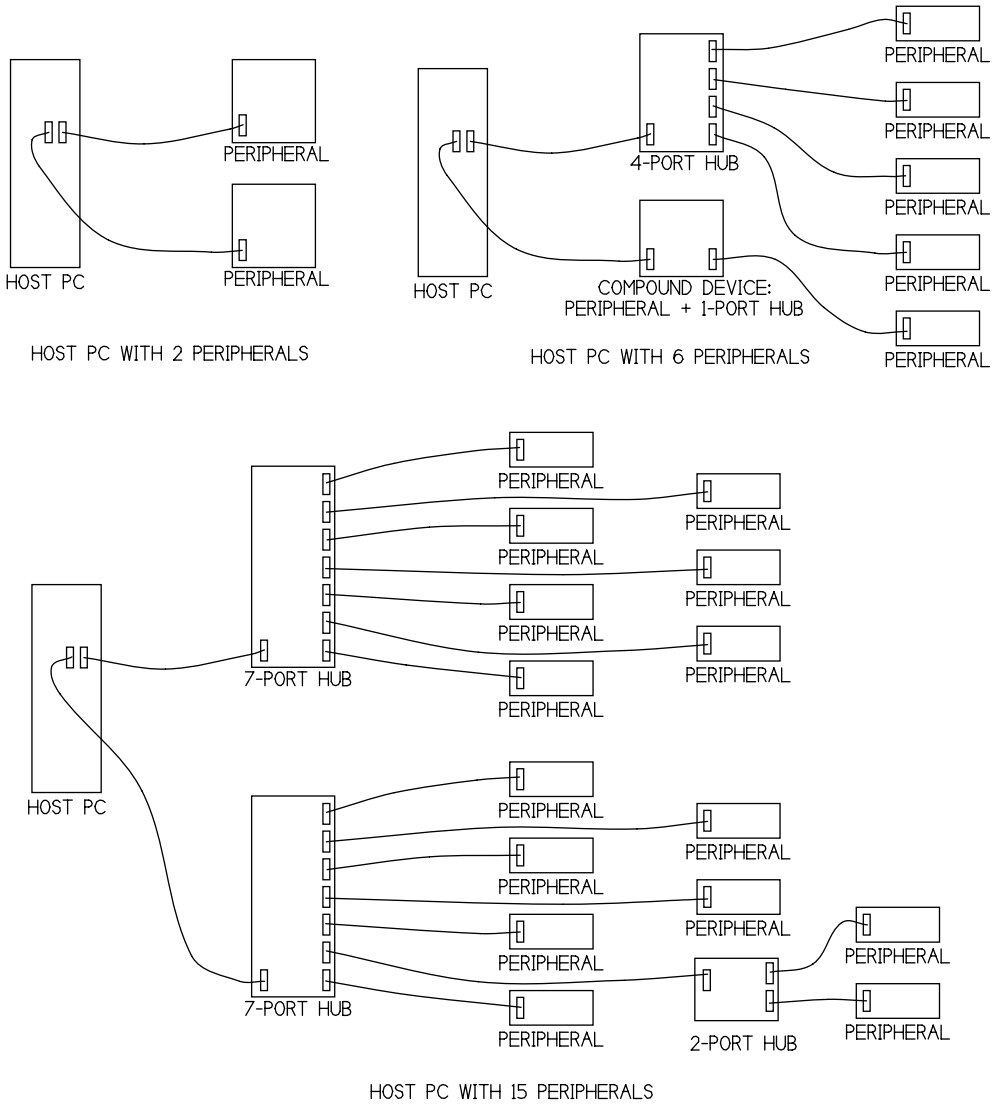

Figure 1-2: There are many possible configurations for connecting USB devices to a host PC. These are a few of the options for a host with two ports.

data-acquisition unit. A single physical device can contain more than one function.

Hub

A hub has one upstream connector for communicating with the host and one or more downstream connectors or internal connections to embedded devices. Each downstream connector or internal connection represents a USB port.

