Chapter 21: Serial Ports

Traditionally, serial communications has been the long distance choice—the best and sometimes only way of getting your message out. Recent advances in technology and standardization have made serial links the new choice for tying your PC to its peripherals. In coming years, serial communication may become synonymous with PC expansion.[/block]

- Background
  - Clocking
  - Frames
  - Packets
    - Background
    - Error Handling
    - History
- RS-232C
  - Electrical Operation
  - Connectors
    - 25-Pin
    - 9-Pin
    - Motherboard Headers
  - Signals
    - Definitions
  - Cables
    - Straight Through Cables
    - Adapter Cables
    - Crossover Cables
  - UARTs

            - 8250
            - 16450
            - 16550A
            - Register Function
            - Buffer Control
            - Identifying UARTs
            - Enhanced Serial Ports
        - Logical Interface
            - Port Names
            - Interrupts
    - ACCESS.bus
        - Architecture
        - Signaling
            - Transfers
            - Arbitration
            - Messages
        - Addresses
        - Connections
    - IrDA
        - History
        - Overview
        - Physical Layer
            - Infrared Light
            - Data Rates
            - Pulse Width
            - Modulation
            - Bit Stuffing
            - Format
            - Aborted Frames
            - Interference Suppression
        - Link Access Protocol
            - Primary and Secondary Stations
            - Frame Types
            - Addressing
            - Error Detection
        - Link Management Protocol
    - Universal Serial Bus
        - Background
        - Connectors
        - Cable
        - Data Coding

        - Protocol
            - Token Packets
            - Data Packets
            - Handshake Packets
    - IEEE-1394
        - Background
        - Performance
        - Timing
        - Setup
        - Arbitration
        - Architecture
            - Bus Management Layer
            - Transaction Layer
            - Link Layer
            - Physical Layer
        - Cabling

21

Serial Ports

For almost two decades, the serial port has been the least common denominator of computer communications, an escape route for your long distance messages but one burdened by its own ball and chain. Today that situation is changing. Where once there was but one "serial port," today several serial communication standards vie for your attention.

Serial communication once hobbled your PC with data rates out of the dark ages. The classic serial port was a carryover from a previous generation of technology. Its low speed was a sad match for the quick pulse of the PC. It was like having a medieval scribe ink out your PC's owner's manual in glorious Gothic script, a trial of your patience that took little advantage of current technology. (Then again, some PC documentation arrives so late you just *might* suspect some scribe to be scrawling it out with quill and oxgall ink.) While PCs generated millions of characters per second, classic serial ports doled out a few hundred or thousand in the same time.

New serial technologies kick communications back into high gear. They are quick enough not only to transfer text messages but to move your voice digitally or even handle full motion video in real time. The also add the versatility of going wireless so you don't have to tie yourself to your desk with a tangle of communications cables.

Today engineers have five chief choices for serial communications between your PC and other devices. The classic serial port (best known by its official EIA standard designation, RS-232C), ACCESS.bus, the IrDA optical connection, the Universal Serial Bus, and P1394.

The least common denominator is the RS-232C port, standard equipment on nearly every PC since 1984. Throughout the first decade and a half of personal computing, serial port meant only RS-232C. But the standard is even older than the first PCs, having been a telephone system standard long before. And, like most of the carryovers from early technology, RS-232C brought its own baggage—a speed limit more severe than a stern first grade teacher who believes that rulers are for discipline rather than measurement.

ACCESS.bus is an inexpensive but low speed serial connection to link multiple undemanding devices with your PC. Rather than speed, its advantage over RS-232C is versatility. It can connect more devices to your PC than all the RS-232C ports you or your system could stand. Moreover, it is a simple standard, one without a confusion of cables and connectors.

IrDA gives the RS-232C standard a new medium, sending signals through the air instead of wires. Using infrared signals exactly like those of television remote controls, IrDA allows you to transfer files between your notebook and desktop PC without wrestling with a cable connection. If the relatively new standard takes hold, you may also link your notebook PC to your printer or other peripherals with invisible light beams. The principal drawback is IrDA's RS-232C heritage. The maximum IrDA data rate matches the low speed of the RS-232C signals.

Universal Serial Bus is the PC generation's answer to serial communications. Unlike the RS-232C based serial systems that are designed to link two devices, USB acts as a true bus that can link as many as 127 devices to your PC without worries about matching connectors and cables (although it remains a wire based design). Speed takes a quantum leap over RS-232C with a peak data rate of 12 megabits per second as well as a low speed mode that operates at 1.5 megabits per second. In order to make the "universal" in the name a reality, the design goal of the new standard aims low: USB was designed to be a low cost interface, cheap enough for every PC.

P1394 pushes serial technology further still, with a maximum data rate of 100 megabits per second currently and rates as high as 400 megabits per second envisioned. More expensive to implement than USB, it fits with the new SCSI-3 scheme of things and

offers a reliable means of linking high speed peripherals such as hard disks and real time video systems to PCs. Table 21.1 compares the characteristics of some of the most common serial port standards.

Table 21.1. A Comparison of Serial Interfaces

Standard	*Data rate (current)*	*Medium*	*Devices per port*
RS-232C	115,200 bps	Twisted pair	1
ACCESS.bus	100 Kbps	4-wire shielded cable	125
IrDA	4 Mbps	Optical	126
USB	12 Mbps	Special 4-wire cable	127
IEEE 1394	100 Mbps	Special 6-wire cable	16

Despite the differences among these standards, all have a common basis. They treat data one-dimensionally, as a long stream or series of bits. From this common ground, each goes its own direction. At heart, however, all involve the same repackaging of data to make it fit a channel with a single stream of data.

Background

No matter the name and standard, all serial ports are the same, at least functionally. Each takes the 8, 16, or 32 parallel bits your computer exchanges across its data bus and turns them sideways—from a broadside of digital blips into a pulse chain that can walk the plank, single file. This form of communication earns its name "serial" because the individual bits of information are transferred in a long series.

The change marks a significant difference in coding. The bits of parallel data are coded by their position. That is, the designation of the bus line they travel confers value. The most significant bit travels down the line designated for the most significant signal. With a serial port, the significance is awarded by timing. The position of a bit in a pulse string gives it its value. The later in the string, the more important the bit.

In a perfect world, a single circuit—nothing more than two wires, a signal line and a ground—would be all that was necessary to move this serial signal from one place to another without further ado. Of course, a perfect world would also have fairies and other benevolent spirits to help usher the data along and protect it from all the evil imps and energies lurking about trying to debase and disgrace the singular purity of serial transfer.

The world is, alas, not perfect, and the world of computers even less so. Many misfortunes can befall the vulnerable serial data bit as it crawls through its connection. One of the bits of a full byte of data may go astray, leaving a piece of data with a smaller value on arrival than it had at departure—a problem akin to shipping alcohol by a courier service operated by dipsomaniacs. With the vacancy in the data stream, all the other bits will slip up a place and assume new values. Or the opposite case—in the spirit of electronic camaraderie, an otherwise well meaning signal might adopt a stray bit like a child takes on a kitten, only to later discover the miracle of pregnancy and a progeny of errors that ripple through the communications stream, pushing all the bits backward. In either case, the prognosis is not good. With this elementary form of serial communications, one mistaken bit either way, and every byte that follows will be in error.

Establishing reliable serial communications means overcoming these bit error problems and many others as well. Thanks to some digital ingenuity, however, serial communications work and work well—well enough that you and your PC can depend on them.

Clocking

In computers, a serial signal is one in which the bits of data of the digital code are arranged in a series. They travel through their medium or connection one after another as a train of pulses. Put another way, the pattern that makes up the digital code stretches across the dimension of time rather than across the width of a data bus. Instead of the bits of the digital code getting their significance from their physical position in the lines of the data bus, the get their meaning from their position in time. Instead of traveling through eight distinct connections, a byte of data, for example, makes up a sequence of eight pulses in a serial communications system. Plot signal to time, and the serial connections turns things sideways from the way they would be inside your PC.

Do you detect a pattern here? Time, time, time. Serial ports make data communications a matter of timing. Defining and keeping time become critical issues in serial data exchanges.

Engineers split the universe of serial communications into two distinct forms, synchronous and asynchronous. The difference between them relates to how they deal with time.

Synchronous communications require the sending and receiving system—for our purposes, the PC and printer—to synchronize their actions. They share a common time base, a serial *clock.* This clock signal is passed between the two systems either as a

separate signal or by using the pulses of data in the data stream to define it. The serial transmitter and receiver can unambiguously identify each bit in the data stream by its relationship to the shared clock. Because each uses exactly the same clock, they can make the match based on timing alone.

In *asynchronous communications* the transmitter and receiver use separate clocks. Although the two clocks are supposed to be running at the same speed, they don't necessarily tell the same time. They are like your wristwatch and the clock on the town square. One or the other may be a few minutes faster even though both operate at essentially the same speed: a day has 24 hours for both.

An asynchronous communications system also relies on the timing of pulses to define the digital code. But they cannot look to their clocks for infallible guidance. A small error in timing can shift a bit a few positions, say from the least significant place to the most significant, which can drastically affect the meaning of the digital message.

If you've ever had a clock that kept bad time—for example, the CMOS clock inside your PC—you probably noticed that time errors are cumulative. They add up. If your clock is a minute off today, it will be two minutes off tomorrow. The longer time elapses, the more the difference in two clocks will be apparent. The corollary is also true: if you make a comparison over a short enough period, you won't notice a shift between two clocks even if they are running at quite different speeds.

Asynchronous communications banks on this fine slicing of time. By keeping intervals short, they can make two unsynchronized clocks act as if they were synchronized. The otherwise unsynchronized signals can identify the time relationships in the bits of a serial code.

Isochronous communications involve time critical data. Your PC uses information that is transferred isochronously in real time. That is, the data are meant for immediate display, typically in a continuous stream. The most common examples are video image data that must be displayed at the proper rate for smooth full motion video and digital audio data that produces sound. Isochronous transmissions may be made using any signaling scheme, be it synchronous or asynchronous. They usually differ from ordinary data transfers in that the system tolerates data errors. It compromises accuracy for the proper timing of information. Whereas error correction in a conventional data transfer may require the retransmission of packets containing errors, an isochronous transfer lets the errors pass through uncorrected. The underlying philosophy is that a bad pixel in an image is less objectionable than image frames that jerk because the flow of the data stream stops for the retransmission of bad packets.

Frames

The basic element of digital information in a serial communication system is the data *frame*. Think of the word as a time frame, the frame bracketing the information like a frame surrounds a window. The bits of the digital code are assigned their value in accordance with their position in the frame. In a synchronous serial communications system, the frame contains the bits of a digital code word. In asynchronous serial communications, the frame also contains a word of data, but it has a greater significance. It is also the time interval in which the clocks of the sending and receiving systems are assumed to be synchronized.

When an asynchronous receiver detects the start of a frame, it resets its clock and then uses its clock to define the significance of each bit in the digital code within the frame. At the start of the next frame, it resets its clock and starts timing the bits again.

The only problem with this system is that an asynchronous receiver needs to know when a frame begins and ends. Synchronous receivers can always look to the clock to know, but the asynchronous system has no such luxury. The trick to making asynchronous communications work is unambiguously defining the frame. Today's asynchronous systems use *start bits* to mark the beginning of a frame and *stop bits* to mark its end. In the middle are a group of *data bits*.

The start bit helps the asynchronous receiver find data in a sea of noise. In some systems, the start bit is given a special identity. In most asynchronous systems, it is twice the length of the other bits inside the frame. In others, the appearance of the bit itself is sufficient. After all, without data, you would expect no pulses. When any pulse pops up, you might expect it to be a start bit.

Each frame ends with one or more stop bits. They assure the receiver that the data in the frame is complete. Most asynchronous communication systems allow for one, one and a half, or two stop bits. Most systems use one because that length makes each frame shorter (which, in turn, means that it takes a shorter time to transmit).

The number of data bits in a frame varies widely. In most asynchronous systems, there will be from five to eight bits of data in each frame. If you plan to use a serial port to connect a modern serial device to your PC, your choices will usually be to use either seven bits or eight bits, the latter being the most popular.

In addition, the data bits in the frame may be augmented by error correction information called a parity bit, which fits between the last bit of data and the stop bit. In modern serial systems, any of five varieties of parity bits are sometimes used: odd, even, space, mark, and none.

The value of the parity bit is keyed to the data bits. The serial transmitter counts the number of digital ones in the data bits and determines whether this total is odd or even. In the odd parity scheme, the transmitter will turn on the parity bit (making it a digital one) only if the total number of digital ones in the data bits is odd. In even priority

systems, the parity bit is set as one only if the data bits contain an even number of digital ones. In mark parity, the parity bit is always a mark, a digital one. In space parity, the parity bit is always a space, a digital zero. With no parity, no parity bit is included in the digital frames, and the stop bits immediately follow the data bits.

By convention, the bits of serial data in each frame are sent least significant bit first. Subsequent bits follow in order of increasing significance. Figure 21.1 illustrates the contents of a single data frame that uses eight data bits and a single stop bit.

Figure 21.1 A serial data frame with eight data bits and one stop bit.

Packets

A frame corresponds to a single character. Taken alone, that's not a whole lot of information. A single character rarely suffices for anything except answering multiple choice tests. To make something meaningful, you combine a sequence of characters to form words and sentences.

The serial communications equivalent of a sentence is a *packet*. A packet is a standardized group of characters or frames that makes up the smallest unit that conveys information through the communications system.

Background

As the name implies, a packet is a container for a message, like a diplomatic packet or envelope. The packet holds the data. In addition, in most packetized systems, the packet also includes an address and, often, a description of its contents. Packets may also include extra data to assure the integrity of their contents—for example, an error detection or error correction scheme of some sort. Figure 21.2 shows a graphical representation of the constituents of a data packet.

Figure 21.2 Constituents of a typical data packet.

The exact constituents of a packet depend on the communication protocol. In general, however, all packets have much the same construction. They begin with a symbol or character string that allows systems listening in to the communication channel to recognize the bit pattern that follows as a packet.

Each packet bears an address that tells where it is bound. Devices listening in on the communication channel check the address. If it does not match their own or does not indicate that the packet is being broadcast to all devices—in other words, the packet wears an address equivalent to "occupant"—the device ignores the rest of the packet. Communications equipment is courteous enough not to listen in on messages meant for someone else.

Most packets include some kind of identifying information that tells the recipient what to do with the data. For example, a packet may bear a marker to distinguish commands from ordinary data.

The bulk of the packet is made from the data being transmitted. Packets vary in size and hence the amount of data that they may contain. Although there are no hard and fast limits, most packets range from 256 to 2048 bytes.

Error Handling

Because no communication channel is error free, most packets include error detection or error correction information. The principal behind error detection is simple: include duplicate or redundant information that you can compare to the original. Because communication errors are random, they are unlikely to affect both of two copies of the transmitted data. Compare two copies sent along and if they do not match, you can be sure one of them changed during transmission and became corrupted.

Many communications systems don't rely on complex error correction algorithms as are used in storage and high quality memory systems. Communications systems have a luxury storage systems do not; they can get a second chance. If an error occurs in transmission, the system can try again—and again—until an error free copy gets through.

As a function of communication protocol, packets are part of the software standard used by the communication system. Even so, they are essential to making the hardware—the entire communication system—work properly and reliably.

History

Nearly every serial communication system now uses packets of some kind. In retrospect, the idea of using packets seems natural, the logical way of organizing data. In fact, however, the concept of using packets as a method of reliably routing data

through communications systems rates as an invention with a clear-cut history. The first inkling of the concept of packet based communications dates back as early as 1960 when Paul Baran, working at RAND Corporation, conceived the idea of a redundant, packet-switched network. At the time nothing came of the idea because the chief telecommunications supplier in the United States, AT&T, regarded a communications system or network based on packets as unbuildable. AT&T preferred switching signals throughout its vast network, a technology with which the company had become familiar after nearly a century of development.

In 1965 Donald Watts Davies, working at the British National Physics Laboratory, independently conceived the idea of packetized communications, and it was he who coined the name "packet." Baran called them data blocks and his version of packet switching was "distributed adaptive message block switching." The direct ancestor of today's Internet, ARPAnet (see Chapter 22, "Modems"), development of which began in 1966, is usually considered the first successful packetized communication system.

RS-232C

The classic serial port in your PC wears a number of different names. IBM, in the spirit of bureaucracy, sanctions an excess of syllables, naming the connection an "asynchronous data communications port." Time pressed PC users clip that to "async port" or "comm" port. Officialdom bequeaths its own term. The variety of serial link accepted by the PC industry operates under a standard called RS-232C (one that was hammered out by an industry trade group, the Electronics Industry Association or EIA), so many folks call the common serial port by its numerical specification, an RS-232C port.

So far we've discussed time in the abstract. But serial communications must occur at very real data rates, and those rates must be the same at both ends of the serial connection, if just within a frame when transmissions are asynchronous. The speed at which devices exchange serial data is called the *bit rate,* and it is measured in the number of data bits that would be exchanged in a second if bits were sent continually. You've probably encountered these bit rates when using a modem. The PC industry uses bit rates in the following sequence: 150; 300; 600; 1200; 2400; 4800; 9600; 19,200; 38,400; 57,600; and 115,200.

This sequence results from both industry standards and the design of the original IBM PC. The PC developed its serial port bit rate by using an oscillator that operates at 1.8432 MHz and associated circuitry that reduces that frequency by a factor of 1,600 to a basic operating speed of 115,200 bits per second. For this base bit rate, a device called a *programmable divider* mathematically creates the lower bit rates used by serial ports. It develops the lower frequencies by dividing the starting rate by an integer. By using a

divisor of three, for example, the PC develops a bit rate of 38,400 (that is, 115200/3). Not all available divisors are used. For example, designers never set their circuits to divide by five.

You may have noticed that some modems use speeds not included in this sequence. For example, today's popular V.34 modems operate at a base speed of 28,800 bits per second. The modem generates this data rate internally. In general, you will connect your PC to the modem so that it communicates at a higher bit rate, and the modem repackages the data to fit the data rate it uses by compressing the data or telling your PC to halt the flow of information until it is ready to send more bits.

The accepted standard in asynchronous communications allows for a number of variables in the digital code used within each data frame. When you configure any serial device and an RS-232C port, you'll encounter all of these variables: speed, number of data bits, parity choices, and number of stop bits. The most important rule about choosing which values to use is that the transmitter and receiver—your PC and the serial device— must use exactly the same settings. Think of a serial communication as being an exchange of coded messages by two spies. If the recipient doesn't use the same secret decoder ring as the sender, he can't hope to make sense out of the message. If your serial peripheral isn't configured to understand the same settings your PC sends out in its serial signals, you can't possible hope to print anything sensible.

Normally you'll configure the bit rate of the serial port on a peripheral using DIP switches or the serial peripheral's menu system. How to make the settings will vary with the serial device you are installing, so you should check its instruction manual to be sure. To use a serial port as a DOS device, you must use the DOS MODE command to set its speed and other communication parameters. This setting only affects what you print from DOS. Most programs and other operating systems take direct control of the PC's serial ports when they need them, and these program override the values set using the MODE command. You adjust the bit rates (and other serial parameters) used by your programs as part of the setup procedures of your applications or operating system. In general you should set the fastest data that both ends of your connection will allow.

Electrical Operation

Serial signals have a definite disadvantage compared to parallel; bits move one at a time. At a given clock rate, fewer bits will travel through a serial link than a parallel one. The disadvantage is on the order of 12 to 1. When a parallel port moves a byte in a single cycle, a serial port take around a dozen—8 for the data bits, 1 for parity, 1 for stop, and 2 for start. That 9,600 bit-per-second serial connection actually moves text at about 800 character per second.

Compensating for this definite handicap, serial connections claim versatility. Their signals can go the distance. Not just the mile or so you can shoot out the signal from your standard serial port, but the thousands of miles you can make by modem—tied, of course, to that old serial port.

The trade-off is signal integrity for speed. As they travel down wires, digital pulses tend to blur. The electrical characteristics of wires tend to round off the sharp edges of pulses and extend their length. The farther a signal travels, the less defined it becomes until digital equipment has difficulty telling where the one pulse ends and the next begins. The more closely spaced the pulses are (and, hence, the higher the bit rate), the worse the problem becomes. By lowering the bit rate and extending the pulses and the time between them, the farther the signal can go before the pulses blend together. (Modems avoid part of this problem by converting digital signals to analog signals for the long haul. PC networks achieve length and speed by using special cables and signaling technologies.)

The question of how far a serial signal can reach depends on both the equipment and wire that you use. You can probably extend a 9,600 bps connection to a hundred feet or more. At a quarter mile, you'll probably be down to 1,200 or 300 bps (slower than even cheap printers can type).

Longer wires are cheaper with serial connections, too, a point not lost on system designers. Where a parallel cable requires 18 to 25 separate wires to carry its signals, a serial link makes do with three: one to carry signals from your PC to the serial peripheral, one to carry signals from the serial peripheral to the PC, and a common or ground signal that provides a return path for both.

The electrical signal on a serial cable is a rapidly switching voltage. Digital in nature, it has one of two states. In the communications industry, these states are termed space and mark like the polarity signals. *Space* is the absence of a bit, and *mark* is the presence of a bit. On the serial line, a space is a positive voltage, a mark is a negative voltage. In other words, when you're not sending data down a serial line, it has an overall positive voltage on it. Data will appear as a series of negative going pulses. The original design of the serial port specification called for the voltage to shift from a positive 12 volts to negative 12 volts. Because 12 volts is an uncommon potential in many PCs, the serial voltage often varies from positive 5 to negative 5 volts.

Connectors

The physical manifestation of a serial port is the connector that glowers on the rear panel of your PC. It is where you plug your serial peripheral into your computer. And it can be the root of all evil—or so it will seem after a number of long evenings during

which you valiantly try to make your serial device work with your PC only to have text disappear like phantoms at sunrise. Again, the principal problem with serial ports is the number of options that it allows designers. Serial ports can use either of two style of connectors, each of which has two options in signal assignment. Worse, some manufacturers venture bravely in their own directions with the all-important flow control signals. Sorting out all of these options is the most frustrating part of serial port configuration.

25-Pin

The basic serial port connector is called a 25-pin D-shell. It earns its name from having 25 connections arranged in two rows that are surrounded by a metal guide that takes the form of a rough letter D. The male variety of this connector—the one that actually has pins inside it—is normally used on PCs. Most, but hardly all, serial peripherals use the female connector (the one with holes instead of pins) for their serial ports. Although both serial and parallel ports use the same style 25-pin D-shell connectors, you can distinguish serial ports from parallel ports because on most PCs the latter use female connectors. Figure 21.3 shows the typical male serial port DB-25 connector that you'll find on the back of your PC.

Figure 21.3 The male DB25 connector used by serial ports on PCs.

Although the serial connector allows for 25 discrete signals, only a few of them are ever actually used. Serial systems may involve as few as three connections. At most, PC serial ports use ten different signals. Table 21.2 lists the names of these signals, their mnemonics, and the pins to which they are assigned in the standard 25-pin serial connector.

Table 21.2. 25-Pin Serial Connector Signal Assignments

Pin	*Function*	*Mnemonic*
1	Chassis ground	None
2	Transmit data	TXD
3	Receive data	RXD
4	Request to send	RTS
5	Clear to send	CTS
6	Data set ready	RTS
7	Signal ground	GND

8	Carrier detect	CD
20	Data terminal ready	DTR
22	Ring indicator	RI

Note that in the standard serial cable, signal ground (which is the return line for the data signals on pins 2 and 3) is separated from the chassis ground on pin one. The chassis ground pin is connected directly to the metal chassis or case of the equipment, much like the extra prong of a three-wire AC power cable, and provides the same protective function. It assures that the case of the two devices linked by the serial cable are at the same potential, which means you won't get a shock if you touch both at the same time. As wonderful as this connection sounds, it is often omitted from serial cables. On the other hand, the signal ground is a necessary signal that the serial link cannot work without. You should never connect the chassis ground to the signal ground.

9-Pin

If nothing else, using a 25-pin D-shell connector for a serial port is a waste of at least 15 pins. Most serial connections use fewer than the complete 10; some as few as 4 with hardware handshaking, 3 with software flow control. For the sake of standardization, the PC industry sacrificed the cost of the other unused pins for years until a larger—or smaller, depending on your point of view—problem arose: space. A serial port connector was too big to fit on the retaining brackets of expansion boards along with a parallel connector. In that all the pins in the parallel connector had an assigned function, the serial connector met its destiny and got miniaturized.

The problem arose when IBM attempted to put both sorts of ports on one board inside its Personal Computer AT when it was introduced in 1984. To cope with the small space available on the card retaining bracket, IBM eliminated all the unnecessary pins but kept the essential design of the connector the same. The result was an implementation of the standard serial port that uses a 9-pin D-shell connector. To trim the 10 connections to 9, IBM omitted the little used chassis ground connection.

As with the 25-pin variety of serial connector, the 9-pin serial jack on the back of PCs uses a male connector. This choice distinguishes it from the female 9-pin D-shell jacks used by early video adapters (the MDA, CGA, and EGA systems all used this style of connector). Figure 21.4 shows the 9-pin male connector that's used on some PCs for serial ports.

Figure 21.4 The male DB-9 plug used by AT-class serial devices.

Besides eliminating some pins, IBM also rearranged the signal assignments used in the miniaturized connector. Table 21.3 lists the signal assignments for the 9-pin serial connector introduced with the IBM PC-AT.

Table 21.3. IBM 9-Pin Serial Connector

Pin	*Function*	*Mnemonic*
1	Carrier detect	CD
2	Receive data	RXD
3	Transmit data	TXD
4	Data terminal ready	DTR
5	Signal Ground	GND
6	Data set ready	DSR
7	Request to send	RTS
8	Clear to send	CTS
9	Ring indicator	RI

Other than the rearrangement of signals, the 9-pin and 25-pin serial connectors are essentially the same. All the signals behave identically no matter the size of the connector on which it appears.

Motherboard Headers

When a serial port is incorporated into motherboard circuitry, the motherboard maker may provide either a D-shell connector on the rear edge of the board or a header from which you must run a cable to an external connector. The pin assignments on these motherboard headers usually conforms to that of a standard D-shell connector, allowing you to use a plain ribbon cable to make the connection.

Intel, however, opts for a different pin assignment on many of its motherboards. Table 21.4 lists the pin assignments of most Intel motherboards.

Table 21.4. Intel Motherboard Serial Port Header Pin Assignments

Motherboard header pin	*Corresponding 9-Pin D-shell Pin*	*Function*

1	1	Carrier detect
2	6	Data set ready
3	2	Receive data
4	7	Request to send
5	3	Transmit data
6	8	Clear to send
7	4	Data terminal ready
8	9	Ring indicator
9	5	Signal ground
10	No connection	No connection

Signals

Serial communication is an exchange of signals across the serial interface. These signals involve not just data but also the flow control signals that help keep the data flowing as fast as possible—but not too fast.

First we'll look at the signals and their flow in the kind of communication system for which the serial port was designed, linking a PC to a modem. Then we'll examine how attaching a serial peripheral to a serial port complicates matters and what you can do to make the connection work.

Definitions

The names of the signals on the various lines of the serial connector sound odd in today's PC-oriented lingo because the terminology originated in the communications industry. The names are more relevant to the realm of modems and vintage teletype equipment.

Serial terminology assumes that each end of a connection has a different type of equipment attached to it. One end has a *data terminal* connected to it. In the old days when the serial port was developed, a terminal was exactly that—a keyboard and a screen that translated typing into serial signals. Today a terminal is usually a PC. For reasons known but to those who revel in rolling their tongues across excess syllables,

the term Data Terminal Equipment is often substituted. To make matters even more complex, many discussions talk about *DTE* devices—which means exactly the same thing as "data terminals."

The other end of the connection had a *data set*, which corresponds to a modem. Often engineers substitute the more formal name Data Communication Equipment or talk about *DCE* devices.

The distinction between data terminals and data sets (or DTE and DCE devices) is important. Serial communications were originally designed to take place between one DTE and one DCE, and the signals used by the system are defined in those terms. Moreover, the types of RS-232C serial devices you wish to connect determine the kind of cable you *must* use.

Transmit Data

The serial data leaving the RS-232C port is called the *transmit data* line, which is usually abbreviated TXD. The signal on it comprises the long sequence of pulses generated by the UART in the serial port. The data terminal sends out this signal, and the data set listens to it.

Receive Data

The stream of bits going the other direction—that is, coming in from a distant serial port—goes through the receive data line (usually abbreviated RXD) to reach the input of the serial port's UART. The data terminal listens on this line for the data signal coming from the data set.

Data Terminal Read

When the data terminal is able to participate in communications, that is, it is turned on and in the proper operating mode, it signals its readiness to the data set by applying a positive voltage to the *data terminal ready* line, which is abbreviated as DTR.

Data Set Ready

When the data set is able to receive data, that is, it is turned on and in the proper operating mode, it signals its readiness by applying a positive voltage to the *data set ready* line, which is abbreviated as DSR. Because serial communications must be two way, the data terminal will not send out a data signal unless it sees the DSR signal coming from the data set.

Request To Send

When the data terminal is on and capable of receiving transmissions, it puts a positive voltage on its *request to send* line, usually abbreviated RTS. This signal tells the data set that it can send data to the data terminal. The absence of an RTS signal across the serial connection will prevent the data set from sending out serial data. This allows the data terminal to control the flow of the data set to it.

Clear To Send

The data set, too, needs to control the signal flow from the data terminal. The signal it uses is called *clear to send*, which is abbreviated CTS. The presence of the CTS in effect tells the data terminal that the coast is clear and the data terminal can blast data down the line. The absence of a CTS signal across the serial connection prevents the data terminal from sending out serial data.

Carrier Detect

The serial interface standard shows its roots in the communication industry with the *carrier detect* signal, which is usually abbreviated CD. This signal gives a modem, the typical data set, a means of signaling to the data terminal that it has made a connection with a distant modem. The signal says that the modem or data set has detected the carrier wave of another modem on the telephone line. In effect, the carrier detect signal gets sent to the data terminal to tell it that communications are possible. In some systems, the data terminal must see the carrier detect signal before it will engage in data exchange. Other systems simply ignore this signal.

Ring Indicator

Sometimes a data terminal has to get ready to communicate even before the flow of information begins. For example, you might want to switch your communications program into answer mode so that it can deal with an incoming call. The designers of the serial port provided such an early warning in the form of the *ring indicator* signal, which is usually abbreviated RI. When a modem serving as a data set detects ringing voltage—the low frequency, high voltage signal that makes telephone bells ring—on the telephone line to which it is connected, it activates the RI signal, which alerts the data terminal to what's going on. Although useful in setting up modem communications, you can regard the ring indicator signal as optional because its absence usually will not prevent the flow of serial data.

Signal Ground

All of the signals used in a serial port need a return path. The signal ground provides this return path. The single ground signal is the common return for all other signals on the serial interface. Its absence will prevent serial communications entirely.

Flow Control

This hierarchy of signals hints that serial communications can be a complex process. The primary complicating factor is handshaking or flow control. The designers of the serial interface recognized that some devices might not be able to accommodate information as fast as others could deliver it, so they built handshaking into the serial communications hardware using several special control signals to compensate.

