Includes **USB 2.0**

USB COMPLETE

SECOND EDITION

Everything You Need to Develop Custom USB Peripherals

With firmware tips & host code in Visual Basic and Visual C++

JAN AXELSON

author of *Parallel Port Complete* and *Serial Port Complete*

USB Complete

Everything You Need to Develop Custom USB Peripherals

Second Edition

Jan Axelson

Lakeview Research
Madison, WI 53704

Published by Lakeview Research
Cover by Rattray Design. Cover Photo by Bill Bilsley Photography.
Index by Broccoli Information Management

Lakeview Research
5310 Chinook Ln.
Madison, WI 53704
USA

Phone: 608-241-5824
Fax: 608-241-5848
Email: info@Lvr.com
Web: http://www.Lvr.com

14 13 12 11 10 9 8 7 6 5 4 3 2

ISBN 0-9650819-5-8 Printed and bound in the United States of America

Table of Contents

Introduction

The Universal Serial Bus (USB) is a fast and flexible interface for connecting devices to computers. Every new PC has at least a couple of USB ports. The interface is versatile enough to use with standard peripherals like keyboards and disk drives as well as more specialized devices, including one-of-a-kind designs. USB is designed from the ground up to be easy for end users, with no user configuring required in hardware or software.

In short, USB is very different from the legacy interfaces it's replacing. A USB device may use any of four transfer types and three speeds. On attaching to a PC, a device must respond to a series of requests that enable the PC to learn about the device and establish communications with it. In the PC, every device must have a low-level driver to manage communications between applications and the system's USB drivers.

Developing a USB device and the software that communicates with it requires knowing something about how USB works and how the PC's operating system implements the interface. In addition, the right choice of con-

troller chip, device class, and tools and techniques can go a long way in avoiding snags and simplifying what needs to be done. This book is a guide for developers of USB devices. Its purpose is to introduce you to USB and to help get your project up and running and troublefree as quickly and easily as possible.

Who should read this book?

This book is for you if you want to know how to design a USB peripheral, or if you want to know how to communicate with USB peripherals from the applications you write. These are some of questions the book answers:

- *What is USB and how do peripherals use it to communicate with PCs?* There's a lot to the USB interface. Learning about it can be daunting at first. This book's focus is on the practical knowledge you'll need to complete your project.
- *How can I decide if my project should use a USB interface?* Maybe your design isn't suited for USB. I'll show you how to decide whether it is. If the answer is yes, I'll help you decide which of USB's speeds and transfer types to use.
- *How do I choose a USB controller chip for my peripheral design?* Every USB peripheral must contain an intelligent controller. There are dozens of controller chips designed for use in USB peripherals. In this book, I compare popular chip families and offer tips on how to decide, based on both your project's needs and your own background and preferences.
- *How do applications communicate with USB peripherals?* To communicate with a USB peripheral, a PC needs two things: a device driver that knows how to communicate with the PC's USB drivers and an application that knows how to talk to the device driver. Some peripherals can use drivers that are built into Windows. Others may require a custom driver. This book will show you when you can use Windows' built-in drivers and how to communicate with devices from Visual Basic and Visual C++ applications. You'll also find out what's involved in writing a device driver and what tools can help to speed up the process.

- *How do USB peripherals communicate?* USB peripherals typically use a combination of hardware and embedded code to communicate with PCs. In this book, I show how to write the code that enables Windows to identify a device and load the appropriate device driver, as well as the code required for exchanging data with applications.
- *How do I decide whether my peripheral can use bus power, or whether it needs its own supply?* A big advantage to USB is that many peripherals can be powered entirely from the bus. Find out whether your device can use this capability and how to manage power use, especially for devices that use battery power.
- *How can I be sure that my device will operate as smoothly as possible for its end users?* On the peripheral side, smooth operation requires understanding the specification's requirements and how the device can meet them. In the PC, proper operation requires a correctly structured information (INF) file that enables Windows to identify the device and software that knows how to communicate with the device as efficiently as possible. This book has information and examples to help with each of these.

What's new in the Second Edition?

