PC Hardware Interfaces: A Developer's Reference

by Michael Gook

A-LIST Publishing
© 2004 (672 pages)
ISBN:193176929X

Focusing on the latest research on hardware interfaces used in modern information technology, this reference covers universal external interfaces, peripheral device interfaces, external memory interfaces, expansion buses, wireless interfaces, and more.

Table of Contents

PC Hardware Interfaces: A Developer's Reference
by Michael Gook
A-LIST Publishing © 2004 (672 pages)
ISBN:193176929X
Focusing on the latest research on hardware interfaces used in modern information technology, this reference covers universal external interfaces, peripheral device interfaces, external memory interfaces, expansion buses, wireless interfaces, and more.

Back Cover

Informatics and data processing dictionaries define an interface as a common boundary shared by two systems, devices, or programs, as well as the elements of this connecting boundary and the auxiliary control circuits used for linking devices. This book deals mainly with interfaces that allow various peripheral devices and their controllers to be connected to personal (and not only personal) computers. Before starting a detailed discussion of specific PC interfaces, a series of general issues regarding the establishment of connections need to be considered.

About the Author

Michael Gook is a robotics and cybernetics hardware and related software developer.

PC Hardware Interfaces—A Developer's Reference

A-LIST, LLC
295 East Swedesford Rd.
PMB #285
Wayne, PA 19087
702-977-5377 (FAX)
mail@alistpublishing.com
http://www.alistpublishing.com

Michael Gook. *PC Hardware Interfaces: A Developer's Reference*

1-931769-29-X

04 7 6 5 4 3 2 1

A-LIST, LLC titles are available for site license or bulk purchase by institutions, user groups, corporations, etc.

Book Editor: Peter Morley

Introduction to Interfaces

Informatics and data processing dictionaries define an *interface* as a common boundary shared by two systems, devices, or programs, as well as the elements of this connecting boundary and the auxiliary control circuits used for linking devices. This book deals mainly with interfaces that allow various peripheral devices and their controllers to be connected to personal (and not only personal) computers. Before starting a detailed discussion of specific PC interfaces, a series of general issues regarding the establishment of connections need to be considered.

General Computer Structure

PC compatible computers, like most computers, are built around the classic design of the von Neumann machine of 1945. According to this design, a computer consists of a central processing unit (CPU), memory, and input/output devices. The *processor* executes programs that are stored in the memory; the *memory* stores the programs and data available to the processor; the *input/output devices* provide a link to the external world. Over time, slight corrections have been made to the names of these three fundamental parts, and now devices that used to fall under the definition of input/output devices are called *peripheral devices.* The processor (one or more), memory, and the necessary elements linking them all each with another and with other devices are called the computer's *central part or core.*

Peripheral Devices

Peripheral devices are all computer components not pertaining to the core. They can be divided into several types:

- *Data storage* devices (external memory devices): disk (magnetic, optical, magneto-optical), tape (streamers), solid state (flash memory cards, modules, and USB devices). These devices are used for nonvolatile storage of information from the memory, and for loading this information back into the main memory. How this information is stored in these devices internally is not really important (the important thing is to correctly read out what has been stored).
- *Input/output* (I/O) devices convert information from its computer-internal representation form (bits and bytes) into a form understandable to man (and other creatures) and to various technical devices, and in the reverse direction. (It is not difficult to adapt a computer to control any equipment; all that is needed are sensors and actuating units.) To this category, displays (output devices), keyboards and mice (input devices), printers and scanners, plotters and digitizers, joysticks, acoustic systems and microphones, televisions and video cameras, remote control and telemetry devices belong. The form, into which these devices convert binary information, is determined by their functions.
- *Communications* devices are used to transfer information between computers and/or their parts. To this category, modems (wire, radio, infrared, etc.) and local and global network adapters belong. In this area, information needs to be converted from one form into another only so that it can be sent over some distance.

The main operation in the computer is execution of program code by the central processor, and the CPU must be able to interact with peripheral devices.

To access peripheral devices in the x86 processors that are used in the PC-compatible computers, the I/O address space allocated is separate from the memory address space. The size of the I/O space is 64 Kbytes; 1-, 2-, or 4-byte registers of peripheral devices can be mapped onto this space and accessed by several special processor instructions. Registers of peripheral devices can also be mapped onto the main memory space, onto the areas not taken by the system random-access (RAM) and read-only (ROM) memory. The I/O space is separated from the memory space in not all processor architectures. In any case, the addresses of different registers of different devices must not overlap in the address space onto which they are mapped: This is the requirement of conflict-free address resource allocation.

In terms of interaction with the rest of the computer's components, the processor cannot do anything other than access a location in memory or I/O space (to read or write a byte, a word, or a double word), and to react to the hardware interrupts. In this way, any peripheral device must present itself to the processor as a set of registers or memory cells, or be an interrupt source (the latter is not a mandatory requirement).