This flow control signal become extremely important when you want to use a serial connection to a slow device such as a plotter. Simply put, plotters aren't as quick as PCs. As you sit around playing Freecell for the fourteenth hand while waiting for the blueprint of your dream house to roll out, that news comes as little surprise. Plotters are mechanical devices that work at mechanical speed. PCs are electronic roadrunners. A modern PC can draw a blueprint in its memory much quicker than your plotter can ink it on paper.

The temptation for your PC is to force feed serial devices, shooting data out like rice

puffs from a cannon. After the first few gulps, however, force fed serial devices choke. With a serial connection, the device might let the next salvo whiz right by. Your plotter may omit something important—like bedroom walls and bathroom plumbing—and leave large gaps in your plan (but, perhaps, making your future life somewhat more interesting). Flow control helps throttle down the onslaught of data to prevent such omissions.

The concept underlying flow control is the same as for parallel and other ports: your peripheral signals when it cannot accept more characters to stop the flow from your PC. When the peripheral is ready for more, it signals its availability to your PC. Where the traditional parallel port uses a simple hardware scheme of this handshaking, flow control for the serial port is a more complex issue. As with every other aspect of serial technology, flow control is a theme overwhelmed by variations.

The chief division in serial flow control is between hardware and software. *Hardware flow control* involves the use of special control lines that can be (but don't have to be) part of a serial connection. Your PC signals whether it is ready to accept more data by sending a signal down the appropriate wire. *Software flow control* involves the exchange of characters between PC and serial peripheral. One character tells the PC your peripheral is ready and another warns that it can't deal with more data. Both hardware and software flow control take more than one form. As a default, PC serial ports use hardware flow control (or hardware handshaking). Most serial peripherals do, too.

Hardware Flow Control

Several of the signals in the serial interface are specifically designed to help handle flow control. Rather than a simple on and off operation, however, they work together in an elaborate ritual.

The profusion of signals seems overkill for keeping a simple connection such as that with a plotter under control, and it is. The basic handshaking protocol for a serial interface is built around the needs of modem communications. Establishing a modem connection and maintaining the flow of data through it is one of the more complex flow control problems for a serial port. Even a relatively simple modem exchange involves about a dozen steps with a complex interplay of signals. The basic steps of the dance would go something like this:

1. The telephone rings when a remote modem wants to make a connection. The data set sends the ring indicator signal to the data terminal to warn of the incoming call.

2. The data terminal switches on or flips into the proper mode to engage in communications. It indicates its readiness by sending the data terminal ready signal to the data set.

3. Simultaneously, it activates its request to send line.

4. When the data set knows the data terminal is ready, it answers the phone and listens for the carrier of the other modem. If it hears the carrier, it sends out the carrier detect signal.

5. The data set negotiates a connection. When it is capable of sending data down the phone line, it activates the data set ready signal.

6. Simultaneously, it activates its clear to send line.

7. The data set relays bytes from the phone line to the data terminal through the receive data line.

8. The data terminal sends bytes to the data set (and thence the distant modem) through the transmit data line.

9. Because the phone line is typically slower than the data terminal-to-data set link, the data set quickly fills its internal buffer. It tells the data terminal to stop sending bytes by deactivating the clear to send line. When its buffer empties, it reactivates clear to send.

10. If the data terminal cannot handle incoming data, it deactivates its request to send line. When it can again accept data, it reactivates the request to send line.

11. The call ends. The carrier disappears, and the data set discontinues the carrier detect signal, clear to send signal, and data set ready signal.

12. Upon losing the carrier detect signal, the data terminal returns to its quiescent state, dropping its request to send and data terminal ready signals.

Underlying the serial dance are two rules. 1. The data terminal must see the data set ready signal as well as the clear to send signal before it will disgorge data. 2. The data set must see the data terminal ready and request to send signals before it will send out serial data. Interrupting either of the first pair of signals will usually stop the data terminal from pumping out data. Interrupting either of the second pair of signals will stop the data set from replying with its own data.

The carrier detect signal may or may not enter into the relationship. Some data terminals require seeing the carrier detect signal before they will transmit data. Others just don't give a byte one way or the other.

Software Flow Control

The alternate means of handshaking, software flow control, requires your serial peripheral and PC to exchange characters or tokens to indicate whether they should transfer data. The serial peripheral normally sends out one character to indicate it can accept data and a different character to indicate that it is busy and cannot accommodate more. Two pairs of characters are often used, XON/XOFF and ETX/ACK.

In the XON/XOFF scheme, the XOFF character sent from your serial peripheral tells your PC that its buffer is full and to hold off sending data. This character is also sometimes called DC1 and has an ASCII value of 19 or 013(Hex). It is sometimes called Control-S. (With some communications programs, you can hold down the Control key and type S to tell the remote system to stop sending characters to your PC). Once your serial peripheral is ready to receive data again, it sends out XON, also known as DC3, to your PC. This character has an ASCII value of 17 or 011(Hex). It is sometimes called Control-Q. When you hold down Control and type Q into your communications program, it cancels the effect of a Control-S.

ETX/ACK works similarly. ETX, which is an abbreviation for End TeXt tells your PC to hold off on sending more text. This character has an ASCII value of 3 (decimal or hexadecimal) and is sometimes called Control-C. ACK, short for Acknowledge, tells your PC to resume sending data. It has an ASCII value of 6 (decimal or hexadecimal), and is sometimes called Control-F.

There's no issue as to whether hardware or software flow control is better. Both work and that's all that's necessary. The important issue is what kind of flow control your serial peripheral and software use. You must assure that your PC, your software, and your serial peripheral use the same kind of flow control.

Your software will either tell you what it prefers or give you the option of choosing when you load the driver for your peripheral. On your serial peripheral, you select serial port flow control when you set it up. Typically, this will involve making a menu selection or adjusting a DIP switch.

Cables

The design of the standard RS-232C serial interface anticipates that you will connect a data terminal to a data set. When you do, all the connections at one end of the cable that links them are carried through to the other end, pin for pin, connection for connection. The definitions of the signals at each end of the cable are the same, and the function and

direction of travel (whether from data terminal to data set or the other way around) of each is well defined. Each signal goes straight through from one end to the other. Even the connectors are the same at either end. Consequently, a serial cable should be relatively easy to fabricate.

In the real world, nothing is so easy. Serial cables are usually much less complicated or much more complicated than this simple design. Unfortunately, if you plan to use a serial connection for a printer or plotter, you have to suffer through the more complex design.

Straight Through Cables

Serial cables are often simpler than pin-for-pin connections from one end to the other because no serial link uses all 25 connector pins. Even with the complex handshaking schemes used by modems, only nine signals need to travel from the data terminal to the data set, PC to modem. (For signaling purposes, the two grounds are redundant—most serial cables do not connect the chassis ground.) Consequently, you need only make these 9 connections to make virtually any data terminal to data set link work. Assuming you have a 25-pin D-shell connector at either end of your serial cable, the essential pins that must be connected are 2 through 8, 20, and 22 on a 25-pin D-shell connector. With 9-pin connectors at either end of your serial cable, all 9 connections are essential.

Not all systems use all the handshaking signals, so you can often get away with fewer connections in a serial cable. The minimal case is a system that uses software handshaking only. In that case, you need only three connections: transmit data, receive data, and the signal ground. In other words, you need only connect pins 2, 3, and 7 on a 25-pin connector or pins 2, 3, and 5 on a 9-pin serial connector—providing, of course, you have the same size connector at each end of the cable.

Although cables with an intermediate number of connections are often available, they are not sufficiently less expensive than the nine-wire cable to justify the risk and lack of versatility. So you should limit your choices to a nine-wire cable for systems that use hardware handshaking or three-wire cables for those that you're certain use only software flow control.

Manufacturers use a wide range of cable types for serial connections. For the relatively low data rates and reasonable lengths of serial connections, you can get away with just about everything, including twisted pair telephone wire. To ensure against interference, you should use shielded cable, which wraps a wire braid or aluminum coated plastic film about inner conductors to prevent signals leaking out or in. The shield of the cable should be connected to the signal ground. (Ideally, the signal ground should have its own wire, and the shield should be connected to chassis ground, but most folks just

don't bother.)

Adapter Cables

If you need a cable with a 25-pin connector at one end and a 9-pin connector at the other, you cannot use a straight through design even when you want to link a data terminal to a data set. The different signal layouts of the two styles of connector are incompatible. After all, you can't possibly link pin 22 on a 25-pin connector to a non-existent pin 22 on a 9-pin connector.

This problem is not uncommon. Even though the 9-pin connector has become a *de facto* standard on PCs, most other equipment, including serial plotter, printers and modems, has stuck with the 25-pin standard. To get from one connector type to another, you need an adapter. The adapter can take the form of a small assembly with a connector on each end or of an adapter cable, typically from six inches to six feet long.

Although commercial adapters are readily available, you can easily make your own. Table 21.5 shows the proper wiring for an adapter to link a 25-pin serial device to a 9-pin jack on a PC, assuming a data terminal-to-data set connection.

Table 21.5. Wiring for 9-to-25 Pin Serial Port Adapter

25-pin connector	*9-pin connector*	*Mnemonic*	*Function*
2	3	TXD	Transmit data
3	2	RXD	Receive data
4	7	RTS	Request to send
5	8	CTS	Clear to send
6	6	RTS	Data set ready
7	5	GND	Signal ground
8	1	CD	Carrier detect
20	4	DTR	Data terminal ready
22	9	RI	Ring indicator

Again, nine wires in a cable will suffice. For systems using only software flow control, you need link only the three essential pins. Note, however, the three pins do not get connected one-for-one. Pin 2 on the 25-pin connector goes to pin 3 on the 9-pin; pin 3 on the 25-pin goes to pin 2 on the 9-pin. The ground on pin 7 of the 25-pin connector

goes to pin 5 of the 9-pin connector.

Crossover Cables

As long as you want to connect a computer serial port that functions to a modem, you should have no problem with serial communications. You will be connecting a data terminal to a data set, exactly what engineers designed the serial systems for. Simply sling a cable with enough conductors to handle all the vital signals between the computer and modem and, *Voila!* Serial communications without a hitch. Try it, and you're likely to wonder why so many people complain about the capricious nature of serial connections.

When you want to connect a plotter or printer to a PC through a serial port, however, you will immediately encounter a problem. The architects of the RS-232C serial system decided that both PCs and the devices are data terminals or DTE devices. The designations actually made sense, at least at that time. You were just as likely to connect a serial printer (such as a teletype) to a modem as you were a computer terminal. There was no concern about connecting a printer to a PC because PCs didn't even exist back then.

When you connect a plotter or printer and your PC—or any two DTE devices—with an ordinary serial cable, you will not have a communication system at all. Neither machine will know that the other one is even there. Each will listen on the serial port signal line that the other is listening to, and each will talk on the line that the other talks on. One device won't hear a bit the other is saying.

The obvious solution to the problem is to switch some wires around. Move the transmit data wire from the PC to where the receive data wire goes on the plotter or printer. Route the PC's receive data wire to the transmit data wire of the plotter or printer. A simple *crossover cable* does exactly that, switching the transmit and receive signals at one end of the connection.

Many of the device that you plug into a PC are classed as DTE or data terminals just like the PC. All of these require a crossover cable. Table 21.6. lists many of the device you might connect to your PC and whether they function as data terminals (DTE) or data sets (DCE).

Table 21.6. Common Serial Device Types

Peripheral	*Device type*	*Cable needed to connect to PC*

PC	DTE	Crossover
Modem	DCE	Straight-through
Mouse	DCE	Straight-through
Trackball	DCE	Straight-through
Digitizer	DCE	Straight-through
Scanner	DCE	Straight-through
Serial printer	DTE	Crossover
Serial plotter	DTE	Crossover

Some serial ports on PCs (and some serial devices, too) offer a neat solution to this problem. They allow you to select whether they function as DTE or DCE with jumpers or DIP switches. To connect one of these to a plotter or printer using an ordinary straight through cable, configure the PC's serial port as DCE.

This simple three-wire crossover cable works if you plan on using only software flow control. With devices that require hardware handshaking, however, the three-wire connection won't work. You need to carry the hardware handshaking signals through the cable. And then the fun begins.

Your problems begin with carrier detect. The carrier detect signal originates on a data set, and many data terminals need to receive it before they send out data. When you connect two data terminals, neither generates a signal anything like carrier detect, so there's nothing to connect to make the data terminals start talking. You have to fabricate the carrier detect signal somehow.

Because data terminals send out their data terminal ready signals whenever they are ready to receive data, you can steal the voltage from that connection. Most crossover cables link their carrier detect signals to the data terminal ready signal from the other end of the cable.

Both data terminals will send out their data terminal ready signals when they are ready. They expect to see a ready signal from a data set on the data set ready connection. Consequently, most crossover cables also link data terminal ready on one end to data set ready (as well as carrier detect) at the other end. Making this linkup allows the two data terminals at either end of the cable to judge when the other is ready.

The actual flow control signals are request to send and clear to send. The typical crossover cable thus links the request to send signal from one end to the clear to send connection at the other end. This link will enable flow control—providing, of course, the two data terminal devices follow the RS-232C signaling standard. Table 21.7 shows summarizes these connections.

Table 21.7. Basic Crossover Cable for Hardware Handshaking (25-Pin Connectors)

PC end	*Function*	*Device end*
2	Transmit data	3
3	Receive data	2
4	Request to send	5
5	Clear to send	4
6	Data set ready	20
7	Signal ground	7
8	Carrier detect	20
20	Data terminal ready	6
20	Data terminal ready	8

Unfortunately, this cable may not work properly when you link many serial devices to the typical PC. A different design that combines the request to send and clear to send signals and links them to carrier detect at the opposite end of the cable often works better than the above by-the-book design. The wiring connections for this variety of crossover cable are listed in Table 21.8.

Table 21.8. Wiring for a Generic Crossover Serial Cable (25-Pin Connectors)

PC end Function Device end 2 Transmit data 3 3 Receive data 2 4 Request to send 8 5 Clear to send 8 6 Data set ready 20 7 Signal ground 7 8 Carrier detect 5 8 Carrier detect 4 20 Data terminal ready 22 20 Data terminal ready 6 22 Ring indicator 20

A number of printers vary from the signal layout ascribed to RS-232C connections and use different connections for flow control. DEC serial printers, among others, use pin 19 instead of pin 20 for hardware flow control. These require another variation on the generic crossover cable to make them work properly with PCs. The proper wiring is shown in Table 21.9.

Table 21.9. Wiring for Crossover Serial Cable (25-Pin to 25-Pin) for DEC Printers Using Pin 19 Handshake

PC end	*Function*	*Device end*

2	Transmit data	3
3	Receive data	2
4	Request to send	8
5	Clear to send	8
6	Data set ready	19
7	Signal ground	7
8	Carrier detect	5
8	Carrier detect	4
20	Data terminal ready	22
20	Data terminal ready	6

Some of the newer and more popular serial printers are in the LaserJet series made by Hewlett-Packard. These use a simplified hardware flow control system that involves only the DTR signal on the printer end of the cable. Earlier printer models use 25-pin connectors, and Hewlett-Packard sells a crossover cable for these as its part number 17255D. Its wiring is shown in Table 21.10.

Table 21.10. HP 25-Pin to 25-Pin Serial Adapter Cable

PC end	*LaserJet end*		
Pin	*Signal*	*Pin*	*Signal*
1	Chassis ground	1	Chassis ground
2	Transmit data	3	Receive data
3	Receive data	2	Transmit data
5	Clear to send	20	Data terminal ready
6	Data set ready	20	Data terminal ready
7	Signal ground	7	Signal ground

You can directly connect a PC-style 9-pin serial port to a LaserJet with a 25-pin serial connector using Hewlett-Packard's adapter cable model 2424G. Its wiring is shown in Table 21.11.

Table 21.11. HP 9-Pin to 25-Pin Serial Adapter Cable

PC end	*LaserJet end*

Pin	*Signal*	*Pin*	*Signal*
2	Receive data	2	Transmit data
3	Transmit data	3	Receive data
5	Signal ground	7	Signal ground
6	Clear to send	20	Data terminal ready
8	Data set ready	20	Data terminal ready

More recent LaserJets use 9-pin serial connectors instead of the 25-pin variety. The machines do not follow the IBM 9-pin standard used by the 9-pin jacks on PCs but instead use its complement. If you consider the IBM-style connector DTE, then the Hewlett-Packard LaserJet 9-pin connector is DCE. It requires its own adapter cable to plug into standard 25-pin PC-style serial ports. The necessary adapter cable is available from Hewlett-Packard as model number C2933A. Table 21.12 shows its wiring.

Table 21.12. HP 25-Pin to 9-Pin Serial Adapter Cable

PC end	*LaserJet end*		
Pin	*Signal*	*Pin*	*Signal*
2	Transmit data	3	Receive data
3	Receive data	2	Transmit data
4	Request to send	7	Not used
5	Clear to send	8	Data terminal ready
6	Data set ready	6	Data terminal ready
7	Signal ground	5	Signal ground
8	Carrier detect	1	Request to send
20	Data terminal ready	4	Data set ready
22	Ring indicator	9	Not used

The Hewlett-Packard redefinition of the 9-pin serial connector for its printers has one big benefit. You can connect a 9-pin PC serial port directly to a 9-pin HP printer port using a straight through cable. Moreover, only HP's printers use only seven of the nine connections. Table 21.13 shows the wiring of this cable, which is available from Hewlett-Packard as its model C2932A.

Table 21.13. HP 9-Pin to 9-Pin Serial Adapter Cable

PC end	*LaserJet end*		
Pin	*Signal*	*Pin*	*Signal*
1	Carrier Detect	1	Carrier detect
2	Receive data	2	Transmit data
3	Transmit data	3	Receive data
4	Data terminal ready	4	Data set ready
5	Signal ground	5	Signal ground
6	Clear to send	6	Data terminal ready
7	Request to send	7	Not used
8	Data set ready	8	Data terminal ready
9	Ring indicator	9	Not used

One way to avoid the hassle of finding the right combination of hardware handshaking connections would appear to be letting software do it—avoiding hardware handshaking and instead using the XON-XOFF software flow control available with most serial devices. Although a good idea, even this expedient can cause hours of head scratching when nothing works as it should—or nothing works at all.

When trying to use software handshaking, nothing happening is a common occurrence. Without the proper software driver, your PC or PS/2 has no idea that you want to use software handshaking. It just sits around waiting for a DSR and a CTS to come rolling in toward it from the connected serial device.

You can sometimes circumvent this problem by connecting the data terminal ready to data set ready and request to send to clear to send within the connectors at each end of the cable. This wiring scheme satisfies the handshaking needs of a device with its own signals. But beware. This kind of subterfuge will make systems that use hardware handshaking print, too, but you'll probably lose large blocks of text when the lack of real handshaking lets your PC continue to churn out data even after your printer shouts "Stop!"

Finally, note that some people call crossover cables *null modem cables*. This is not correct. A null modem is a single connector used in testing serial ports. It connects the transmit data line to the receive data line of a serial port as well as crossing the handshaking connections within the connector. Correctly speaking, a null-modem cable is equipped with this kind of wiring at both ends. It forces both serial ports constantly on and prevents any hardware flow control from functioning at all. Although such a cable can be useful, it is not the same as a crossover cable. Substituting one for the other will lead to some unpleasant surprises—text dropping from sight from within documents as mysteriously and irrecoverably as D. B. Cooper.

UARTs

A serial port has two jobs to perform. It must re-package parallel data into serial form, and it must send power down a long wire with another circuit at the end, which is called driving the line.

Turning parallel data into serial is such a common electrical function that engineers created special integrated circuits that do exactly that. Called *Universal Asynchronous Receiver/Transmitter* chips or UARTs, these chips gulp down a byte or more of data and stream it out a bit at a time. In addition, they add all the other accouterments of the serial signal—the start, parity, and stop bits. Because every serial practical connection is bi-directional, the UART works both ways, sending and receiving, as its name implies.

Because the UART does all the work of serializing your PC's data signals, its operation is one of the limits on the performance of serial data exchanges. PCs have used three different generations of UARTs, each of which imposes its own constraints.

The choice of chip is particularly critical when you connect your serial port to modem to it. When you communicate on-line with a modem, you're apt to receive long strings of characters through the connection. Your PC must take each character from a register in the UART and move it into memory. When your PC runs a multitasking system, it may be diverted for several milliseconds before it turn its attention to the UART and gathers up the character. Older UARTs must wait for the PC to take away one character before they can accept another from the communications line. If the PC is not fast enough, the characters pile up. The UART doesn't know what to do with them, and some of the characters simply get lost. The latest UARTs incorporate small buffers, or memory areas, that allow the UART to temporarily store characters until the PC has time to take them away. These newer UARTs are more immune to character loss and are preferred by modem users for high speed communications.

When you connect a printer to a serial port, you don't have such worries. The printer connection is more a monologue than a dialogue—your PC chatters out characters and gets very little backtalk from your printer. Typically, it will get only a single XOFF or XON character to tell the PC to stop or start the data flow. Because there's no risk of a pileup of inbound characters, there's no need for a buffer in the UART.

If you have both a modem and a serial printer attached to your PC, your strategy should be obvious; the modem gets the port with the faster UART. Your printer can work with whatever UART is left over.

The three UART chips that PC and peripheral makers install in their products are the

8250, 16450, and 16550A.

8250

The first UART used in PCs was installed in the original IBM PC's Asynchronous Communications Adapter card in 1981. Even after a decade and a half, it is still popular on inexpensive port adapter expansion boards because it is cheap. It has a one-byte internal buffer that's exactly what you need for printing or plotting applications. It can hold the XOFF character until your PC gets around to reading it. It is inadequate for reliable two way communications at high modem speeds.

16450

In 1984, designers first put an improved version of the 8250, the 16450 UART, in PCs. Although the 16450 has a higher speed internal design, it still retains the one-byte buffer incorporated into its predecessor. Serial ports built using it may still drop characters under some circumstances at high data rates. Although functionally identical, the 16450 and 8250 are physically different (they have different pin-outs), and you cannot substitute one in a socket meant for the other.

16550A

The real UART breakthrough came with the introduction of the 16550 to PCs in 1987. The first versions of this chip proved buggy, so it was quickly revised to produce the 16550A. It is commonly listed as 16550AF and 16550AFN, with the last initials indicating the package and temperature rating of the chip. The chief innovation incorporated into the 166550 was a 16-byte *first-in, first out buffer* (or FIFO). The buffer is essential to high speed modem operating in multitasking systems, making this the chip of choice for communications.

To maintain backward compatibility with the 16450, the 16550 ignores its internal buffer until it is specifically switched on. Most communications programs activate the buffer automatically. Physically, the 16550 and 16450 will fit and operate in the same sockets, so you can easily upgrade the older chip to the newer one.

Register Function

The register at the base address assigned to each serial port is used for data communications. Bytes are moved to and from the UART using the microprocessor's OUT and IN instructions. The next six addresses are used by other serial port registers, in order: the Interrupt Enable Register, the Interrupt Identification Register, the Line Control Register, the Modem Control Register, the Line Status Register, and the Modem Status Register. Another register, called the Divisor Latch, shares the base address used by the Transmit and Receive registers and the next higher register used by the interrupt enable register. It is accessed by toggling a setting in the line control register.

This latch stores the divisor that determines the operating speed of the serial port. Whatever value is loaded into the latch is multiplied by 16. The resulting product is used to divide the clock signal supplied to the UART chip to determine the bit rate. Because of the factor of 16 multiplication, the highest speed the serial port can operate at is limited to 1/16 the supplied clock (which is 1.8432 MHz). Setting the latch value to its minimum, one, results in a bit rate of 115,200.

Registers not only store the values used by the UART chip but also are used to report to your system how the serial conversation is progressing. For example, the line status register indicates whether a character that has been loaded to be transmitted has actually been sent. It also indicates when a new character has been received.

Although you can change the values stored in these registers manually using Debug or your own programs, for the most part you'll never tangle with these registers. They do, however, provide flexibility to the programmer.

Instead of being set with DIP switches or jumpers, the direct addressability of these registers allows all the vital operating parameters to be set through software. For instance, by loading the proper values into the line control register, you alter the word length, parity, and number of stop bits used in each serial word.

Buffer Control

Operating system support for the buffer in the 16550 appeared only with Windows 3.1. Even then it was limited in support to Windows applications only. DOS applications require internal FIFO support even when they run inside Windows 3.1. Windows for Workgroups (Version 3.11) extended buffer support to DOS applications running within the operating environment. The standard communications drivers for OS/2 Warp

and Windows 95 operating systems will automatically take advantage of the 16550 buffer when the chip is present.

Windows 3.1 uses its COMM.DRV for controlling the buffer of the 16650. You control whether the buffer is activated by altering the COMxFIFO entries for each of your serial ports in the [386Enh] Section of your SYSTEM.INI file of any member of the Window 3.1 family. To activate the buffer for a specific port, set COM*x*FIFO to one where the *x* is the port designation. To deactivate the buffer, make the entry zero. For example, the following entries in SYSTEM.INI will switch on the buffer for COM3 only:

```
[386Enh]
COM1FIFO=0
COM2FIFO=1
COM3FIFO=0
COM4FIFO=0
```

By default, Windows will activate the buffers in any 16550 chip that it finds.

Under Windows 95, you can control the FIFO buffer through the Device Manager section of the System folder found in Control Panel. Once you open Control Panel, click on the System icon. Click on the Device Manager tab, then the entry for Ports, which will then expand to list the ports available in your system. Click on the COM port you want to control, then click on the Properties button below the listing, as shown in Figure 21.5.

Figure 21.5 Windows 95 Device Manager folder.

From the Communications properties screen, click on the Port Settings tab. In addition to the default parameters set up for your chosen port, you'll see a button labeled Advanced. Clicking on it will give you control of the FIFO buffer, as shown in Figure 21.6.

Figure 21.6 Disabling or enabling your UART FIFO buffer under Windows 95.

Windows 95 defaults to setting the FIFO buffer on if you have a 16550 UART or the equivalent in your PC. To switch the buffer off, click on the checked box labeled User FIFO buffers. The settings shown in Figure 21.6 are the defaults.

Identifying UARTs

One way of identifying the type of UART installed in your PC is to look at the designation stenciled on the chip itself. Amid a sea of prefixes, suffixes, data codes, batch numbers, and other arcana important only the chip makers, you'll find the model number of the chip.

First, of course, you must find the chip. Fortunately, UARTs are relatively easy to find. All three basic types of UART use exactly the same package, a 40-pin DIP (Dual In-line Package) black plastic shell that's a bit more than 2 inches long and 0.8 inch wide. Figure 21.7 shows this chip package. These large chips tend to dominate multifunction or port adapter boards on which you'll typically find them. Some older PCs have their chips on their motherboards.

Figure 21.7 The 40-pin DIP package used by UARTs.

Unfortunately, the classic embodiment of the UART chip is disappearing from modern PCs. Large ASICs (Application-Specific Integrated Circuits) often incorporate the circuitry and functions of the UART (or, more typically, two of them). Most PCs consequently have no UARTs for you to find, even though they have two built-in serial ports.

The better way to identify your UARTs is by checking their function. That way you don't have to open up your PC to find out what you've got. For example, when you set up a serial port, Windows 95 will tell you whether you have a 16550A UART and allow you to adjust its buffer.

If you're stuck with an older operating system, you can still use software to be sure that the chip will act the way it's supposed to. Snooper programs will check out your UART quickly and painlessly. Better still, you can determine the kind of UARTs in your PC (as well as a wealth of other information) using Microsoft Diagnostics, the program MSD.EXE, which is included with the latest versions of Windows. After you run MSD, you'll see a screen like that shown in Figure 21.8.

Figure 21.8 The opening screen of the Microsoft Diagnostics program.

From the main menu, choose the COM ports option by pressing "C" on your keyboard. The program will respond by showing you the details of the serial ports that you have installed in your system with a screen like that shown in Figure 21.9.

Figure 21.9 The Microsoft Diagnostics display of communications port parameters.

The last line of the display lists the UART chip used by each of your PC's serial ports. Use the more recent chip for your external modem if you have the choice; the 8250 chips (as shown in the example screen) are suitable for plotters, printers, mice, and

other slow moving critters.

Enhanced Serial Ports

Serious modem users may install an Enhanced Serial Port in their PCs, which acts like a 16550 but incorporates higher speed circuitry and a much larger buffer. Because you have to install such an option yourself, you should know if you have one. Most Enhanced Serial Ports are identified as 16550 UARTs by snooping programs. They were introduced primarily to take advantage of higher modem speeds. Parallel modems and new serial port designs such as USB provide a means of achieving the same end using recognized industry standards, so you may want to avoid enhanced serial ports.

Logical Interface

Your PC controls the serial port UART through a set of seven registers built into the chip. Although your programs could send data and commands to the UART (and, through it, to your serial device) by using the hardware address of the registers on the chip, this strategy has disadvantages. It requires the designers of systems to allocate once and forever the system resources used by the serial port. The designers of the original IBM PC were loathe to make such a permanent commitment. Instead they devised a more flexible system that allows your software to access ports by name. In addition, they worked out a way that port names would be assigned properly and automatically even if you didn't install ports in some predetermined order.

Port Names

The names they assigned were COM1 and COM2. In 1987, the designers of DOS expanded the possible port repertory to include COM3 and COM4. Under Windows 3.1, up to nine serial ports could be installed in a PC using DOS conventions, although no true standards for the higher port values exist. Windows 95 has enhanced its support of serial ports to extend to 128 potential values. The implementation of these ports is handled by the device driver controlling them.

Under the DOS conventions, the names that get assigned to a serial port depend on the input/output port addresses used by the registers on the UART. PCs reserve a block of eight I/O ports for the seven UART registers. These eight addresses are sequential, so

they can be fully identified by the first, the *base address* of the serial port.

Because of their long use, the first two base addresses used by the serial ports are invariant, 3F8(Hex) and 2F8(Hex). Although the next two sometimes vary, the most widely used values for the next two base addresses are 3E8(Hex) and 2E8(Hex). Windows automatically assumes you will use these base addresses.

IBM devised an elaborate scheme for assigning port names to base addresses. When a PC boots up, DOS reads all base address values for serial ports, then assigns port names to them in order. Serial port names are consequently always sequential. In theory, you could skip a number from the ordered listing of base addresses and still get a sequence of serial port names starting with COM1. In practice, however, this can create setup difficulties. You're better off assuming that the defaults listed in Table 21.14 for DOS and Windows default serial port parameters are a hard and fast rule.

Table 21.14. Default Settings for DOS and Windows Serial Ports

Port name	*Base address*	*Interrupt*
COM1	03F8(Hex)	4
COM2	02F8(Hex)	3
COM3	03E8(Hex)	4
COM4	02E8(Hex)	3

Assigning alternate base addresses for higher port names when using the Windows 3.1 family requires adjusting settings in your PC's SYSTEM.INI file which is normally located in your WINDOWS directory. If you look under the [386Enh] section of SYSTEM.INI, you'll find entries for the variables *COM3Base* and *COM4Base*. You can change the base address assignments by altering these values. Windows 95 allows much more versatility. Normal hardware installation and Device Manager allow you to configure additional serial ports to use any of a range of base addresses within the range allowed by your PC and the driver software accompanying the port hardware.

Interrupts

Serial ports normally operate as interrupt-driven devices. That is, when they must perform an action immediately, they send a special signal called an interrupt to your PC's microprocessor. When the microprocessor receives an interrupt signal, it stops the work it is doing, saves its place, and execute special software routines called *interrupt handlers*.