In the months after the publication of the first edition of *USB Complete*, much happened in the world of USB, including the release of version 2.0 of the USB specification. USB 2.0 supports a bus rate of 480 Megabits per second, forty times faster than USB 1.1. This and other developments in hardware and software prompted this second edition of the book.

Rather than just tacking on a chapter about USB 2.0, I've revised the book from start to finish to reflect the changes in 2.0. By popular request, another addition is Visual C++ code to accompany the Visual Basic examples for application communications with USB devices. I've also expanded the material about Windows drivers and applications to include Windows 2000, and have added information on new controller chips and development tools. Other additions and updates are sprinkled throughout, many prompted by reader suggestions.

Is this book really complete?

Although the title is *USB Complete*, please don't expect this book to contain every possible fact about USB. That would take a library. The *Complete* in the title means that this book will guide you from knowing nothing about USB to developing all of the code required to get a USB peripheral up and communicating with a PC.

There are many other worthy topics related to USB, but limitations of time and space prevent me from including them all.

My focus is on communicating with Windows PCs. Although the basic principles are the same, I don't include details about how to communicate with peripherals on a Macintosh or a PC running Linux or other non-Windows operating systems.

I cover the basics of the device driver's responsibilities and what's involved in writing a driver, but the details of driver writing can easily fill a book (and in fact there are excellent—and lengthy—books on this topic). This book will help you decide when you need to write a custom driver and when and how to use a class driver included with Windows.

To understand the material in the book, it's helpful to have basic knowledge in a few areas. I assume you have some experience with digital logic, application programming for PCs and writing embedded code for peripherals. You don't have to know anything at all about USB.

Additional Resources, Updates, and Corrections

For more about using USB, I invite you to visit my USB Central page at Lakeview Research's website, *www.Lvr.com.* This is where you'll find complete code examples, updates, links to vendors, information and tools from other sources, as well as links to anything else I find that's relevant to developing USB products. If you have a suggestion, code, or other information that you'd like me to post or link to, let me know at *jan@lvr.com*.

In spite of my very best efforts, I know from experience that errors will slip through in this book. As they come to light, I'll document them and make a

list available at Lakeview Research's website. If you find an error in the book, please let me know and I'll add it.

Thanks!

USB is way too complicated to write about without help. I have many people to thank.

I owe an enormous thank you to my technical reviewers, who generously read my rough and rocky drafts and provided feedback that has improved the book enormously. (With that said, every error in this book is mine and mine alone.)

I thank Paul E. Berg of PEB Consulting; Brian Buchanan, Mark Hastings, Lane Hauck, Bijan Kamran, Kosta Koeman, Tim Williams, and Dave Wright of Cypress Semiconductor; Joshua Buergel of BSQUARE Inc.; Gary Crowell of Micron Technology; Fred Dart of Future Technology Devices International (FTDI); Dave Dowler; Mike Fahrion and the engineers at B&B Electronics; John M. Goodman, author of *Hard Disk Secrets, Peter Norton's Inside the PC, Memory Management for All of Us,* and other books; John Hyde, USB enthusiast and author of *USB Design by Example;* David James of 1Zero1 Technologies; Christer Johansson of High Tech Horizon; Jon Lueker of Intel Corporation; Bob Nathan of NCR Corporation; Robert Severson of USBMicro; and Craig R. Smith of Ford Motor Company, R&VT department.

Others I want to thank for their help in my researching and writing this book are Walter Banks of Byte Craft; Jason Bock; Michael DeVault of DeVaSys Embedded Systems; Pete Fowler, Joseph McCarthy, and Don Parkman of Cypress Semiconductor; Brad Markisohn of INDesign LLC; Daniel McClure of Tyco Electronics; Tawnee McMullen of Belkin Components; Rich Moran of RPM Systems Corporation; Dave Navarro of PowerBasic; and Amar Rajan of American Concepts Consulting.

I hope you find the book useful. Comments invited!

Jan Axelson, June 2001
jan@lvr.com

1

A Fresh Start

Computer hardware doesn't often get a chance to start fresh. Anything new usually has to remain compatible with whatever came before it. This is true of both computers and the peripherals that connect to them. Even the most revolutionary new peripheral has to use an interface supported by the computers it connects to.