Hierarchy of Connections

The components of a computer system are connected with each other in a hierarchy, at the top of which the *system-level connection interfaces* are located. To these, the following belong:

- Front Side Bus (FSB) to which the central processor (processors in complex systems) is connected
- Memory Bus
- I/O buses providing communication between the computer core and the peripheral devices

This level of connections being of the system type means that physical memory and I/O addresses (if there are any) are used here.

The most representative specimens of the I/O buses in IBM PC are the ISA (rapidly becoming extinct) and PCI (evolving into PCI-X and further) buses. All processor accesses to the peripheral devices go through the I/O buses. *Controllers* and *adapters*[1] of peripheral devices or their interfaces are connected to I/O buses. Some peripheral devices are combined with their controllers (adapters) (e.g., Ethernet network adapters that are connected to the PCI bus). Other peripheral devices are connected to their controllers via intermediate *peripheral interfaces,* which make up the low level of the connection hierarchy.

Peripheral interfaces are the most varied of all hardware interfaces. Most storage devices (disk, tape type), I/O devices (displays, keyboards, mice, printers, plotters), and some communications devices (external modems) are peripheral devices that are connected via intermediate interfaces. To interact with a peripheral device, the processor accesses registers of the controller that represents the device connected to it.

In terms of the functions they perform, peripheral interfaces can be divided into specialized and universal, dedicated and shared:

- *Specialized* interfaces are oriented at connecting a certain narrow class of devices, and use very specific data transfer protocols. Examples of such protocols include the very popular VGA monitor interface, the floppy disk drive interface, the traditional keyboard and mouse interfaces, and the interface of the IDE/ATA hard disk controllers.
- *Universal* interfaces are more widely used; their protocols deliver all types of data. Examples include the communication port (COM), SCSI, USB, and FireWire protocols.
- *Dedicated* interfaces allow only one device to be connected to one port (connection point) of the adapter (controller); the number of devices that can be connected is limited by the number of ports. Examples are the COM port, VGA monitor interface, AGP, and Serial ATA.
- *Shared* interfaces allow multiple devices to be connected to one adapter port. Physical connections can take many forms: a bus (printed circuit like ISA or PCI; cable bus like SCSI or IDE/ATA), a daisy chain of devices (SCSI, IEEE 1284.3), a logical bus implemented with hubs (USB), or built-in repeaters (IEEE 1394 FireWire).

[1]A controller is more "intelligent" than an adapter.

Interface Organization

The task of each of the interfaces considered here is to transfer information between some devices. To get the whole picture, the type of information that can be transmitted and the physical signals used to do this will be considered. Then, the types of relationships into which the interconnected devices can enter and the tasks various protocols need to accomplish will be looked into.

Types of Transferred Information

Information (data) that needs to be sent over interfaces can be of various natures. Some of the types of transmitted information are listed below:

- Analog information reflects a process continuous in time and magnitude (i.e., it can take on any of an indefinite number of values even within a definite interval). One example, sound (including speech) is continuous changes in (usually) air pressure. The task of how to transfer such information arises, for example, when a microphone (a device for converting air pressure changes to electric voltage changes) is connected to a computer.
- Discrete information depicts a process by a finite number of values. The elementary discrete information unit is a bit, which can take on one of two logical values: 1 (true, yes) or 0 (false, no). One bit can be used to reflect the state of a mouse button (whether it is pressed or not). Discrete binary information is natural to most computers as it is the easiest to obtain, process, store, and transmit. Discrete information can be not only binary; of interest is the ternary system, in which a tribit[1] can have three states—yes, no, and don't know.
- Digital information is a sequence (set) of values that have finite width (and, accordingly, finite number of possible values). An example is digitized sound, which is a sequence of samples of instantaneous pressure values taken at equal time intervals.

Discrete and digital information are often confused (although, it is not always necessary to tell them apart) because they look similar: in the binary number system, both of them are represented by collections of ones and zeroes. Digital information is a special case of discrete information. Concerning data transfer over different interfaces, the division of information into analog (continuous) and discrete is of importance.

In order to transmit data, they must be represented as a *signal:* a physical process (electric, optical, or electromagnetic, although others also are possible). Signals can be of different types: analog (continuous), discrete, or digital. The type of signal may not necessarily correspond to the type of the transferred data. So, for example, a telephone modem analog signal carries discrete (digital) data. The type and nature of the signal used is determined by the requirements of the interface: transmission range, data transfer rate, data reliability and validity, security, cost, connection ease, power consumption, and others.