A serial port generates a hardware interrupt by sending a signal down an *interrupt request line* or IRQ. PC expansion buses typically have six to twelve separate interrupt request lines. The ISA expansion bus is designed to allow one interrupt and one device to use each IRQ line. More advanced expansion buses like EISA and PCI allow more than one device to share each IRQ line.

Unless interrupts are shared, each serial port needs its own interrupt. Unfortunately, the number of available interrupts is typically too small for four serial ports to get their own IRQs. In most ISA computers, only interrupts 3 and 4 are typically used by serial ports. When you have more than two serial ports installed in such a system, some ports must share interrupts. Because ISA makes no provision for the sharing of these interrupts, conflicts can arise when two devices ask the microprocessor to use the same interrupt at the same time. The result can be characters lost in serial port communications, mice that refuse to move, and even system crashes.

The usual culprits in such problems are modems and mice sharing a port. Printers rarely cause problems because their interrupt demands are small—they usually need interrupt service only for flow control. If you must share interrupts between devices, making the serial port used by your plotter or printer share usually is the best choice. It will happily share an interrupt with your modem's serial port and will rarely cause problems, particularly if you refrain from printing and using your modem at the same time.

Windows can help resolve serial port interrupt conflict problems. It lets you designate your own choice of interrupt for serial ports three and four. As with the base addresses used by these ports, you tell Windows the values you want to use in your PC's SYSTEM.INI file. You'll find entries for COM3Int and COM4Int under the heading [386Enh] in SYSTEM.INI. Window 95 normally will automatically identify your serial ports. However, you can change the settings using the Resources tab in the Communications Port properties folder, as shown in Figure 21.10. You access this folder exactly as you would change FIFO buffers settings as described earlier in this section.

Figure 21.10 The Resources tab on the Communications Port Properties folder.

In any case, you cannot haphazardly assign any value to serial port interrupts and expect the serial port to automatically work. You also have to configure the port hardware to use the same address you've indicated to Windows. Most serial ports allow you to choose the interrupt from a short list using jumpers or DIP switches. You'll have to consult the manual accompanying your PC or serial port adapter to determine the proper settings.

ACCESS.bus

Not appreciably faster than an RS-232C connection, ACCESS.bus earns interest with the second half of its name. Unlike the RS-232C connection design to bridge two devices together, ACCESS.bus acts as a bus and links as many as 125 devices to a single port. In other words, although it is not faster than the old-fashioned serial connection, it is more versatile.

Moreover, ACCESS.bus is already part of the PC universe, adopted as a means of monitoring the condition of Smart batteries and by the Video Electronic Standards Association as part of the Display Data Channel interface. Although its use there is now chiefly for identifying the monitor connected to your PC, the versatility of the interface promises to allow your monitor to serve as the centerpiece for the desktop accessories of your PC—the monitor can act as the hub to which your keyboard, mouse, and joystick connect to your PC through DDC and ACCESS.bus.

Two words summarize the ACCESS.bus design—simple and slow. It uses a simple serial interface with a well defined protocol. Its aim is to connect one or more low speed input/output devices to a host computer. Instead of hardware signals for controlling transfers, ACCESS.bus uses messages sent through the data channel.

The name ACCESS.bus is derived from its purpose, a *bus* for connecting *ACCESS*ory devices to a host computer system. Although based on a physical interface developed by Philips Electronics called I^2C for Inter-Integrated Circuit, the actual ACCESS.bus specification was developed by Digital Equipment Corporation. Offered to the computer industry as an open standard, ACCESS.bus does not require fees or royalties to use despite its proprietary origins.

Architecture

ACCESS.bus is a multiple master design. Devices that connect using ACCESS.bus operate either as masters or slaves, and these definitions can change dynamically. The difference is that a master controls the transfer while the slave only receives data. The master sends out both the serial clock and serial data signals.

Your PC, as part of the ACCESS.bus system, serves a special role. It is the *host*. As such, it sets up the ACCESS.bus system, assigning addresses to individual devices every time you switch the system on or add a new device to a running system. All transmissions across the ACCESS.bus are between the host and another device, although the host may act as master or slave during these transfers.

The ACCESS.bus system has three layers. The *Physical Layer* controls the signals and

transfer protocol, including the how packets are defined using the basic signals. The *Base Protocol* outlines the essential content the messages, including the message format—the function of each bit within a message, including headers and error detection—and defines the commands that can be relayed through across the bus within messages. The Application Protocol defines how information from different kinds of devices gets packaged into messages. The current ACCESS.bus specification specifically defines three classes of devices: keyboards, locators (essentially pointing devices such as mice), and text devices (which can be either devices that sent textual data, such as a bar code reader, or simply the identification text sent by a monitor as part of DDC).

Signaling

The ACCESS.bus uses four signals. One, SDA, transmits data across the bus. A second line, SCLK, is a clock signal that defines when the data is valid and can be read. The bus also provides a five-volt power connection cable of running low power devices. A minimum of 50 milliamps must be supplied to the ACCESS.bus by the computer host. All three signals share a common ground. Table 21.15 summarizes these signals, their normal pin assignments, and wiring color code.

Table 21.15. ACCESS.bus Signals

Pin	*Function*	*Mnemonic*	*Color code*
1	Ground	GND	Black
2	Serial data	SDA	Green
3	+5 VDC	+5V	Red
4	Serial clock	SCL	White

The data and clock signals operate at 100 kilohertz. Because the protocol adds an acknowledge bit for every byte transferred and the overhead involved in packet headers and error detection, the actual throughput of the ACCESS.bus is about 80 kbits/sec.

The ACCESS.bus data and clock signals are normally held at a logical high voltage at the host computer either through a voltage source or by simply connecting the lines to a positive voltage through a resistor. All devices monitor the state of these lines, detecting high and low. Any device connected to the bus can pull this signal low, and it will appear low to all devices regardless of which of them pulls it low.

Transfers

A *start condition* begins on the bus when a device pulls the data line low while leaving the clock line high. When devices sense a start condition, they regard the bus as busy and no other master will attempt to send data down the bus. The master in control sends a stop signal to indicate it has finished with the bus. The stop signal is a low to high transition on the data line while the clock line is high.

After sending the start signal, the master clock forces the clock line low. It then pulses the clock high to indicate valid data on the data line. Each byte of data is sent as a series of eight bits, most significant first, delineated by eight pulses of the clock signal.

Following the data bits, the master sends out an acknowledge bit, essentially a pulse of the clock. The slave acknowledges receipt of the byte by having pulled the data line low during the acknowledge pulse of the clock. If the slave allows the data line to remain high during the acknowledge clock pulse, the transfer is considered to have been not acknowledged. Under this system, a device can actively generate a "not acknowledged" signal. A device that is not available or even not turned on automatically generates a "not acknowledged" indication.

Arbitration

Arbitration on the ACCESS.bus is quite simple. Should two masters attempt to send data at the same time, both can start transmitting. As long as both send the same data down the bus both can continue transmitting because the signals are the same and it simply makes no difference where they come from. As soon as the signals differ—one master puts a zero on the data line while the other puts a

one there—the master that puts the one on the bus loses the arbitration and stops transmitting. The other master completes its transmission normally.

Messages

Packets on the ACCESS.bus are termed messages. Each message has a three-byte header, from one to 127 bytes of data, and a one-byte check sum. The first byte indicates the address of the destination device. The second, the address of the source device. The third includes a one-bit protocol flag and seven bits specifying the number of data bits in the message.

Both data exchanges and control functions get carried through the ACCESS.bus system as messages. Most of the command messages are used during system configuration. These include host messages to force all devices on the bus to reset to their power-on condition, a command to assign an address to a device, a command for each device to identify itself, another to identify device capabilities, and one that simply detects whether a given device is present on the bus. Devices can respond with messages identifying themselves or their capabilities. Devices can also send an attention message to let the host know that a new device has been plugged into the bus.

Addresses

Of the 128 addresses possible with seven-bit encoding, three are reserved. The host computer is always assigned the address 50(Hex). The default address of all devices when they power up is 6E(Hex). The system management device always is assigned 10(Hex). The remaining 125 addresses are available for

ACCESS.bus devices. Most are dynamically assigned by the host computer, although a few are used for specific devices such as Smart Battery Specification chargers and VESA-standard monitors using the Display Data Channel identification system.

To assign addresses, the host computer relies on the ACCESS.bus arbitration procedure. The host broadcasts a message requesting each device to send out its unique identification by addressing the message to the default address, 6E(Hex). All devices respond at once, each sending a message containing its identification back to the host. As the bits flow, the devices that have a one in their identifications at a given position but detect one or more other devices putting a zero on the bus drop out. They lose the arbitration. Because each identification is unique, eventually one device is left and completes sending the identification. The master then assigns an address to that device, instructing the device to respond only to that address. The master then broadcasts the command to request identification and repeats the process until all devices have been assigned addresses.

Connections

The physical embodiment of ACCESS.bus is a modular jack with four connections. The basic ACCESS.bus jack is shown in Figure 21.11.

Figure 21.11 The ACCESS.bus female jack.

To allow for multiple devices on the ACCESS.bus, each device may have two connectors, allowing one to serve as input and the other as output. Electrically, the two jacks are identical—the same signals are present in the same locations on each. Devices with only one connector or a permanently attached cable can attach as the last device on the bus or through a T-connector that allows for further device additions.

The ACCESS.bus cable has four conductors, corresponding to the four signals on the jacks. An overall shield prevents interference. The signals are not paired. The maximum length of a cable is about 33 feet (ten meters), limited by the capacitance of the cable. The overall reach of an ACCESS.bus connection can be extended with a repeater.

Cable connectors use special shielded modular-style connectors. The connectors, as shown in Figure 21.12, are identical at both ends of the cable. The connections run pin-for-pin straight through the cable.

Figure 21.12 The ACCESS.bus cable connector.

IrDA

The one thing you don't want with a portable PC is a cable to tether you down, yet most of the time you must plug into one thing or another. Even a simple and routine chore like downloading files from your notebook machine into your desktop PC gets tangled in cable trouble. Not only do you have to plug both ends in, reaching behind your desktop machine only a little more elegantly than fishing into a catch basin for a fallen quarter—and, more likely than not, unplugging something else that you'll inevitably need later only to discover the dangling cord—but you've got to tote that writhing cable along with you wherever you go. There has to be a better way.

There is. You can link your PC to other systems and components with a light beam. On the rear panel of many notebook PCs, you'll find a clear LED or a dark red window through which your system can send and receive invisible infrared light beams. Although originally introduced to allow you to link portable PCs to desktop machines, the same technology can tie in peripherals like modems and printers, all without the hassle of plugging and unplugging cables.

History

On June 28, 1993, a group of about 120 representatives from 50 computer-related companies got together to take the first step in cutting the cord. Creating what has come to be known as the *Infrared Developers Association* or IrDA, they aimed at more than making your PC more convenient to carry. They also saw a new versatility and, hardly incidentally, a way to trim their own costs.

The idea behind the get together was to create a standard for using infrared light to link your PC to peripherals and other systems. The technology had already been long established, not only in television remote controls but also in a number of notebook PCs already in the market. Rather than build a new technology, the goal of the group was to find common ground, a standard so that the products of all manufacturers could communicate with the computer equivalent of sign language.

Hardly a year later on June 30, 1994, the group approved its first standard. The original specification, now known as IrDA Version 1.0, essentially gave the standard RS-232C port an optical counterpart, one with the same data structure and, alas, speed limit. In August 1995, IrDA took the next step and approved high speed extensions that pushed the wireless data rate to four megabits per second.

Overview

More than a gimmicky cordless keyboard, IrDA holds an advantage that makes computer manufacturers—particularly those developing low cost machines—eye it with interest. It can cut several dollars from the cost of a complex system by eliminating some expensive hardware, a connector or two, and a cable. Compared to the other wireless technology, radio, infrared requires less space because it needs only a tiny LED instead of a larger and more costly antenna. Moreover, infrared transmissions are not regulated by the FCC as are radio transmissions. Nor do they cause interference to radios, televisions, pacemakers, and airliners. The range of infrared is more limited than radio and restricted to line-of-sight over a narrow angle. However, these weaknesses can become strengths for those who are security conscious.

The original design formulated by IrDA was for a replacement for serial cables. The link was envisioned as a half-duplex system. Although communications go in both directions, only one end of the conversation sends out data at any given time.

To make the technology easy and inexpensive to implement with existing components, it was based on the standard RS-232C port and its constituent components, such as UARTs. The original IrDA standard called for asynchronous communication using the same data frame as RS-232C and the most popular UART data rates from 2400 to 115,200 bits per second.

To keep power needs low and prevent interference among multiple installations in a single room, IrDA kept the range of the system low. The expected separation between devices using IrDA signals to communicate was about one meter (three feet). Some links are reliable to two meters.

Similarly, the IrDA system concentrates the infrared beam used to carry data because diffusing the beam would require more power for a given range and be prone to causing greater interference among competing units. The laser diodes used in the IrDA system consequently focus their beams into a cone with a spread of about 30 degrees.

After the initial serial port replacement design was in place, IrDA worked to make its interface suitable for replacing parallel ports as well. That goal lead to the creation of

the IrDA high speed standards for transmissions at data rates of 0.576, 1.152 and 4.0 megabits per second. The two higher speeds use a packet based synchronous system that requires a special hardware based communication controller. This controller monitors and controls the flow of information between the host computer's bus and communications buffers.

A watershed of differences separate low speed and high speed IrDA systems. Although IrDA designed the high speed standard to be backwardly compatible with old equipment, making the higher speeds work requires special hardware. In other words, although high speed IrDA devices can successfully communicate with lower speed units, such communications are constrained to the speeds of the lower speed units. Low speed units cannot operate at high speed without upgrading their hardware.

IrDA defines not only the hardware but also the data format used by its system. The group has published six standards to cover these aspects of IrDA communications. The hardware itself forms the *physical layer*. In addition, IrDA defines a *link access protocol* termed IrLAP and a *link management protocol* called IrLMP that describe the data formats used to negotiate and maintain communications. All IrDA ports must follow these standards. In addition, IrDA has defined an optional *transport protocol* and optional *Plug-and-Play* extensions to allow the smooth integration of the system into modern PCs. The group's IrCOMM standard describes a standard way for infrared ports to emulate conventional PC serial and parallel ports.

Physical Layer

The physical layer of the IrDA system encompasses the actual hardware and transmission method. Compared to other serial technologies, the hardware you need for an IrDA port would appear to be immaterial. After all, it *is* wireless. However, your PC still needs a port capable of sending and receiving invisible light beams.

A growing number of notebook computers have built-in IrDA facilities. In fact, IrDA is one of the more important features to look for in a new notebook computers.

Desktop machines are another matter. The IrDA wave hasn't yet struck them. Fortunately, you can add an IrDA port as easily as plugging in a serial cable. For example, the Adaptec AirPort plugs into a serial port and gives you an optical eye to send and receive IrDA signals.

The AirPort is only the first generation of IrDA accessories for desktops. By its nature, it is limited to standard serial port speeds. After all, it can't run faster than the signals coming to it. IrDA ports that take advantage of the newer, higher speeds will require direct connection to your PC's expansion bus (or a more advanced serial port like the

Universal Serial Bus, discussed in the following "Universal Serial Bus" section).

In any case, the optical signals themselves form what the IrDA called the Physical Layer of the system. IrDA precisely defines the nature of these signals.

Infrared Light

Infrared light is invisible electromagnetic radiation that has a wavelength longer than that of visible light. Where you can see light that ranges in wavelength from 400 angstroms (deep violet) to 700 angstoms (dark red), infrared stretches from 700 angstroms to 1,000 or more. IrDA specifies that the infrared signal used by PCs for communication have a wavelength between 850 and 900 angstroms.

Data Rates

All IrDA ports must be able to operate at one basic speed, 9600 bits per second. All other speeds are optional.

The IrDA specification allows for all the usual speed increments used by conventional serial ports from 2400 bps to 115,200 bps. All of these speeds use the default modulation scheme, RZI. High speed IrDA Version 1.1 adds three additional speeds, 576 kbps, 1.152Mbps, and 4.0 Mbps.

No matter the speed range implemented by a system or used for communications, IrDA devices first establish communications at the mandatory 9600 bps speed using the link access protocol. Once the two devices establish a common speed for communicating, they switch to it and use it for the balance of their transmissions.

Pulse Width

The infrared cell of an IrDA transmitter sends out its data in pulses. Unlike the electronic logic signals inside your PC, which are assumed to remain relatively constant throughout a clock interval, the IrDA pulses last only a fraction of the basic clock period or bit cell. The relatively wide spacing between pulses makes each pulse easier for the optical receiver to distinguish.

At speeds up to and including 115,200 bits per second, each infrared pulse must be at least 1.41 microseconds long. Each IrDA data pulse nominally lasts just 3/16 of the length of a bit cell, although pulse widths a bit more than 10 percent greater remain acceptable. For example, each bit cell of a 9600 bps signal would occupy 104.2 microseconds (that is, one second divided by 9600). A typical IrDA pulse at that data rate would last 3/16 that period or 19.53 microseconds.

At higher speeds, the pulse minima are substantially shorter ranging from 295.2 nanoseconds at 576 Kbps to only 115 nanoseconds at 4.0 Mbps. At these higher speeds, the nominal pulse width is one quarter of the character cell. For example, at 4.0 megabits per second each pulse is only 125 nanoseconds long. Again, pulses about 10 percent longer remain permissible. Table 21.16 summarizes the speeds and pulse lengths.

Table 21.16. IrDA Speeds and Modulation

Signaling Rate	*Modulation*	*Pulse Duration*
2.4 kb/s	RZI	78.13 us
9.6 kb/s	RZI	19.53 us
19.2 kb/s	RZI	9.77 us
38.4 kb/s	RZI	4.88 us
57.6 kb/s	RZI	3.26 us
115.2 kb/s	RZI	1.63 us
0.576 Mb/s	RZI	434.0 ns
1.152 Mb/s	RZI	217.0 ns
4.0 Mb/s	4PPM, single pulse	125 ns
4.0 Mb/s	4PPM, double pulse	250.0 ns

Modulation

Depending on the speed at which a link operates, it may use one of two forms of modulation. At speeds lower than 4.0 megabits per second, the system employs *Return-to-Zero Invert* (RZI) modulation. Actually, RZI is just a fancy way of describing a simple process. Each pulse represents a logical zero in the data stream. A logical one gets no infrared pulse. Figure 21.13 shows the relation between the original digital code, the equivalent electrical logic signal, and the corresponding IrDA pulses.

Figure 21.13 Relationship between original code and the IrDA pulses.

At the 4.0 megabit per second data rate, the IrDA system shifts to *pulse position modulation*. Because the IrDA system involves four discrete pulse positions, it is abbreviated 4PPM.

Pulse position modulation uses the temporal position of a pulse within a clock period to indicate a discrete value. In the case of IrDA's 4PPM, the length of one clock period is termed the *symbol duration* and is divided into four equal segments termed *chips*. A pulse can occur in one and only one of these chip segments, and the chip the pulse appears in— its position inside the symbol duration or clock period—encodes its value. For example, the four chips may be numbered 0, 1, 2, and 3. If the pulse appears in chip 2, it carries a value 2 (in binary, that's 10). Figure 21.14 shows the pulse position for the four valid code values under IrDA at 4.0 Mbps.

Figure 21.14 Pulse positions for the four valid code values of 4PPM under IrDA.

The IrDA system uses group coding under 4PPM. Each discrete pulse in one of the four possible positions indicates one of four two-bit patterns. Each pulse in the IrDA 4PPM system thus encodes two bits of data and four clock periods suffice for the transfer of a full byte of data. Table 21.17 lists the correspondence between pulse position and bit patterns for IrDA's 4PPM.

Table 21.17. Data to Symbol Translation for IrDA 4PPM Code

Data Bit Pair	*4PPM Symbol*
00	1000
01	0100
10	0010
11	0001

IrDA requires data to be transmitted only in eight-bit format. In terms of conventional serial port parameters, a data frame for IrDA comprises a start bit, eight data bits, no parity bits, and a stop bit for a total of ten bits per character. Note, however, that zero insertion may increase the length of a transmitted word beyond this minimum. Any inserted zeroes are removed automatically by the receiver and do not enter the data stream. No matter the form of modulation used by the IrDA system, all byte values are transmitted the least significant bit first.

Bit Stuffing

Note that with RZI modulation a long sequence of logical ones will suppress pulses for the entire duration of the sequence. For example, a sequence of the byte value 0FF(Hex) will include no infrared pulses. If this suppression extends for a long enough period during synchronous communications, the clocks in the transmitter and receiver may become unsynchronized. To prevent the loss of sync, moderate speed IrDA systems use a technique called *bit stuffing*.

Moderate speed IrDA systems operate synchronously using long data frames that are self-clocking. To avoid long periods without pulses, these systems rely on *bit stuffing*. After a predetermined number of logical ones appear in the data stream, the system automatically inserts a logical zero. The zero adds a pulse that allows the transmitter and receiver to synchronize their clocks. The receiver, when detecting an extended period without pulses automatically removes the extraneous stuffed pulse from the data stream.

Moderate speed systems stuff a zero at the conclusion of each string of five logical ones. The system calculates its CRC error correction data before the data gets stuffed. The receiver strips off the stuffed bits before performing its CRC.

Bit stuffing is only required during synchronous transmissions. Because low speed IrDA systems operate asynchronously, there is no need for bit stuffing. Nor is bit stuffing necessary at IrDA's highest speed because 4PPM modulation guarantees a pulse within each clock period.

Format

The IrDA system doesn't deal with data at the bit or byte level but instead arranges the data transmitted through it in the form of packets, which the IrDA specification also terms *frames*. A single frame can stretch from 5 to 2050 bytes (and sometimes more) in length. As with other packetized systems, an IrDA frame includes address information, data, and error correction, the last of which is applied at the frame level. The format of the frame is rigidly defined by the IrDA Link Access Protocol standard, discussed in the following "Link Access Protocol" section.

Aborted Frames

Whenever a receiver detects a string of seven or more consecutive logical ones—that is, an absence of optical pulses—it immediately terminates the frame in progress and disregards the data it received (which is classed as invalid because of the lack of error correction data). The receiver then awaits the next valid frame, signified by a start-of-frame flag, address field, and control field. Any frame that ends in this summary manner is termed an *aborted frame*.

A transmitter may intentionally abort a frame or a frame may be aborted because of an interruption in the infrared signal. Anything that blocks the light path will stop infrared pulses from reaching the receiver and, if long enough, abort the frame being transmitted.

Interference Suppression

High speed systems automatically mute lower speed systems that are operating in the same environment to prevent interference. To stop the lower speed link from transmitting, the high speed system sends out a special *Serial Infrared Interaction Pulse* at intervals no

longer than half a second. The SIP is a pulse 1.6 microseconds long followed by 7.1 microseconds of darkness, parameters exactly equal to a packet start pulse. When the low speed system sees what it thinks is a start pulse, it automatically starts looking for data at the lower rates, suppressing its own transmission for half a second. Before it has a chance to start sending its own data (if any), another SIP quiets the low speed system for the next half second.

Link Access Protocol

To give the data transmitted through the optical link a common format, the Infrared Data Association created its own protocols. Its Link Access Protocol describes the composition of the data packets transmitted through the system. IrLAP is broadly based on the asynchronous data communications standards used in RS-232C ports (no surprise here) that is, in turn, adapted from the more general HDLC, High Level Data Link Control. Note that although the overall operation of the IrDA link is the same at all speeds, IrDA defines different protocols for its low and high speeds.

Primary and Secondary Stations

The foundation of IrLAP is distinguishing primary and secondary stations. A *primary station* is the device that takes command and controls the link transfers. Even when a given link involves more than two devices, all transmissions must either go to or come from the primary station. That is, all secondary devices send data only to the primary station. The primary station, however, can target a single secondary station or broadcast its data to all secondary stations.

In any IrDA connection, there can be only one primary station. The role of primary station does not automatically fall on a given device. For example, your PC is not automatically the primary station in the sessions in which it becomes involved. At the beginning of any IrDA link, the stations negotiate for the roles which they will play. After one device becomes the primary station, however, it retains that role until the link is ended.

An IrDA link begins when a device seeks to connect with another. The first device may directly request a link to a known device or it may sniff out and discover a device to which to make a connection. For example, a notebook PC could continuously search for a desktop mate, and when it finally comes into range with one, it begins to negotiate a link.

To begin the link, the first device sends out a connection request at the universal 9600 bps speed. This request includes the address of the initiating device, the speed at which it wants to pursue the linkup, and other parameters. The responding device assumes the role of the secondary station and sends back identifying information, including its own address and its speed capabilities. The two devices then change to a mutually compatible speed and the initiating device, as the primary station, takes command of the link. Data transfer begins.

Frame Types

Three types of frames are used by IrLAP, following the model of HDLC. *Information frames* transfer the actual data in a connection. *Unnumbered frames* perform setup and management tasks. For example, an unnumbered frame can establish a connection, remove a connection, or search out or discover devices with which to make a connection. *Supervisory frames* help control the flow of data between stations. For example, a supervisory frame may be used to

acknowledge the receipt of a given frame or to warn that a given station is busy.

Each frame is bracketed by two or more start fields at its beginning and one or more stop fields at its conclusion. If a frame is not bracketed by the proper flags, it will be ignored.

Next comes the address field, which identifies the source and destination of the packet. The control field identifies the type of packet, whether it contains data or control information. The data field is optional because some packets used to control the link need no data. The data field can be any length up to and including 2045 bytes but must be a multiple of eight bits. The frame ends with a frame check sequence used to detect errors and one or more stop flags. Figure 21.15 shows the layout of a typical IrDA frame.

Figure 21.15 Constituents of a high speed IrDA data frame (packet).

The start flag is simply a specific bit pattern that indicates the beginning of a field. It serves to get the attention of the receiver. The exact pattern used for the start flag depends on the speed of the data. At speeds of 115,200 bps and below, the start flag is the byte values 0C0(Hex). A frame may include more than one start flag. The stations participating in a link negotiate the number to use. Table 21.18 summarizes the components in a IrDA synchronous data frame at low data rates.

Table 21.18. IrDA Low Speed (2400 to 115,200 bps) Frame Components

Mnemonic	*Definition*	*Length*	*Value*
STA	Start flag	8 bits	0C0(Hex)
ADDR	Address field	8 bits	Varies
DATA	Control field	8 bits	Varies

DATA	Information field	2045 bytes	Varies
FCS	Error correction field	16 bits	CRC
STO	Stop flag	8 bits	0C1(Hex)

At higher speeds, both starting and end flags are given the value 07E(Hex). Each frame must have at least two start flags. A transmitter can insert additional start flags, which are ignored by the receiver. If a transmitter outputs two IrDA frames back to back, the data in the two frames will be separated by at least one stop flag and two start flags. If the transmitter sends two frame that are not back to back, the stop flag of the first frame and the first start flag of the second frame must be separated by at least seven pulse-free clock cycles. Table 21.19 summarizes the components in a IrDA synchronous data frame.

Table 21.19. IrDA Synchronous Data Frame Components at 576Kbps and Higher

Mnemonic	*Definition*	*Length*	*Value*
STA	Start flag	8 bits	07E(Hex)
ADDR	Address field	8 bits	Varies
DATA	Control field	8 bits	Varies
DATA	Information field	<2046 bytes	Varies
FCS	Frame check sequence	16 bits	CRC error detection
STO	Stop flag	8 bits	07E(Hex)

Addressing

The address field identifies the secondary station participating in the communications link. When the primary station transmits a packet, the address identifies the secondary station to which the packet is

destined. When a secondary station transmits a packet, the address identifies the secondary station. The first bit identifies the direction of the transmission, 1 from the primary station, 0 from the secondary station. The remaining seven bits are the actual address. Two of the 128 possible addresses are reserved. The address 0000000(Binary) identifies the primary station. The address 1111111(Binary) identifies a packet as global, which means that it is transmitted to all secondary nodes participating in the link. The addressing scheme constrains the number of IrDA devices in a given system to 127—the primary station and 126 secondary stations.

Error Detection

The error correction field is two bytes long. Its value is computed from the bits in the entire frame except for the starting and ending flags. In other words, the error correction covers not only the information field but also the address and control fields.

IrDA error correction is based on the cyclical redundancy check adopted by the CCITT. It is similar but not identical to the error detection used in the XMODEM file transfer protocol.

The value of the cyclical redundancy check is computed using the following algorithm:

```
CRC(x) = x16 + x12 + x5 + 1
```

Link Management Protocol

Before an IrDA system can exchange data, it must establish a link.

The IrDA created a standard procedure for creating and managing links call the Link Management Protocol. This standard covers all aspects of establishing and ending communication between IrDA devices, including such aspects as link initialization, discovering the addresses of available devices, negotiating the speed and format of the link, disconnection, shutdown, and the resolution of device conflicts. Many aspects of IrDALMP were derived from the more general HDLC protocols. Because of the broadcast nature of the infrared signal, however, the IrDA group developed its own system for detecting the presence of devices and resolving conflicts among them.

One of the biggest problems faced by IrDALMP is the dynamic nature of wireless connections. You can bring new devices into the range of a master at any time, or similarly, remove them, changing the very nature of the connection. The master must be able to determine the devices with which it can communicate and keep alert for changes. It must be able not only to search out those device within its range (and operating in a compatible mode) but also new devices that you suddenly introduce. Moreover, it must be able to determine when each device involved in a communication session ceases to participate—either by being turned off or by being moved out of range. Where traditional hard wired serial connections had the luxury of status signals that allow the easy monitoring of the status of the connection and the devices linked through it, infrared devices have nothing but pulses of light linking them.

Unlike the traditional point to point serial connection that involved merely two devices exchanging data, the master in an IrDA link must be able to manage and control the signals from multiple devices at the same time. All must share the same optical channel. The master is charged with doling out bandwidth to suit the connection and prevent contention between the various devices.

Universal Serial Bus

Three drawbacks head any list of the most aggravating aspects of serial ports: low speed, complex cabling, and the limited number of ports. The Universal Serial Bus breaks through all three, combining a signaling rate of 12 Mbits/sec with a mistake proof wiring system and almost unlimited number of connections. The standard also supports lower speed devices sharing the same wiring system along with high speed devices. The low speed signaling rate is 1.5 Mbits/sec.

First introduced in 1996, the USB is more than a successor to the RS-232C serial port. It provides the basic mechanism for connecting most, if not all, peripherals to your PC. Everything from your keyboard to cash register drawer can connect simply and quickly with a USB plug.

Background

Designed for those who would rather compute than worry about hardware, the premise underlying USB is the substitution of software intelligence for cabling confusion. USB handles all the issues involved in linking multiple devices with different capabilities and data rates with a layer cake of software. Along the way, it introduces its own new technology and terminology.

USB divides serial hardware into two classes, hubs and functions. A USB *hub* provides jacks into which you can plug functions. A USB *function* is a device that actually does something. USB's designers imagined that a function may be anything that you can connect to your computer including keyboards, mice, modems, printers, plotters, scanners, or whatever.

Rather than a simple point to point port, the USB acts as an actual bus that allows you to connect multiple peripherals to one jack on your PC with all of the linked devices sharing exactly the same signals. Information passes across the bus in the form of packets, and all functions receive all packets. Your PC accesses individual functions by adding a specific address to the packets, and only the function with the correct address acts on the packets addressed to it.