But what if you had the chance to design a peripheral interface from scratch? What qualities and features would you include? It's likely that your wish list would include these:

- **Easy to use**, so there's no need to fiddle with configuration and setup details.
- **Fast**, so the interface doesn't become a bottleneck of slow communications.
- **Reliable**, so that errors are rare, with automatic correction of errors that do occur.
- **Flexible**, so many kinds of peripherals can use the interface.

- **Inexpensive**, so users (and the manufacturers who will build the interface into their products) don't balk at the price.
- **Power-conserving**, to save battery power on portable computers.
- **Supported by the operating system**, so developers don't have to struggle with writing low-level drivers for the peripherals that use the interface.

The good news is that you don't have to create this ideal interface, because the developers of the Universal Serial Bus (USB) have done it for you. USB was designed from the ground up to be a simple and efficient way to communicate with many types of peripherals, without the limitations and frustrations of existing interfaces.

Every new PC has a couple of USB ports that you can connect to a keyboard, mouse, scanners, external disk drives, printers, and standard and custom hardware of all kinds. Inexpensive hubs enable you to add more ports and peripherals as needed.

But one result of USB's ambitious goals has been challenges for the developers who design and program USB peripherals. A result of USB's versatility and ease of use is an interface that's more complicated than the interfaces it replaces. Plus, any new interface will have difficulties just because it's new. When USB first became available on PCs, Windows didn't yet include device drivers for all popular peripheral types. Protocol analyzers and other development tools couldn't begin to be designed until there was a specification to follow, so the selection of these was limited at first. Problems like these are now disappearing, and the advantages are increasing with the availability of more controller chips, new development tools, and improved operating-system support. This book will show you ways to get a USB peripheral up and running as simply and quickly as possible by making the best possible use of tools available now.

This chapter introduces USB, including its advantages and drawbacks, a look at what's involved in designing and programming a device with a USB interface, and a bit of the history behind the interface.

What USB Can Do

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

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 to 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 many kinds of peripherals. Instead of having a different connector type and supporting hardware for each peripheral, one interface serves many.

Automatic configuration. When a user connects a USB peripheral to a powered system, Windows automatically 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 locate and run a setup program or restart the system before using the peripheral.

No user settings. USB peripherals don't have user-selectable settings such as port addresses and interrupt-request (IRQ) lines. Available IRQ lines are in short supply on PCs, and not having to allocate one for a new peripheral is often reason enough to use USB.

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

Interface	Format	Number of Devices (maximum)	Length (maximum, feet)	Speed (maximum, bits/sec.)	Typical Use
USB	asynchronous serial	127	16 (or up to 96 ft. with 5 hubs)	1.5M, 12M, 480M	Mouse, keyboard, disk drive, modem, audio
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
IrDA	asynchronous serial infrared	2	6	115k	Printers, hand-held computers
Microwire	synchronous serial	8	10	2M	Microcotroller communications
SPI	synchronous serial	8	10	2.1M	Microcotroller communications
I^2C	synchronous serial	40	18	3.4M	Microcotroller communications
IEEE-1394 (FireWire)	serial	64	15	400M (increasing to 3.2G with IEEE-1394b	Video, mass storage
IEEE-488 (GPIB)	parallel	15	60	8M	Instrumentation
Ethernet	serial	1024	1600	10M/100M/ 1G	Networked PC
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

