
CHAPTER 1

ADDING I/O DEVICES TO A MODERN PC

The personal computer (PC) has been around for a very long time—I think that Noah must have had one on his Ark! The first IBM PC was announced in 1981, and since then we’ve all wanted to add hardware to our PCs. The power of the PC has grown; the number of tasks we want the PC to accomplish has grown; and the number of devices we want to connect to the PC has grown. But until recently there has been a practical limit on the number of available ports to which we could connect devices.

Enter the Universal Serial Bus, USB. You may have heard USB referred to as “the best thing” to happen to the personal computer for some time. The list of USB features is impressive:

- Hot-pluggable: I/O devices can be added while the PC is running.
- Ease of use: I/O device attachment is recognized by the PC and appropriate device drivers, and configuration is done automatically.
- Single connector type: all devices plug into the same socket type.
- High performance: up to 480 Mbps transfer rate.
- Three speed choices: match I/O device speed to one of the standard speeds for optimal design.
- Up to 126 devices: there is no practical limit to I/O device expandability.
- Power supplied by cable: most devices will not need an additional power source.
- Power management: devices automatically power down when not in use.
- Error detection and recovery: errors are detected and transactions are retried to ensure that data is delivered reliably.
- External to the PC: there is no need to open the PC or design cards that must be installed in the PC.

Acknowledging the benefits of USB and designing an I/O device interface are two different things, however. Until now, you’ve had only the USB Specification to read for technical details. A specification by its nature usually provides few, if any, implementation examples. So, even after reading the specification, you still might not know how to design a simple I/O port for USB, let alone design a telephone or a video-based I/O device.

THE REASONS FOR THIS BOOK

Hands-on Examples

Most USB books, including the specification, position USB as a technical wonder, which it is. But these books lack practical, how-to examples that include schematics and code. That's where this book comes in. I designed it to help you add I/O devices to your PC, and I have tried to make it as much of a “cookbook” as I could. (I use the term “PC” to refer to Intel-based personal computers running an operating system that supports USB. I use Windows[†] 98 and Windows 2000 in most of the examples, and cover other operating systems in Chapter 16.) All the examples are fully documented, and prototype boards are available for you to build upon and expand your ideas and solutions. A variety of vendor solutions are demonstrated, so that you can choose the solution closest to your application.

- Most of the PC host software examples are implemented in Visual Basic[†] and Visual C++[†], so that you can choose the language you are most familiar with.
- The microcontroller examples are mainly in MCS[®]51 assembler code, but some of the more complex examples are implemented in C; I chose the MCS 51 architecture because many manufacturers have used it as the basis for their intelligent USB controllers.
- The MCS 51 family has an uncomplicated architecture with few programming “tricks,” so the examples should be easy to port to another microcontroller family.

I do not repeat sections from the USB specification, but have included it on the companion CD-ROM for the interested reader.

This book describes how to connect a PC host to the usual PC-type peripherals (such as scanners, proximity detectors, keypads, and printers), and also how to connect to everyday items such as lights, switches, motors, temperature sensors, speakers, video components, and telephones.

You will soon discover that adding I/O devices to a modern PC host is both easy and fun. The range of devices is limited only by your imagination.

How Much Technical Background Do You Need?

I assume that you have some fundamental electronics and programming skills, but I don't expect these to be your major field. Instead, I assume that you want to *use* the PC to do something useful. This book takes a practical approach to adding devices to a PC. This task is much easier now than it was with previous generations of personal computers. Today we connect to an external, standard USB socket and don't even have to open up the case of the computer—a much more civilized approach, and one far less prone to error.

Focus of the Text

This book focuses on I/O device design. It also covers PC host applications and driver software that will let us view and control our I/O devices. There is a lot of USB software on a PC host—we'll see that the implementation of the Win32 Driver Model (WDM) in the Windows family of operating systems uses a layered approach so that we can access USB features and services at a level suitable for our application. One of Intel's goals in working with Microsoft on their USB software was to eliminate the need for most users to write *any* OS-level software; many of the PC host software examples in this book use only applications-level software. When a device driver needs to be written, I cover this in greater detail.

When USB was being developed, a base assumption was the creation of “easy, low-cost I/O devices.” The resulting standard is asymmetric with most of the complexity on the PC host side. The operating system and available host controller silicon take care of this complexity, so we need deal with only the so-called “easy part,” the I/O device. Hub design is also presented, but this is mainly from an I/O device point of view.

THE MODERN PC: A SHORT HISTORY

What is a modern PC? And if you have an “old” PC, how do you make it modern? This section gives a brief history of the modern PC, and introduces USB as a solution to simple, low-cost I/O expansion.