A 1.x hub repeats received USB traffic in both directions, manages power, and sends and responds to status and control messages. A 2.0 hub does all of this and also supports high speed, converting as needed between speeds.

Device

The USB specification's definition of a device is a function or a hub, except for the special case of the compound device, which contains a hub and one or more functions. The host treats a compound device in much the same way as if the hub and its functions were separate physical devices. Every device on the bus has a unique address, except again for a compound device, whose hub and functions each have unique addresses.

A composite device is a multi-function device with multiple, independent interfaces. The interfaces are defined by interface descriptors stored in the device. A composite device has one address on the bus but each interface has a different function and specifies its own device driver on the host. For example, a composite device could have one interface for an audio device and another interface for a control panel. Some Microsoft documentation uses the term composite device to refer to any device whose function is defined by its interface descriptor, rather than its device descriptor, whether or not there is more than one active interface.

Port

In a general sense, a computer port is an addressable location that is available for attaching additional circuits. Usually the circuits terminate at a connector that enables attaching a cable to a peripheral. Some peripheral circuits are hard-wired to a port. Software can monitor and control the port circuits

by reading and writing to the port's address. Computer memory also consists of addressable locations, but the CPU typically accesses memory with different machine instructions than are used for accessing ports.

USB ports differ from many other ports because all of the ports on a bus share a single path to the host and aren't directly addressable. With the RS-232 interface, each port is on the PC and independent from the others. If you have two RS-232 ports, each has its own data path, and each cable carries its own data and no one else's. The two ports can send and receive data at the same time.

With USB, each host controller manages a single bus, or data path. Each connector on a bus represents a USB port, but unlike RS-232, all devices share the bus's bandwidth. So even though a USB host controller can communicate with multiple ports, each with its own connector and cable, one data path serves all. Only one device or the host may transmit at a time. A single computer can have multiple USB host controllers, however, each with its own bus. Other interfaces where multiple devices can share a data path include IEEE-1394, SCSI, and Ethernet.

Another difference with USB is that a bus can have ports on hubs that are external to the host controller's PC.

Division of Labor

The host and its devices each have defined responsibilities. The host bears most of the burden of managing communications, but a device must have the intelligence to respond to communications and other bus events from the host and the hub the device attaches to.

The Host's Duties

To communicate with USB devices, a computer needs hardware and software support that enable the computer to function as a USB host. The hardware consists of a USB host controller and a root hub with one or more USB ports. The software support is an operating system that provides a mechanism for drivers to communicate with the USB hardware.

Just about any recent PC will have a USB host controller and two or more USB-port connectors. Many PCs have multiple host controllers. If a computer doesn't have USB support built into its motherboard, you can add a host controller on an expansion card that plugs into a slot on the PCI bus. For portable computers, USB controllers on PC cards are available.

The host is in charge of the bus. The host has to know what devices are on the bus and the capabilities of each device. The host must also do its best to ensure that all devices on the bus can send and receive data as needed. A bus may have many devices, each with different requirements, and all wanting to transfer data at the same time. The host's job is not trivial.

Fortunately, the host-controller hardware and the host-controller drivers in Windows do much of the work of managing the bus. Each device attached to the host must also have a device driver that enables applications to communicate with the device. Some peripherals can use device drivers included with Windows. Other devices use drivers provided by the device manufacturer. Various system-level software components manage communications between the device driver and the host-controller and root-hub hardware.

Applications don't have to worry about the USB-specific details of communicating with devices. All the application has to do is send and receive data using standard operating-system functions that are accessible from just about all programming languages. Often the application doesn't have to know or care whether the device uses USB or another interface.

The host performs the tasks below. The descriptions are in general terms. Later chapters have more specifics.

Detect Devices

On power-up, the hubs make the host aware of all attached USB devices. In a process called enumeration, the host assigns an address and requests additional information from each device. After power-up, whenever a device is removed or attached, the host learns of the event and enumerates any newly attached device and removes any detached device from its list of devices available to applications.