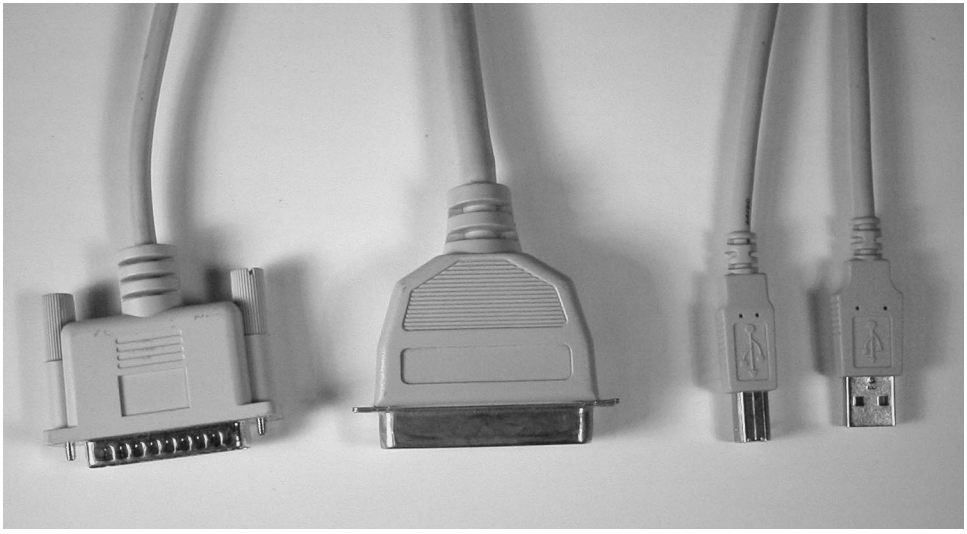

Figure 1-1: The two USB connectors (right) are much more compact than typical RS-232 serial (left) and Centronics parallel (center) connectors.

Frees hardware resources for other devices. Using USB for as many peripherals as possible frees up IRQ lines for the peripherals that do require them. The PC dedicates a series of port addresses and one interrupt-request (IRQ) line to the USB interface, but beyond this, individual peripherals don't require additional resources. In contrast, each non-USB peripheral requires dedicated port addresses, often an IRQ line, and sometimes an expansion slot (for a parallel-port card, for example).

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 at least two USB ports. You can expand the number of ports by connecting a USB hub to an existing port. Each hub has additional ports for attaching more peripherals or hubs.

Simple cables. The USB's cable connectors are keyed so you can't plug them in wrong. Cables can be as long as 5 meters. With hubs, a device can be as far as 30 meters from its host PC. Figure 1-1 shows that the USB connectors are small and compact in contrast to typical RS-232 and parallel

connectors. To ensure reliable operation, the specification includes detailed requirements that all cables and connectors must meet.

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

No power supply required (sometimes). The USB interface includes power-supply and ground lines that provide +5V from the computer's or hub's supply. A peripheral that requires up to 500 milliamperes can draw all of its power from the bus instead of having its own supply. In contrast, most other peripherals have to choose between including a power supply in the device or using a bulky and inconvenient external supply.

Speed

USB supports three bus speeds: high speed at 480 Megabits per second, full speed at 12 Megabits per second, and low speed at 1.5 Megabits per second. Every USB-capable PC supports low and full speeds. High speed was added in version 2.0 of the specification, and requires USB 2.0-capable hardware on the motherboard or an expansion card.

These speeds are signaling speeds, or the bit rates supported by the bus. The rates of data transfer that individual devices can expect are lower. In addition to data, the bus must carry status, control, and error-checking signals. Plus, multiple peripherals may be sharing the bus. The theoretical maximum rate for a single transfer is over 53 Megabytes per second at high speed, about 1.2 Megabytes per second at full speed, and 800 bytes per second at low speed.

Why three speeds? Low speed was included for two reasons. Low-speed peripherals can often be built more cheaply. And for mice and devices that require flexible cables, low-speed cables can be more flexible because they don't require as much shielding.

Full speed is comparable to or better than the speeds attainable with existing serial and parallel ports and can serve as a replacement for these.

After the release of USB 1.0, it became clear that a faster interface would be useful. Investigation showed that a speed increase of forty times was feasible while keeping the interface backwards-compatible with low- and full-speed devices. High speed became an option with the release of version 2.0 of the USB specification.

Reliability

The reliability of USB results from both the hardware design and the data-transfer protocols. The hardware specifications for USB drivers, receivers, and cables eliminate most noise that could otherwise cause data errors. In addition, the USB protocol enables detecting of data errors and notifying the sender so it can retransmit. The detecting, notifying, and retransmitting are typically done in hardware and don't require any programming or user intervention.