When IBM announced the PC, the company documented the PC internals in their typically thorough style. The availability of this information, along with a strong desire to add hardware to the PC, allowed many people to develop plug-in cards so that the PC could monitor and even control their environment. Initially, the slots inside the IBM PC didn’t even have a name. The name ISA, for Industry Standard Architecture, was coined in the early 1980s. There were no official guidelines on how a PC should be expanded, so cards made by vendor A could not be used at the same time as cards made by vendor B. It took until the early 1990s for the ISA bus to be documented (*ISA & EISA Theory and Operation* by Edward Solari is the classic text for this information). Some developers took a lower-risk approach and created attachments that plugged into the serial and parallel ports. But this port resource was quickly depleted, and when more serial and parallel ports were added via plug-in cards, the internal design of the PC could not support enough interrupts, Direct Memory Access (DMA) channels, or other system resources. Sometimes the interactions were subtle and failed only under certain conditions. The bandwidth of these data paths in and out of the PC was also quite low. A better solution was required.

The PCI Bus

Many PC and PC component vendors, including Compaq, Digital Equipment, NCR, IBM, and Intel Corporation, worked together to define a high-bandwidth expansion bus that was eventually called the Peripheral Components Interconnect bus, or PCI. This bus definition included configuration information and controls to alleviate the vendor-to-vendor conflicts of ISA. The PCI bus was included beside the ISA bus inside the PC host. Software support for automatic configuration was added to Windows so that PCI cards became easy to install. This was the start of “Plug and Play,” an industry-wide initiative that governs the way add-in hardware should identify itself.

PCI was an instant success for high-bandwidth devices, but was seen by many as excessive and complex for the simpler I/O devices. The PCI interface components typically cost more than a simple I/O device. Another main disadvantage was the location of the PCI connectors; just like those of their ISA predecessor, they were inside the PC case. This meant that the case had to be opened, clamping screws undone, boards plugged exactly right into connectors that were difficult to identify, and then the system reassembled. This was discouraging for many would-be users because of their fears that the PC could be easily broken by these actions. PCI was a better solution but did not address “easy expansion of simple I/O.”

Several PC-industry leaders worked together to define an external expansion bus. Driven by customer demand, it was essential that this bus be simple and low cost. Consumer focus groups demanded that PC expansion be as easy as connecting a VCR to a television—you should not need to set switches or run software, and you must be allowed to plug and swap cables while the equipment is turned on.

Easy to Use and Low Cost

With the goal of designing an architecture that would be easy to use and low cost to implement, industry leaders turned their attention to a high-speed serial bus that came to be called the Universal Serial Bus (USB). Serial was preferred over parallel because the cables would be cheaper, and it would be easier to implement dynamic configuration. “Dynamic configuration” meant that the I/O subsystem could be extended or reconfigured by swapping cables while the PC was running. Rebooting a modern PC takes several minutes, so this situation had to be avoided in every solution.

A typical configuration included one PC host and many I/O devices. To reduce the overall system costs, a master-slave implementation was chosen for USB. The PC host would be the master controlling all traffic on the serial bus—the additional silicon complexity required to implement the controlling protocols would need to be implemented only once in the host. The slave I/O device could then be made simpler and, therefore, cheaper. This asymmetric solution means that the two ends of a cable are *not* equal—we need to know which end connects to the master and which end to the slave! The terminology adopted by the USB specification is “upstream” (toward the PC host) and “downstream” (toward the I/O device); see Figure 1-1. The upstream end of the cable controls the protocol and instructs the downstream end to reply at defined times.

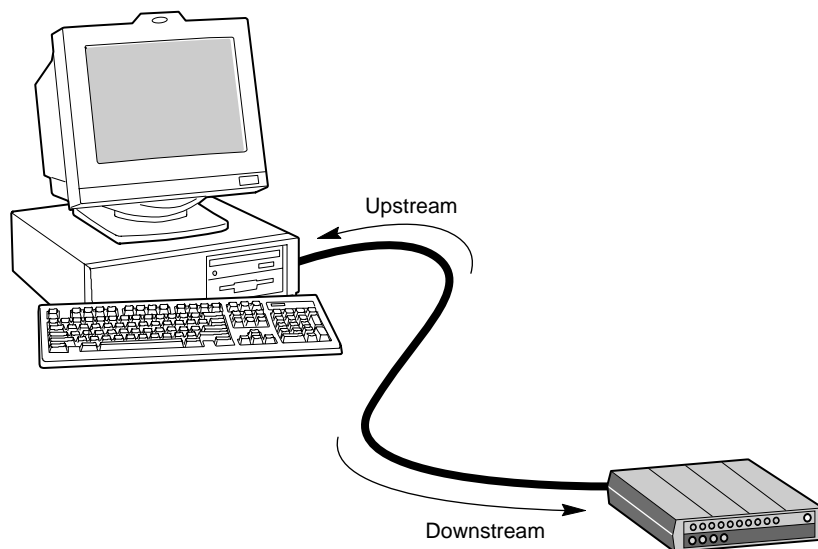


Figure 1-1. The two ends of a USB cable are not the same!