The physical manifestation of USB is a port, a jack that's part of a hub. Each physical USB port connects to a single device, and a hub offers multiple jacks to let you plug in several devices. You can plug one hub into another to provide several additional jacks and ports to connect more devices. The USB design envisions a hierarchical system with hubs connected to hubs connected to hubs. In that each hub allows multiple connections, the reach of the USB system branches out like a tree—or a tree's roots. Figure 21.16 gives a conceptual view of the USB wiring system.

Figure 21.16 USB hierarchical interconnection scheme.

Your PC acts as the base hub for a USB system and is termed the *host*. The circuitry in your PC that controls this integral hub and the rest of the USB system is called the *bus controller*. Each USB system has one and only one bus controller.

The USB system doesn't care which device you plug into which hub or how many levels down the hub hierarchy you put a particular device. All the system requires is that you properly plug everything together following its simple rule—each device must plug into a hub—and the USB software sorts everything out. This software, making up the *USB protocol*, is the most complex part of the design. In comparison, the actual hardware is simple—but the hardware won't work without the protocol.

The wiring hardware imposes no limit on the number of devices and functions that you can connect in a USB system. You can plug hubs into hubs into hubs fanning out into as many ports as you like. You do face limits, however. The protocol limits the number of functions on one bus to 127 because of addressing limits. Seven bits are allowed for encoding function addresses, and one of the potential 128 is reserved.

In addition, the wiring limits the distance at which you can place functions from hubs. The maximum length of a USB cable is five meters. Because hubs can regenerate signals, however, your USB system can stretch out for greater distances by making multiple hops through hubs.

As part of the Plug-and-Play process, the USB controller goes on a device hunt when you start your PC. It interrogates each device to find out what it is. It then builds a map that locates each device by hub and port number. These become part of the packet address. When the USB driver sends data out the port, it routes it to the proper device by this hub and port address.

Wiring with USB is, by design, trouble free. Because all devices receive all signals, you face no issues of routing. Because each port has a single jack that accepts one and only one connector—and a connector of a specific matching type—you don't have to worry about adapters, crossover cables or the other minutiae required to make old style serial connections work.

On the other hand, USB requires specific software support. Any device with a USB connector has the necessary firmware to handle USB built in. But your PC also requires software to make the USB system work. Your PC's operating system must know how to send the appropriate signals to its USB ports. In addition, each function must have a matching software driver. The function driver creates the commands or packages the data for its associated device. An overall USB driver acts as the delivery service, providing the channel—called, in USB terminology, a *pipe*—for routing the data to the various functions. Consequently, each USB you add to your PC requires software

installation along with plugging in the hardware.

Connectors

The USB system involves four different styles of connectors, two chassis-mounted jacks and two plugs at the ends of cables. Each jack and plug comes in two varieties, A and B.

Hubs have A jacks. These are the primary outward manifestation of the USB port. The matching A plug attaches to the cable that leads to the USB device. In the purest form of USB, this cable is permanently affixed to the device, and you need worry about no other plugs or jacks.

The USB standard allows for a second, different style of plug and jack meant only to be used for inputs to USB devices. If a USB device (other than a hub) requires a connector so that, as a convenience, you can remove the cable, it uses a USB "B" jack. The mating plug is a "B" plug.

The motivation behind this multiplicity of connectors is to prevent rather than cause confusion. All USB cables have an A plug at one end and a B plug at the other. One end must attach to a hub and the other to a device. You cannot inadvertently plug things together incorrectly.

Because all A jacks are outputs and all B jacks are inputs, only one form of detachable USB cable exists—one with an A plug at one end and a B plug at the other. No crossover cables or adapters are needed for any USB wiring scheme.

Cable

The physical USB wiring uses a special four-wire cable. Two conductors in the cable transfer the data as a differential digital signal. That is, the voltage on the two conductors is of equal magnitude and opposite polarity so that when subtracted from one another (finding the difference) the result cancels out any noise that ordinarily would add equally to the signal on each line. In addition, the USB cable includes a power signal, nominally five volts DC, and a ground return. The power signal allows you to supply power for external serial devices through the USB cable.

The two data wires are twisted together as a pair. The power cables may or may not be.

To achieve its high data rate, the USB specification requires that certain physical characteristics of the cable be carefully controlled. Even so, the maximum length permitted any USB cable is five meters.

One limit on cable length is the inevitable voltage drop suffered by the power signal. All wires offer some resistance to electrical flow, and the resistance is proportional to the wire gauge. Hence, lower wire gauges (thicker wires) have lower resistance. Longer cables require lower wire gauges. At maximum length, the USB specification requires 20-gauge wire, which is one step (two gauge numbers) thinner than ordinary lamp cord.

The individual wires in the USB cable are color coded. The data signals form a green-white pair, the +Data signal on green. The positive five-volt signal rides on the red wire. The ground wire is black. Table 21.20 sums up this color code.

Table 21.13. USB Cable Color Code

Signal	*Color*
+ Data	Green
- Data	White
VCC	Red
Ground	Black

Data Coding

To help ensure the integrity of the high speed data signal, the USB system uses a combination of NRZI data encoding and bit stuffing. NRZI coding (inverse no return to zero) uses a change in signal during a given period to indicate a logical zero and no change in a period to indicate a logical one. Figure 21.17 illustrates the NRZI translation scheme. Note that a zero in the code stream triggers each transition in resulting the NRZI code. A continuous stream of logical zeros results in an on-off pattern of voltages on the signal wires, essentially a square wave.

Figure 21.17 NRZI coding scheme used by USB.

The NRZI signal is useful because it is self-clocking. That is, it allows the receiving system to regenerate the clock directly from the signal. For example, the square wave of a stream of zeros acts as the clock signal. The receiver adjusts its timing to fit this interval. It keeps timing even when a logical one in the signal results in no transition. When a new transition occurs, the timer resets itself, making whatever small adjustment

might be necessary to compensate for timing differences at the sending and receiving end.

Ordinarily, a continuous stream of logical ones would result in a constant voltage, an extended stream without transitions. If the length of such a series of logical ones were long enough, the sending and receiving clocks in the system might wander and lose their synchronicity. *Bit stuffing* helps keep the connection in sync.

The bit stuffing technique used by USB system injects a zero after every continuous stream of six logical ones. Consequently, a transition is guaranteed to occur at least every seven clock cycles. When the receiver detects a lack of transitions for six cycles, then receives the transition on the seventh, it can reset its timer. It also discards the stuffed bit and counts the transition (or lack of it) occurring at the next clock cycle to be the next data bit.

Protocol

As with all more recent interface introductions, the USB design uses a packet based protocol.

All message exchanges require the exchange of three packets. The exchange begins with the host sending out a *token packet*. The token packet bears the address of the device meant to participate in the exchange as well as control information that describes the nature of the exchange. A *data packet* holds the actual information that is to be exchanged. Depending on the type of transfer, either the host or the device sends out the data packet. Despite the name, the data packet may contain no information. The exchange ends with a *handshake packet* which acknowledges the receipt of the data or other successful completion of the exchange. A fourth type of packet, called Special, handles additional functions.

All packets must start with two components, a Sync Field and a Packet Identification. Each of these components is one byte long.

The *Sync Field* is a series of bits that produces a dense string of pulse transitions using the NRZI encoding scheme required by the

USB standard. These pulses serve as a consistent burst of clock pulses that allow all the devices connected to the USB bus to reset their timing and synchronize themselves to the host. As encoded, the Sync Field appears as three on/off pulses followed by a marker two pulses wide. The raw data before encoding takes the value 00000001(binary), although the data is meaningless because it is never decoded.

The *Packet Identifier* byte includes four bits to define the nature of the packet itself and another four bits as check bits that confirm the accuracy of the first four. Rather than a simple repetition, the check bits take the form of a one's complement of the actual identification bits (every zero is translated into a one). The four bits provides a code that allows the definition of 16 different kinds of packet.

USB uses the 16 values in a two step hierarchy. The two more significant bits specify one of the four types of packet. The two less significant bits subdivide the packet category. Table 21.21 lists the PIDs of the four basic USB packet types.

Table 21.21. USB Packet Identifications

Bit pattern	*Packet type*
XX00XX11	Special packet
XX01XX10	Token packet
XX10XX01	Handshake packet
XX11XX00	Data packet

Token Packets

Only the USB host sends out Token Packets. Each Token Packet takes up four bytes, which are divided into five functional parts.

Figure 21.18 graphically shows the layout of a Token Packet.

Figure 21.18 Functional parts of a USB Token Packet.

The two bytes take the standard form of all USB packets. The first byte is a Sync Field that marks the beginning of the token's bit stream. The second byte is the *Packet Identification*.

The PID byte defines four types of token packets. These include an Out packet that carries data from the host to a device; an In packet that carries data from the device to the host; a Setup packet that targets a specific Endpoint; and a Start of Frame packet that helps synchronize the system. Table 21.22 matches the PID code with the Token Packet type.

Table 21.22. Token Packet Types

Packet identification byte	*Token packet type*
00011110	Out
01011010	Start of Frame (SOF)
10010110	In
11010010	Setup

For In, Out, and Setup Token Packets, the seven bits following the PID encode the *Address Field,* which identifies the device that the host wants to command or send data to. Four additional bits supply an *Endpoint* number. An Endpoint is an individually addressable section of a USB function. Endpoints give hardware designers the flexibility to divide a single device into logically separate units. For example, a keyboard with a built-in trackball might have one overall address to act as a single USB function. Assigning individual Endpoints to the keyboard section and the trackball section allows device designers to individually address each part of the overall keyboard.

Start of Frame packets differ from other USB packets in that they are broadcast. All devices receive and decode, but do not acknowledge, them. The 11 bits that would otherwise make up the Address and Endpoint fields indicate a Frame Number. The host sends out one Start of Frame packet each millisecond, as the name suggests defining the beginning of the USB's one-millisecond frame. The host assigns frame numbers incrementally, starting with zero and adding one for each subsequent frame. When it reaches the maximum 11-bit value (3072 in decimal), it starts over from zero. Figure 21.19 is a graphical representation of the Start of Frame type of Token packet.

Figure 21.19 Constituents of a USB Start of Frame form of Token Packet.

All Token Packets end with five bits of cyclic redundancy check information. The CRC data provides an integrity check of the Address Field and Endpoint. It does not cover the PID, which has its own, built-in error correction.

Data Packets

The actual information transferred through the USB system takes the form of Data Packets.

As with all USB packets, a Data Packet begins with a one-byte Sync Field followed by the Packet Identification. The actual data follows as a sequence of zeroes to 1,023 bytes. A two-byte cyclic redundancy check verifies the accuracy of only the data field. The PID field relies on its own redundancy check mechanism. Figure 21.20 shows a graphical representation of a USB Data Packet.

Figure 21.20 Constituents of a USB Data Packet.

The PID field ostensibly defines two types of Data Packet, Data 0 and Data 1. Functionally, however, the two data types (and hence the PID) form an additional error checking system between the data transmitter and receiver. The transmitter toggles between Data 0 and Data 1 to indicate that it has received a valid acknowledgment of

receipt of the preceding data packet. In other words, it confirms the confirmation. Table 21.23 summarizes these USB data packet types.

Table 21.23. USB Data Packet Types

Packet identification	*Data packet type*
00110011	Data 0
10110010	Data 1

For example, the transmitter sends out a Data Packet of the type Data 0. After the receiver successfully decodes the packet, it sends an acknowledgment signal back to the transmitter in the form of a Handshake Packet. If the transmitter successfully receives and decodes the acknowledgment, the next Data Packet it sends will be Data 1. From this change in Data Packet type, the receiver knows that its acknowledgment was properly received.

Handshake Packets

Handshake packets are two bytes long, comprising a Sync Field and a Packet Identification. Figure 21.21 graphically illustrates a USB Handshake Packet, and Table 21.24 lists the three forms of this packet type.

Figure 21.21 Constituents of a USB Handshake Packet.

Table 21.24. USB Handshake Packet Types

Packet Identification Byte	*Handshake type*
00101101	ACK
01011010	NAK
11100001	STALL

IEEE-1394

Compared to the performance you've come to expect from your PC, serial ports are

slow at best. They are constrained not only by the pragmatic aspects of their design—UART chips and the clocks that control them—but also by the medium through which the signals travel. Single-ended signals and cables of dubious quality are the communications equivalent of unleashing a go-kart with a lawnmower engine on an Autobahn that's unfettered by speed limits. The chances your data will get where its going unscathed are slim and, even if successful, the trip will be slow. As long as the medium remains the same, improvements in serial signal speed will be chancy, if possible at all.

The best way to accelerate the serial trip is to redefine both the medium and the method. The IEEE embarked on exactly that goal and is working on a proposal called P1394 to be the serial port of the future. The goal of the effort is to give computer and peripheral makers a low cost but high speed interface for linking devices and systems. Rather than replacing the RS-232C port alone, proponents see P1394 as a substitute for all the odd and varied ports on the back of your PC. P1394 has the potential for replacing not only your serial port but the parallel port, SCSI port, even the video connector.

Cross the slowest port in your PC with the most cantankerous one, and what do you get? Not an engineer's nightmare but a vision of the future called P1394. Although this up and coming standard combines the serial technology of today's laggardly RS-232C port with the intelligence of SCSI protocol, it takes the best instead of worst of each and makes an interconnection system with the speed of local bus, the wiring ease of MIDI, and economy in keeping with today's plunging PC prices. Add to the list of mandatory equipment on your next PC another port.

More than the next generation of serial communications, P1394 will likely be the connection that brings mass market simplicity to multimedia. One connection could do it all, linking as many as 16 peripherals. Advocates of P1394 imagine it linking PCs not just to traditional devices like CD ROM drives, hard disks, modems, printers, and scanners but also to video cameras and stereo systems. Easier to plug together than ordinary stereo components, P1394 eliminates the wiring confusion that scares technophobes from trying and using computer technology. If you can manage plugging your PC into a wall outlet, you can connect the most elaborate multimedia system. In short, P1394 is key to pushing computing technology into home and everyday entertainment.

Background

Development of the new standard began nearly a decade ago in September 1986 when the IEEE (Institute of Electrical and Electronic Engineers) assigned a study group the task of clearing the murk of thickening morass of serial standards. Hardly four months later (in January 1987), the group had already outlined basic concepts underlying

P1394, some of which still survive in today's standard—including low cost, a simplified wiring scheme, and arbitrated signals supporting multiple devices. Getting the devil out of such details as operating speed and the technologies needed to achieve it, took years because needs, visions, and visionaries changed. Consensus on the major elements of the standard—including the connector and the bus management—came only in 1993, and the standard reached final form in late 1994. Achieving its worthy goals required four breakthrough new technologies, including a novel encoding system that made high speed safe for serial data, a self-configuration system that moved the headaches of setup from users to the port circuitry, a time based arbitration system that guarantees all of the many devices linked to a single port have fair and guaranteed access, and a means of delivering time critical data like video without affecting the transfer of serial data.

P1394 truly offers something for everyone—today's relatively skilled PC user, tomorrow's casual home user, and even machine makers.

For manufacturers, the cost of P1394 may prove most alluring. P1394 has the potential of reducing the cost of external connections to PCs both in terms of money spent and panel usage. Both of these savings originate in the design of the P1394 connector. P1394 envisions a single 6-wire plastic connector replacing most if not all of the standard port connectors on a PC. As with today's SCSI, one P1394 port on a PC allows you to connect multiple devices, up to 16 in current form.

The connector itself will cost manufacturers a few cents while the connectors alone for an RS-232C port can cost several dollars (and that can be a significant portion of the price of a peripheral or even PC). Moreover, a standard serial connector—that 25-pin D-shell connector—by itself is much too large for today's miniaturized systems. It can't fit a PCMCIA card by any stretch of the imagination or plastic work.

As less skilled people start tinkering with PCs and try linking them into multimedia systems, the simplified setup and wiring of P1394 should earn their praises. Today's high performance interface choice, SCSI, is about as friendly as a hungry bear awakened from hibernation. Although backed by strong technology, SCSI is a confusion of connectors, cables, terminators, and ID numbers. Wise folks find the best strategy is to stay out of the way. Where cabling a SCSI system means following rules more obscure than those of a fantasy adventure game, P1394 has exactly one wiring requirement: all P1394 devices in a system must connect without loops. There are no terminations to worry about, no different cable types like straight through and crossover, no cable length concerns, no identification numbers, and no connector genders to change. You simply plug one end of a P1394 cable into a jack on the back of the two devices you want to link. Most P1394 cables will have two or three jacks, so you can wire together elaborate webs. As long as no more than one circuit runs between any two P1394 devices, the system will work. It's even easier than a stereo system because there are no worries about input and output jacks.

Down deeper, however, P1394 is more complex. Instead of simply needing a UART,

P1394 is a complex communications system with its own transfer protocol requiring new, application specific, integrated circuits. Although those initially will be expensive—estimated $15 per P1394 device—throughout the history of PCs, the cost of standard silicon circuits has plummeted while the cost of connectors continued to climb. Moreover, the current cost isn't entirely out of line with today's serial technology where a 16550AFN UART alone can cost $5 to $10. Just as the electrically more complex AT interface replaced older interfaces for hard disks, P1394 stands to step in place of the serial port.

Performance

For today's PC users, speed is probably the most important aspect of P1394. Serial connections exchange simplicity for the constraints of moving one bit at a time through a narrow data channel. For example, the standard RS-232C port on your PC tops out at 115,200 bits per second. Although the top rate is set by the timebase design of the original IBM PC of 1981, electrical issues like interference and wire capacitance constrain RS-232C transmissions to substantially slow data rates on longer connections.

In contrast, P1394 starts with a raw data rate of 100 megabits per second and some devices will be able to shift bits at speeds up to four times that rate. Although P1394 imposes substantial software overhead because of its packet based nature and the needs for addressing and arbitration, it still offers enough bandwidth to carry three simultaneous video signal or 167 CD-quality audio signals at its base 100 Mbits/sec rate. In current form, it allows hard disks to matches the 10MByte/sec transfer rate of Fast SCSI-2 connections.

Timing

Reliability is a problem with any high speed circuit, and the designers of P1394 faced a formidable challenge. More than does any comedian, P1394 depends on precise timing. The meaning of each bit in a transmission depends on when the bit gets registered. At the high data rates of P1394, signal jitter becomes a major problem. Each bit must be defined to fit precisely into a frame 10 billionths of a second long; the slightest timing error can cause an error. In designing P1394, engineers tried elaborate coding schemes to eliminate jitter problems. In the end, they created an entirely new signaling system.

To minimize noise, data connections in P1394 use differential signals. Ordinary RS-232C serial ports use single-ended signals. One wire carries the data, and the ground connection serves as the return path. Differential signaling uses two wires that carry the

same signal but of different polarities. Receiving equipment subtracts the signal on one wire from that on the other to find the data as the difference between the two signals. The benefit of this scheme is that any noise gets picked up by the wires equally. When the receiving equipment subtracts the signals on the two wires, the noise gets eliminated—the equal noise signals subtracted from each other equal zero.

P1394 goes further, using two differential wire pairs. One pair carries the actual data; the second pair, called the *strobe lines*, complements the state of the data pair so that one and only one of the pairs changes polarity every clock cycle. For example, if the data line carries two sequential bits of the same value, the strobe line reverses polarity to mark the transition between them. If a sequence of two bits changes the polarity of the data lines (a one followed by a zero or zero followed by a one), the strobe line does not change polarity. Summing the data and strobe lines together exactly reconstructs the clock signal of the sending system, allowing the sending and receiving devices to precisely lock up.

Setup

As with existing SCSI systems, P1394 allows you to connect multiple devices and uses an addressing system so the signals sent through a common channel are recognized only by the proper target device. The linked devices can independently communicate among themselves without the intervention of your PC.

In order to communicate, however, devices must be able to identify one another. Providing proper ID has been one of the recurring problems with SCSI, requiring you to set switches on every SCSI device and then indicate your choices when configuring software. P1394 eliminates such concerns with its own automated configuration process.

Whenever a new devices gets plugged into a P1394 system (or when the whole system gets turned on), it starts its automatic configuration process. By signaling through the various connections, each device determines how it fits into the system, either as a root node, a branch, or a leaf. Each P1394 system has only one root, which is the foundation around which the rest of the system organizes itself. The node also sends out a special clock signal). P1394 devices with only one connection are leaves; those that link to multiple devices are branches. Once the connection hierarchy is setup, the P1394 devices determine their own ID numbers from their location in the hierarchy and send identifying information (ID and device type) to their host.

Arbitration

P1394 also relies on timing for its arbitration system. As with a SCSI or network connection, P1394 transfers data in packets, a block of data preceded by a header that specifies where the data goes and its priority. In the basic cable based P1394 system, each device sharing a connection gets a chance to send one packet in an arbitration period that's called a fairness interval. The various devices take turns until all have had a chance to use the bus. After each packet gets sent, a brief time called the *sub-action gap* elapses, after which another device can send its packet. If no device starts to transmit when the sub-action gap ends, all devices wait a bit longer, stretching the time to an arbitration reset gap. After that time elapses, a new fairness interval begins, and all devices get to send one more packet. The cycle continues.

To handle devices that need a constant stream of data for real time display, such as video or audio signals, P1394 uses a special isochronous mode. Every 125 microseconds, one device in the P1394 that needs isochronous data sends out a special timing packet that signals isochronous devices that they can transmit. Each takes a turn in order of its priority, leaving a brief isochronous gap delay between their packets. When the isochronous gap delay stretches out to the sub-action gap length, then the devices using ordinary asynchronous transfers take over until the end of the 125 microsecond cycle when the next isochronous period begins.

The scheme guarantees that video and audio gear can move its data in real time with a minimum of buffer memory. (Audio devices require only a byte of buffer; video may need as many as six bytes!) The 125-microsecond period matches the sampling rate used by digital telephone systems to help P1394 mesh with ISDN (Integrated Service Digital Network) telephone systems.

The key to the power of P1394 is speed. The initial design of P1394 sets up a 100 megabit-per-second data transfer protocol. In addition, the standard defines two higher speed data rates for future upgrades, 200 and 400 megabits per second.

From a manufacturer's standpoint, size and cost are as important as speed. P1394 has the potential of reducing the cost of external connections to PCs both in terms of money spent and panel usage. Both of these savings originate in the design of the P1394 connector. P1394 envisions a single 6-wire plastic connector replacing most, if not all, of the standard port connectors on a PC.

The connector itself will cost manufacturers a few cents while the connectors alone for an RS-232C port can cost several dollars (and that can be a significant portion of the price of a peripheral or even PC). Moreover, a standard serial connector —25-pin D-shell—by itself is much too large for today's miniaturized systems. It can't fit a PCMCIA card by any stretch of the imagination or plastic work.

Those savings have a price. Far from simply needing a UART, P1394 is a complex

communications system with its own transfer protocol. It will require complex circuits to work. As the interface becomes popular, however, the cost of this circuitry will quickly plunge below the cost of the more sophisticated connectors used by other interfaces. Just as the electrically more complex AT interface replaced older interfaces for hard disks, P1394 stands to step in place of the serial port.

The future imagined by P1394 advocates is much like the early Macintosh computers that depended on a single SCSI port for all system expansion. P1394 beats old SCSI both in connector simplicity and cost. But it also joins SCSI—P1394 is one of the hardware channels that are incorporated into the proposed SCSI-3 standard.

As with existing SCSI systems, P1394 allows you to connect multiple devices and uses an addressing system so the signals sent through a common channel are recognized only by the proper target device. The linked devices can independently communicate among themselves without the intervention of your PC. But P1394 gives you greater wiring flexibility than the current SCSI standards. To link multiple peripherals, you can daisy chain them or split the cable into branches. In effect, the P1394 connection behaves like a small (but fast) network.

Architecture

P1394 is a true architecture that is built from several layers, each of which defines one aspect of the serial connection. These layers include a bus management layer, a transaction layer, a link layer, and a physical layer.

Bus Management Layer

This part of the P1394 standard defines the basic control functions as well as the control and status registers required by connected devices to operate their ports. This layer handles channel assignments, arbitration, mastering, and errors.

Transaction Layer

The protocol that governs transactions across the P1394 connection is called the transaction layer. That is, this layer mediates the read and write operations. To match modern PCs, the transaction layer is optimized to work with 32-bit double-words, although the standard also allows block operations of variable length. The operation of this layer was derived from the IEEE 1212 parallel data-transfer standard.

Link Layer

The logical control on the data across the P1394 wire is the link layer, making the transfer for the transaction layer. Communications are half-duplex transfers, but the link layer provides a confirmation of the reception of data. Double-word transfers are favored, but the link layer also permits exchanges in variable length blocks.

Physical Layer

The actual physical connections made by P1394 are governed by the physical layer. This part of the standard includes both a protocol and the medium itself. The physical protocol sublayer controls access to the connection with full arbitration. The physical medium sublayer comprises the cable and connectors.

Cabling

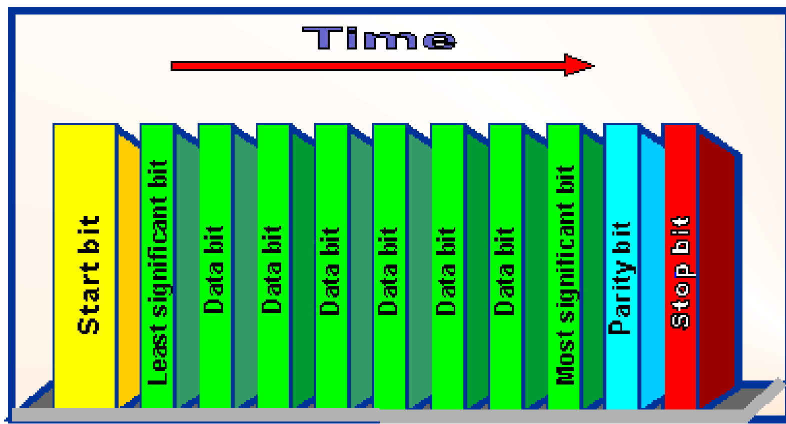
In initial form, the physical part of P1394 will be copper wires. The standard cable is a complex weaving of six conductors. Data will travel down two shielded twisted pairs.

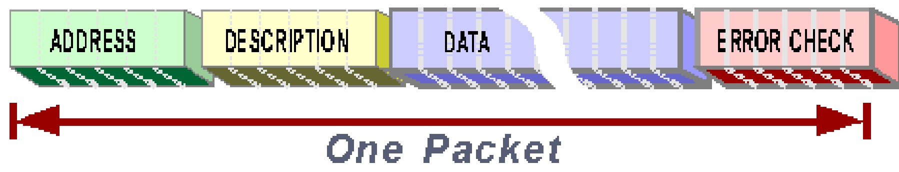
Two wires will carry power at 8 to 40 volts with sufficient current to power a number of peripherals. Another shield will cover the entire collection of conductors. A small, 6-pin connector will link PCs and peripherals to this cable.

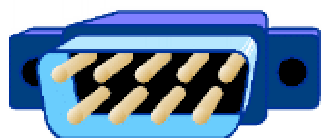
The P1394 wiring standard allows for up to 32 hops of 4.5 meters (about 15 feet) each. As with current communications ports, the standard allows you to connect and disconnect peripherals without switching off power to them. You can daisy chain P1394 devices or branch the cable between them. When you make changes, the network of connected devices will automatically reconfigure itself to reflect the alterations.

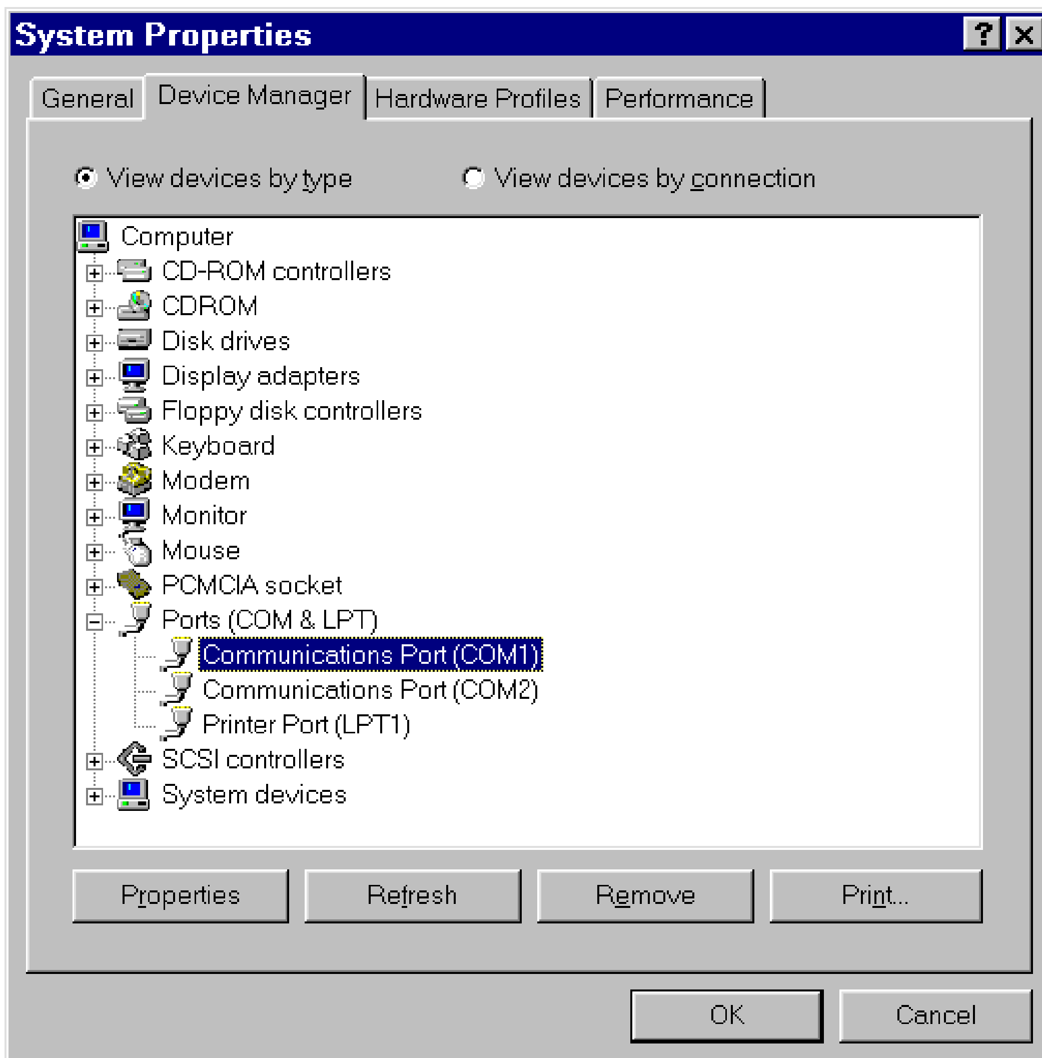
The P1394 wiring scheme depends on each of the connected devices to relay signals to the others. Pulling the plug to one device could potentially knock down the entire connection system. To avoid such difficulties and dependencies, P1394 uses its power connections to keep in operation the interface circuitry in otherwise inactive devices. These power lines could also supply enough current to run entire devices. No device may draw more than three watts from the P1394 bus, although a single device may supply up to 40 watts. The P1394 circuitry itself in each interface requires only about 2 milliwatts.