Low Cost

Even though USB is more complex than earlier interfaces, its components and cables are inexpensive. A device with a USB interface is likely to cost the same or less than its equivalent with an older interface. For very low-cost peripherals, the low-speed option has less stringent hardware requirements that may reduce the cost further.

Low Power Consumption

Power-saving circuits and code automatically power down USB peripherals when not in use, yet keep them ready to respond when needed. In addition to the environmental benefits of reduced power consumption, this feature is especially useful on battery-powered computers where every milliampere counts.

Benefits for Developers

The above advantages for users are also important to hardware designers and programmers. The advantages make users eager to use USB peripherals, so there's no need to fear wasting time developing for an unpopular interface. And many of the user advantages 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 also has advantages that benefit developers specifically. The developers include the hardware designers who select components and design the circuits, the programmers who write the software that communicates with USB peripherals, and the programmers who write the embedded code inside the peripherals.

The benefits to developers result from the flexibility built into the USB protocol, the support in the controller chips and operating system, and the fact that the interface isn't controlled by a single vendor. Although users aren't likely to be aware of these benefits, they'll enjoy the result, which is inexpensive, trouble-free, and feature-rich peripherals.

Flexibility

USB's four transfer types and three speeds make it 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 a transfer rate or maximum time between transfers. These abilities are especially welcome under Windows, where accessing peripherals in real time is often a challenge. The operating system, device drivers, and application software can still introduce unavoidable delays, but 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 signals 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, the limitation of having just five inputs was an obstacle. (Eventually the interface was expanded to allow eight

bits of input.) USB makes no such assumptions and is suitable for just about any device type.

For communicating with common device types such as printers and modems, there are USB classes with defined device requirements and protocols. This saves developers from having to re-invent these.

Operating System Support

Windows 98 was the first Windows operating system to reliably support USB, and its successors, including Windows 2000 and Windows Me, support USB as well. This book focuses on Windows programming for PCs, but other computers and operating systems also have USB support. On Apple's iMac, the only peripheral connectors are USB. Other Macintoshes also support USB, and support is in progress for Linux, NetBSD, and FreeBSD.

However, a claim of operating-system support can mean many things. The level of support can vary! At the most fundamental level, an operating system that supports USB must do three things:

- Detect when a device is attached to or 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 host 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 software device drivers that enable application programmers to access devices by calling functions supported by the operating system. If the operating system doesn't include a device driver appropriate for a specific peripheral, the peripheral vendor has to provide one.

Microsoft has added class drivers with each release of Windows. Device types with included drivers now include human interface devices (keyboards, mice, joysticks), audio devices, modems, still-image cameras and scanners, printers, and mass-storage devices. A filter driver can support

device-specific features and abilities. Applications use Applications Program Interface (API) functions or other operating-system components to communicate with the device drivers.

In the future, Windows will likely include support for more device classes. In the meantime, some chip vendors provide drivers that developers can use with their chips, either as-is or with minimal modifications.

USB device drivers use the new Win32 Driver Model (WDM), which defines an architecture for drivers that run under Windows 98, Windows 2000, Windows Me, and future Windows editions. The aim is to enable developers to support all of the operating systems with a single driver. The reality is that some devices still require two, though similar, WDM drivers, one for Windows 98/Windows Me and one for Windows 2000.

Because Windows includes low-level drivers that handle communications with the USB hardware, writing a USB device driver is easier than writing a driver for devices that use other interfaces.

Peripheral Support

On the peripheral side, each USB device's hardware must include a controller chip that handles the details of USB communications. Some controllers are complete microcomputers that include a CPU and memory to store device-specific code that runs inside the peripheral. Others handle only USB-specific tasks, with a data bus that connects to another microcontroller that performs non-USB related functions and communicates with the USB controller as needed.

The peripheral is responsible for responding to requests to send and receive configuration data, and for reading and writing other data when requested. In some chips, some functions are microcoded in hardware and don't need to be programmed.

Many USB controllers are based on popular architectures such as Intel's 8051, with added circuits and machine codes to support USB. If you're already familiar with a chip architecture that has a USB-capable variant, there's no need to learn an entirely new architecture in order to use USB.