Another major decision that would support both design goals was to officially supply power from the PC host. Operating modes that specify differing power limits are defined, and a good understanding of this feature allows cheaper and easier-to-use I/O devices to be built. You can, for example, eliminate the cost of a power supply in simpler I/O devices and ease the design of more complex I/O devices. Some implementation trade-offs in the I/O devices can also be made when the system specifies minimum and maximum power levels.

USB TERMINOLOGY

The USB specification introduced new terms that are used throughout the USB literature. This section introduces those terms and presents an overview. Later chapters discuss each element in detail.

A typical configuration has a single PC host with multiple **devices** interconnected by USB **cables** (Figure 1-2). The PC host has an embedded **hub**, also called the root hub, which typically contains two or more USB ports.

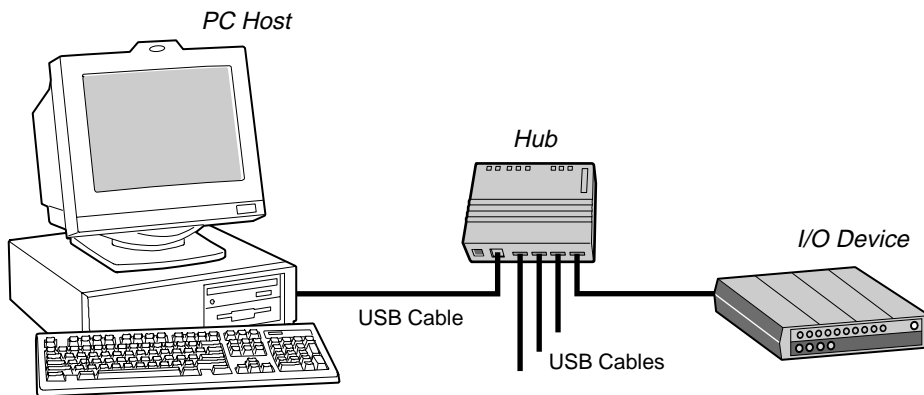


Figure 1-2. Standard USB designations

Device configurations range from simple to complex:

- **Hub:** If a device contains only additional downstream USB ports, then it is called simply a hub.
- **I/O device:** An I/O device adds a capability to the PC host. It has a single upstream connection and interacts with the real world to create or consume data on behalf of the PC host.
- **Compound device:** If a device includes both I/O and hub functionality, it is called a compound device (for example, a keyboard that includes additional USB downstream ports).
- **Composite device:** If a single device implements two or more sets of diverse functions, it is called a composite device (for example, a telephone with keypad and dual audio channels).

As far as the PC host is concerned, devices are the important feature, and as many as 126 devices can be interconnected using external hubs up to five levels deep (Figure 1-3).