Parallel and Serial Interfaces

For computers and devices connected to them, the most common task is transfer of, as a rule, significant amounts (more than one bit) of discrete data. The most common way to represent data by signals is the binary method: For example, logical one corresponds to the high (above the threshold) voltage level; logical zero corresponds to the low (below the threshold) voltage level. A reverse assignment is also possible. One binary signal transfers one bit of information within one unit of time. As already mentioned, the processor exchanges information with peripheral devices in bytes (8 bits), words (in the x86 world, one word is two bytes or 16 bits), or double words (32 bits). There are two approaches to organize the interface to transfer a group of bits:

- *Parallel interface.* A separate line (usually binary) is used for each transmitted bit of the group, and all bits of the group are transmitted simultaneously within one unit of time (i.e., they move over the interface lines in parallel). The 8-bit parallel printer port (LPT port), the 16-bit ATA/ATAPI interface, the 8- or 16-bit SCSI port, and the 32- or 64-bit PCI bus all are examples of the parallel interface.

- *Serial interface.* Only one signal line is used, and bits of the group are sent in turn one after another; each of them is assigned a unit of time (bit interval). The serial communications (COM) port, the serial USB and FireWire buses, and local and wide area network interfaces are all examples of the serial interface.

At first glance, the parallel interface seems to be easier and straightforward to organize, as there is no need to queue the bits for transmission and then assemble bytes from the received serial bits. Also, at first glance, the parallel interface is a faster way to transmit data, because bits are sent in packets. The obvious shortcoming of the parallel interface is the large number of wires and connector contacts in the connecting cable (at least one for each bit). For this reason, parallel interface cables and interface circuits of devices are cumbersome and expensive, which is tolerated for the sake of the desired transmission speed. The transceiver of the serial interface is more complex functionally, but its cables and connectors are much simpler and less expensive. Naturally, it is not wise (not to mention impossible) to lay multiple-wire parallel interface cables over long distances, the serial interface is much more suitable here. These considerations were decisive in selecting an interface until about the early 1990s. Then, the choice was simple: parallel interfaces were employed over short distances (a few or tens of meters maximum) where high speeds were required; over long distances, as well as when parallel cables were unacceptable, serial interfaces were used, sacrificing the transfer speed.

That the data transfer rate or speed is defined as the number of bits sent over a unit of time divided by the unit's duration. For simplicity, the interface *clock rate* or *frequency*—a unit that is the inverse of the unit's length (period)—can be used. This concept is natural for synchronous interfaces, which have a synchronization (clocking) signal that determines the allowable starting moments of all events (state changes). For asynchronous interfaces, the equivalent clock rate—a unit that is the inverse of the minimal length of one interface state—can be used. Using these definitions, it can be said that the maximum (peak) data transfer rate equals the product of the frequency and the interface width. The width of the serial interface is one bit; the width of the parallel interface equals the number of the signal lines used to transfer data bits. With the speed issue resolved, only the questions of maximum frequency and width remain. For both the parallel and the serial interfaces, the maximum frequency is determined by the attainable efficiency (at reasonable monetary and energy expenses) of the transceiver circuits of the devices and the cables' frequency characteristics. The advantages of the serial interface can already be seen here: Expenses for building high-speed elements do not have to be multiplied by the interface's width, as is to be done with the parallel interface.

There is a phenomenon in the parallel interface called skew that significantly affects the achievable frequency limit. In essence, skew means that signals sent simultaneously from one end of the interface do not arrive to the opposite end simultaneously because of the differences in the individual signal circuits' characteristics. The transit time is affected by the length of wires, insulation characteristics, connecting elements, etc. Obviously, the signal skew (difference in the arrival time) of different bits must be explicitly smaller than the time unit, otherwise the bits will be scrambled with the bits of the same name from previous and successive transfers. It is totally clear that the signal skew limits the allowable interface cable length: At the same relative signal propagation speed difference, as the transmission range increases, so does the skew.

The skew also hinders increasing the interface width: The more parallel circuits are used, the more difficult it is to make them identical. Because of this, wide interfaces even have to be broken down into several narrower groups, for each of which their own control signals are used. In the 1990s, frequencies of hundreds of megahertz and higher started to be employed (i.e., the time unit started to be measured in nanoseconds or fractions of nanoseconds). A proportionally small skew could be attained only within the confines of rigid compact constructions (like printed circuit boards), while to connect devices by cables tens of centimeters long, frequencies of up to tens of megahertz were the maximum. For the purposes of orientation among the numbers, in one nanosecond a signal travels 20-25 centimeters over electrical conductor.