Manage Data Flow

The host manages the flow of data on the bus. Multiple peripherals may want to transfer data at the same time. The host controller divides the available time into segments called frames and microframes and gives each transmission a portion of a frame or microframe.

Transfers that must occur at specific rate are guaranteed to have the amount of time they need in each frame. During enumeration, a device's driver requests the bandwidth it will need for any transfers that must have guaranteed timing. If the bandwidth isn't available, the host doesn't allow communications to begin. The driver must then request a smaller portion of the bandwidth or wait until the requested bandwidth is available. Transfers that have no guaranteed timing use the remaining portion of the frames and must wait if the bus is busy.

Error Checking

When transferring data, the host adds error-checking bits. On receiving data, the device performs calculations on the data and compares the results with the received error-checking bits. If the results don't match, the device doesn't acknowledge receiving the data and the host knows that it should retransmit. USB also supports one transfer type that doesn't allow re-transmitting, in the interest of maintaining a constant transfer rate. In a similar way, the host error-checks the data received from devices.

The host may receive other indications that a device can't send or receive data. The host can then inform the device's driver of the problem, and the driver can notify the application so it can take appropriate action.

Provide Power

In addition to its two signal wires, a USB cable has +5V and ground wires. Some devices draw all of their power from these wires. The host provides power to all devices on power-up or attachment, and works with the devices to conserve power when possible. A high-power, bus-powered device can draw up to 500 milliamperes. The ports on a battery-powered host or hub

may support only low-power devices, which are limited to 100 milliamperes. A device may also have its own power supply.

Exchange Data with Peripherals

All of the above tasks support the host's main job, which is to exchange data with peripherals. In some cases, a device driver requests the host to attempt to send or receive data at defined intervals, while in others the host communicates only when an application or other software component requests a transfer. The device driver reports any problems to the appropriate application.

The Peripheral's Duties

In many ways, the peripheral's duties are a mirror image of the host's. When the host initiates communications, the peripheral must respond. But peripherals also have duties that are unique. A peripheral can't begin USB communications on its own. Instead, the peripheral must wait and respond to a communication from the host. (An exception is the remote wakeup feature, which enables a peripheral to request communications from the host.)

The USB controller in the peripheral handles many of the device's responsibilities in hardware. The amount of support required by device firmware varies with the chip.

The peripheral must perform all of the tasks described below. The descriptions are in general terms. Later chapters have more specifics.

Detect Communications Directed to the Chip

Each device monitors the device address contained in each communication on the bus. If the address doesn't match the device's stored address, the device ignores the communication. If the address matches, the device stores the data in its receive buffer and triggers an interrupt to indicate that data has arrived. In almost all chips, these functions are built into the hardware and require no support in code. The firmware doesn't have to take action or make decisions until the chip has detected a communication containing the device's address.

Respond to Standard Requests

On power-up, or when the device attaches to a powered system, a device must respond to standard requests sent by the host during enumeration. The host may also send requests any time after enumeration completes.

All devices must respond to these requests, which query the capabilities and status of the device or request the device to take other action. On receiving a request, the device places data or status information in a transmit buffer to send to the host. For some requests, such as selecting a configuration, the device takes other action in addition to responding with information.

The USB specification defines eleven requests, and a class or vendor may define additional requests. A device doesn't have to carry out every request, however; the device just has to respond to the request in an understandable way. For example, when the host requests a configuration that the device doesn't support, the device responds with a code that indicates that the configuration isn't supported.

Error Check

Like the host, a device adds error-checking bits to the data it sends. On receiving data that includes error-checking bits, the device does the error-checking calculations. The device's response or lack of response tells the host whether to re-transmit. These functions are typically built into the controller's hardware and don't need to be programmed. When appropriate, the device also detects the acknowledgement the host returns on receiving data from the device.