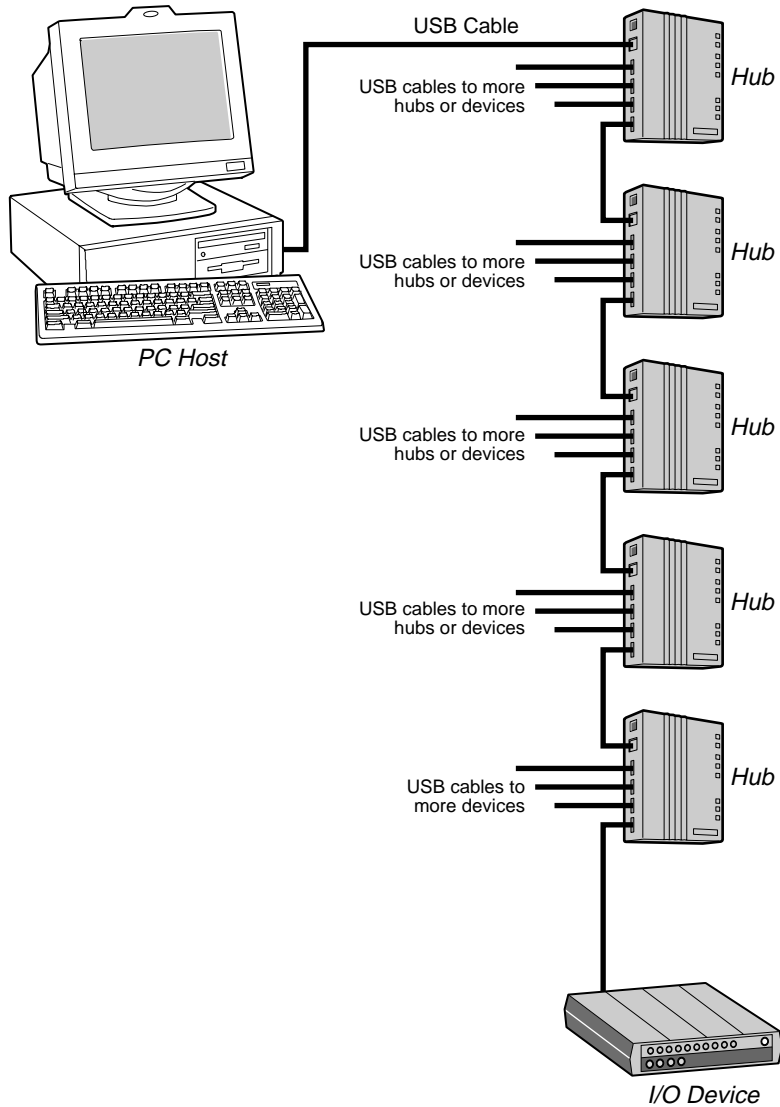


Figure 1-3. Devices can be nested five hub levels deep

USB now supports three device speeds. The USB specification initially defined low speed (LS) at 1.5 Mbps and full speed (FS) at 12 Mbps. A new high speed (HS) at 480 Mbps was recently added to Version 2.0 of the specification. Low-speed devices are the cheapest to manufacture and are adequate for supporting low-data-rate devices such as mice, keyboards, and a wealth of other equipment designed to interact with people. The addition of a high-speed data rate enables high-bandwidth devices—such as full-color page scanners and printers, and mass storage devices—to be designed and added to a PC that supports a high-speed hub.

We shall see that a high-speed hub is more complex than a full-speed hub, and it is anticipated that it will cost a little more initially. A typical system, therefore, will have a mixture of HS and FS hubs, as shown in Figure 1-4.

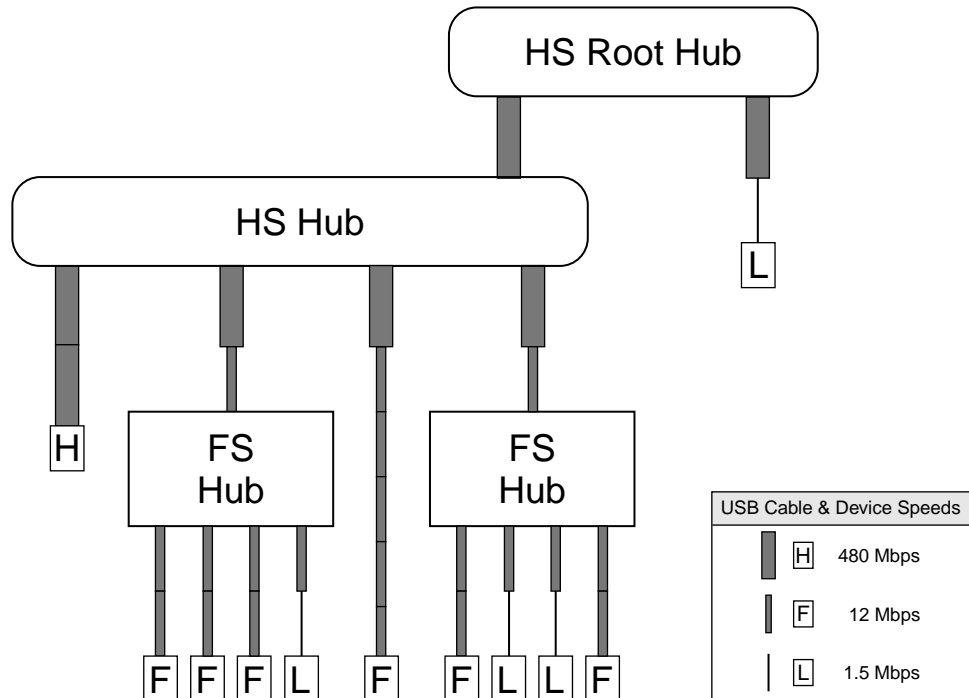


Figure 1-4. A typical system using two hub types

Note that connecting a low-speed device onto the root hub, as I have shown, is a legal connection; it does, however, preclude connecting high- and full-speed devices to this node. It would be better, from a system perspective, to connect the low-speed device onto an FS hub.

In the figure, the cables are labeled in different ways for illustrative purposes only—in fact, all of the USB cables look and operate the same. Note that the HS hubs communicate with each other at 480 Mbps—they also communicate with HS devices at 480 Mbps. An FS hub can be connected to an HS hub, but it will only communicate at 12 Mbps. However, multiple FS hubs may be attached to an HS hub and each will receive their own 12 Mbps channel—the bandwidth is not shared at this level. If an FS hub is connected to another FS hub, then the 12 Mbps channel is shared, as defined by the original USB specification.

Any speed of I/O device may be connected to any downstream port of any hub. The speed of the I/O device will determine the data rate on the cable to the hub. The only poor choice is attaching an LS device to an HS hub, as this will limit the speed of the USB segment to 1.5 Mbps. Fortunately, the operating system will recognize this suboptimal connection during enumeration of the device, and will recommend that you detach the LS device and reattach it to an FS hub.