In order to increase the bandwidth of parallel interfaces, Dual Data Rate (DDR) was employed in the mid-1990s. In essence, it consists of making the switching frequencies of the data signal lines and the clock signal lines equal. In the classical version, the data of the information lines were latched only at one transition (positive or negative) of the clock signal, which doubles the switching speed of the clock lines relative to the

data lines. With double clocking, data are latched at both the positive and the negative transitions; therefore, the switching frequency for all lines becomes the same, which with the identical physical characteristics of cables and interface circuits makes it possible to double the bandwidth. This modernization wave started with the ATA interface (UltraDMA modes) and has already rolled over the SCSI (Ultra160 and higher) and the memory (DDR SDRAM) interfaces.

Moreover, *Source Synchronous transfer* is employed at high frequencies: The clocking signal, which determines the moments at which switching is done and the data are valid, is generated by the source of data itself. This makes it possible to align the data and clock signals more precisely in time, because they propagate over the interface in parallel and in one direction. The alternative—a common source clock—cannot handle high switching frequencies, because the time relations between data and clock signals at different (geographical) points will be different.

Increasing the interface signal switching frequency, as a rule, is accompanied by lowering the levels of the signals that are generated by the interface circuitry. This tendency is explained by power considerations: A frequency increase means that the time needed to switch a signal decreases. At higher signal amplitudes, a higher signal rise rate is required and, consequently, a higher transmitter current. Increasing output current (pulsing) is not desirable for various reasons: It causes great crosstalk interferences in the parallel interface, necessitates the use of powerful output signal drivers, and produces excessive heat. The tendency to lower signal voltages can be traced to the examples of the AGP interface (3.3 V—1.5 V—0.8 V), the PCI/PCI-X buses (5 V—3.3 V—1.5 V), the SCSI and other memory and processor buses.

The skew problem does not exist in serial interfaces; therefore, the frequency can be increased up to the capability limits of the transceiver circuits. Of course, cable frequency characteristics impose limitations, but it is much easier to fashion a good cable for one circuit than for several, and with high identity requirements to boot. And when the electric cable cannot handle the required frequency and range, a switch to optical cable can be made, whose undeveloped frequency reserves are huge. A parallel optical interface, on the other hand, is too much of a luxury[1].

The reasons stated above explain the tendency to switch to serial methods of data transfer. Table 1 lists the comparative characteristics of the parallel and serial interfaces presently employed to connect peripheral devices. To complete the picture of parallel data transfer method achievements, parallel buses used to connect processors to the memory must be mentioned. Their width reaches up to 128 bits and their data transfer rates can reach several gigabytes per second. However, these performance characteristics are obtained only over very short distances (some ten centimeters or even less). Serial network interfaces (Ethernet 10/100/1000/10000 Mbps, transfer technologies of global networks such as ATM, SONET, and SDH), with their impressive combination of gigabit speeds and communication range (hundreds of meters to tens and hundreds of kilometers) are not included in this table. There are some nuances when comparing parallel interface speeds, which are expressed in megabytes per second (MBps), and serial interface speeds, which are expressed in megabits per second (Mbps); they will be explained further on. For the time being, megabits per second can be divided by 10 (not by 8) to obtain an approximate value of megabytes per second.

Table 1: Parallel and Serial Interface Comparisons

Name, width	Use	Speed limits	Range	Prospects
Parallel interfaces and buses				
IEEE 1284 (LPT port), 8 bit	Connecting printers, plotters, scanners, and others.	2 MBps	1-10 m	Abandonment in favor of USB
SCSI, 8/16 bit	Connecting any internal and external devices	10-20-40-80-160-320 MBps	Up to 6-12-25 m	Move to the Serial SCSI
ATA/ATAPI,	Connecting internal data	16-33-66-10-133 MBps	Up to 0.5 m	Move to the

16 bit	storage devices			Serial ATA
ISA, 8/16 bit	Connecting internal devices (expansion cards)	8-16 MBps	15-20 cm (within limits of the PCB with expansion slots)	Abandonment
PCI, 32/64 bit	Connecting internal devices (expansion cards)	66-133-266-533 MBps	10-15 cm (within limits of the PCB with expansion slots)	Move to PCI-X
PCI-X, 32/64/16 bit	Connecting internal devices (expansion cards)	533-4256 MBps	10-15 cm (within limits of the PCB with expansion slots)	
Serial interfaces				
RS-232C (COM port)	Connecting communications and other external devices	115200 bps (11.5Kbps)	25 m	Abandonment in favor of USB
RS-422/485	Connecting industrial automation devices	10 Mbps	1,200 m	
USB 1.0-1.1	Connecting external devices	1.5-12 Mbps (up to1 MBps)	30 m	Move to USB 2.0
USB 2.0	Connecting external devices	1,5-12-480 Mbps (up to 24 MBps)	30 m	
IEEE 1394 (FireWire)	Connecting external devices, organizing multimedia device networks	100-200-400 Mbps	72 m	Moving to 800 Mbps and 1,600 Mbps.
Serial ATA	Connecting internal data storage devices	1.5 Gbps (150 MBps)	1 m	Moving to higher speeds
Serial SCSI	Connecting internal and external devices	3 Gbps	6m	Moving to 6 Gbps
Fibre Channel	Connecting external data storage devices	2 Gbps	15m (copper cable) – 10 km (optical cable)	Moving to 4 Gbps
PCI Express	Connecting components on PCBs interblock connection	2,5-80 Gbps	Within the limits of the chassis	
Hyper Transport	Connecting components on PCBs	Up to 6.4 GBps (total of the two oncoming lines up to 12.8 GBps)	Within the limits of the PCB	
InfiniBand	Connecting components on PCBs; interblock and intercomputer connections (in clusters)	2.5, 10, and 30 Gbps	Up to 17 m over a copper cable; up to 10 km over optical cable.	