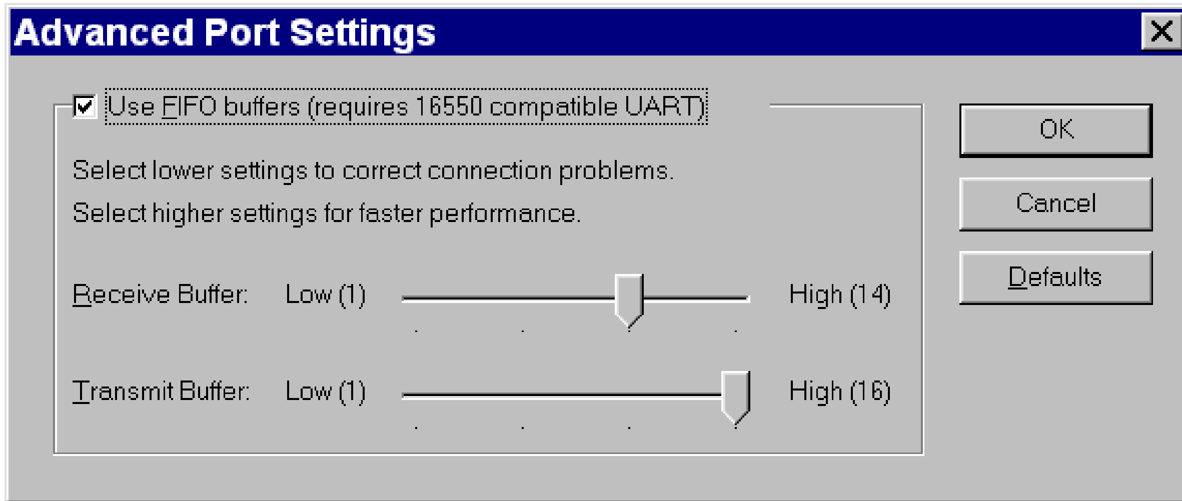
The P1394 wiring standard allows for up to 16 hops of 4.5 meters (about 15 feet) each. As with current communications ports, the standard allows you to connect and disconnect peripherals without switching off power to them. You can daisy chain P1394 devices or branch the cable between them. When you make changes, the network of connected devices will automatically reconfigure itself to reflect the alterations.

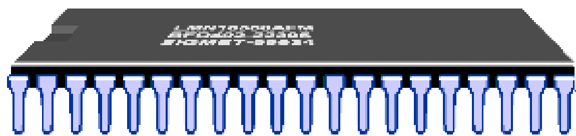

Time
Start bit
Least significant bit
Data bit
Data bit
Data bit
Data bit
Data bit
Data bit
Most significant bit
Parity bit
Stop bit


ADDRESS
DESCRIPTION
DATA
ERROR CHECK
One Packet




System Properties
General
Device Manager
Hardware Profiles
Performance
View devices by type
View devices by connection
Computer
CD-ROM controllers
CDROM
Disk drives
Display adapters
Floppy disk controllers
Keyboard
Modem
Monitor
Mouse
PCMCIA socket
Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Printer Port (LPT1)
SCSI controllers
System devices
Properties
Refresh
Remove
Print...
OK
Cancel

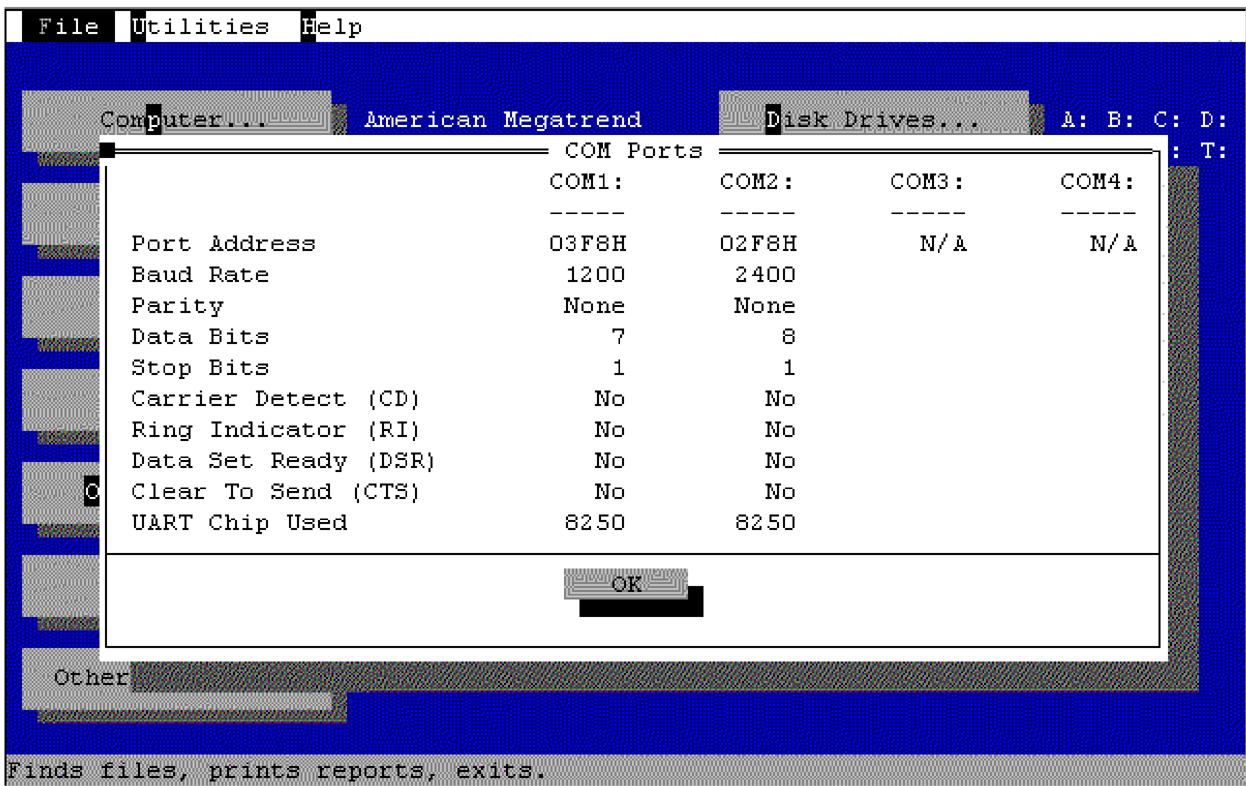

Advanced Port Settings
Use FIFO buffers (requires 16550 compatible UART)
Select lower settings to correct connection problems.
Select higher settings for faster performance.
Receive Buffer: Low (1) High (14)
Transmit Buffer: Low (1) High (16)
OK
Cancel
Defaults

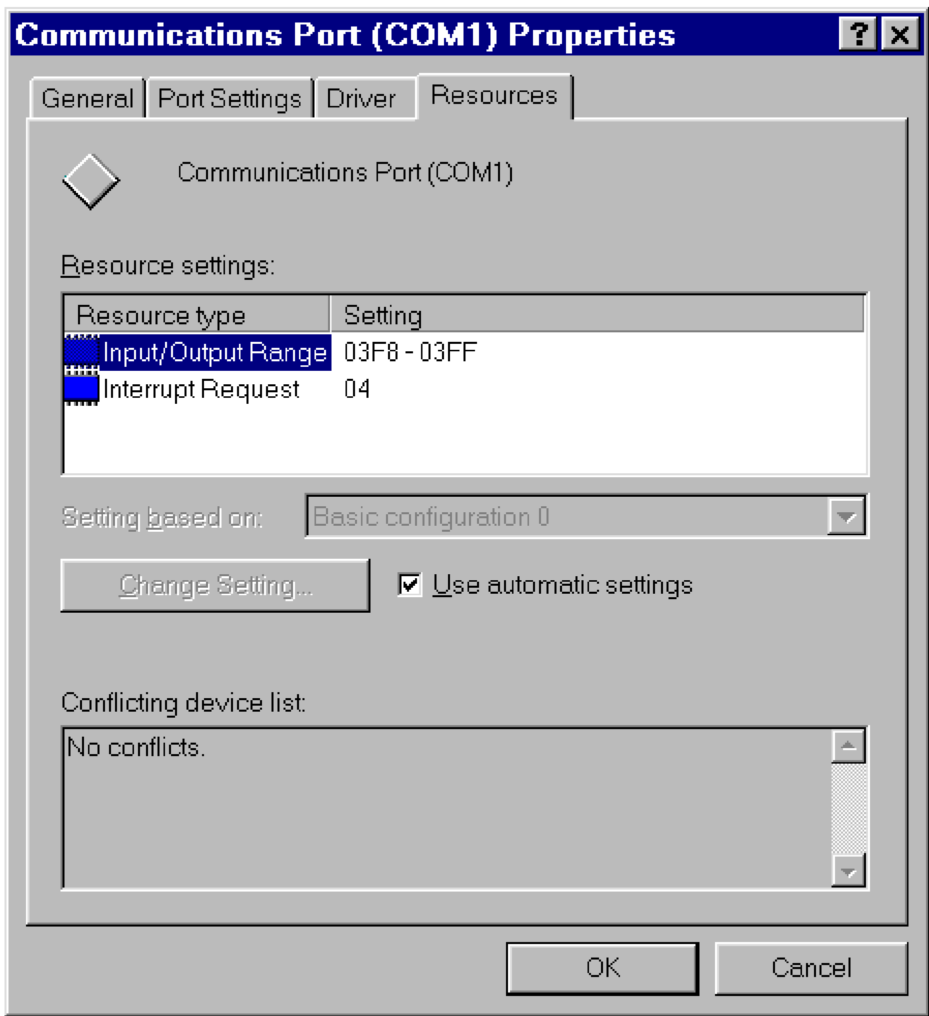


File Utilities Help

Computer...	American Megatrend 486DX	Disk Drives...	A: B: C: D: E: F: S: T:
Memory...	636K, 15360K Ext, 15120K XMS	LPT Ports...	1
Video...	VGA, Matrox Vision	COM Ports...	2
Network...	LANtastic	IRQ Status...	
OS Version...	MS-DOS 7.00 Windows 3.10	TSR Programs...	
Mouse...	No Mouse Installed 6.04	Device Drivers...	
Other Adapters...	Game Adapter		

Press ALT for menu, or press highlighted letter, or F3 to quit MSD.


File Utilities Help
Computer...
American Megatrend
Disk Drives...
A: B: C: D:
: T:
COM Ports
COM1: COM2: COM3: COM4:
Port Address 03F8H 02F8H N/A N/A
Baud Rate 1200 2400
Parity None None
Data Bits 7 8
Stop Bits 1 1
Carrier Detect (CD) No No
Ring Indicator (RI) No No
Data Set Ready (DSR) No No
Clear To Send (CTS) No No
UART Chip Used 8250 8250
OK
Other
Finds files, prints reports, exits.

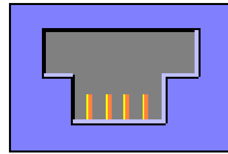

Communications Port (COM1) Properties
General
Port Settings
Driver
Resources
Communications Port (COM1)
Resource settings:
Resource type
Setting
Input/Output Range
03F8 - 03FF
Interrupt Request
04
Setting based on:
Basic configuration 0
Change Setting...
Use automatic settings
Conflicting device list:
No conflicts.
OK
Cancel

1 — Ground

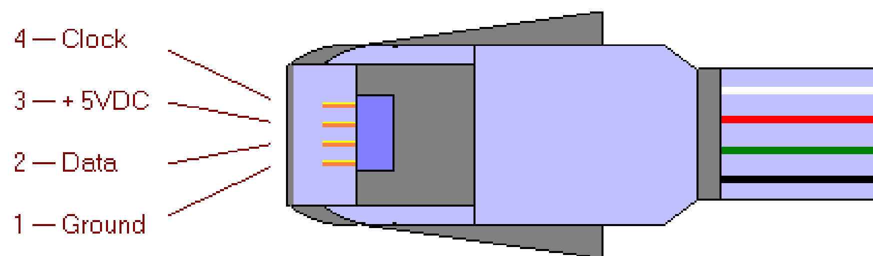
2 — Data

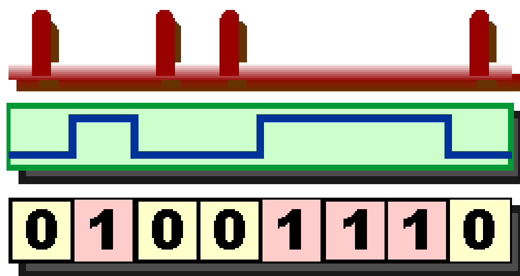
3 — + 5VDC

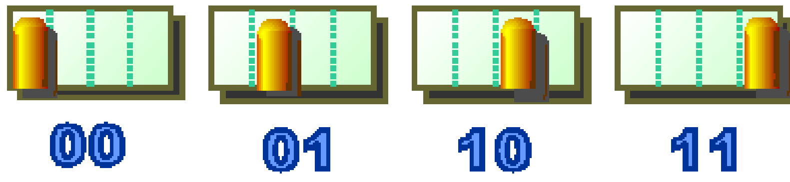
4 — Clock

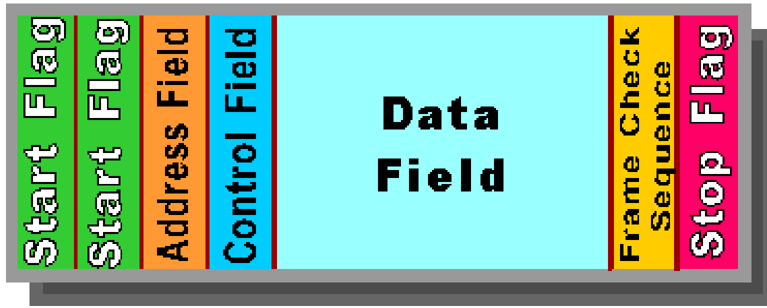


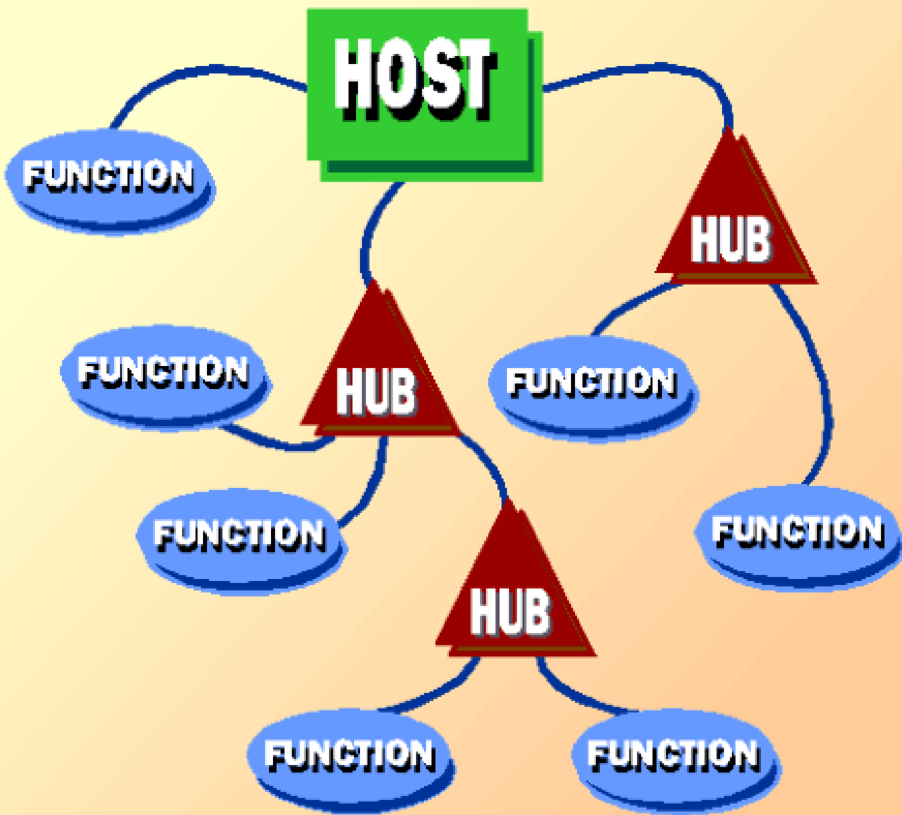
4 3 2 1

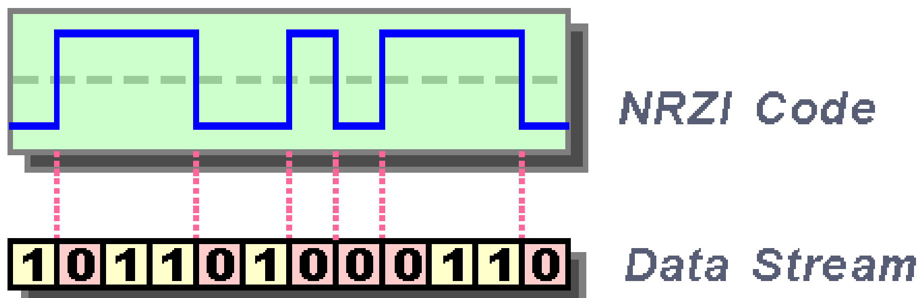

4 — Clock
3 — + 5VDC
2 — Data
1 — Ground

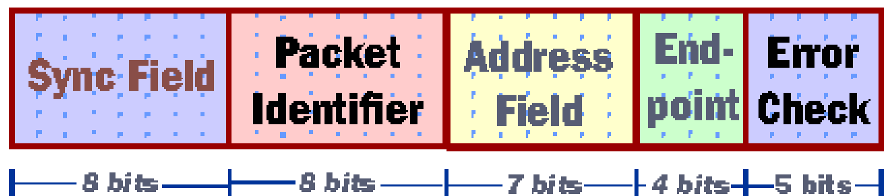

IrDA Pulses
Logic Level
0 1 0 0 1 1 1 0
Original Code

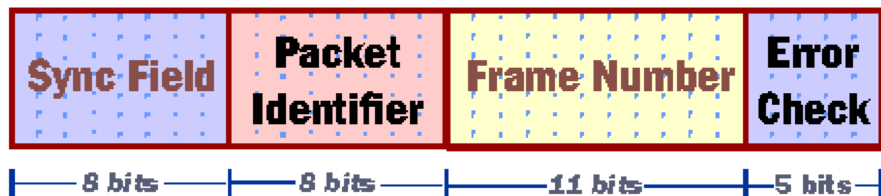

00
01
10
11

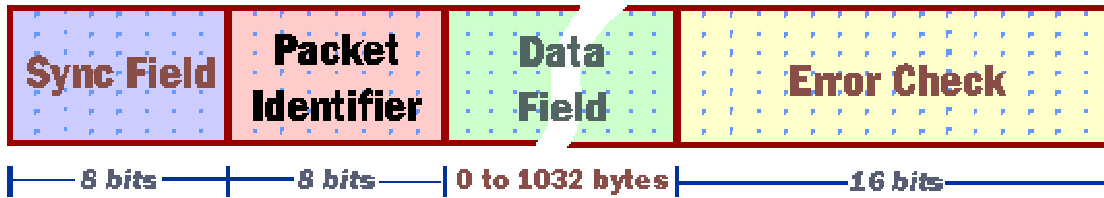

Start Flag
Start Flag
Address Field
Control Field
Data Field
Frame Check Sequence
Stop Flag

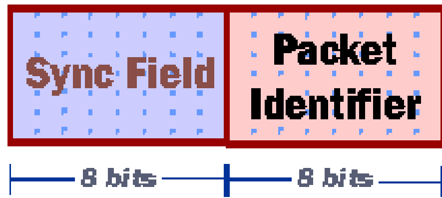

HOST
FUNCTION
HUB
FUNCTION
HUB
FUNCTION
FUNCTION
FUNCTION
HUB
FUNCTION
FUNCTION


NRZI Code
1 0 1 1 0 1 0 0 0 1 1 0
Data Stream


Sync Field
Packet Identifier
Address Field
End-point
Error Check
8 bits
8 bits
7 bits
4 bits
5 bits


Sync Field
Packet Identifier
Frame Number
Error Check
8 bits
8 bits
11 bits
5 bits


Sync Field
Packet Identifier
Data Field
Error Check
8 bits
8 bits
0 to 1032 bytes
16 bits


Sync Field
Packet Identifier
8 bits
8 bits

Chapter 19: Parallel Ports

Parallel ports are well-defined, convenient, and quick—probably the most trouble free connection you can make with your PC. Once the exclusive province of printers, with the advent of the Enhanced Parallel Port they promise to be the universal interface—the duct tape of PC ports. An increasing number of peripherals are taking advantage of the fast, sure, parallel connection. But all parallel ports are not the same, nor are all parallel connections. A port that works for a printer may fail dismally when you attempt to transfer files across it. It is all a matter of design.

- IEEE 1284
    - History
    - Connectors
        - The A connector
        - The B Connector
        - The C Connector
        - Adapters
    - Cable
    - Electrical Operation
        - Compatibility Mode
        - Nibble Mode
        - Byte Mode
        - Enhanced Parallel Port Mode
        - Extended Capabilities Port Mode
    - Logical Interface
        - Input/Output Ports
        - Device Names
        - Interrupts
        - Port Drivers

- Control
  - Traditional Parallel Ports
  - Enhanced Parallel Ports
- Performance Issues
  - Timing
  - Data Compression
  - Bus Mastering
- Plug-and-Play
  - Benefits
  - Requirements
  - Operation
- GP-IB Interface

19

Parallel Ports

When it comes to connecting peripherals to your PC, the parallel port appears to be the answer to your prayers. At one time, the chief value of the parallel port was its being a foolproof connection for printers. You could plug in a cable and everything would work. No switches to worry about, no mode commands, no breaking out the break-out box to sort through signals with names that sound suspiciously similar to demons from Middle Earth.

The parallel connection proved too intriguing for engineers, however. It was inherently faster than the only other standard PC port at the time, the RS-232 serial port. By tinkering with the parallel port design, the engineers first broke the intimate link between the port and your printer, and opened it as a general purpose high speed connection. Not satisfied with a single standard, they developed several. In the process, the port lost its trouble free installation.

Now new interconnection standards, designed for utter simplicity so that even the basest fool can connect them properly, stand to steal the parallel port's role as the highest speed player. While the industry tries to catch up with the engineers, however, the parallel port will likely remain an important interconnection standard.

Today the parallel port is not a singular thing. Through the years, a variety of parallel port standards

have developed. Today you face questions about three standard connectors and four operational standards, not to mention a number of proprietary detours that have appeared along the way. Despite their innate differences—and often great performance differences—all are termed *parallel ports*. You may face combinations of any of the three connectors with any of the four operational standards in a given connection. With just a look, you won't be able to tell the difference between them, or what standard the connection uses. In fact, you might become aware of the differences only when you start to scratch your head and wonder why your friends can unload files from their notebook computers ten times faster than you can.

Beside the traditional parallel port, several other PC interface systems use parallel connections. For example, the classic SCSI port is a parallel interface. Most of these other parallel designs are most applicable for specific hardware. For example, because SCSI finds its most important application in linking hard disk drives, we've already discussed it in the chapter about disk interfaces (Chapter 9, "Storage Interfaces"). One other parallel interface system, the General Purpose Interface Bus or GPIB, has been used among PCs to link a variety of peripherals. Because of its parallel nature—and because we couldn't think of a better place to put it—we've included a brief discussion of GPIB in this chapter, too.

IEEE 1284

The defining characteristic of the parallel port design is implicit in its name. The port is parallel because it conducts its signals through eight separate wires—one for each bit of a byte of data—that are enclosed together in a single cable. The signal wires literally run in parallel from your PC to their destination—or at least they did. Better cables twist the physical wires together but keep their signals straight (and parallel).

In theory, having eight wires means you can move data eight times faster through a parallel connection than through a single wire. All else being equal, simple math would make this statement true. Although a number of practical concerns make such extrapolations impossible, throughout its life, the parallel port has been known for its speed. It beat its original competitor, that RS-232 port hands down, outrunning the serial port's 115.2 kbit/sec maximum by factors from two to five even in early PCs. The latest incarnations of parallel technology put the data rate through the parallel connection to over 100 times faster than the basic serial port rate.

In simple installations, for example when used for its original purpose of linking a printer to your PC, the parallel port is a model of installation elegance. Just plug your printer in, and the odds are it will works flawlessly—or that whatever flaws appear won't have anything to do with the interconnection.

Despite such rave reviews, parallel ports are not trouble free. All parallel ports are not created equal. A number of different designs have appeared during the brief history of the PC. Although new PCs usually incorporate the latest, most versatile, and highest speed of these, some manufacturers skimp. Even when you buy a brand new computer, you may end up with a simple printer port that steps back

to the first generation of PC design.

A suitable place to begin this saga is to sort out this confusion of parallel port designs by tracing its origins. As it turns out, the history of the parallel port is a long one, older even than the PC, although the name, and our story, begins with its introduction.

History

Necessity isn't just the mother of invention. It also spawned the parallel port. As with most great inventions, the parallel port arose with a problem that needed to be solved. When IBM developed its first PC, its engineers looked for a simplified way to link to a printer, something without the hassles and manufacturing costs of a serial port. The simple parallel connection, already used in a similar form by some printers, was an elegant solution. Consequently, IBM's slightly modified version became standard equipment on the first PCs. Because of its intended purpose, it quickly gained the "printer port" epithet. Not only were printers easy to attach to a parallel port, they were the only thing that you could connect to these first ports at the time.

In truth, the contribution of PC makers to the first parallel port was minimal. They added a new connector that better fit the space available on the PC. The actual port design was already being used on computer printers at the time. Originally created by printer maker Centronics Data Computer Corporation and used by printers throughout the 1960s and 70s, the connection was electrically simple, even elegant. It took little circuitry to add to a printer or PC even in the days when designers had to use discrete components instead of custom designed circuits. A few old-timers still cling to history and call the parallel port a *Centronics* port.

The PC parallel port is not identical to the exact Centronics design, however. In adapting it to the PC, IBM substituted a smaller connector. The large jack used by the Centronics design had 36 pins and was too large to put where IBM wanted it—sharing a card retaining bracket with a video connector on the PC's first Monochrome Display Adapter. In addition, IBM added two new signals to give the PC more control over the printer and adjusted the timing of the signals traveling through the interface. All that said, most Centronics-style printers worked just fine with the original PC.

At the time, the PC parallel port had few higher aspirations. It did its job, and did it well, moving data one direction—from PC to printer—at rates from 50 to 150 kilobytes per second. It, or subtle variations of it, became ubiquitous if not universal. Any printer worth connecting to a PC used a parallel port (or so it seemed).

In 1987, however, IBM's engineers pushed the parallel port in a new direction. The motivation for the change came from an odd direction. The company decided to adopt the 3.5-inch floppy disk drives for its new line of PS/2 computers at a time when all the world's PC data was mired on 5.25-inch diskettes. The new computers made no provision for building in the bigger drives. Instead, IBM believed that the entire world would instantly switch over to the new disk format. People would need

to transfer their data once and only once to the new disk format. To make the transfer possible, the company released its *Data Migration Facility*, a fancy name for a cable and a couple disks. You used the cable to connect your old PC to your new PS/2, and software on the disks to move files through the parallel port from the old machine and disks to the new ones.

Implicit in this design is the ability of the PS/2 parallel port to receive data as well as send it out, as to a printer. The engineers tinkered with the port design and made it work both ways, creating a *bi-directional parallel port*. Because of the design's intimate connection with the PS/2, it is sometimes termed the *PS/2 parallel port*.

The Data Migration Facility proved to be an inspirational idea despite its singular shortcoming of working in only one direction. As notebook computers became popular, they also needed a convenient means to move files between machines. The makers of file transfer programs like *Brooklyn Bridge* and *LapLink* knew a good connection when they saw it. By tinkering with parallel port signals, they discovered that they could make any parallel port operate in both directions and move data to and from PCs.

The key to making bi-directional transfers on the old-fashioned one way ports was to redefine signals. They redirected four of the signals in the parallel connector that originally had been designed to convey status information back from the printer to your PC. These signals already went in the correct direction. All that the software mavens did was to take direct control of the port and monitor the signals under their new definitions. Of course, four signals can't make a byte. They were limited to shifting four bits through the port in the backward direction. Because four bits make a nibble, the new parallel port operating mode soon earned the name *nibble mode*.

Four-bits-at-a-time had greater implications than just a new name. Half as many bits also means half the speed. Nibble mode operates at about half the normal parallel port rate—still faster than single-line serial ports but no full parallel speed.

If both sides of a parallel connection had bi-directional ports, however, data transfers ran at full speed both ways. Unfortunately, as manufacturers began adapting higher performance peripherals to use the parallel port, what they once thought was fast performance became agonizingly slow. Although the bi-directional parallel port more than met the modest data transfer needs of printers and floppy disk drives, it lagged behind other means of connecting hard disks and networks to PCs.

Engineers at network adapter maker Xircom, Incorporated decided to do something about parallel performance and banded together with notebook computer maker Zenith Data Systems to find a better way. Along the way, they added Intel Corporation, and formed a triumvirate called Enhanced Parallel Port Partnership. They explored two ways of increasing the data throughput of a parallel port. They streamlined the logical interface so that your PC would need less overhead to move each byte through the port. In addition, they tightly defined the timing of the signals passing through the port, minimizing wasted time and helping assure against timing errors. They called the result of their efforts the *Enhanced Parallel Port*.

On August 10, 1991, the organization released its first description of what they though the next

generation of parallel port should be and do. They continued to work on a specification until March 1992, when they submitted Release 1.7 to the Institute of Electrical and Electronic Engineers (the IEEE) to be considered as an industry standard.

Although the EPP version of the parallel port can increase its performance by nearly tenfold, that wasn't enough to please everybody. The speed potential made some engineers see the old parallel port as an alternative to more complex expansion buses like the SCSI system. With this idea in mind, Hewlett-Packard joined with Microsoft to make the parallel port into a universal expansion standard called the *Extended Capabilities Port* (or ECP). In November 1992, the two companies released the first version of the ECP specification aimed at computers that use the ISA expansion bus. This first implementation adds two new transfer modes to the EPP design—a fast two way communication mode between a PC and its peripherals, and another two way mode with performance further enhanced by simple integral data compression—and defines a complete software control system.

The heart of the ECP innovation is a protocol for exchanging data across a high speed parallel connection. The devices at the two ends of each ECP transfer negotiate the speed and mode of data movement. Your PC can query any ECP device to determine its capabilities. For example, your PC can determine what language your printer speaks and set up the proper printer driver accordingly. In addition, ECP devices tell your PC the speed at which they can accept transmissions and the format of the data they understand. To assure the quality of all transmissions, the ECP specification includes error detection and device handshaking. It also allows the use of data compression to further speed transfers.

On March 30, 1994, the IEEE Standards Board approved its parallel port standard, *IEEE-1284-1994*. The standard included all of the basic modes and parallel port designs including both ECP and EPP. It was submitted to the American National Standards Institute and approved as a standard on September 2, 1994.