PC Host

A typical configuration has a single PC host. (You can interconnect two PC hosts using USB, and this special case is discussed in Chapter 10.) The PC host runs a USB-aware operating system software that supports two distinct functions—initialization and run time.

The USB initialization software is active at all times and not just during PC host power-on. Because the initialization software is always active, USB devices can be added and removed at any time. Once a device is added to a PC host, the device is **enumerated** by the USB initialization software and assigned a unique identifier that is used during run time. (This enumeration process is described in detail in Chapter 3.)

Figure 1-5 shows how the USB host software is layered; layering supports many different software solutions. A **class** is a grouping of devices, with similar characteristics, that can be controlled by a generic class device driver. Examples of classes include mass-storage devices, communications devices, audio devices, and human-interface devices. A single I/O device can belong to multiple classes. If a device fits neatly into one or more of these predefined classes, then you don't need to write operating system software, such as a device driver. The software structure allows you to focus more of your I/O device design effort on the function and usability of the I/O device and less on the inner workings of an operating system. Vendor-defined commands are also available for those who want specific functionality for their device only.

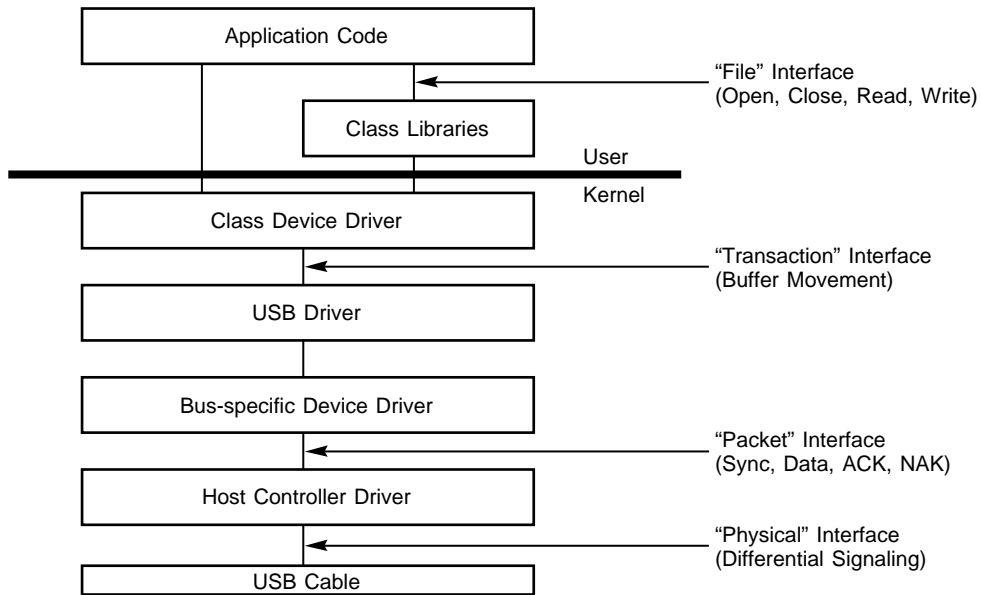


Figure 1-5. PC host software for USB is defined in layers

USB Cable

A USB cable transports both power supply and data signals.

The power supplied by the USB cable is an important benefit of the USB specification. A simpler I/O device can rely on the USB cable for all its power needs and will not require the traditional “black brick” plugged into the wall. The power resource is carefully managed by USB, with the hub device playing the major role.

A hub or I/O device can be self-powered or bus-powered.

- Self-powered is the traditional approach in which the hub or I/O device has an additional power cable attached to it.
- A bus-powered device relies solely on the USB cable for its power needs and is often less expensive.

The USB cable connectors were specifically designed with the power pins longer than the signal pins so that power would always be applied before signals. Two different connector types were defined, to ensure that illegal configurations could not be made (Figure 1-6). An “A”-type connector defines the downstream end of the cable, and a “B”-type connector defines the upstream end.



Figure 1-6. Different connectors define the ends of a USB cable

The design center for a USB application is a “desktop PC external expansion bus,” so a length of 5 meters was chosen as the maximum cable length between a hub and a device. With five levels of hubs, this means that any device will be located within a 30 meter radius of the PC Host. Many applications require a greater distance, and the industry has responded with “repeater/extender” products (these are covered in Chapter 9).

A cable with a 480 Mbps or even a 12 Mbps data rate makes a good antenna, unfortunately, so cable shielding is required. Shielding adds to the cost of the cable, but is not required for the 1.5 Mbps signaling rate, so this approach is cheaper. The USB specification allows a system to use a mixture of 480 Mbps, 12 Mbps and 1.5 Mbps devices on two cable types (shielded and unshielded), but I recommend that you *only use shielded cables*.