Signals and Transmission Medium

The most common physical process used to transmit interface signals is electromagnetic oscillations of various frequencies. The most usual electrical signals are electromagnetic oscillations of relatively low frequency range (up to tens or hundreds of megahertz) transmitted over electric wires. This type of transmitter places its signal in the form of certain voltage of current levels on one end of the electric communications line (cable) while the receiver on the other end of the line receives a signal that is like the signal sent to a greater

or lesser extent. The degree of similarity (or dissimilarity) is determined by the cable's performance characteristics, its length, the frequency spectrum of the signal, and the amount of external interference. When high frequency is combined with great length (of interest is the LF factor: the product of the length L and the frequency F), wave characteristics of electromagnetic signals, such as propagation signal decay and reflections from the line's non-uniformities, must be taken into account.

Signal decay occurs not only because of electromagnetic losses (cable heating), but because of spurious radiation: electromagnetic signals emanating beyond the confines of the cable into the ether. Because of wave effects, electric cables employed to transmit the signal must be of special construction: coaxial cables, twisted pairs, and some others. The task of these constructions is to preserve the shape of the transmitted signal as faithfully as possible, not to allow it to escape the cable limits, and, as far as possible, not to let external interferences penetrate the cable. The last two items are especially important in providing information transmission security and confidentiality: preventing outside eavesdropping on and malicious damage to the information (or making it difficult). The overwhelming majority of peripheral device interfaces use wires to transmit electric signals, which provides transmission ranges of several, tens, and hundreds of meters at speeds of up to several gigabits per second; wire transmission also dominates in computer networks.

Electromagnetic oscillations in the hundreds of megahertz to tens of gigahertz range can also be employed for wireless radio transmission of signals. Frequencies lower than these require unacceptably large antennas for efficient emission and reception; with higher frequencies, difficulties exist so far in implementing transceivers, and moreover, signal propagation in this frequency range has some unattractive specifics (such as being absorbed by fog or rain). The first computer radio interface to have been widely used—Bluetooth—employs a microwave range frequency of 2.4 GHz. The radio waves propagate in a straight line in this range (there is no bending effect peculiar to low frequency radio waves) and, with some attenuation, can penetrate through walls. The unpleasant part is that the signal is reflected from various objects, so that the receiver receives not only the direct transmitter signal but also the bounced signals, which arrive with some delay relative to the original signal. Because of this multipath reception effect, communication on some frequencies is impossible at some points, but as soon as the receiver or transmitter is moved a little or the frequency is changed, the connection resumes.

Various methods are used to combat the problem of the signal decay due to the multipath reception effect; the Bluetooth technology employs the carrier hopping method. It uses omnidirectional antennas; the communication range, depending on how powerful the transmitter is, is limited to tens or hundreds of meters. Wireless interfaces are attractive because of the absence of cables and connectors, which have to be laid and connected to organize communications: In order to establish a radio connection, devices only have to be within the coverage area. However, they also have their demerit: The transmission medium is totally open to everyone including wrongdoers, who can intercept the signal to obtain information or introduce their own signals with malevolent purposes. The solution to this security problem lies beyond the scope of physical signal transmission. Another problem with radio interfaces is the high and constantly growing occupancy of the radio spectrum, which causes interference (unwanted interaction) of different equipment, including wireless and cellular phones, wireless local network equipment, microwave ovens, and other equipment.

Further up the electromagnetic oscillation frequency spectrum is, first, the infrared spectrum, which adjoins the visible optical range. These ranges are also used for optical transmission of signals both over wires (optical fiber) and without them.