Manage Power

A device may be bus-powered or have its own power supply. For devices that use bus power, when there is no bus activity, the device must limit its use of bus current. When the host enters a low-power state, all communications on the bus cease, including the periodic timing markers the host normally sends. On detecting the absence of bus activity for three milliseconds, a device must enter the Suspend state and limit the current drawn from the

bus. While in the Suspend state, the device must continue to monitor the bus and exit the Suspend state when bus activity resumes.

Devices that don't support the remote-wakeup feature should consume no more than 500 microamperes from the bus in the Suspend state. If a device supports the remote-wakeup feature and the host has enabled the feature, the limit is 2.5 milliamperes. These are average values over 1 second; the peak current can be greater.

Exchange Data with the Host

All of the above tasks support the main job of the device's USB port, which is to exchange data with the host. After being configured, the device must respond to communications that may contain data and may require the device to return data or status information. The device's capabilities, the host's device driver, and the applications that use the device together determine the type of communications and when they occur.

For most transfers where the host sends data to the device, the device must respond to each transfer attempt by sending a code that indicates whether the device accepted the data or was too busy to handle it. For most transfers where the device sends data to the host, the device must respond to each attempt by returning data or a code indicating there was no data to send or the device was busy. Typically, the hardware responds automatically according to settings made previously in firmware. Some transfers don't use acknowledgements and the sender assumes the receiver has received all transmitted data.

The controller chip's hardware handles the details of formatting the data for the bus. The formatting includes adding error-checking bits to data to transmit, checking for errors in received data, and sending and receiving the individual bits on the bus.

Of course, the device must also do anything else it's responsible for. For example, a mouse must be ready to detect movement and button clicks, a data-acquisition unit has to read the data from its sensors, and a printer must translate received data into images on paper.

What about Speed?

A device controller may support one or more bus speeds. Virtually all hubs support low- and full-speed devices. The exception is a hub in a compound device whose functions use a single speed. A low- or full-speed peripheral can connect to any USB hub. Users don't have to know whether a device is low or full speed because there are no user settings or configurations for different speeds.

High-speed peripherals are likely to be dual-speed devices that also function at full speed. A 1.x host or hub doesn't support high speed because high speed didn't exist when the 1.x specifications were written. To ensure that high-speed devices don't confuse 1.x hosts and hubs, all high-speed devices must at least respond to standard enumeration requests at full speed. Any host can thus identify any attached device.

Other than responding to bus reset and standard requests at full speed, a high-speed device doesn't have to function at full speed. But because adding support for full speed is easy to do and is required to pass the USB IF's compliance tests, most high-speed devices also function at full speed.

The actual rate of data transfer between a peripheral and host is less than the bus speed and isn't always predictable. Some of the transmitted bits are used for identifying, synchronizing, and error-checking, and the data rate also depends on the type of transfer and how busy the bus is.

For time-sensitive data, USB supports transfer types that have a guaranteed rate or guaranteed maximum latency. Isochronous transfers have a guaranteed rate, where the host can request a specific number of bytes to transfer to or from a peripheral at defined intervals. The intervals can be as often as every millisecond at full speed or every 125 microseconds at high speed. Isochronous transfers have no error correcting, however. Interrupt transfers have error correcting and guaranteed maximum latency, which means that a precise rate isn't guaranteed, but the time between transfer attempts will be no greater than a specified period. At low speed, the requested maximum interval can range from 10 to 255 milliseconds. At full speed, the range is 1 to 255 milliseconds. At high speed, the range is 125 microseconds to 4.096 seconds.

Because the bus is shared, there's no guarantee that a particular rate or maximum latency will be available to a device. If the bus is too busy to allow a requested rate or maximum latency, the host refuses to complete the configuration process that enables the host to attempt the transfers. To take full advantage of reserved bandwidth, the device driver and application software or device firmware must ensure that data is available to send when the host controller is ready to initiate the transfer. The receiver of the data must also be ready to accept the data when it arrives.