The USB specification defines differential signaling on its data wires to reduce the effects of induced system noise. The USB bus is a point-to-point connection that operates in half-duplex mode. There is no “direction” bit that identifies the current transmitter or receiver—this information is embedded in the bus protocol. While this does simplify the implementation, it also makes it difficult to build a bus repeater or a bus isolation unit (other solutions to these requirements are presented in Chapter 9). The fundamental element of communication on the USB bus is a **packet**, and defined sequences of packets are used to build a robust communications channel. Chapter 2 describes packet types and the communications protocol in detail.

Hub Device

The hub has two major roles: power management and signal distribution.

An external hub has one upstream connection and multiple downstream connections. The USB specification does not limit the number of downstream connections, but seven seems to be the practical limit. The most popular hub size has four.

A hub can be self-powered or bus-powered:

- The self-powered approach is recommended, in which the hub has an additional power cable attached to it. If the hub is self-powered, it can provide up to 500 mA to each of its downstream ports.
- A bus-powered device relies solely on the USB cable for its power needs. If the hub is bus-powered, then it has a maximum of 500 mA available. The hub will use 100 mA for itself, and have only 100 mA available for each of four downstream USB ports (unless the socket is suspended). The situation is worse if the hub has more than four downstream ports, in which case less than 100 mA will be available for each port.

Because of the power limitation, I only recommend bus-powered hubs in exceptional situations. A bus-powered hub can support only 100 mA on each of its downstream USB ports. Although this is enough to enumerate all I/O devices, it is typically not enough for most I/O devices to operate. The current version of Windows 98 does not flash lights or give audible warning if an enumerated device can't operate because of lack of hub power, so the user is left wondering why the newly attached device doesn't work—not a good user experience. This situation can get worse: Nothing prevents a user from adding a second bus-powered hub onto a port of the first bus-powered hub; the second hub uses all of the 100 mA for itself and cannot even supply enough power to enumerate additional I/O devices. This confuses a user even more. Note that Windows 98, and Windows 2000, will alert the user if a high-power device is attached to a low-power hub, and will recommend a better system configuration.

When first attached to a USB socket, any I/O device (which includes a hub) can always expect 100 mA to be available. The device uses this power to operate during the enumeration stage. The device may *not* use more than 100 mA until it is configured; if it does, the power source on the USB socket will be removed and an error sent to the PC host. If the I/O device requires more than 100 mA for run-time operation, it can request up to 500 mA from the hub. If the hub can supply this power, it does so; otherwise, the I/O device is not configured, and an error message is sent to the PC host. If the I/O device requires more than 500 mA for run-time operation, the device must be self-powered and will need a power cord.

It is worth the design effort to reduce your I/O device power to less than 500 mA, because this eliminates the need for an external power source.

An I/O device is required to power itself down, or suspend, when there is no activity on the USB bus. In the USB specification, “no activity” is defined to mean no bus signaling for 3 ms. The maximum amount of power that may be drawn during a suspend is 0.5 mA; this is a larger design challenge, as we shall see in later chapters.

I/O Device

A PC host creates data for, or consumes data from, the real world, as shown in Figure 1-7. A scanner is a good example of a data-creating I/O device, and a printer is a good example of a data consuming I/O device.

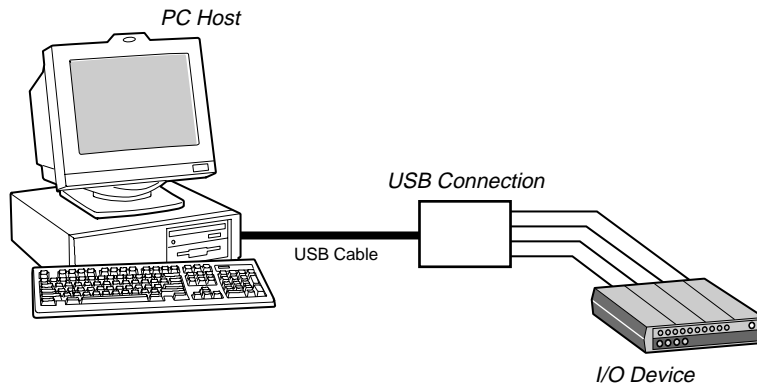


Figure 1-7. An I/O device connects the USB to the real world

A single I/O device could support both a data creator and a data consumer, so a different software driver may be required for each of these two distinct tasks. The software (or logical) view of the USB connection is shown in Figure 1-8. The logical view is deliberately general in nature, so that all types of real-world connections can be made. This diagram is best explained from the bottom up.