The infrared port—the standard IrDA and its forerunners, HP-SIR and ASK IR—have been used for wireless connection of peripheral equipment (printers and other devices) to computers for many years. This type of connection looks especially effective with compact devices, whose size is proportionate to (or even smaller than) the cable and connectors of traditional interfaces. The IrDA port has directed emitters and receivers with a coverage angle of 15-30° and a range of about 1 meter, so to connect devices, they must be close enough to each other and properly oriented. The small coverage area (as compared to the radio interface) is not always a drawback: It is easier to control against unsanctioned connections, and users can be certain that no-one from behind the wall will tap into their line and eavesdrop on their traffic. The attainable maximum transfer speed

is not high (4 Mbps); higher speeds are problematic because of the low signal strength that reaches the photoreceiver. Infrared communication also is employed in wireless local networks (IEEE 802.11 DFIR) where omnidirectional transceivers using the signal bounce off the walls and ceiling make it possible to provide a coverage area of about 10 meters; however, speeds are even lower here (1-2 Mbps).

The undisputed leader in terms of the LF factor is the fiber optic link, in which infrared light impulses are sent over glass or plastic optical fiber. Glass fiber is mostly used in telecommunications, where transmissions over long distances, from hundreds of meters to tens (or even hundreds) of kilometers, are required. For the right wavelengths, certain types of optic fibers pass the signal with acceptable attenuation and shape distortion, making speeds in the gigabit range over distances measured in kilometers possible. Optical lines have a throughput reserve in the form of wave multiplexing: Many optical signals of different frequencies can be sent over one fiber without interfering with each other. Similar signal multiplexing has far more modest possibilities in electric cables. The shortcoming of the glass optic is that its end devices (transceivers) and the connecting equipment are quite expensive; the cable itself can be even cheaper than copper cable. For interfaces that do not require signals to be transmitted over long distances (up to several tens of meters), plastic optic fiber can be used; its cables and connectors are significantly cheaper than for glass fiber. An example of an optic interface in modern personal computers is the Toslink interface, the optic version of the digital S/PDIF audio interface. The Fibre Channel (FCAL) interface can be encountered in servers; data storage devices that may be located kilometers away from the computer are connected using this interface, providing speeds of about 1 Gbps.

While on the subject of optic interfaces, it must be noted that they provide complete *galvanic decoupling* of connected devices. Moreover, optic interfaces are not sensitive to electromagnetic interference. In some cases, these characteristics play a decisive role, for example, when connecting equipment at power plants, on industrial premises with strong interferences, and the like. Fiber optic interfaces are most secure against unauthorized connection. Information cannot be taken off the channel without physically tapping into it, and should it be necessary, line state can be monitored and an unauthorized tapping attempt detected in due time.

Galvanic Decoupling of Connected Devices

Galvanic decoupling means that the circuit grounds of the connected devices do not electrically connect with each other over the interface circuits. Moreover, the devices can be under significantly different potentials.

Most electrical interfaces do not provide for galvanic decoupling. For example, the circuit grounds of the devices connected to a COM or LPT computer port are linked with the computer's circuit ground (and over the interface cables, their grounds are linked with each other). If there is a potential difference before the connection, then equalizing current will flow over the interface cable common wire, which is no good for several reasons. A voltage drop on the common wire due to this current flow causes signal levels to shift, and the flow of alternating current leads to the mixing of the useful signal with the variable component of interference. TTL interfaces are especially sensitive to this interference, whereas in the RS-232C interface the dead zone will absorb up to 2 V of interference or signal shift. If the common wire is broken or there is a bad connection, but much more often when devices are connected or disconnected under the power on, the potential difference is applied to the signal circuits and the equalizing current flowing through them often causes damages to the devices. In audio equipment, equalizing currents cause audible interference (background hum).

Interface signals are decoupled from the device's ground using optoelectronic devices (in the MIDI and current loop interfaces) or transformers (FireWire bus, Ethernet interfaces). Sometimes, direct current decoupling is achieved using decoupling capacitors (in cheap versions of the FireWire and PCI Express interface).

Potential difference between devices connected by a galvanically decoupled interface is limited by the maximum *dielectric breakdown voltage* for the given interface. For example, Ethernet adapters (for twisted pair) must be able to withstand voltages of up to 1.5 KV; decoupling implemented using optrons withstands

500-1,000 V; the capacitor decoupling used in FireWire provides up to a 60 V safety margin. Fiber optic interfaces provide galvanic decoupling for any voltages: thousands or even millions of volts. Any wireless interface also provides galvanic decoupling.

Device Interaction and Device Topology

The interfaces considered here are used to exchange information between connected devices; the exchange is conducted using transactions. An *interface transaction* is a finite operation of transferring a certain amount of information. Each transaction involves a *master device* (exchange initiator), which controls the interface during the given time period, and a *slave device* (target device), the second participant in the transaction that is selected by the master device and follows its commands.

A transaction can consist of a series of phases (states, steps) in each of which some elementary objective is accomplished. The set of phases depends on the complexity of the interface, and in the simplest case it comes down to simple data transfer phases. The *interface protocol* is a set of rules for interaction between master and slave devices for executing transactions. The complexity of the protocol depends on the potential number of involved devices, their interrelations, and the characteristics of the exchange that the protocol has to provide.