The IEEE 1284 standard marks a watershed in parallel port design and nomenclature. The standard defines (or redefines) all aspect of the parallel connection, from the software interface in your PC to the control electronics in your printer. It divides the world of parallel ports in two: *IEEE 1284-compatible devices*, which are those that will work with the new interface, which in turn includes just about every parallel port and device ever made; and *IEEE 1284-compliant devices,* those which understand and use the new standard. This distinction is essentially between pre- and post-standardization ports. You can consider IEEE 1284-*compatible* ports to be "old technology" and IEEE 1284-*compliant* ports to be "new technology."

Before IEEE 1284, parallel ports could be divided into four types: Standard Parallel Ports, Bi-directional Parallel Ports (also known as PS/2 parallel ports), Enhanced Parallel Ports, and Extended Capabilities Ports. The IEEE specification redefines the differences in ports, classifying them by the transfer mode they use. Although the terms are not exactly the same, you can consider a Standard Parallel Port one that is able to use only nibble-mode transfers. A PS/2 or Bi-directional Parallel Port from the old days is one that can also make use of byte-mode transfers. EPP and ECP ports are those that use EPP and ECP modes, as described by the IEEE 1284 specification.

EPP and ECP remain standards separate from IEEE 1284, although they have been revised to depend

on it. Both EPP and ECP rely on their respective modes as defined in the IEEE specification for their physical connections and electrical signaling. In other words, IEEE 1284 describes the physical and electrical characteristics of a variety of parallel ports. The other standards describe how the ports operate and link to your applications.

Connectors

The best place to begin any discussion of the function and operation of the parallel port is the connector. After all, the connector is what puts the port to work. It is the physical manifestation of the parallel port, the one part of the interface and standard you can actually touch or hold in your hand. It is the only part of the interface that most people will ever have to deal with. Once you know the ins and outs of parallel connectors, you'll be able to plug in the vast majority of PC printers and the myriad other things that now suck signals from what was once the printer's port.

Unfortunately, as with the variety of operating modes, the parallel port connector itself is not a single thing. Parallel port connectors come in enough different and incompatible designs to make matters interesting, enough subtle wiring variations to make troubleshooting frustrating, and enough need for explanation that this book can find its way into a fourth edition. Although the long-range prognosis for those suffering from connector confusion is good—eventually parallel ports will gravitate to a single connector design—in the short-term matters are destined only to get more confusing.

Before the IEEE-1284 standard was introduced, equipment designers used either of two connectors for parallel ports. On the back of your PC you would find a female 25-pin D-shell connector, IBM's choice to fit a parallel port within the confines allowed on the MDA video adapter. On your printer you would find a female 36-pin ribbon connector patterned after the original Centronics design. In its effort to bring standardization to parallel ports, the IEEE used these two designs as a foundation, formalizing them into the standard and giving them official names, the 1284-A and 1284-B connectors. The standard also introduced a new, miniaturized connector, 1284-C, similar to the old Centronics ribbon connector but about half the size.

The A connector

The familiar parallel port on the back of your PC was IBM's pragmatic innovation. At the time of the design of the original PC, many computers used a 37-pin D-shell connector for their printer ports that mated with Centronics-style printers. This connector was simply too long (about four inches) for where IBM wanted to put it. Slicing off 12 pins would make a D-shell connector fit, and it still could provide sufficient pins for all the essential functions required in a parallel port as long as some of the ground return signals were doubled (and tripled) up. Moreover, the 25-pin D-shell was likely in stock on the shelves wherever IBM prototyped the PC because the mating connector had long been used in

serial ports. IBM chose the opposite gender (a female receptacle on the PC) to distinguish it from a serial connection.

To retain compatibility with the original IBM design, other computer makers also adopted this connector. By the time the IEEE got around to standardizing the parallel port, the 25-pin D-shell was the standard. The IEEE adopted it as its 1284-A connector. Figure 19.1 shows a conceptual view of the A-connector.

Figure 19.1 The IEEE-1284 A connector, a female 25-pin D-shell jack.

The individual contacts appear as socket holes, spaced at intervals of one-tenth inch, center-to-center. On the printer jack as it appears in the illustration, pin one is on the upper right, and contacts are consecutively numbered right to left. Pin 14 appears at the far right on the lower row, and again the contacts are sequentially numbered right to left. (Because you would wire this connector from the rear, the contact number would appear there in more familiar left to right sequence.) The socket holes are encased in plastic to hold them in place, and the plastic filler itself is completely surrounded by a metal shell that extends back to the body of the connector. The entire connector measures about two inches wide and half an inch tall when aligned as shown in the illustration.

The studs at either side of the connector are 4-40 jack screws, which are essentially extension screws. They fit into the holes in the connector and attach it to a chassis. Instead of slotted heads, they provide another screw socket to which you can securely attach screws from the mating connector.

As a receptacle or jack for mounting on a panel such as the back of your PC, this connector is available under a number of different part numbers, depending on their manufacturer. Some of these include AMP 747846-4, Molex 82009, and 3M Company 8325-60XX and 89925-X00X. Mating plugs are available as AMP 747948-1, Molex 71527, and 3M 8225-X0XX.

Of the 25 contacts on this parallel port connector, 17 are assigned individual signals for data transfer and control. The remaining eight serve as ground returns. Table 19.1 lists the functions assigned to each of these signals as implemented in the original IBM PC parallel port and most compatible computers until the IEEE 1284 standard was adopted. In its compatibility mode, the IEEE standard uses essentially these same signal assignments.

Table 19.1. The Original IBM PC Parallel Port Pin-Out

Pin	*Function*
1	Strobe
2	Data bit 0
3	Data bit 1
4	Data bit 2
5	Data bit 3

6	Data bit 4
7	Data bit 5
8	Data bit 6
9	Data bit 7
10	Acknowledge
11	Busy
12	Paper end (Out of paper)
13	Select
14	Auto feed
15	Error
16	Initialize printer
17	Select input
18	Strobe ground
19	Data 1 and 2 ground
20	Data 3 and 4 ground
21	Data 5 and 6 ground
22	Data 7 and 8 ground
23	Busy and Fault ground
24	Paper out, Select, and Acknowledge ground
25	AutoFeed, Select input, and Initialize ground

Under the IEEE 1284 specification, the definition of each signal on each pin is dependent on the operating mode of the port. Only the definitions change; the physical wiring inside your PC and inside cables does not change—if it did, shifting modes would be far from trivial. The altered definitions change the protocol, the signal handshaking that mediates each transfer.

A single physical connector on the back of your PC can operate in any of these five modes, and the signal definitions and their operation will change accordingly. Table 19.2 lists these five modes and their signal assignments.

Table 19.2. IEEE 1284-A Connector Signal Assignments in All Modes

Pin	*Compatibility mode*	*Nibble mode*	*Byte mode*	*EPP mode*	*ECP mode*
1	nStrobe	HostClk	HostClk	nWrite	HostClk

2	Data 1	Data 1	Data 1	AD1	Data 1
3	Data 2	Data 2	Data 2	AD2	Data 2
4	Data 3	Data 3	Data 3	AD3	Data 3
5	Data 4	Data 4	Data 4	AD4	Data 4
6	Data 5	Data 5	Data 5	AD5	Data 5
7	Data 6	Data 6	Data 6	AD6	Data 6
8	Data 7	Data 7	Data 7	AD7	Data 7
9	Data 8	Data 8	Data 8	AD8	Data8
10	nAck	PtrClk	PtrClk	Intr	PeriphClk
11	Busy	PtrBusy	PtrBusy	nWait	PeriphAck
12	PError	AckDataReq	AckDataReq	User defined 1	nAckReverse
13	Select	Xflag	Xflag	User defined 3	Xflag
14	nAutoFd	HostBusy	HostBusy	nDStrb	HostAck
15	nFault	nDataAvail	nDataAvail	User defined 2	nPeriphRequest
16	nInit	nInit	nInt	nInt	nReverseRequest
17	nSelectIn	1284 Active	1284 Active	nAStrb	1284 Active
18	Pin 1 (nStrobe) ground return				
19	Pins 2 and 3 (Data 1 and 2) ground return				
20	Pins 4 and 5 (Data 3 and 4) ground return				
21	Pins 6 and 7 (Data 5 and 6) ground return				
22	Pins 8 and 9 (Data 7 and 8) ground return				
23	Pins 11 and 15 ground return				
24	Pins 10, 12, and13 ground return				
25	Pins 14, 16, and 17 ground return				

Along with standardized signal assignments, IEEE 1284 also gives us a standard nomenclature for describing the signals. In Table 19.2 and all following that refer to the standard, signal names prefaced with a lower-case "n" indicate the signal goes negative when active—that is, the absence of a voltage means the signal is present.

Mode changes are negotiated between your PC and the printer or other peripheral connected to the parallel port. Consequently, both ends of the connection switch modes together so the signal assignments remain consistent at both ends of the connection. For example, if you connect an older printer that only understands Compatibility Mode, your PC cannot negotiate any other operating mode with the printer. It will not activate its EPP or ECP mode, so your printer will never get signals it cannot understand. This negotiation of the mode assures backward compatibility among parallel devices.

The B Connector

The parallel input on the back of your printer is the direct heir of the original Centronics design, one that has been in service for more than two decades. Figure 19.2 offers a conceptual view of this connector.

Figure 19.2 The IEEE-1284 B connector, a 36-pin ribbon jack.

At one time this connector was called an "Amphenol" connector after the name of the manufacturer of the original connector used on the first ports, an Amphenol 57-40360. Amphenol used the trade name "Blue Ribbon" for its series of connectors that included this one, hence the ribbon connector name.

Currently this style of connector is available from several makers, each of which uses its own part number. In addition to the Amphenol part, some of these include AMP 555119-1, Molex 71522, and 3M Company 3367-300X and 3448-62. The mating cable plug is available as AMP 554950-1 or Molex 71522.

The individual contacts in the 36-pin receptacle take the form of fingers or ribbons of metal. In two 18-contact rows they line the inside of a rectangular opening that accepts a matching projection on the cable connector. The overall connectors from edge to edge measures about 2.75 inches long and about 0.66 inch wide. The individual contacts are spaced at 0.085 inch center-to-center. On the printer jack as it appears in the illustration, pin one is on the upper right, and contacts are consecutively numbered right to left. Pin 19 appears at the far right on the bottom row, and again the contacts are sequentially numbered right to left. (In wiring this connector, you would work from the rear, and the numbering of the contacts would rise in the more familiar left to right.)

The assignment of signals to the individual pins of this connector has gone through three stages. The first standard was set by Centronics for its printers. In 1981, IBM altered this design somewhat by redefining several of the connections. Finally, in 1994, the IEEE published its standard assignments which, like those of the A-connector, vary with operating mode.

The Centronics design serves as the foundation for all others. It, with variations, was used by printers through those made in the early years of the PC. Table 19.3 shows its signal assignments. This basic

arrangement of signals has been carried through, with modification, to the IEEE 1284 standard. As far as modern printers go, however, this original Centronics design can be considered obsolete. Those printers not following the IEEE standard invariably use the IBM layout.

Table 19.3. Centronics Parallel Port Signal Assignments

Pin	*Function*
1	Strobe
2	Data bit 0
3	Data bit 1
4	Data bit 2
5	Data bit 3
6	Data bit 4
7	Data bit 5
8	Data bit 6
9	Data bit 7
10	Acknowledge
11	Busy
12	Paper end (Out of paper)
13	Select
14	Signal ground
15	External oscillator
16	Signal Ground
17	Chassis ground
18	+5 VDC
19	Strobe ground
20	Data 0 ground
21	Data 1 ground
22	Data 2 ground
23	Data 3 ground
24	Data 4 ground
25	Data 5 ground

26	Data 6 ground
27	Data 7 ground
28	Acknowledge ground
29	Busy ground
30	Input prime ground
31	Input prime
32	Fault
33	Light detect
34	Line count
35	Line count return (isolated from ground)
36	Reserved

The Centronics layout includes some unique signals not found on later designs. The Line Count (pins 34 and 35) connections provide an isolated contact closure each time the printer advances its paper by one line. The Light Detect signal (pin 33) provides an indication whether the lamp inside the printer for detecting the presence of paper is functioning. The External Oscillator signal (pin 15) provides a clock signal to external devices, one generally in the range of 100 KHz to 200 KHz. The Input Prime signal (pin 31) serves the same function as the later Initialize signal. It resets the printer, flushing its internal buffer.

The IBM design eliminates the signals (but essentially only renames Input Prime) and adds two new signals, Auto feed and Select input, discussed in the following "Operation" section. This layout remains current as IEEE 1284 compatibility mode on the 1284-B connector. Its signal assignments are listed in Table 19.4.

Table 19.4. IBM Parallel Printer Port Signal Assignments

Pin	*Function*
1	Strobe
2	Data bit 0
3	Data bit 1
4	Data bit 2
5	Data bit 3
6	Data bit 4
7	Data bit 5

8	Data bit 6
9	Data bit 7
10	Acknowledge
11	Busy
12	Paper end (Out of paper)
13	Select
14	Auto feed
15	No connection
16	Ground
17	No connection
18	No connection
19	Strobe ground
20	Data 0 ground
21	Data 1 ground
22	Data 2 ground
23	Data 3 ground
24	Data 4 ground
25	Data 5 ground
26	Data 6 ground
27	Data 7 ground
28	Paper end, Select, and Acknowledge ground
29	Busy and Fault ground
30	Auto feed, Select in, and Initialize ground
31	Initialize printer
32	Error
33	No connection
34	No connection
35	No connection
36	Select input

As with the A-connector, the IEEE 1284 signal definitions on the B-connector change with the operating mode of the parallel port. The signal assignments for each of the five IEEE operating modes

are listed in Table 19.5.

Table 19.5. IEEE 1284-B Connector Signal Assignments in All Modes

Pin	*Compatibility mode*	*Nibble mode*	*Byte mode*	*EPP mode*	*ECP mode*
1	nStrobe	HostClk	HostClk	nWrite	HostClk
2	Data 1	Data 1	Data 1	AD1	Data 1
3	Data 2	Data 2	Data 2	AD2	Data 2
4	Data 3	Data 3	Data 3	AD3	Data 3
5	Data 4	Data 4	Data 4	AD4	Data 4
6	Data 5	Data 5	Data 5	AD5	Data 5
7	Data 6	Data 6	Data 6	AD6	Data 6
8	Data 7	Data 7	Data 7	AD7	Data 7
9	Data 8	Data 8	Data 8	AD8	Data8
10	nAck	PtrClk	PtrClk	Intr	PeriphClk
11	Busy	PtrBusy	PtrBusy	nWait	PeriphAck
12	PError	AckDataReq	AckDataReq	User defined 1	nAckReverse
13	Select	Xflag	Xflag	User defined 3	Xflag
14	nAutoFd	HostBusy	HostBusy	nDStrb	HostAck
15	*Not defined*				
16	Logic ground				
17	Chassis ground				
18	Peripheral logic high				
19	Ground return for pin 1 (nStrobe)				
20	Ground return for pin 2 (Data 1)				
21	Ground return for pin 3 (Data 2)				
22	Ground return for pin 4 (Data 3)				
23	Ground return for pin 5 (Data 4)				

24	Ground return for pin 6 (Data 5)				
25	Ground return for pin 7 (Data 6)				
26	Ground return for pin 8 (Data 7)				
27	Ground return for pin 9 (Data 8)				
28	Ground return for pins 10, 12, and13 (nAck, PError, and Select)				
29	Ground return for pins 11 and 32 (Busy and nFault)				
30	Ground return for pins 14, 31, and 36 (nAutoFd, nSelectIn, and nInit))				
31	nInit	nInit	nInit	nInit	nReverseRequest
32	nFault	nDataAvail	nDataAvail	User Defined 2	nPeriphRequest
33	*Not defined*				
34	*Not defined*				
35	*Not defined*				
36	nSelectIn	1284 Active	1284 Active	nAStrb	1284 Active

Again, the port modes and the associated signal assignments are not fixed in hardware but change dynamically as your PC uses the connection. Although your PC acts as host and decides which mode to use, it can only negotiate those that your printer or other parallel device understands. Your printer (or whatever) determines which of these five modes could be used while your PC and its applications picks which of the available modes to use for transferring data.

The C Connector

Given a chance to start over with a clean slate and no installed base, engineers would hardly come up with the confusion of two different connectors with an assortment of different, sometimes compatible operate modes. The IEEE saw the creation of the 1284 standard as such an opportunity, one which they were happy to exploit. To eliminate the confusion of two connectors and the intrinsic need for adapters to move between them, they took the logical step: they created a third connector, IEEE 1284-

C.

All devices compliant with IEEE 1284 Level 2 must use this connector. That requirement is the IEEE's way of saying, "Let's get rid of all these old, confusing parallel ports with their strange timings and limited speed and get on with something new for the next generation." Once the entire world moves to IEEE 1284 Level 2, you'll have no need of compatibility, cable adapters, and other such nonsense. In the meantime, as manufacturers gradually adopt the C-connector for their products, you'll still need adapters but in even greater variety.

All that said, the C-connector has much to recommend it. It easily solves the original IBM problem of no space. Although it retains all the signals of the B-connector, the C-connector is miniaturized, about half the size of the B-connector. As a PC-mounted receptacle, it measures about 1.75 inches long by .375 inch wide. It is shown in a conceptual view in Figure 19.3.

Figure 19.3 Conceptual view of the 1284-C parallel port connector.

The actual contact area of the C-connector is much like that of the B-connector with contact fingers arranged inside a rectangular opening that accepts a matching projection on the mating plug. The spacing between the individual contacts is reduced, however, to 0.05 inch, center-to-center. This measurement corresponds to those commonly used on modern printed circuit boards.

The C-connector provides a positive latch using clips that are part of the shell of the plug. The clips engage latches on either side of the contact area, as shown in the figure. Squeezing the side of the plug spreads the clips and releases the latch.

The female receptacle (as shown) is available from a number of manufacturers. Some of these include AMP 2-175925-5, Harting 60-11-036-512, Molex 52311-3611, and 3M 10236-52A2VC. Part numbers of the mating plug include AMP 2-175677-5, Harting 60-13-036-5200, Molex 52316-3611, and 3M 10136-6000EC.

Every signal on the C-connector gets its own pin and all pins are defined. As with the other connectors, the signal assignments depend on the mode in which the IEEE 1284 port is operating. Table 19.6 lists the signal assignments for the 1284-C connector in each of the five available modes.

Table 19.6. IEEE 1284-C Connector Signal Assignments in All Modes

Pin	*Compatibility mode*	*Nibble mode*	*Byte mode*	*EPP mode*	*ECP mode*
1	Busy	PtrBusy	PtrBusy	nWait	PeriphAck
2	Select	Xflag	Xflag	User defined 3	Xflag
3	nAck	PtrClk	PtrClk	Intr	PeriphClk
4	nFault	nDataAvail	nDataAvail	User Defined 2	nPeriphRequest

5	PError	AckDataReq	AckDataReq	User defined 1	nAckReverse
6	Data 1	Data 1	Data 1	AD1	Data 1
7	Data 2	Data 2	Data 2	AD2	Data 2
8	Data 3	Data 3	Data 3	AD3	Data 3
9	Data 4	Data 4	Data 4	AD4	Data 4
10	Data 5	Data 5	Data 5	AD5	Data 5
11	Data 6	Data 6	Data 6	AD6	Data 6
12	Data 7	Data 7	Data 7	AD7	Data 7
13	Data 8	Data 8	Data 8	AD8	Data8
14	nInit	nInit	nInit	nInit	nReverseRequest
15	nStrobe	HostClk	HostClk	nWrite	HostClk
16	nSelectIn	1284 Active	1284 Active	nAStrb	1284 Active
17	nAutoFd	HostBusy	HostBusy	nDStrb	HostAck
18	Host logic high				
19	Ground return for pin 1 (Busy)				
20	Ground return for pin 2 (Select)				
21	Ground return for pin 3 (nAck)				
22	Ground return for pin 4 (nFault)				
23	Ground return for pin 5 (PError)				
24	Ground return for pin 6 (Data 1)				
25	Ground return for pin 7 (Data 2)				
26	Ground return for pin 8 (Data 3)				
27	Ground return for pin 9 (Data 4)				
28	Ground return for pin 10 (Data 5)				
29	Ground return for pin 11 (Data 6)				

30	Ground return for pin 12 (Data 7)				
31	Ground return for pin 13 (Data 8)				
32	Ground return for pin 14 (nInit)				
33	Ground return for pin 15 (nStrobe)				
34	Ground return for pin 16 (nSelectIn)				
35	Ground return for pin 17 (nAutoFd)				
36	Peripheral logic high				

Adapters

The standard printer cable for PCs is an adapter cable. It rearranges the signals of the A-connector to the scheme of the B-connector. Ever since the introduction of the first PC, you needed this sort of cable just to make your printer work. Over the years they have become plentiful and cheap.

Unfortunately, as cables get cheaper and sources become more generic and obscure, quality is apt to slip. Printer cables provide an excellent opportunity for allowing quality to take a big slide. If you group all the grounds together as a single common line, you're left with only 18 distinct signals on a printer cable. In that some of the grounds are naturally group together, this approach might seem feasible, particularly since you can save the price of a 25-conductor cable. In fact, IBM took this approach with its first printer cable. Low cost printer cables still retain this design. The wiring of this adapter cable is given in Table 19.7.

Table 19.7. Printer Cable, 18-Wire Implementation

PC end 25-pin connector	*Function*	*Printer end 36-pin connector*
1	Strobe	1
2	Data bit 0	2
3	Data bit 1	3
4	Data bit 2	4
5	Data bit 3	5

6	Data bit 4	6
7	Data bit 5	7
8	Data bit 6	8
9	Data bit 7	9
10	Acknowledge	10
11	Busy	11
12	Paper end (Out of paper)	12
13	Select	13
14	Auto feed	14
15	Error	32
16	Initialize printer	31
17	Select input	36
18	Ground	19-30,33
19	Ground	19-30,33
20	Ground	19-30,33
21	Ground	19-30,33
22	Ground	19-30,33
23	Ground	19-30,33
24	Ground	19-30,33
25	Ground	19-30,33

Some PC and printer manufacturers did not exploit all the control signals that were part of the basic parallel port design. In fact, many early printers would not function properly if they received these control signals. Many of these printers (and some early PCs) required proprietary adapter cables to make them work.

A modern printer cable contains a full 25 connections with the ground signals divided among separate pins. For example OS/2, unlike DOS, requires the use of all 25 pins in the IBM parallel printer connection. A generic printer cable which makes only 18 connections may not work with OS/2. If your printer doesn't work properly with OS/2 and does with DOS, the cable is the first place to suspect a problem.

Using all 25 wires is the preferred and correct wiring for a classic parallel printer adapter cable. If you buy or make a cable and plan to use it with classic parallel connections, it should connect all 25 leads at both ends. The IEEE recognizes this cable layout to adapt 1284-A to 1284-B connectors. Table 19.8 shows the wiring of this adapter. (The different nomenclature given for names of signal functions

reflects the official IEEE usage. We've modified a few of the official IEEE signal designations for clarity, particularly those of the ground return lines.)

Table 19.8. 25-Wire Parallel Printer Adapter (IEEE 1284-A to 1284-B)

Host end A connector	*Function*	*Peripheral end* B connector
1	nStrobe	1
2	Data bit 1	2
3	Data bit 2	3
4	Data bit 3	4
5	Data bit 4	5
6	Data bit 5	6
7	Data bit 6	7
8	Data bit 7	8
9	Data bit 8	9
10	nAck	10
11	Busy	11
12	PError	12
13	Select	13
14	nAutoFd	14
15	nFault	32
16	nInit	31
17	nSelectIn	36
18	Pin 1 (nStrobe) ground return	19
19	Pins 2 and 3 (Data 1 and 2) ground return	20 and 21
20	Pins 4 and 5 (Data 3 and 4) ground return	22 and 23
21	Pins 6 and 7 (Data 5 and 6) ground return	24 and 25
22	Pins 8 and 9 (Data 7 and 8) ground return	26 and 27
23	Pins 11 and 15 ground return	29
24	Pins 10, 12, and 13 ground return	28
25	Pins 14, 16, and 17 ground return	30

As new peripherals with the 1284-C connector become available, you'll need to plug them into your

PC. To attach your existing PC to a printer or other device using the C-connector, you'll need an adapter cable to convert the A-connector layout to the C-connector design. Table 19.9 shows the proper wiring for such an adapter as adopted in the IEEE 1284 specification (again with a modification in signal nomenclature from the official standard for clarity).

Table 19.9. Parallel Interface Adapter, 1284-A to 1284-C Connectors

Host end A connector	*Function*	*Peripheral end C connector*
1	nStrobe	15
2	Data bit 1	6
3	Data bit 2	7
4	Data bit 3	8
5	Data bit 4	9
6	Data bit 5	10
7	Data bit 6	11
8	Data bit 7	12
9	Data bit 8	13
10	nAck	3
11	Busy	1
12	PError	5
13	Select	2
14	nAutoFd	17
15	nFault	4
16	nInit	14
17	nSelectIn	16
18	Pin 1 (nStrobe) ground return	33
19	Pins 2 and 3 (Data 1 and 2) ground return	24 and 25
20	Pins 4 and 5 (Data 3 and 4) ground return	26 and 27
21	Pins 6 and 7 (Data 5 and 6) ground return	28 and 29
22	Pins 8 and 9 (Data 7 and 8) ground return	30 and 31
23	Pins 11 and 15 ground return	19 and 22
24	Pins 10, 12, and 13 ground return	20, 21, and 23
25	Pins 14, 16, and 17 ground return	32, 34, and 35

If your next PC or parallel adapter uses the C-connector and you plan to stick with your old printer, you'll need another variety of adapter, one that translates the C-connector layout to that of the B-connector. Table 19.10 lists the wiring required in such an adapter.

Table 19.10. Parallel Interface Adapter, 1284-C to 1284-B

Host end C connector	*Function*	*Peripheral end B connector*
1	Busy	11
2	Select	13
3	nAck	10
4	nFault	32
5	PError	12
6	Data 1	2
7	Data 2	3
8	Data 3	4
9	Data 4	5
10	Data 5	6
11	Data 6	7
12	Data 7	8
13	Data 8	9
14	nInit	31
15	nStrobe	1
16	nSelectIn	36
17	nAutoFd	14
18	Host logic high	No connection
19	Ground return for pin 1 (Busy)	29
20	Ground return for pin 2 (Select)	28
21	Ground return for pin 3 (nAck)	28
22	Ground return for pin 4 (nFault)	29
23	Ground return for pin 5 (PError)	28
24	Ground return for pin 6 (Data 1)	20
25	Ground return for pin 7 (Data 2)	21

26	Ground return for pin 8 (Data 3)	22
27	Ground return for pin 9 (Data 4)	23
28	Ground return for pin 10 (Data 5)	24
29	Ground return for pin 11 (Data 6)	25
30	Ground return for pin 12 (Data 7)	26
31	Ground return for pin 13 (Data 8)	27
32	Ground return for pin 14 (nInit)	30
33	Ground return for pin 15 (nStrobe)	19
34	Ground return for pin 16 (nSelectIn)	30
35	Ground return for pin 17 (nAutoFd)	30
36	Peripheral logic high	18
The following pins on the 1284-B connector are not connected: 15, 16, 17, 33, 34, and 35. Connector shields are connected at each end.		

Note that although both the B- and C-connectors have 36-pins, they do not have the same signals. Several signals share ground connections on the B-connector while several other pins are not connected.

Cable

The nature of the signals in the parallel port are their own worst enemy. They interact with themselves and the other wires in the cable to the detriment of all. The sharp transitions of the digital signals blur. The farther the signal travels in the cable, the greater the degradation that overcomes it. For this reason, the maximum recommended length of a printer cable was ten feet. Not that longer cables will inevitably fail—practical experience often proves otherwise—but some cables in some circumstances become unreliable when stretched for longer distances.

The lack of a true signaling standard before IEEE 1284 made matters worse. Manufacturers had no guidelines for delays or transition times, so these values varied among PC, printer, and peripheral manufacturers. Although the signals might be close enough matches to work through a short cable, adding more wire could push them beyond the edge. A printer might then misread the signals from a PC, printing the wrong character or nothing at all.

Traditional printer cables are notoriously variable. As noted in the discussion of adapters, manufacturers scrimp where they can to produce low cost adapter cables. After all, cables are commodities and the market is highly competitive. When you pay under $10 for a printer cable that comes without a brand name, you can never be sure of its electrical quality.

For this reason, extension cables are never recommended for locating your printer more than ten feet from your PC. Longer distances require alternate strategies—opting for another connection (serial or network) or getting a printer extension system that alters the signals and provides a controlled cable environment.

What length you can get away with depends on the cable, your printer, and your PC. Computers and printers vary in their sensitivity to parallel port anomalies like noise, crosstalk, and digital blurring. Some combinations of PCs and printers will work with lengthy parallel connections, up to fifty feet long. Other match-ups may balk when you stretch the connection more than the recommended ten feet.

The high speed modes of modern parallel ports make them even more finicky. When your parallel port operates in EPP or ECP modes, cable quality becomes critical even for short runs. Signaling speed across one of these interfaces can be in the megahertz range. The frequencies far exceed the reliable limits of even short runs of the dubious low cost printer cables. Consequently, the IEEE 1284 specification precisely details a special cable for high speed operation. Figure 19.4 offers a conceptual view of the construction of this special parallel data cable.

Figure 19.4 IEEE 1284 cable construction details.

Unlike standard parallel wiring, the data lines in IEEE 1284 cables must be double shielded to prevent interference from affecting the signals. Each signal wire must be twisted with its ground return. Even though the various standard connectors do not provide separate pins for each of these grounds, the ground wires must be present and run the full length of the cable.

The difference between old-fashioned "printer" cables and those that conform to the IEEE 1284 standard is substantial. Although you can plug in a printer with either a printer- or IEEE 1284-compliant cable, devices that exploit the high speed potentials of the EPP or ECP designs may not operate properly with a non-compliant cable. Often even when a printer fails to operate properly, the cable may be at fault. Substituting a truly IEEE 1284-compliant cable will bring reluctant connections to life.

Electrical Operation

In each of its five modes, the IEEE 1284 parallel port operates as if it were some kind of completely different electronic creation. When in compatibility mode, the IEEE 1284 port closely parallels the operation of the plain vanilla printer port of bygone days. It allows data to travel in one direction only, from PC to printer. Nibble mode gives your printer (or more likely, another peripheral) a voice, and

allows it to talk back to your PC. In nibble mode, data can move in either of two directions, although asymmetrically. Information flows faster to your printer than it does on the return trip. Byte mode makes the journey fully symmetrical.

With the shift to EPP mode, the parallel port becomes a true expansion bus. A new way of linking to your PC's bus gives it increased bi-directional speed. Many systems can run their parallel ports ten times faster in EPP mode than in compatibility, nibble, or byte modes. ECP mode takes the final step, giving control in addition to speed. ECP can do just about anything any other expansion interface (including SCSI) can do.

Because of these significant differences, the best way to get to know the parallel port is by considering each separately as if it were an interface unto itself. Our examination will follow from simple to complex, which also mirrors the history of the parallel port.

Note that IEEE 1284 deals only with the signals traveling through the connections of the parallel interface. It establishes the relationship between signals and their timing. It concerns itself neither with the data that is actually transferred, command protocols encoded in the data, nor with the control system that produces the signals. In other words, IEEE 1284 provides an environment under which other standards such as EPP and ECP operate. That is, ECP and EPP modes are not the ECP and EPP standards, although those modes are meant to be used by the parallel ports operating under respective standards.