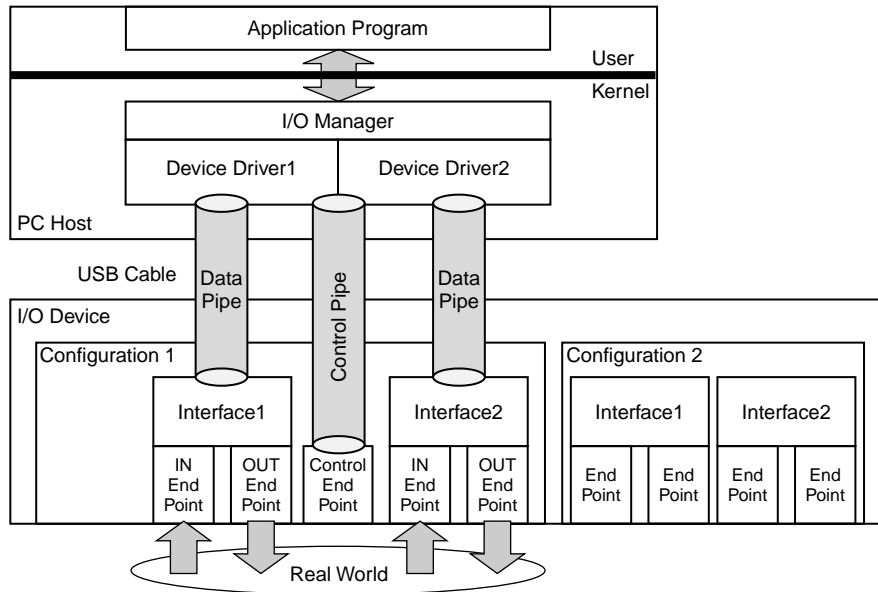


Figure 1-8. Logical view of an I/O device

The term **endpoint** is used to describe a point at which data enters or leaves a USB system. An IN endpoint is a data creator, and an OUT endpoint is a data consumer. Note that the data direction is relative to the PC host—if you remember that the PC host is the “master” controlling all data movements, then the data direction is easy to understand.

A typical real-world connection may need multiple IN and/or OUT endpoints to implement a reliable data-delivery scheme. This collection of endpoints is called an **interface**, and is directly related to a real-world connection. The operating system will have a software driver that corresponds to each interface. The operating system uses the term **pipe** to describe the logical connection between a software driver on the PC host and the interface on the I/O device. There is always a one-to-one mapping between software driver pipes and interfaces.

Real-world devices may have multiple interfaces—a telephone, for example, has a keypad interface and an audio interface. The operating system will manage the keypad and audio using two separate device drivers. This decomposition of complex devices into smaller logical interfaces means that building-block software elements can be quickly used to manage these complex devices. We don’t have to invest the time and effort to write a special telephone device driver—we can be operational with the class drivers already included in the operating system. All the interfaces run concurrently.

A collection of interfaces is called a **configuration**, and only one configuration can be active at a time. A configuration defines the attributes and features of a specific model. Using configurations allows a single USB connection to serve many different roles, and the modularity of this system solution saves in development time and support costs.

ADDING USB TO AN “OLD” PC

All desktop PCs and laptops manufactured today contain a USB host controller and one or more downstream ports. An older PC or laptop may be upgraded easily with a PCI add-in card or PCMCIA card, as shown in Figure 1-9. You should also upgrade your operating system software to ensure that you have the best support for USB.



Figure 1-9. Adding USB capability to a PC host

Not all host controllers are equal! The first differentiation is between a V2.0 (480 Mbps) and a V1.1 (12 Mbps) root hub—this is clearly marked on the product. There are also two types of V1.1 add-in boards—some contain a single host controller and thus provide 12 Mbps added USB bandwidth, while others (the one shown in the center of Figure 1-9 is from Belkin) provide four host controllers or 48 Mbps of added bandwidth for your PC host.

IMPACT OF USB ON PC HOST

As well as simplifying I/O device design, USB can have a major impact on the PC host design. All of the basic features of a PC are implemented in highly integrated components, and a view inside today's PC shows a motherboard similar to Figure 1-10; most of the physical space is taken up by PCI and/or ISA slots!