Most peripheral manufacturers provide sample code for their chips. Using this code as a starting point for your own developing can give you a quick start.

USB Implementers Forum

Unlike other interfaces, where you're pretty much on your own when it comes to getting a design up and running, USB offers plenty of help via the USB Implementers Forum, Inc. (USB-IF) and its website (*www.usb.org*). The Forum is the non-profit corporation founded by the companies that developed the USB specification. The Forum's mission is to support the advancement and adoption of USB technology.

To that end, the Forum offers information, tools, and testing. The information includes the specification documents, white papers that delve into specific topics in detail, FAQs, and a web board where developers can post and answer questions on any USB-related topic. The tools include software and hardware to help in developing and testing products. Testing includes developing compliance tests to verify proper operation, holding compliance workshops where developers can have their products tested, and granting the rights to use the USB Logo on products that pass the tests.

It's Not Perfect

All of USB's advantages mean that it's a good candidate for use with many peripherals. But one interface can't do it all.

User Challenges

From the user's perspective, the downside to USB includes lack of support in older hardware and operating systems, speed and distance limits that make USB impractical for some uses, and problems with some products due to difficulties experienced by the developers of early USB products.

Lack of Support for Legacy Hardware

Older ("legacy") computers and peripherals don't have USB ports. If you want to connect a non-USB peripheral to a USB port, a solution is a con-

verter 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. However, the converter solution is useful only for peripherals that use conventional protocols supported by the converter's device driver. For example, a parallel-port converter is likely to support printers but not other peripheral types.

If you want to use a USB peripheral with a PC that doesn't support USB, the solution is to add USB capabilities to the PC. This requires two things: USB host-controller hardware and 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 version of Windows should be Windows 98 or later. A few peripherals have drivers for use with later releases of Windows 95, but it's best not to count on these being available. If the hardware doesn't meet Windows 98's minimum requirements, it will need upgrades. The upgrades may end up costing more than a new system with USB, so replacing the system may be the best option.

If upgrading the PC to support USB isn't feasible, what about using a converter to translate the peripheral's USB interface to the PC's RS-232, parallel, or other interface? Interface converters are generally designed for use between a USB port on a PC and a peripheral with a legacy interface. A converter for the other direction would be much more complicated because the peripheral would have to contain the host-controller hardware and code that normally resides in the PC. So a converter isn't normally an option when the PC has the legacy interface.

Even on new systems, users may occasionally run applications on older operating systems such as MS-DOS. But the drivers that Windows 98 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, the reality is that few peripherals provide one.

However, for the mouse and keyboard, which are standard, essential peripherals, the system's BIOS is likely to include support to ensure that the peripheral is usable any time, including from within DOS, from the BIOS

screens that you can view on bootup, and from Windows' Safe mode (used in system troubleshooting). If there is no BIOS or other support, the system will need to have an old-style keyboard interface and a spare keyboard for these uses.

Speed Limits

USB is versatile, but it's not designed to do everything. USB's high speed makes it competitive with the IEEE-1394 (Firewire) interface's 400 Megabits per second, but IEEE-1394b will be faster still, at 3.2 Gigabytes per second.

Distance Limits

USB was designed as a desktop bus, with the expectation that peripherals would be relatively close at hand. A cable segment can be as long as 5 meters. Other interfaces, such as RS-232, RS-485, 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, using 6 cable segments of 5 meters each.

To extend the range beyond this, 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

The assumption that USB is a desktop bus also means that every USB system has a host computer to manage the bus communications. Peripherals can't talk to each other directly. All communications are to or from the host computer. Other interfaces, such as IEEE-1394, allow direct peripheral-to-peripheral communications.

USB provides a partial solution with USB On-The-Go, introduced in 2001 in a supplement to the 2.0 specification. USB On-The-Go defines a host computer with reduced capabilities, suitable for use in embedded devices that need to connect to a single USB peripheral.

Products with Problems

When USB works, it's great. But the reality is that some USB products don't work as well as they should. When something misbehaves, the result can be an inability to communicate with a peripheral or an application or system crash. The source of the problem may be in hardware or software, in the PC or in the peripheral. Problems like these are a result of USB's complexity and newness combined with inadequate testing.