Compatibility Mode

The least common denominator among parallel ports is the classic design that IBM introduced with the first PC. It was conceived strictly as an interface for the one way transfer of information. Your PC sends data to your printer and expects nothing in return. After all, a printer neither stores information nor creates it on its own.

In conception, this port is like a conveyor that unloads ore from a bulk freighter or rolls coal out of a mine. The raw material travels in one direction. The conveyor mindlessly pushes out stuff and more stuff, perhaps creating a dangerously precarious pile, until its operator wakes up and switches it off before the pile gets much higher than his waist.

If your printer had unlimited speed or an unlimited internal buffer, such a one way design would work. But like the coal yard, your printer has a limited capacity and may not be able to cart off data as fast as the interface shoves it out. The printer needs some way of sending a signal to your PC to warn about a potential data overflow. In electronic terms, the interface needs feedback of some kind—it needs to get information from the printer that your PC can use to control the data flow.

To provide the necessary feedback for controlling the data flow, the original Centronics port design and IBM's adaptation of it both included several control signals. These were designed to allow your

PC to monitor how things are going with your printer—whether data is piling up, whether it has sufficient paper or ribbon, whether the printer is even turned on. Your PC can use this information to moderate the outflowing gush of data or to post a message warning you that something is wrong with your printer. In addition, the original parallel port included control signals sent from your PC to the printer to tell it *when* the PC wants to transfer data and to tell the printer to reset itself. The IEEE 1284 standard carries all of these functions into compatibility mode.

Strictly speaking, then, even this basic parallel port is not truly a one way connection, although its feedback provisions were designed strictly for monitoring rather than data flow. For the first half of its life, the parallel port kept to this design. Until the adoption of IEEE-1284, this was the design you could expect for the port on your printer and, almost as likely, those on your PC.

Each signal flowing through the parallel port in compatibility mode has its own function. These signals include the following.

Data Lines

The eight *data lines* of the parallel interface convey data in all operating modes. In compatibility mode, they carry data from the host to the peripheral on connector pins 2 through 9. The higher numbered pins are the more significant to the digital code. To send data to the peripheral, the host puts a pattern of digital voltages on the data lines.

Strobe Line

The presence of signals on the data lines does not, in itself, move information from host to peripheral. As your PC gets its act together, it may change the pattern of data bits. No hardware can assure that all eight will always pop to the correct values simultaneously. Moreover, without further instruction your printer has no way knowing whether the data lines represent a single character or multiple repetitions of the same character.

To assure reliable communications, the system requires a means of telling the peripheral that the pattern on the data lines represents valid information to be transferred. The *strobe line* does exactly that. Your PC pulses the strobe line to tell your printer that the bit pattern on the data lines is a single valid character that the printer should read and accept. The strobe line gives its pulse only after the signals on the data lines have settled down. Most parallel ports delay the strobe signal by about half a microsecond to assure that the data signals have settled. The strobe itself lasts for at least half a microsecond so that your printer can recognize it. (The strobe signal can last up to 500 microseconds.) The signals on the data lines must maintain a constant value during this period and slightly afterward so that your printer has a chance to read them.

The strobe signal is negative going. That is, a positive voltage (+5VDC) stays on the strobe line until your printer wants to send the actual strobe signal. Your PC then drops the positive voltage to near zero for the duration of the strobe pulse. The IEEE 1284 specification calls this signal *nStrobe*.

Busy Line

Sending data to your printer is thus a continuous cycle of setting up the data lines, sending the strobe signal, and putting new values on the data lines. The parallel port design typically requires about two microseconds for each turn of this cycle, allowing a perfect parallel port to dump out nearly half a million characters a second into your hapless printer. (As we will see, the actual maximum throughput of a parallel port is much lower than this.)

For some printers, coping with that data rate is about as daunting as trying to catch machine gun fire with your bare hands. Before your printer can accept a second character, its circuitry must do something with the one it has just received. Typically it will need to move the character into the printer's internal buffer. Although the character moves at electronic speeds, it does not travel instantaneously. Your printer needs to be able to tell your PC to wait for the processing of the current character before sending the next.

The parallel port's *busy line* gives your printer the needed breathing room. Your printer switches on the busy signal as soon as it detects the strobe signal and keeps the signal active until it is ready to accept the next character. The busy signal can last for a fraction of a second (even as short as a microsecond) or your printer could hold it on indefinitely while it waits for you to correct some error. No matter how long the busy signal is on, it keeps your PC from sending out more data through the parallel port. It functions as the basic flow control system.

Acknowledge Line

The final part of the flow control system of the parallel port is the *acknowledge line*. It tells your PC that everything has gone well with the printing of a character, or its transfer to the internal buffer. In effect, it is the opposite of the busy signal, telling your PC that the printer is ready rather than unready. Where the busy line says "Whoa!" the acknowledge line says "Giddyap!" The acknowledge signal is the opposite in another way; it is negative going where busy is positive going. The IEEE 1284 specification calls this signal *nAck*.

When your printer sends out the acknowledge signal, it completes the cycle of sending a character. Typically the acknowledge signal on a conventional parallel port lasts about eight microseconds, stretching a single character cycle across the port to ten microseconds. (IEEE 1284 specifies the length of nAck to be between 0.5 and 10 microseconds.) If you assume the typical length of this signal for a

conventional parallel port, the maximum speed of the port works out to about 100,000 characters per second.

Select

In addition to transferring data to the printer, the basic parallel port allows your printer to send signals back to your PC so your computer can monitor the operation of the printer. The original IBM design of the parallel interface includes three such signals that tell your PC when your printer is ready, willing, and able to do its job. In effect, these signals give your PC the ability to remote sense the condition of your printer.

The most essential of these signals is *select*. The presence of this signal on the parallel interface tells your PC that your printer is online. That is, that your printer is switched on and is in its online mode, ready to receive data from your PC. In effect, it is a remote indicator for the online light on your printer's control panel. If this signal is not present, your PC assumes that nothing is connected to your parallel port and doesn't bother with the rest of its signal repertory.

Because the rest state of a parallel port line is an absence of voltage (which would be the case if nothing were connected to the port to supply the voltage), the select signal takes the form of a positive signal (nominally +5VDC) that *in compatibility mode* under the IEEE 1284 specification stays active the entire period your printer is online.

Paper Empty

To print anything your printer needs paper, and the most common problem that prevents your printer from doing its job is running out of paper. The *paper empty* signal warns your PC when your printer runs out. The IEEE 1284 specification calls this signal *PError* for "paper error," although it serves exactly the same function.

Paper empty is an information signal. It is not required for flow control because the busy signal more than suffices for that purpose. Most printers will assert their busy signals for the duration of the period they are without paper. Paper empty tells your PC about the specific reason that your printer has stopped data flow. This signal allows your operating system or application to flash a message on your monitor to warn you to load more paper.

Fault

The third printer-to-PC status signal is *fault*, a catch-all for warning of any other problems that your printer may develop—out of ink, paper jams, overheating, conflagrations, and other disasters. In operation, fault is actually a steady state positive signal. It dips low (or off) to indicate a problem. At the same time, your printer may issue its other signals to halt the data flow including busy and select. It never hurts to be extra sure. Because this signal is negative going, the IEEE specification calls it *nFault*.

Initialize Printer

In addition to the three signals your printer uses to warn of its condition, the basic parallel port provides three control signals that your PC can use to command your printer without adding anything to the data stream. Each of these three provides its own hard-wired connection for a specific purpose. These include one to initialize the printer, another to switch it to online condition if the printer allows a remote control status change, and a final signal to tell the printer to feed the paper up one line.

The *initialize printer* signal helps your computer and printer keep in sync. Your PC can send a raft of different commands to your printer to change its mode of operation, change font, alter printing pitch, and so on. Each of your applications that share your printer might send out its own favored set of commands. And many applications are like sloppy in-laws that comes for a visit and fail to clean up after themselves. The programs may leave your printer in some strange condition, such as set to print underscored boldface characters in agate size type with a script typeface. The next program you run might assume some other condition and blithely print out paychecks in illegible characters.

Initialize printer tells your printer to step back to ground zero. Just as your PC boots up fresh and predictably, so does your printer. When your PC sends your printer the initialize printer command, it tells the printer to boot up, that is, reset itself and load its default operating parameters with its start up configuration of fonts, pitches, typefaces, and the like. The command has the same effect as you switching off the printer and turning it back on and simply substitutes for adding a remote control arm on your PC to duplicate your actions.

During normal operation, your PC puts a constant voltage on the initialize printer line. Removing the voltage tells your printer to reset. The IEEE 1284 specification calls this negative going signal *nInit*.

Select Input

The signal that allows your PC to switch your printer online and offline is called *select input*. The IEEE 1284 specification calls it *nSelectIn*. It is active, forcing your printer online, when it is low or off. Switching it high deselects your printer.

Not all printers obey this command. Some have no provisions for switching themselves on and offline. Others have setup functions (such as a DIP switch) that allow you to defeat the action of this signal.

Auto Feed XT

At the time IBM imposed its print system design on the rest of the world, different printers interpreted the lowly carriage return in one of two ways. Some printers took it literally. Carriage return mean to move the printhead carriage back to its starting position on the left side of the platen. Other printers thought more like typewriters. Moving the printhead full left also indicated the start of a new line, so they obediently advance the paper one line when they got a carriage return command. IBM, being a premiere typewriter maker at the time, opted for this second definition.

To give printer developers flexibility, however, the IBM parallel port design included the *Auto Feed XT* signal to give your PC command of the printer's handling of carriage returns. Under the IEEE 1284 specification, this signal is called *nAutoFd*. By holding this signal low or off, your PC commands your printer to act in the IBM and typewriter manner, adding a line feed to every carriage return. Making this signal high tells your printer to interpret carriage returns literally and only move the printhead. Despite the availability of this signal, most early PC printers ignored it and did whatever their setup configuration told them to do with carriage returns.

Nibble Mode

Early parallel ports used uni-directional circuitry for their data lines. No one foresaw the need for your PC to acquire data from your printer, so there was no need to add the expense or complication of bi-directional buffers to the simple parallel port. This tradition of single-direction design and operation continues to this day in the least expensive (which, of course, also means "cheapest") parallel ports.

Every parallel port does, however, have five signals that are meant to travel from the printer to your PC. These include (as designated by the IEEE 1284 specification) nAck, Busy, PError, Select, and nFault. If you could suspend the normal operation of these signals temporarily, you could use four of them to carry data back from the printer to your PC. Of course, the information would flow at half speed, four bits at a time.

This means of moving data is the basis of *nibble mode*, so called because the PC community calls half a byte (or those four bits) a nibble. Using nibble mode, any parallel port can operate bi-directionally—full speed forward but half speed in reverse.

Nibble mode requires that your PC take explicit command and control the operation of your parallel port. The port itself merely monitors all of its data and monitoring signals and relays the data to your

PC. Your PC determines whether to regard your printer's status signals as backward-moving data. Of course this system also requires that the device at the other end of the parallel port—your printer or whatever—know that it has switched into nibble mode and understand what signals to put where and when. The IEEE 1284 specification defines a protocol for switching into nibble mode and how PC and peripherals handle the nibble-mode signals.

The process is complex, involving several steps. First your PC must identify whether the peripheral connected to it recognizes the IEEE standard. If not, all bets are off for using the standard. Products created before IEEE 1284 was adopted relied on the software driver controlling the parallel port to be matched to your parallel port peripheral. Because the two were already matched, they knew everything they needed to know about each other without negotiation. The pair could work without understanding the negotiation process or even the IEEE 1284 specification. Using the specification, however, allows your PC and peripherals to do the matching without your intervention.

Once your PC and peripheral decide they can use nibble mode, your PC signals to the peripheral to switch to the mode. Before the IEEE 1284 standard, the protocol was proprietary to the parallel port peripheral. The standard gives all devices a common means of controlling the switchover.

After both your PC and parallel port peripheral have switched to nibble mode, the signals on the interface get new definitions. In addition, nibble mode itself operates in two modes or phases, and the signals on the various parallel port lines behave differently in each mode. These modes include reverse idle phase and reverse data transfer phase.

In *reverse idle phase,* the PtrClk signal (nAck in compatibility mode) operates as an attention signal from the parallel port peripheral. Activating this signal tells the parallel port to issue an interrupt inside your PC, signaling that the peripheral has data available to be transferred. Your PC acknowledges the need for data and requests its transfer by switching the HostBusy signal (nAutoFd in compatibility mode) low or off. This switches the system to *reverse data transfer phase*. Your PC switches the HostBusy signal high again after the completion of the transfer of a full data byte. When the peripheral has mode data ready and your PC switches Host Busy back low again, another transfer begins. If it switches low without the peripheral having data available to send, the transition re-engaged reverse idle phase.

During reverse data transfer phase, information is coded across two transfers as listed in Table 19.11. In effect, each transfer cycle involves to epi-cycles which move one nibble. First your peripheral transfers the four bits of lesser significance, then the bits of more significance.

Table 19.11. Data Bit Definitions in Nibble Mode

Signal	*First epi-cycle contents*	*Second epi-cycle contents*
nFault	Least significant bit	Data bit 5
Xflag	Data bit 2	Data bit 6
AckDataReq	Data bit 3	Data bit 7

PtrBusy	Data bit 4	Most significant bit

Because moving a byte from peripheral to PC requires two nibble transfers, each of which requires the same time as one byte transfer from PC to peripheral, reverse transfers in nibble mode operate at half speed at best. The only advantage of nibble mode is its universal compatibility. Even before the IEEE 1284 specification, it allowed any parallel port to operate bi-directionally. Because of this speed penalty alone, if you have a peripheral and parallel port that lets you choose the operating mode for bi-directional transfers, nibble mode is your *least* attractive choice.

Byte Mode

Unlike nibble mode, byte mode requires special hardware. The basic design for byte mode circuitry was laid down when IBM developed its PS/2 line of computers and developed the Data Migration Facility. By incorporating bi-directional buffers in all eight of the data lines of the parallel port, IBM enabled them to both send and receive information on each end of the connection. Other than that change, the new design involved no other modifications to signals, connector pin assignments, or the overall operation of the port. Before the advent of the IEEE standard, these ports were known as PS/2 parallel ports or bi-directional parallel ports.

IEEE 1284 does more than put an official industry imprimatur on the IBM design, however. The standard redefines the bi-directional signals and adds a universal protocol of negotiating bi-directional transfers.

As with nibble mode, a peripheral in byte mode uses the PtrClk signal to trigger an interrupt in the host PC to advise that the peripheral has data available for transfer. When the PC services the interrupt, it checks the port nDataAvail signal, a negative going signal which indicates a byte is available for transfer when it goes low. The PC can then pulse off the HostBusy signal to trigger the transfer using the HostClk (nStrobe) signal to read the data. The PC raises the HostBusy signal again to indicate the successful transfer of the data byte. The cycle can then repeat for as many bytes as need to be sent.

Because byte mode is fully symmetrical, transfers occur at the same speed in either direction. The speed limit is set at the performance of the port hardware, the speed at which the host PC handles the port overhead, and by the length of timing cycles set in the IEEE 1284 specification. Potentially the design could requires as little as four microseconds for each byte transferred, but real world systems peak at about the same rate as conventional parallel ports, 100,000 bytes per second.

Enhanced Parallel Port Mode

When it was introduced, the chief innovation of the Enhanced Parallel Port was its improved performance, thanks to a design that hastened the speed at which your PC could pack data into the port. The EPP design altered port hardware so that instead of using byte-wide registers to send data through the port, your PC could dump a full 32-bit word of data directly from its bus into the port. The port would then handle all the conversion necessary to repackage the data into four byte-wide transfers. The reduction in PC overhead and more efficient hardware design enabled a performance improvement by a factor of ten in practical systems. This speed increase required more stringent specifications for printer cables. The IEEE 1284 specification does not get into the nitty-gritty of linking the parallel port circuitry to your PC, so it does not guarantee that a port in EPP mode will deliver all of this speed boost. Moreover, the IEEE 1284 cable specs are not as demanding as the earlier EPP specs.

EPP mode of the IEEE 1284 specification uses only six signals in addition to the eight data lines for controlling data transfers. Three more connections in the interface are reserved for use by individual manufacturers and are not defined under the standard.

A given cycle across the EPP mode interface performs one of four operations: writing an address, reading an address, writing data, or reading data. The address corresponds to a register on the peripheral. The data operations are targeted on that address. Multiple data bytes may follow a single address signal as a form of burst mode.

nWrite

Data can travel both ways through an EPP connection. The *nWrite* signal tells whether the contents of the data lines are being sent from your PC to a peripheral or from a peripheral to your PC. When the nWrite signal is set low, it indicates data is bound for the peripheral. When set high, it indicates data sent from the peripheral.

nDStrobe

Sound boards are heavy feeders when it comes to system resources. A single sound board may require multiple interrupts, a wide range of input/output ports, and a dedicated address range in High DOS memory. Because of these extensive resource demands, the need for numerous drivers, and often poor documentation, sound boards are the most frustrating expansion products to add to a PC. In fact, a sound board may be the perfect gift to surreptitiously gain revenge letting you bury the hatchet with an estranged friend without the friend knowing you've sliced solidly into his back.

As with other parallel port transfers, your system needs a signal to indicate when the bits on the data lines are valid and accurate. EPP mode uses a negative going signal called *nDStrobe* for this function

in making data operations. Although this signal serves the same function as the strobe signal on a standard parallel port, it has been moved to a different pin, that used by the nAutoFd signal in compatibility mode.

nAStrobe

To identify a valid address on the interface bus, the EPP system uses the nAStrobe signal. This signal uses the same connection as does nSelectIn during compatibility mode.

nWait

To acknowledge that a peripheral has properly received a transfer, it deactivates the negative going *nWait* signal (making it a positive voltage on the bus). By holding the signal positive, the peripheral signals the host PC to wait. Making the signal negative indicates that the peripheral is ready for another transfer.

Intr

To signal the host PC that a peripheral connected to the EPP interface requires immediate service, it sends out the *Intr* signal. The transition between low and high states of this signal indicates a request for an interrupt (that is, the signal is edge-triggered). EPP mode does not allocate a signal to acknowledge that the interrupt request was received.

nInit

The escape hatch for EPP mode is the *nInit* signal. When this signal is activated by making it low, it forces the system out of EPP mode and back to compatibility mode.

Extended Capabilities Port Mode

When operating in ECP mode, the IEEE 1284 port uses seven signals to control the flow of data through the standard eight data lines. ECP mode defines two data transfer signaling protocols—one for forward transfers (from PC to peripheral) and one for reverse transfers (peripheral to PC)—and the transitions between them. Transfers are moderated by closed loop handshaking that guarantees that all bytes get where they are meant to go, even should the connection be temporarily disrupted.

Because all parallel ports start in compatibility mode, your PC and its peripherals must first negotiate with one another to arrange to shift into ECP mode. Your PC and its software initiate the negotiation (as well as managing all aspects of the data transfers). Following a successful negotiation to enter ECP mode, the connection enters its forward idle phase.

HostClk

To transfer information or commands across the interface, your PC starts from the forward idle phase and puts the appropriate signals on the data line. To signal to your printer or other peripheral that the values on the data lines are valid and should be transferred, your PC activates its *HostClk* signal, setting it to a logical high.

PeriphAck

The actual transfer does not take place until your printer or other peripheral acknowledges the HostClk signal by sending back the *PeriphAck* signal, setting it to a logical high. In response, your PC switches the HostClk signal low. Your printer or peripheral then knows it should read the signals on the data lines. Once it finishes reading the data signals, the peripheral switches the PeriphAck signal low. This completes the data transfers. Both HostClk and PeriphAck are back to their forward idle phase norms, ready for another transfer.

nPeriphRequest

When a peripheral needs to transfer information back to the host PC or to another peripheral, it makes a request by driving the *nPeriphRequest* signal low. The request is a suggestion rather than a command because only the host PC can initiate or reverse the flow of data. The nPeriphRequest typically causes an interrupt in the host PC to make this request known.

nReverseRequest

To allow a peripheral to send data back to the host or to another device connected to the interface, the host PC activates the *nReverseRequest* signal by driving it low, essentially switching off the voltage that otherwise appears there. This signals to the peripheral that the host PC will allow the transfer.

nAckReverse

To acknowledge that it has received the nReverseRequest signal and that it is ready for a reverse-direction transfer, the peripheral asserts its *nAckReverse* signal, driving it low. The peripheral can then send information and commands through the eight data lines and the PeriphAck signal.

PeriphClk

To begin a reverse transfer from peripheral to PC, the peripheral first loads the appropriate bits onto the data lines. It then signals to the host PC that it has data ready to transfer by driving the *PeriphClk* signal low.

HostAck

Your PC responds to the PeriphClk signal by switching the *HostAck* signal from its idle logical low to a logical high. The peripheral responds by driving PeriphClk high. When the host accepts the data, it responds by driving the HostAck signal low. This completes the transfer and returns the interface to the reverse idle phase.

Data Lines

Although the parallel interface uses the same eight data lines to transfer information as do other IEEE 1284 port modes, it supplements them with an additional signal to indicate whether the data lines contain data or a command. The signal used to make this nine-bit information system changes with the direction of information transfer. When ECP mode transfers data from PC host to a peripheral (that is, during a forward transfer), it uses the HostAck signal to specific command or data. When a peripheral

originates the data being transferred (a reverse transfer), it uses the PeriphAck signal to specify command or data.

Logical Interface

All parallel ports, no matter the speed, technology, or operating mode, must somehow interface with your PC, its operating system, and your applications. After all, you can't expect to print if you can't find your printer, so you shouldn't expect your programs to do it, either. Where you might need a map to find your printer, particularly when your office makes the aftermath of a rock concert seem organized, your programs need something more in line with their logical nature that serves the same function. You look for a particular address on a street. Software looks for function calls, interrupt routines, or specific hardware parameters.

That list represents the steps that get you closer and closer to the actual interface. A function call is a high level software construct, part of your operating system or a driver used by the operating system or your applications. The function call may in turn ask for an interrupt, which is a program routine that either originates in the firmware of your PC or is added by driver software. Both the function call and interrupt work reach your interface by dipping down to the hardware level and looking for specific features. Most important of these are the input/output ports used by your parallel interface.

Input/Output Ports

The design of the first PC linked the circuitry of the parallel port to the PC's microprocessor through a set of *input/output ports* in your PC. These I/O ports are not ports that access the outside world but rather are a special way a microprocessor has to connect to circuitry. An I/O port works like a memory address—the microprocessor signals address value to the PC's support circuitry, then it sends data to that address. The only difference between addressing memory and I/O ports is that data for the former goes to the RAM in your PC. In the later case, the addressing is in a separate range that's links to other circuitry. In general, the I/O port addresses link to registers, a special kind of memory that serves as a portal for passing logical values between circuits.

The traditional design for a parallel port used three of these I/O ports. The EPP and ECP designs use more. In any case, however, the I/O ports take the form of a sequential block. The entire range of I/O ports used in a parallel connection usually gets identified by the address of first of these I/O ports (which is to say the one with the lowest number or address). This number is termed the *base address* of the parallel port. Every parallel port in a given PC must have a unique base address. Two parallel ports inside a single PC cannot share the same base address, nor can they share any of their other I/O ports. If you accidentally assign two parallel ports the same base address when configuring your PC's hardware, neither will likely work.

The original PC design made provisions for up to three parallel ports in a single system, and this limit has been carried through to all IBM-compatible PCs. Each of these has its own base address. For the original PC, IBM chose three values for these base addresses, and these remain the values used by most hardware makers. These basic base addresses are 03BC(Hex), 0378(Hex), and 0278(Hex).

Manufacturers rarely use the first of these, 03BC(Hex). IBM originally assigned this base address to the parallel port that was part of the long obsolete IBM Monochrome Display Adapter or MDA card. IBM kept using this name in its PS/2 line of computers, assigning it to the one built-in parallel port in those machines. There was no chance of conflict with the MDA card because the MDA cannot be installed inside PS/2s. Other computer makers sometimes use this base address for built-in parallel ports. More often, however, they use the base address of 0378(Hex) for such built-in ports. Some allow you to assign either address—(or even 0278(Hex)—using their setup program, jumpers, or DIP switches.

Device Names

These base address values are normally hidden from your view and your concern. Most programs and operating systems refer to parallel ports with port names. These names take the familiar Line PrinTer form: LPT1, LPT2, and LPT3. In addition, the port with the name LPT1 can also use the alias PRN.

The correspondence between the base address of a parallel port and its device name varies with the number of ports in your PC. There is no direct one to one relationship between them. Your system assigns the device names when it boots up. One routine in your PC's BIOS code searches for parallel ports at each of the three defined base addresses in a fixed order. It always looks first for 03BC(Hex), then 0378(Hex), then 0278 (Hex). The collection of I/O ports at the first base address that's found gets assigned the name LPT1; the second, LPT2; the third, LPT3. The BIOS stores the base address values in a special memory area called the *BIOS data area* at particular absolute addresses. Because I/O port addresses are 16 bits long, each base address is allocated two bytes of storage. The base address of the parallel port assigned the LPT1 is stored at absolute memory location 0000:0408; LPT2, at 0000:040A; LPT3, 0000:040C. This somewhat arcane system assures you that you will always have a device called LPT1 (and PRN) in your PC if you have a parallel interface at all, no matter what set of I/O ports it uses.

Interrupts

The design of the original PC provided two interrupts for use by parallel ports. Hardware interrupt 07(Hex) was reserved for the first parallel port, and hardware interrupt 05(Hex) was reserved for the second.

DOS, Windows, and most applications do not normally use hardware interrupts to control printers. When interrupts run short in your PC and you need to find one for a specific feature, you can often steal one of the interrupts used by a printer port.

The key word in the discussion of parallel port interrupts is *printer*. If you use your parallel port for some other purpose, you may not be able to steal its interrupt. Drivers for EPP and ECP ports may use interrupts, and modems that use parallel ports usually make use of interrupts. If you need an interrupt and you're not sure whether your parallel port needs it, try reassigning it where you need it. Then try to print something while you use the feature that borrowed the interrupt. If there's a problem, you'll know it before you risk your data to it.

Port Drivers

The PC printer port was designed to be controlled by a software driver. Under DOS, you might not notice these drivers because they are part of your PC's ROM BIOS. The printer interrupt handler is actually a printer driver.

In reality, only a rare program uses this BIOS-based driver. It's simply too slow. Because the hardware resources used by the parallel port are well known and readily accessed, most programmers prefer to directly control the parallel port hardware to send data to your printer. Many applications incorporate their own print routines or use printer drivers designed to take this kind of direct control.

More advanced operating systems similarly take direct hardware control of the parallel port through software drivers which take over the functions of the BIOS routines. Windows, up through Version 3.11, automatically used its own integral drivers for your printer ports, although EPP and ECP operation require that you explicitly load drivers to match. More advanced operating systems, including OS/2 and Windows 95, always use external drivers to take control of your PC's ports.

You can check or change the parallel port driver your system uses when you run Windows 95 through the Printer Port properties folder. To access this folder, run Device Manager. From the Start button, select Settings, then Control Panel. Click on the System icon in Control Panel. Select the Device Manager tab. Click on the line for Ports (COM and LPT), then highlight the LPT port for which you want to check the driver. Finally, click on the properties button. You'll see a screen like that shown in Figure 19.5.

Figure 19.5 The Windows 95 parallel port properties folder.

Under the heading Driver files:, you'll see your parallel port drivers listed. You can change the driver used by this port by clicking on the Change Driver button. When you do, you'll see a window like that shown in Figure 19.6.

Figure 19.6 Updating your parallel port driver under Windows 95.

By default, the Models list will include only those drivers that are compatible with the port that the Windows Plug-and-Play system has detected in your PC. The list will change with the Manufacturer you highlight. You can view all the available drivers from a given manufacturer (or the standard drivers) by selecting Show all devices. If the driver you want is not within the current repertory of your Windows 95 system, you can install a driver from a floppy or CD ROM disk (or one that you've copied to your hard disk) by clicking on the Have Disk button, which prompts you for the disk and path name leading to the driver.

To install a new driver, highlight it and click on the OK button. Windows takes care of the rest.

Control

Even in its immense wisdom, a microprocessor can't fathom how to operate a parallel port by itself. It needs someone to tell it how to move the signals around. Moreover, the minutiae of constantly taking care of the details of controlling a port would be a waste of the microprocessor's valuable time. Consequently, system designers created help systems for your PC's big brain. Driver software tells the microprocessor how to control the port. And port hardware handles all the details of port operation.

As parallel ports have evolved, so have these aspects of their control. The software that controls the traditional parallel port that's built into the firmware of your PC has given way to a complex system of drivers. The port hardware, too, has changed to both simplify operation and to speed it up.

These changes don't follow the neat system of modes laid down by IEEE 1284. Instead, they have undergone a period of evolution in reaching their current condition.

Traditional Parallel Ports

In the original PC, each of its parallel ports linked to the PC's microprocessor through three separate I/O ports, each controlling its own register. The address of the first of these registers serves as the base address of the parallel port. The other two addresses are the next higher in sequence. For example, when the first parallel port in a PC had a base address of 0378(hex), the other two I/O ports assigned it had addresses of 0379(Hex) and 037A(Hex).

The register at the base address of the parallel port serves a data latch called the *printer data register* which temporarily holds the values passed along to it by your PC's microprocessor. Each of the eight bits of this port is tied to one of the data lines leading out of the parallel port connector. The correspondence is exact. For example, the most significant bit of the register connects to the most significant bit on the port connector. When your PC's microprocessor writes a value to the base

register of the port, the register latches those values until your microprocessor sends newer values to the port.

Your PC uses the next register on the parallel port, corresponding to the next I/O port, to monitor what the printer is doing. Termed the *printer status register,* the various bits that your microprocessor can read at this I/O port carry messages from the printer back to your PC. The five most significant bits of this register directly correspond to five signals appearing in the parallel cable: bit 7 indicates the condition of the busy signal; bit 6, acknowledge; bit 5, paper empty; bit 4, select; and bit 3, error. The remaining three bits of this register (bits 2, 1, and 0—the least significant bits) served no function in the original PC parallel port.

To send commands to your printer, your PC uses the third I/O port, offset two ports from the base address of the parallel port. The register there, called the *printer control register*, relays commands through its five least significant bits. Of these, four directly control corresponding parallel port lines. Bit 0 commands the strobe line; bit 1 the Auto Feed XT line; bit 2, the initialize line; and Bit 3, the select line.

To enable your printer to send interrupts to command the microprocessor's attention, your PC uses bit 4 of the printer control register. Setting this bit high causes the acknowledge signal from the printer to trigger a printer interrupt. During normal operation your printer, after it receives and processes a character, changes the acknowledge signal from a logical high to a low. Set bit 4, and your system detects the change in the acknowledge line through the printer status register and executes the hardware interrupt assigned to the port. In the normal course of things, this interrupt simply instructs the microprocessor to send another character to the printer.

All of the values sent to the printer data register and the printer control register are put in place by your PC's microprocessor, and the chip must read and react to all the values packed into the printer status register. The printer gets its instructions for what to do from firmware that is part of your system's ROM BIOS. The routines coded for interrupt vector 017(Hex) carry out most of these functions. In the normal course of things, your applications call interrupt 017(Hex) after loading appropriate values into your microprocessors registers, and the microprocessor relays the values to your printer. These operations are very microprocessor intensive. They can occupy a substantial fraction of the power of a microprocessor (particularly that of older, slower chips) during print operations.