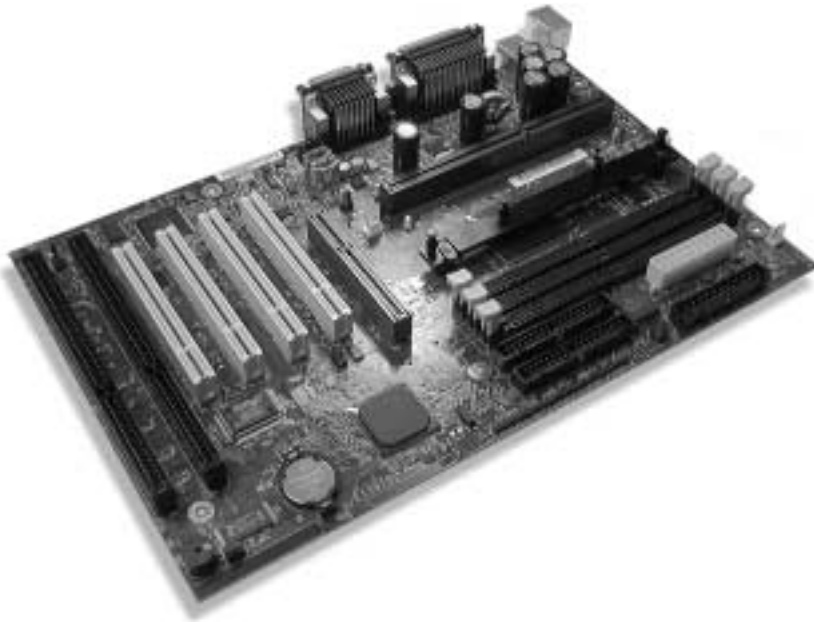


Figure 1-10. Typical PC host motherboard

If all I/O expansion could be external, then there would be no requirement for expansion slots on the motherboard. As well as saving physical space, we would also save in the power supply design. ISA slots have a power allowance of 25 watts, and PCI slots have a power allowance of 10 watts. This means that a PC with no I/O expansion slots (“slotless”) requires only a 50–80-watt power supply rather than a 180- or 240-watt unit. Less heat will be generated inside the system, so a single cooling fan will be adequate. Figure 1-11 shows an example of a slotless motherboard that is half the size of a standard motherboard.



Figure 1-11. Slotless motherboard

Figure 1-12 shows two concept platforms built around a slotless motherboard. Each platform contains the same motherboard and the same peripherals, a 42-GB hard drive, and a DVD-ROM, and they differ only in the physical placement of the subunits. These PCs are very quiet and don't look like typical "beige-box" PCs. They will be attractive for several new applications, such as home use.



Figure 1-12. Concept PCs built with slotless motherboard

These concept platforms also introduce new I/O paradigms. Many peripherals available today attach to the serial or parallel ports of the traditional PC. A peripheral device expander (Figure 1-13) could be used to attach these peripherals to a concept platform. Software drivers on the PC host would redirect I/O requests to this expander, so that no changes in applications software would be required. The upper device shown in Figure 1-13 is from Fujitsu Ltd.; it additionally includes a three-port USB hub, an Ethernet connection, and audio connections. The lower device is from Belkin. Although targeted at USB-enabled laptop computers, these device expanders would complement a slotless desktop implementation.



Figure 1-13. Integrated hub and I/O expander products

IMPACT OF USB ON EMBEDDED PCs AND OTHER DEVICES

USB was originally conceived as a desktop PC expansion bus. A large spectrum of devices have been developed; this, in turn, is expanding the role of the personal computer. The Windows 95, now 98, operating system was the first to include software support for USB but this, too, has expanded to include Windows 2000, Linux, MacOS, Windows CE and a variety of other operating systems.

Embedded PCs on industrial buses, such as the compact PCI from Radisys shown in Figure 1-14, were also quick to adopt the USB expansion connector. A large array of data acquisition and process control devices, also shown in Figure 1-14, were developed for this industrial application range.



Courtesy of Radisys Corporation and National Instruments

Figure 1-14. Embedded PC with industrial USB devices.

Embedded PCs typically run a real-time operating system (such as Wind River's VxWorks), so a variety of vendors now supply USB class and device drivers in this context as well. The volume economics of the PC industry is reducing the price of all USB-related products, and this is further expanding USB's application range.

A portable/handheld segment is also being fueled by USB. A variety of data collection and data storage devices have been developed, and newer reprogrammable devices (such as the Journada from Hewlett Packard and the iPAQ 3650 from Compaq, shown in Figure 1-15) include a USB connection to allow data synchronization with a desktop PC. All such devices are USB slaves that require the PC host to implement the data transfer.



Figure 1-15. The range of available portable/handheld USB devices is widening

A new connector type, the mini-B, has been approved by the USB Implementors Forum for use in portable devices. This connector, shown in Figure 1-16, is about 12% of the size of a standard B connector.



Courtesy of Molex

Figure 1-16. A mini-B connector for portable devices is available

CHAPTER SUMMARY

The Universal Serial Bus was the result of a tremendous amount of cooperative industry effort, and its inclusion (with operating system support) defines a modern PC. If your old notebook or desktop PC does not have USB sockets, these can be added via a PCMCIA or PCI card.

PC host software is layered, and interfaces are defined to allow the simple addition of application programs. The higher up the USB software stack we go, the farther we are from the actual I/O devices we are controlling. This abstraction allows the software and hardware to be developed on different schedules by different people. The hardware and software communicate via standardized interfaces that have the added benefit that they can be swapped-out or upgraded independently.

Portable/handheld products now include USB as the standard method of communication—the possibilities for USB-based devices continue to broaden, limited only by one’s creativity.

My primary goal in writing this book is to enable you to quickly take advantage of this ever-increasing opportunity. The rigid USB specification defines exactly how a USB I/O device can correctly interoperate with a wide spectrum of USB hosts; this book adds the practical details of what must be done to create a compliant USB I/O device.

Read through the examples—they start simple and gradually become more complex. Choose the example closest to what you would like to build, and augment from there.

You’ll soon discover that USB is both easy and fun.