If the interface links more than two devices, then the protocol has to tackle the addressing issue: selecting the device that will respond to the given transaction. At the same time, the enumeration task needs to be addressed: assigning each device a unique address, at least within the limits of the given grouping. The user must play a minimal role in assigning addresses, or, even better, not to have any part in it at all: This is one of the principles of the Plug and Play system. Modern interfaces—the PCI, USB, FireWire—were from the beginning developed with the objective of totally automatic address assignment taken into account. Automatic address assignment capabilities were added to several old interfaces later on and they have remained in some (the Microsoft Plug and Play specification for the ISA bus, for example) but not in others (attempts to automate the SCSI were abandoned rather quickly).

While on the subject of the Plug and Play, one more objective needs to be mentioned: automatic detection of connected devices. Master devices must have a means to find out exactly what devices are connected to their interface, and must support configuration tables: lists of correspondences between device addresses and their identifiers. Solving the automatic device enumeration and detection issues allows the issue of automatic configuration, static or dynamic, to be discussed. *Static configuration* systems allow powered-down devices to be connected and disconnected; after all configuration changes have been made, these devices need to be reinitialized. This is the case with the PCI and SCSI buses, although special hot plug versions exist for them. *Dynamic configuration* systems allow devices to be connected on the fly, almost not affecting the operation of other devices. The USB, FireWire buses and, of course, the Bluetooth and IrDA wireless interfaces can be configured automatically.

Relations between connected devices can be *peer-to-peer or master/slave* types. In the case of peer-to-peer relations, any device (at least not only one) can attempt to become the master and in fact become it to carry out some transactions. This creates the *arbitration* objective: deciding which device becomes master for the next transaction. In the case of master/slave relations, only one of the devices connected by the interface can become master. The *host-centric* interface means that one central node, called the host, controls all transaction executions. The host role in the system-level interfaces is played by the computer's central part (the core with the closest surrounding); for peripheral level interfaces, the host role is played by a computer with an interface adapter (controller). The PCI, FireWire, and SCSI buses are examples of peer-to-peer interfaces; the USB and ATA are examples of host-centric interfaces.

The *topology* determines the configuration of connections between connected devices: point-to-point, bus, star, mixed, or daisy-chain.

The *point-to-point,* or dedicated, interface is the simplest connection, in which only two devices are present. Each of the devices knows that only one device can be on the opposite end and no addressing problem arises.

If relations in this connection are of the master/slave type, then the arbitration issue does not exist either. A point-to-point interface with a slave device will have the simplest protocol. One example of such interface is the standard mode LPT port, to which only one device (printer) is connected.

Three possible exchange modes are distinguished for the point-to-point interface:

- The *duplex* mode allows information to be sent over one channel in both directions simultaneously. It may be asymmetrical, if the direct and reverse bandwidth are significantly different, or symmetrical. On the subject of the bandwidth, for duplex connections it often means the summary bandwidth of both the direct and reverse channels.
- The *half-duplex* mode allows information to be sent in the direct and reverse directions in turns; the interface is equipped with a means to switch the channel transfer direction.
- The *simplex* (unidirectional) mode provides only one data transfer direction (in the opposite direction, only auxiliary signals of the interface can be sent).

Bus type topology means connecting several devices to one bus. Bus in the given context means the collection of the signal lines connecting several devices. All devices use the same bus line to exchange information; they can all hear each other simultaneously. For an exchange transaction to be successful, only one device on the bus can be the master (initiator) at this moment; if more than one device on the bus is trying to become the master, than the arbitration protocol and mechanism—granting bus control rights—must also be implemented. The arbitration can be centralized, with one dedicated arbiter bus node doing it, or distributed, when the function is performed by all potential masters. Arbitration can be simple or prioritized, with various mechanisms to control node priority. Examples of buses with centralized arbitration are PCI and ISA; SCSI and FireWire employ distributed arbitration. Bus type topology is inexpensive and simple to implement, which is especially noticeable in parallel interfaces with a large number of signal lines. However, this is achieved at the expense of a more complex protocol and greater vulnerability, because the failure of one device (or a communications line) can paralyze the entire bus.

Star topology is a way to connect more than two devices in which each peripheral device is connected to the central device by its set of interface lines. As a rule, the central node is also the only master device, so the arbitration issue is excluded from the protocol. The addressing issue is solved exclusively by means of the central node (star topology can be considered as a collection of point-to-point connections), which also simplifies the protocol. Besides the simple protocol, star topology possesses a series of other benefits: high survivability (the failure of one of the peripheral devices or its communication lines does not affect the rest); the longest communication range (matching devices with the lines is easier to accomplish in point-to-point connections, in which the loads on the transmitters are minimal). However, when the number of signal lines is large, star interfaces turn out to be too expensive, so the star topology is mostly used for serial interfaces. An example of star topology is the new Serial ATA interface.