Enhanced Parallel Ports

Intel set the pattern for Enhanced Parallel Port by integrating the design into the 386SL chip set (which comprised a microprocessor and a support chip, the 386SL itself and the 82360SL I/O subsystem chip, which together required only memory to make a complete PC). The EPP was conceived as a superset of the standard and PS/2 parallel ports. As with those designs, compatible transfers require the use of the three parallel port registers at consecutive I/O port addresses. However, it adds five new registers to the basic three. Although designers are free to locate these registers wherever they want because

they are accessed using drivers, in the typical implementation, these registers occupy the next five I/O port addresses in sequence.

EPP Address Register

The first new register (offset three from the base I/O port address) is the called the *EPP address register*. It provide a direct channel through which your PC can specify addresses of devices linked through the EPP connection. By loading an address value in this register, your PC could select among multiple devices attached to a single parallel port, at least once parallel devices using EPP addressing become available.

EPP Data Registers

The upper four ports of the EPP system interface (starting at offset four from the base port) link to the *EPP data registers* which provide a 32-bit channel for sending data to the EPP data buffer. The EPP port circuitry takes the data from the buffer, breaks it into four separate bytes, then sends the bytes through the EPP data lines in sequence. Substituting four I/O ports for the one used by standard parallel ports moves the conversion into the port hardware, relieving your system from the responsibility of formatting the data. In addition, your PC can write to the four EPP data registers simultaneously using a single 32-bit double-word in a single clock cycle in computers that have 32-bit data buses. In lesser machines, the EPP specification also allows for byte-wide and word-wide (16-bit) write operations through to the EPP data registers.

Unlike standard parallel ports that require your PC's microprocessor to shepherd data through the port, the Enhanced Parallel Port works automatically. It needs no other signals from your microprocessor after it loads the data in order to carry out a data transfer. The EPP circuitry itself generates the data strobe signal on the bus almost as soon as your microprocessor writes to the EPP data registers. When your microprocessor reads data from the EPP data registers, the port circuitry automatically triggers the data strobe signal to tell whatever device that's sending data to the EPP connection that your PC is ready to receive more data. The EPP port can consequently push data through to the data lines with a minimum of transfer overhead. This streamlined design is one of the major factors that enables the EPP to operate so much faster than standard ports.

Fast Parallel Port Control Register

To switch from standard parallel port to bi-directional to EPP operation requires only plugging values

into one of the registers. Although the manufacturers can use any design they want, needing only to alter their drivers to match, most follow the pattern set in the SL chips. Intel added a software controllable *fast parallel port control register* as part of the chipset. This corresponds to the unused bits of the standard parallel port printer control register.

Setting the most significant bit (bit 7) of the fast parallel port control register high engages EPP operation. Setting this bit low (the default) forces the port into standard mode. Another bit controls bi-directional operation. Setting bit 6 of the fast parallel port control register high engages bi-directional operation. When low, bit 6 keeps the port unidirectional.

In most PCs, an EPP doesn't automatically spring to life. Simply plugging your printer into EPP hardware won't guarantee fast transfers. Enabling the EPP requires a software driver which provides the link between your software and the EPP hardware.

Extended Capabilities Ports

As with other variations on the basic parallel port design, your PC controls an Extended Capabilities Port through a set of registers. To maintain backward compatibility with products requiring access to a standard parallel port, the ECP design starts with the same trio of basic registers. However, it redefines the parallel port data in each of the port's different operating modes.

The ECP design supplements the basic trio of parallel port registers with an additional set of registers offset at port addresses 0400(Hex) higher than the base registers. One of these, the *extended control register*, controls the operating mode of the ECP port. Your microprocessor sets the operating mode by writing to this port, which is located offset by 0402(Hex) from the base register of the port. The ECP port uses additional registers to monitor and control other aspects of the data transfer. Table 19.12 lists the registers used by the ECP, their mnemonics, and the modes in which they function.

Table 19.12. Extended Capabilities Port Register Definitions

Name	*Address*	*Mode*	*Function*
Data	Base	PC, PS/2	Data register
ecpAFifo	Base	ECP	ECP FIFO (Address) buffer
DSR	Base+1	All	Status register
DCR	Base+2	All	Control register
cFifo	Base+400	EPP	Enhanced Parallel Port FIFO (data) buffer
ecpDFifo	Base+400	ECP	ECP FIFO (data) buffer
tFifo	Base+400	Test	Test FIFO

cnfgA	Base+400	Configuration	Configuration register A
cnfgB	Base+401	Configuration	Configuration register B
ecr	Base+402	All	Extended control register

As with other improved parallel port designs, the ECP behaves exactly like a standard parallel port in its default mode. Your programs can write bytes to its data register (located at the port's base address just as with a standard parallel port) to send the bits through the data lines of the parallel connection. Switch to EPP or ECP mode, and your programs can write at high speed to a register as wide as 32 bits. The ECP design allows for transfers 8, 16, or 32 bits wide at the option of the hardware designer.

To allow multiple devices to share a single parallel connection, the ECP design incorporates its own addressing scheme that allows your PC to separately identify and send data to as many as 128 devices. When your PC wants to route a packet or data stream through the parallel connection to a particular peripheral, it sends out a channel address command through the parallel port. The command includes a device address. When an ECP parallel device receives the command, it compares the address to its own assigned address. If the two do not match, the device ignores the data traveling through the parallel connection until your PC sends the next channel address command through the port. When your PC fails to indicate a channel address, the data gets broadcast to all devices linked to the parallel connection.

Performance Issues

As with any interface, you want your parallel connection to operate at the highest possible speed. The speed of a parallel connection can be difficult to pin down. Several variables affect it. For example, the parallel cable itself sets the upper limit on the frequencies of the signals that the port can use, which in turn limits the maximum data rate. At practical cable lengths, which means those less then the recommended 10-foot maximum, cable effects on parallel port throughput are minimal. Other factors that come into play include: the switching speed of the port circuitry itself, the speed at which your PC can write to various control and data registers, the number of steps required by the BIOS or software driver to write a character, the ability of the device at the other end of the connection to accept and process the data sent to it, and the delays necessary in the timing of the various parallel port signals that are necessary to insure the integrity of the transfer.

Timing

The timing of parallel port signals is actually artificially slow to accommodate the widest variety of parallel devices. Because the timing was never standardized before the IEEE 1284 specification, manufacturers had to rely on loose timing—meaning a wider tolerance of errors achieved through a

slower signaling rate—to assure any PC could communicate with any printer or other parallel peripheral.

When system timing of an older standard parallel port is set at the minimum that produces the widest compatibility, the transmission of a single character requires about ten microseconds. That speed yields a peak transfer rate of 100,000 bytes per second. Operated at the tightest timing allowed by the IEEE 1284 specification, a conventional parallel port can complete a single character transfer cycle in four microseconds, yielding a peak throughput of 250,000 bytes per second.

Add in all the overhead at both ends of the connection, and those rates can take a bad tumble. With a fast PC and fast peripheral, you can realistically expect 80 to 90 kilobytes per second through the fastest conventional parallel port.

The EPP specification allows for a cycle time of one-half microsecond in its initial implementations. That translates to a peak transfer rate of two megabytes per second. In actual operation with normal processing overhead, EPP ports come close to half that rate, around 800 kilobytes per second.

Such figures do not represent the top limit for the EPP design, however. In future versions of the EPP standard, timing constraints may be tightened to require data on the interface to become valid within 100 nanoseconds. Such future designs allow for a peak transfer rate approaching eight megabytes per second. Such a rate actually exceeds the speed of practical transfers across the ISA bus. Taking full advantage of an EPP connection will require a local bus link.

Data Compression

One very effective way of increasing the speed of information through any interface is to minimize the number of bytes you have to move. By compressing the digital code—that is, reducing it to a more bit efficient format—you can reduce the number of bytes needed to convey text, graphics, and files. Already popular in squeezing more space from disks (for example, with DriveSpace and Stacker), tapes in backup systems, and modem connections, data compression is also part of the Extended Capabilities Port standard.

As an option, the ECP system allows you to compress the data you send through the parallel interface to further increase the speed of transfers. The port circuitry itself handles the compression and decompression, invisible to your PC and its software as well as to the peripheral at the other end of the connection. The effect on your transfers is the same as increasing the speed of the signals across the parallel cable but without all the electrical problems.

The ECP design uses a simple form of compression called *Run Length Encoding* or RLE. As with any code, RLE can take many different forms but the basic principle is the same. Long repetitions of the same digital pattern get reduced to a single occurrence of the pattern and a number indicating how many times the pattern is repeated. The specific RLE algorithm used by the ECP system works at the

byte level. When the same byte is repeated in a sequence of data, the system translates it into two bytes: one indicating the original code and a multiplier. Of course, if bytes do not repeat, this basic form of RLE is counterproductive. Using two bytes to code one increases the number of bytes required for a given amount of data. To minimize the impact of this expansion, the RLE algorithm used by the ECP system splits the difference. Half the possible byte values are kept untouched and are used by the code to represent the same single byte values as in the incoming data stream. The other byte values serve as multipliers. If one of the byte values that are reserved for multipliers appears in the incoming data stream, it must be represented by two bytes (the byte value followed by a multiplier of one). This system allows two bytes to encode repeated character streams up to 128 bytes long.

At its best, this system can achieve a compression ratio of 64 to 1 on long repetitions of a single byte value. At worst, the system expands data by a ratio of 1 to 2. With real world data, the system achieves an overall compression ratio approaching 2 to 1, effectively doubling the speed of the parallel interface whatever its underlying bit-per-second transfer rate.

RLE data compression can be particularly effective when you transfer graphics images from your PC to your printer. Graphic images often contain long sequences of repeated bytes representing areas of uniform color. RLE encoding offers little benefit to textual exchanges because text rarely contains long repetitions of the same byte or character. Of course, sending ordinary text to a printer usually doesn't strain the capabilities of even a standard parallel port, so the compression speed boost is unnecessary.

Bus Mastering

System overhead is the bane of performance in any data transfer system. The more time your PC's microprocessor spends preparing and moving data through the interface, the less of its time is available for other operations. The problem is most apparent during background printing in PCs using older, slower microprocessors. Most applications give you the option of printing in the background so you can go on to some other task while your PC slowly spools out data to your printer. All too often, the PC slows down so much during background printing that it's virtually useless for other work. This problem occurs in PCs as powerful as 486-based machines.

The slowdown has several sources. Your microprocessor may have to rasterize a full-page image itself (as it often does when printing from Windows) or it may spend its time micro-managing the movement of bytes from memory to the registers of the parallel interface. Although system designers can do anything to improve the speed of the former case, short of using a more powerful microprocessor, they have developed several schemes to minimize system overhead. One dramatic improvement comes with sidestepping the printer BIOS routines and taking direct control of the interface circuitry. Another is to take the transfer job from the microprocessor and give it to some other circuit. This last expedient underlies the technology of *bus mastering*.

Bus mastering can improve overall system (and printing) performance two ways. The circuit managing the transfers can be more efficient than your microprocessor at the chore. It may be able to move bytes

faster. And, by removing responsibility from your microprocessor, it prevents data transfers from bogging down the rest of your PC. Your microprocessor has more of its time for doing whatever a microprocessor does.

In systems that allow the bus mastering of parallel ports, the transfers are typically managed by your system's DMA (Direct Memory Access) controller. Your microprocessor sets up the transfer—specifying where the bytes are coming from, where they are to go, and how many to move—and lets the DMA controller take over the details. The DMA controller then takes control of the bus, becoming its master, and moving the bytes across it.

Bus mastered parallel transfers have not won wide favor. The technology does not work well on the ISA expansion bus, and IBM introduced it late in the life of the Micro Channel system. Moreover, the high processing speed of modern 486 and better microprocessors coupled with comparatively low throughput of the standard parallel interface makes bus mastering an unnecessary complication. Although not currently applied to PCI or VL Bus systems (both of which support bus mastering), the technology could give a boost to EPP and ECP performance because of the higher throughput and simplified means of transfer bytes to those interfaces.

Plug-and-Play

The Plug-and-Play system developed by computer manufacturers with the intention of making your life simpler—or at least dealing with the setup of your PC easier—extends to input/output ports and printers. Plug-and-Play technology allows your PC to detect and identify the various hardware devices that you connect to your computer. For example, a printer that understands and uses the Plug-and-Play system can identify itself to your PC and tell your PC which software driver is best to use.

The basic mechanism required for the Plug-and-Play system to work for printers is built into the IEEE 1284 specification. The actual identification and matching of drivers gets handled by your PC's operating system.

Benefits

Equipment made in accord with the Plug-and-Play specification tells your PC the system resources it needs, and your PC can then automatically assign those resources to the equipment. Unlike when you set up hardware yourself, your PC can infallibly (or nearly so) keep track of the resource demands and usages of each device you connect. Plug-and-Play technology lets your PC not only resolve conflicts between devices that need the same or similar hardware resources, but also the system prevents conflicts from occurring in the first place.

You only need to concern yourself, if at all, with two aspects of Plug-and-Play when you connect your printer—how it configures your ports and how it deals with your printer itself. Although you shouldn't even have to worry about these details most of the time, understanding the magic can help you better understand your PC and subvert the system when it creates instead of eliminates a problem.

Printers that conform to the requirements of the Plug-and-Play system enable several automatic features. A printer can then specify its *device class*, and the Plug-and-Play operating system will install features and drivers that work with that device class. The system allows your printer to identify itself with a familiar name instead of some obscure model number and use that name throughout the configuration process. That way you can understand what's going on instead of worrying about some weird thing in your computer with a name that looks eerily like the markings on the side of a UFO. And the Plug-and-Play printer can tell you what other peripherals that it works with.

Requirements

For the Plug-and-Play system to work at all, you need to run an operating system that has Plug-and-Play capabilities. Windows 95 is the first operating system to fully support the technology. DOS offers no Plug-and-Play support, and OS/2 Warp includes only a trifling bit of Plug-and-Play technology, used chiefly in administering PC Card slots. It cannot automatically identify your printer.

Ideally, your PC and all the peripherals connected to it will comply with the Plug-and-Play specifications. If you buy a new PC in these enlightened times, you should expect that level of compliance. If you have an older system (or a new system into which you've installed old peripherals), however, you probably won't have full Plug-and-Play compliance. That's okay because in most cases a Plug-and-Play operating system can make do with what you have. For example, Windows 95 can identify your printer as long as it follows the Plug-and-Play standard even if you have cluttered your PC with old expansion boards that don't mesh with the standard.

To automatically identify your printer, the Plug-and-Play system needs only to be able to signal to your printer and have it send back identification data. Your parallel port is key to this operation, but the demands made from it for Plug-and-Play operation are minimal. The port may use any of the standard IEEE connector designs. It must also support, at minimum, nibble-mode bi-directional transfers. Nearly every parallel port ever made fits these requirement. Plug-and-Play prefers a port that follows the ECP design, and for the sake of maximum printer performance, so should you.

Of course, a printer must have built-in support of the Plug-and-Play standard if it is to take advantage of the technology. The primary need is simple. Your printer must be able to send to your PC Plug-and-Play identification information so your PC will know what kind of printer you've connected. So that your system can be certain about the kind of printer you have, it requires three forms of identification called *key values*. Three additional key values optimize the operation of the Plug-and-Play system.

Operation

The IEEE 1284 specification provides a mechanism through which your PC's operating system can query a device connected to a parallel port. When your PC sends out the correct command, the printer responds first by sending back two bytes indicating how much identification data it has stored. This value is the length of the identification data in bytes, including the two length-indicating bytes. The first byte of these is the more significant.

After your PC gets the length information, it can query your printer for the actual data with another command. Your Plug-and-Play printer responds by sending back the key value information stored inside its configuration memory (which may be ROM, Flash RAM, or EEPROM).

The three required identifications for Plug-and-Play to work are the Manufacturer, Command Set, and Model of your printer. Each of these is stored as a string of case-sensitive characters prefaced by the type of identification. The IEEE 1284 specification abbreviates these identifications as MFG, CMD, and MDL. For example, your printer might respond with these three required values like this:

MFG: Acme Printers; CMD: PCL; MDL: Roadrunner 713

The manufacturer and model identifications are unique to each manufacturer. These values should never change and typically will be stored in ROM inside your printer. Ideally, the command set identification tells your computer what printer driver to use, Hewlett-Packards Printer Control Language (PCL) in the example line. Although it is often a fixed value in a given printer, if your printer allows you to plug in additional emulations or fonts, the value of the command set identifier should change to match. Note that Windows 95 ignores the command set key value. Instead, when it automatically sets up your printer driver, it relies on manufacturer and model information to determine which driver to use.

Windows 95 generates its own internal Plug & Play identification for working with key value data. It generates its ID value by combining the Manufacturer and Model values and appending a four-digit checksum. If the Manufacturer and Model designations total more than 20 characters, Windows 95 cuts them off at 20 characters but only *after* it calculates the checksum. The result is a string 24 or fewer characters long. Finally Windows 95 adds the preface LPTENUM\ (indicating the parallel port enumerator) so that it knows the path through which to find the printer. The result is the printer's Plug-and-Play ID that Windows 95 uses internally when matching device drivers to your printer. For example, the internal Windows 95 ID for a Hewlett-Packard LaserJet 4L printer would be the following character string:

```
LPTENUM\Hewlett-PackardLaserC029
```

Printer manufacturers can add, at their option, other identification information to the Plug-and-Play key values. The IEEE 1284 specification envisions Comment and Active Command Set entries.

Microsoft defines its own trio of options: Class (abbreviated CLS), Description (or DES), and Compatible ID (or CID). These values are not case-sensitive.

The Class key value describes the general type of device. Microsoft limits the choices to eight: FDC, HDC, Media, Modem, Net, Ports, or Printer.

The Description key value is a string of up to 128 characters that is meant to identify the Plug-and-Play device in a form that human beings understand. Windows 95 uses the Description when referring to the device on screen when it cannot find a data (INF) file corresponding to the device. Normally Windows would retrieve the onscreen identification for the device from the file. The Description key value keeps things understandable even if you plug in something Windows has never encountered before.

The Compatible ID key value tells Windows if your printer or other device will work exactly like some other product for which Windows might have a driver. For example, it allows the maker of a printer cloned from an Epson MX-80 to indicate it will happily use the Epson printer driver.

Once Windows 95 has identified your printer, its command set, and compatibilities, it uses these values to search through for the data it needs to find the drivers required by your printer and properly configure them. Of course, you always have the option to override the automatic choices when you think you know better than Mother Microsoft.

GP-IB Interface

Long before the standard parallel port had evolved its high speed extensions and enhancements, even before IBM unleashed its first PC onto the market, a number of applications had need for a relatively high speed yet simple data interface. Among the first areas that embraced such connections was scientific instrumentation. Engineers, scientists, and technicians often need to take extensive series of measurements at regular intervals to explore phenomena and test their latest creations. They were early to embrace the idea of automating measurements, both to

ensure reliability and to gain a good night's sleep without the alarm going off at 3 AM to warn of the need to make the next series of measurements. Connecting the various test and measurement instruments together and to a computer programmed to operate them assure researchers both peaceful slumber and results not compromised by bleary eyes and a lack of caffeine.

The leading manufacturer of scientific tests and measurements, Hewlett-Packard Company, developed its own parallel interface to link together its test equipment dubbed the *Hewlett-Packard Interface Bus* or HP-IB. The design became so popular that other companies including HP's competitors adopted it. In 1978 the design was sanctioned by the Institute of Electrical and Electronic Engineers as a formal standard, known as IEEE-488. In this guise, it wears a less proprietary common name, the General Purpose Interface Bus.

The HP-IB (Hewlett-Packard Interface Bus) name survives today. Rather than an alias for the industry standard, Hewlett-Packard maintains that HP-IB is a proprietary interface but one that provides for compatibility and two way communication between devices that follow the IEEE-488 standard. Devices that commonly use IEEE-488 include automated test and measurement equipment, printers, plotters, and PCs.

No matter the name you give it, the basic IEEE-488 design comprises 16 separate connections to move data and commands between electronic devices. Eight of these connections carry data in a true byte-wide bus. Three lines provide handshaking and flow control between the various devices that are linked together. The remaining five lines allow for arbitration and management of the bus connections. The standard connector also provides eight ground connections, one of which is a chassis or earth ground. Data and commands flow between the linked devices on the eight data lines asynchronously, governed by the handshake signals.

The standard connector used by the GPIB system resembles a parallel port B connector but has only 24 connections. It has two parallel rows of ribbon contacts arranged around a tab that's inside a shield. To assure the physical integrity of the connection, female plugs and jacks have bail wires that latch into male connectors. Figure 19.7 shows one form of female GPIB jack.

Figure 19.7 A 24-place GP-IB female jack.

As a true bus, the cables used in a GP-IB system are all straight-through, connected pin-for-pin one end to another. No twists, turns, flip-flops, or crossovers are required for linking together various devices. Table 19.13 lists the signal assignments for the various pins in a standard GP-IB

connector.

Table 19.13. GP-IB Signal Assignments

Pin	*Signal name*	*Function*
1	D101	Data Bus 1
2	D102	Data Bus 2
3	D103	Data Bus 3
4	D104	Data Bus 4
5	EO1	End or Identify
6	DAV	Data Valid
7	NRFD	Not Ready for Data
8	NDAC	Not Data Accepted
9	IFC	Interface Clear
10	SRQ	Service Request
11	ATN	Attention
12	SHIELD	Earth Ground
13	D105	Data Bus 5
14	D106	Data Bus 6
15	D107	Data Bus 7
16	D108	Data Bus 8
18	GND	Signal Ground
19	GND	Signal Ground
20	GND	Signal Ground
21	GND	Signal Ground
22	GND	Signal Ground
23	GND	Signal Ground
24	GND	Signal Ground

Devices are wired together into a bus by daisy chaining. A

device may have two connectors to facilitate the daisy chain connection or the cable may have two interface connections at its ends, one male (which plugs into a device) and, on the opposite side, a female connector that allows you to plug in a second cable to run to the next device.

The GP-IB standard defines three types of devices that operate on the bus: talkers, listeners, and controllers.

A *talker* is a device that sends data out or talks to any of the other devices linked by the bus. A *listener* is the device that receives the data from the talker. A *controller* manages the interactions between the various devices that are linked by the bus.

These definitions are dynamic. A given device may operate one moment as a talker and the next moment as a listener, depending on its current function in the system. The controller sends out commands that make a given device act as talker or listener.

GP-IB specifies not only the bus hardware but also a transfer protocol and set of commands. These commands are sent to devices like data across the data lines. To identify commands and distinguish them from data, the controller activates the Attention (ATN, pin 11) line on the bus to indicate a command byte.

Because GP-IB is a bus, all devices linked by it share the

same signals. To route data or commands to a specific device, each device attached to the bus must be given a unique address. Addresses on the GP-IB range from zero to seven, allowing for a maximum of eight devices. Each address corresponds to one of the data lines, which is used as a signal to address the device. Some equipment puts restrictions on some addresses. For example, some HP printers assign address seven as "listen only." In this mode, the printer listens to all the traffic on the bus regardless of its address. This allows the printer to serve as a log of all bus traffic, a useful function in monitoring scientific apparatus.

GP-IB addresses are typically set with a DIP switch on the back of the device near the interface connector.

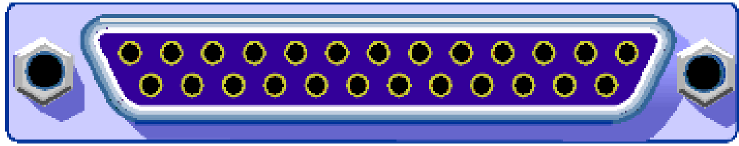
In normal operation, a controller manages the bus by *polling*. That is, it periodically sends out commands to each device to see which require service. For example, during an experiment the controller may poll a thermocouple linked to the bus to determine (and log) the current temperature of an experiment.

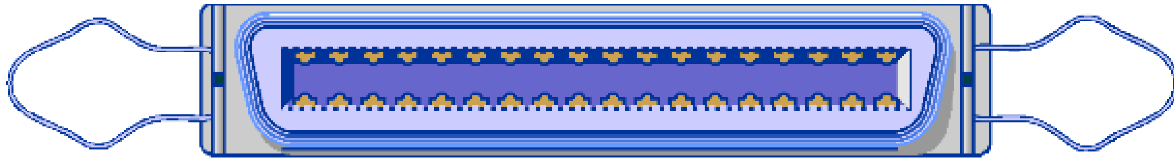
Individual devices can also require immediate service by activating the Service Request (SRQ, pin 10) line on the bus. The controller then responds by polling the bus to determine which device made the request, and finally servicing the request.

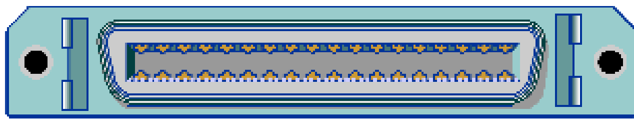
The controller can monitor the bus with either parallel or serial polling.

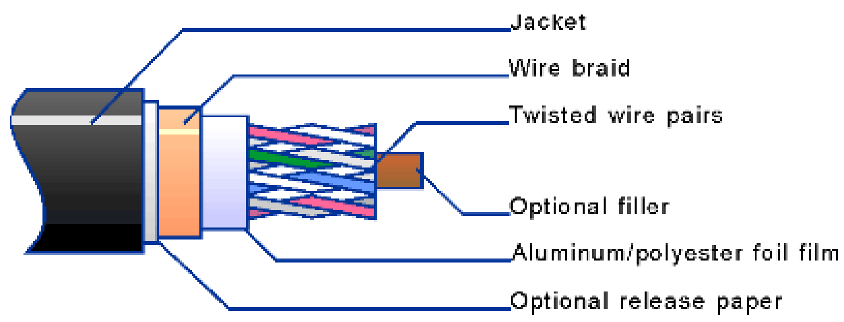
The controller uses parallel polling when it needs to check all devices on the bus or wants to determine which device activated the SRQ bus line. To start a parallel poll, the controller activates both the End or Identify (EOI, pin 5) and Attention (ATN, pin 11) bus lines at the same time. Each device that has both been enabled to respond and requires service activates the data line with the number corresponding to its address. The controller enables devices with GP-IB commands.

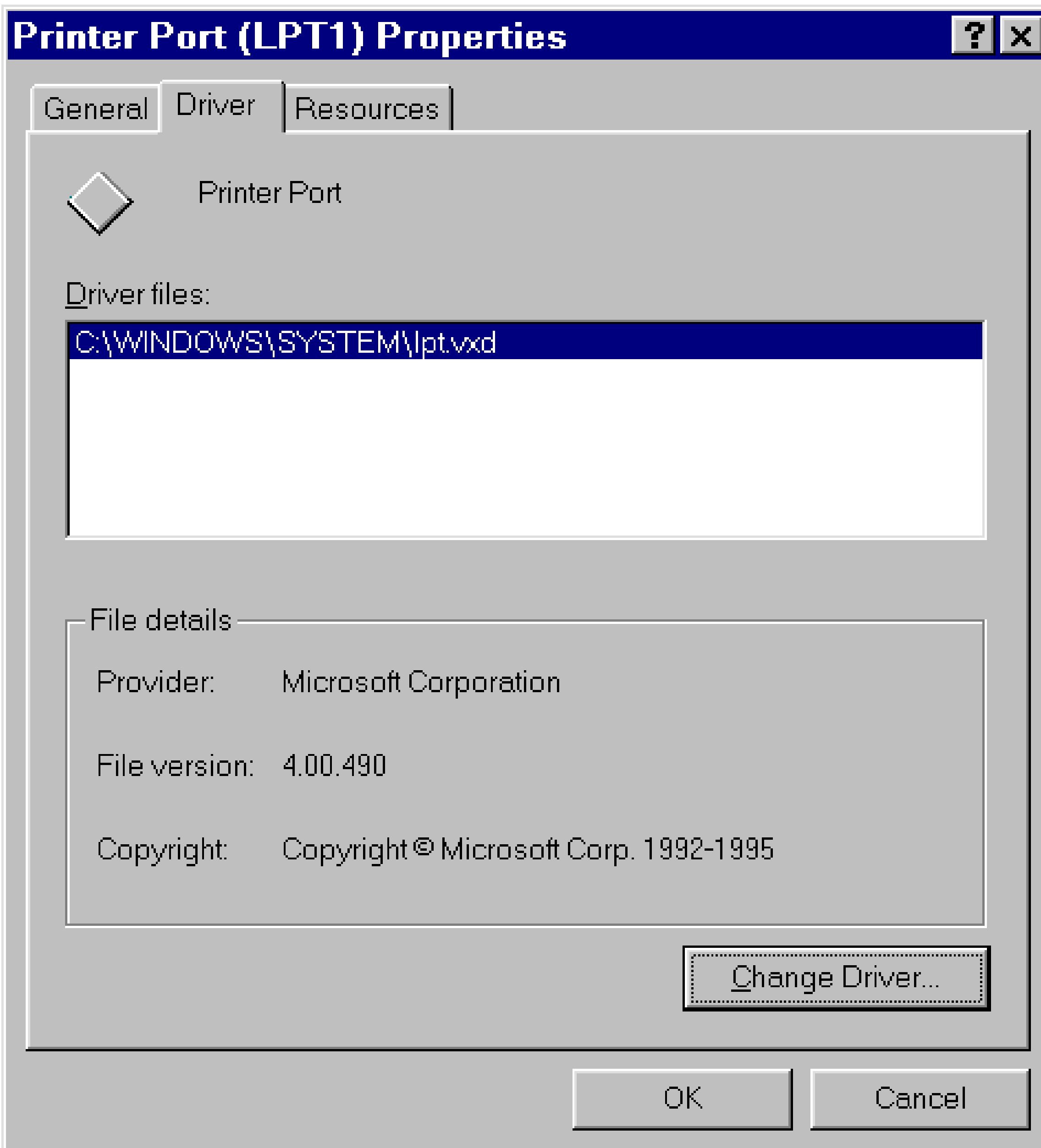
In serial polling, the controller addresses each device sequentially, using the bus data lines to send out the address of the device to be polled. As it is identified, each device responds in turn, sending out data should it have been programmed to do so.



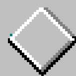





Jacket
Wire braid
Twisted wire pairs
Optional filler
Aluminum/polyester foil film
Optional release paper


Printer Port (LPT1) Properties
General
Driver
Resources
Printer Port
Driver files:
C:\WINDOWS\SYSTEM\lpt.vxd
File details
Provider: Microsoft Corporation
File version: 4.00.490
Copyright: Copyright © Microsoft Corp. 1992-1995
Change Driver...
OK
Cancel

Select Device



Click the Ports (COM LPT) that matches your hardware, and then click OK. If you don't know which model you have, click OK. If you have an installation disk for this device, click Have Disk.

Manufacturers:

(Standard port types)
DBC
Hewlett Packard
IBM
Rockwell
SMART Modular Technologie
Socket Communications

Models:

Communications Port
ECP Printer Port
Generic IRDA Compatible Device
Printer Port

Show compatible devices

Show all devices

Have Disk...

OK

Cancel