But there are plenty of products that do perform exactly as they should. The problems are diminishing as the operating-system support has improved and developers have become more familiar with USB.

Developer Challenges

From the developer's perspective, the main downside to USB is the increased complexity of the programming. Bugs in the USB hardware in the peripheral or PC can also slow project development and cause problems after a product is released. However, these problems are also diminishing as the operating-system support increases, more chips and development tools are available, and everyone gains more experience.

Protocol Complexity

To program a USB peripheral, you need to know a fair amount about the USB's protocols (the rules for exchanging data on the bus). The controller chips handle much of the communications automatically, but they still must be programmed, and this requires the knowledge to write the programs and the tools to do the programming. Chips vary in how much support they require to perform USB communications. 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 and no device driver required on the PC (just direct port reads and writes).

Evolving Support in the Operating System

Windows includes class drivers that enable applications to communicate with some devices. This is great if you can design your device to use one of the provided drivers. If not, in many cases you can use or adapt a driver provided by the controller-chip vendor, so you don't have write a driver from scratch. Several vendors offer toolkits that make the job of writing USB drivers easier.

Hardware Bugs

Some early host-controller hardware wasn't bugfree, and some peripheral chips have had problems as well. In most cases, the manufacturers make fixes available with new drivers or coding workarounds. The way to keep on top of these problems is to choose your hardware carefully and visit manufacturers' websites for the latest information and fixes.

Fees

The USB Implementers Forum provides the USB specification, related documents, software for compliance testing, and much more, all for free on its website. Anyone can develop USB software without paying a licensing fee.

However, anyone who sells a device with a USB interface must obtain legal access to use a Vendor ID. The administrative fee for obtaining a Vendor ID from the Forum is $1500. Or if you join the Forum at $2500/year, the Vendor ID is free, along with many other benefits such as compliance workshops. The Vendor ID and a Product ID assigned by the vendor are embedded in each device to identify it to the operating system. The fee is no problem for developers of high-volume products, but it can be an impediment to developers for the hobbyist market who expect to sell only small quantities of inexpensive devices. Some chip manufacturers will assign their Vendor ID and a block of Product IDs to customers for use with the manufacturer's chips.

History

To understand what USB is all about, it helps to know a little history. The main reason that new interfaces don't come around very often is that existing interfaces have the irresistible pull of all of the existing peripherals that users don't want to scrap. Also, using an existing interface saves the time and expense of designing something new. 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 computer power and the number of peripherals have increased, the older interfaces have became a bottleneck of slow communications, with limited options for expansion.

The Motivation for Change

A break with tradition is justified when the desire for enhancements overshadows the inconvenience and expense of changing. This is the situation that prompted the development of USB. The result is a versatile interface that can replace existing interfaces to standard and custom peripherals on computers of all types.

In the past, development of a new interface was often the work of a single company. Hewlett Packard developed the HP Interface Bus (HPIB), which came to be known as the GPIB (general-purpose interface bus) for lab equipment, and the Centronics Data Computer Corporation popularized a printer interface that is still referred to as the Centronics interface.

But an interface controlled by a single company isn't ideal. The company may forbid others from using the interface, or charge licensing fees. Even if the interface is freely available, a company may be reluctant to commit its products to an interface controlled by another company who may be a competitor and may change the interface without warning.

For these reasons, more recent interfaces are often the product of a collaboration of manufacturers who share a common interest. In some cases, an

organization like the IEEE (Institute of Electrical and Electronics Engineers) or TIA (Telecommunications Industry Association) sponsors committees to develop specifications and publishes the results. In fact, many of the older manufacturers' standards have been taken over by these organizations. The IEEE-1284 standard evolved from the Centronics interface, and the GPIB was the basis for IEEE-488.

In other cases, the developers of a standard form a new organization to release the standard and handle other development issues. This is the approach used for 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. All have agreed to make the specification available without charge (which is a refreshing change from the standards published by other organizations). The USB Implementers Forum's website has the latest versions of all USB specifications and other information for both developers and end users.