At full and high speeds, the fastest transfers on an otherwise idle bus are bulk transfers, with a theoretical maximum of 1.216 Megabytes/sec. at full speed and 53.248 Megabytes/sec. at high speed. The host controller may limit a single bulk transfer to a slower rate, however. The transfers with the most guaranteed bandwidth are high-speed interrupt and isochronous transfers at 24.576 Megabytes/second.

Although the low-speed bus speed is 1.5 Megabits/sec., the fastest guaranteed delivery for the data in a single transfer is 8 bytes every 10 milliseconds, or just 800 bytes/sec.

Developing a Device

Designing a USB product for PCs involves both getting the peripheral up and running and developing or obtaining PC software needed to communicate with the peripheral.

Elements in the Link

A USB peripheral needs all of the following:

- A controller chip with a USB interface.
- Code in the peripheral to carry out the USB communications.
- Whatever hardware and code the peripheral needs to carry out its other functions (processing data, reading inputs, writing to outputs).
- A host that supports USB.

- Device-driver software on the host to enable applications to communicate with the peripheral.
- If the peripheral isn't a standard type supported by the operating system, the host must have application software to enable users to access the peripheral. For standard peripheral types such as a mouse, keyboard, or disk drive, you don't need custom application software, though you may want to write a test application.

Tools for Developing

To develop a USB peripheral, you need the following tools:

- An assembler or compiler to create the device firmware (the code that runs inside the device's controller chip).
- A device programmer or development kit that enables you to store the assembled or compiled code in the controller's program memory.
- A programming language and development environment on the host for writing and debugging the host software. The host software may include a device driver or filter driver and/or application code. To write a device driver, you'll need Microsoft's Windows Driver Development Kit (DDK).
- Also recommended are a monitor program for debugging the device firmware and a protocol analyzer to enable viewing USB traffic.

Steps in Developing a Project

For a project of any size, you'll want to create the project in modules and get each piece working before moving on to the next. In writing the firmware, you can begin by writing just enough code to enable Windows to detect and enumerate the device. When that code is working, you can move on to exchanging small blocks of data with applications. From there you can add specific code for your application. The steps in project development include initial decisions, enumerating, and exchanging data.

Initial Decisions

Before you begin the developing, you need to gather data and make some decisions:

1. Specify the requirements of your device. For the USB interface, how much data does it need to transfer, and how fast? Do you need error correcting? How much power will the device draw? What else does the device need to do?

2. From your requirements, decide whether the PC will communicate with the peripheral using Windows' built-in drivers, a generic device driver from another source, or a custom driver.

3. Select a controller chip that matches your requirements.

Enumerating

Here's what you need to do to get Windows to enumerate your device:

1. Write the code the controller chip needs to be enumerated by its host. The details vary with the chip, but every chip must be able send a series of descriptors to the host. The descriptors are data structures that describe the device's USB capabilities and how they'll be used. The chip must have program code or hardware that decodes and responds to the requests that the host sends and other events that occur when the host enumerates the device. Chip vendors generally provide example code that you can modify. A few controllers can enumerate with no user code required.

2. Identify or create a device driver and INF (information) file so that Windows can identify the device and assign a driver. The INF file is a text file that names the driver the device will use on the host computer. If your device fits a class supported by Windows, you may be able to use an INF file included with Windows.

3. If necessary, design and build a circuit to enable testing the chip and your firmware. In many cases, you can use a development board available from the chip's manufacturer.

4. Load the code into the device and plug the device into the host's bus. Windows should enumerate the device, adding it to the Control Panel and identifying it correctly.

5. Debug and repeat as needed!

Exchanging Data

When the device enumerates successfully, you can add components and code to enable the device to carry out its intended function. If needed, write application code to communicate with and test the device. When the code is debugged, you're ready to program the code into the chip and test on your final hardware.

But before you begin, it's useful to know a little more about how the host enumerates and transfers data with devices, so you can make the right choices about controller chips and drivers. This is the purpose of the following chapters.