An early specification with many USB-like features was the ACCESS.bus sponsored by Philips and Digital Equipment Corporation, who made it available as an open standard. ACCESS.bus was in turn derived from the I^2C synchronous serial bus. Although the electrical interface is different, many of the functions and features are a lot like what ended up in USB.

ACCESS.bus was designed for interfacing keyboards, pointing devices, and other devices at speeds of 100 kilobits per second. The bus supports up to 125 devices and 10-meter cables. Devices are hot-pluggable. The cable includes +5V and ground wires. Classes are defined for keyboards, pointing devices (called locators), monitor/display control and text devices. Unlike USB, ACCESS.bus uses open-collector drivers, with one data wire and one clock wire.

ACCESS.bus never caught on with PCs, but is still used in other applications, including smart battery control.

The Specification's Release

Release 1.0 of the USB specification in January 1996 followed several years of development and preliminary releases. The 1.1 release is dated September 1998. USB 1.1 fixed problems identified in release 1.0 and added one new transfer type (Interrupt OUT). In this book, *1.x* refers to USB 1.0 and 1.1. April 2000 saw the release of USB 2.0 with the new high-speed option. An Engineering Change Notice (ECN) in December 2000 provided corrections and defined a new mini-B connector.

Although companies may begin designing products while a specification is still under development, by necessity, the availability of products on the market lags the specification's release.

USB capability first became available on PCs with the release of Windows 95's OEM Service Release 2. There were at least two editions of this release, OSR 2.1 and 2.5. Neither was available directly to retail customers. They were sold only to vendors who installed 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. A service pack for Windows 98 and the release of 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 have USB support built in, but its successor, Windows 2000, does. Windows 98's successor, Windows Me, also supports USB. Generally, Windows 2000 is more stable and is targeted for business users, while Windows 98 and Windows Me are more flexible and targeted for home users.

Following these editions is Windows XP, which is based on the Windows 2000 kernel but includes editions for both home and business users, with the goal of replacing both Windows 98/Windows Me and Windows 2000.

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, and Windows XP, 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.

USB 2.0

A big step in USB's evolution was version 2.0, whose main added feature is support for *much* faster transfers. The original hope when researching the new high speed was a 20-times increase in speed, but studies and tests showed that this estimate was low. In the end, a 40-times increase was found to be feasible, for a bus speed of 480 Megabits per second. This makes USB much more attractive for peripherals such as printers, scanners, drives, and even video.

USB 2.0 is backwards compatible with USB 1.1. Version 2.0 peripherals can use the same connectors and cables as 1.x peripherals. To use the new, higher speed, peripherals must connect to 2.0-compliant hosts and hubs. 2.0 hosts and hubs can also communicate with 1.x peripherals. A 2.0-compliant hub with a slower peripheral attached will translate as needed between the peripheral's speed and high speed. This increases the hub's complexity but makes good use of the bus time without requiring different hubs for different speeds.

USB versus IEEE-1394

The other major interface choice for new peripherals is IEEE-1394. Apple Computer's implementation of the interface is called Firewire. USB and IEEE-1394 take complimentary approaches, with IEEE-1394 being faster and more flexible, but more expensive. IEEE-1394 is best suited for video and other links where speed is essential or a host PC isn't available. USB is best suited for typical peripherals such as keyboards, printers, scanners, and disk drives as well as low- to moderate-speed, cost-sensitive applications. For many devices, either interface would work.

With USB, a single host controls communications with many peripherals. The host handles most of the complexity, so the peripherals' electronics can be relatively simple and inexpensive. IEEE-1394 uses a peer-to-peer model, where peripherals can communicate with each other directly. A single communication can also be directed to multiple receivers. The result is a more flexible interface, but the peripherals' electronics are more complex and expensive.

IEEE-1394's 400 Megabits per second is more than 30 times faster than USB 1.x's 12 Megabits per second. As USB is getting faster with version 2.0, IEEE-1394 is getting faster with the proposed IEEE-1394.b. Its 3.2 Gigabits per second is over six times faster than USB 2.0's 480 Megabits per second.