Daisy-chain topology connects one device to another in a chain in which each device has a pair of connectors and relays the interface signals from one connector to the other. Physical daisy-chain topology can have different logical organization. For example, in case of the SCSI, the chain provides bus connection but with the number of clients not set in advance (i.e., all the nodes are connected to one set of the signal lines at the same time). Another case is connecting a chain of devices to the LPT port; for example, a scanner, followed by an external hard disk drive, and finally a regular printer. In this case, the internal logic of the interface part of devices in the middle of the chain (the IEEE 1284.3 standard) relays signals from one connector to the other selectively, under the control of the special device selection protocol. As a result of the execution of this protocol, a point-to-point connection is established between the host and one of the chained devices, and the subsequent transfer is conducted as if there were only one device connected to the computer's port.

Some interfaces have hybrid topology; for example, in the PCI bus most of the signal lines have bus organization, but some signals are routed to each device (slot) radially from the central device (the bridge). The USB uses tree-like physical (external) topology (connection of stars with a USB hub at the center of each star), but logical hubs connect all devices in bus topology with the only master: the host controller.

Transfer Validity and Reliability and Flow Control

Data transfer integrity control involves detecting and sometimes correcting errors that occur during transfers. Far from all interfaces possess this control: In some of them, data integrity is not important, while in others the probability of errors is negligibly small. In new interfaces, transfer integrity is given serious attention as they, as a rule, are intended to work in extreme conditions (high frequencies, long distances, and interference).

A *parity check* is the simplest way to detect errors. In it, a parity bit is added to each transmitted information element (as a rule, a byte or a word) that supplements the number of one-value information bits to even (even parity) or odd (odd parity). The receiver checks the number of one-value bits, including the parity bit, for parity (odd or even, depending on what was agreed) and if it detects that the received parity does not match the parity that was sent, it considers the received data erroneous. It must inform the transmitting device about this in order to attempt to retransmit information that is distorted. A parity check is the most primitive and least effective way to control data integrity; with a noticeable overhead (usually one bit for each byte), this check does not catch all even parity errors (distortion of the even number of bits). Parity checking is employed in serial interfaces (COM port) and the SCSI bus; it used also to be employed for memory.

Information doubling is a more wasteful (but also more reliable) control method employed when small amounts of information are transferred. In it, each information element (usually a several-bit long bit field) is repeated twice; moreover, one of the copies can be sent inverted. If the two received copies differ, the reception is considered an error. This method is used to protect identifier packets in the USB interface. A development of this idea is repeating the block three times: If two of the three received copies match, they are considered the right ones (this can also be considered an error-correction method). This method is employed in the Bluetooth radio interface.

A more complex, but also more efficient, control version is calculating Cyclic Redundant Code (CRC) and adding it to the transmitted information. For example, using 16-bit CRC, errors in data blocks up to 4 KB can be detected with very high probability. CRC is practical to calculate with serial data transmission: for this, only simple hardware circuitry (feedback shift registers) is needed. Calculating CRC with parallel transmission (and by software utilizing the CPU) is resource-intensive. Nevertheless, CRC control is used in the parallel IDE/ATA interface, but only in the Ultra DMA mode (other transmission modes of this bus have no error control of any kind).

To correct transmission errors, Error Correction Codes (ECC) are employed. Here, the idea consists of calculating several control bits, each of which is calculated under the parity check rules for certain information group bits. Special division into groups (that partially overlap) makes use of the received information and control bits to detect and even correct errors. The number of control bits depends on the number of the information bits and the desired ratio of the corrected and detected errors. For example, to correct a single occurrence and to detect all double occurrence errors (and most errors of higher occurrence), four control bits are needed for every eight information bits, five for 16, six for 32, and seven for 64, respectively. The ECC control is widely used in memory operations (especially for cache) and in some interfaces (such as PCI-X).

Ensuring *transfer reliability* means informing the transaction's initiator whether or not it has been successfully executed, which in case of failure allows the initiator to undertake some corrective measures (an attempt to retransmit the data, for example). Some interfaces (and protocols) do not provide transfer reliability: For example, in the ISA bus even a non-existent device can be accessed. In this case, write operations are performed into nothing, while read operations usually return empty data (FFh), which the initiator cannot distinguish from real data. The PCI bus is reliable: the initiator always knows what has happened to its transactions; data transfer integrity is controlled (by parity check or ECC methods).

The issue of coordinating the work pace of devices connected by an interface is solved with the help of handshaking and/or flow control mechanisms.

Handshaking is mutual acknowledgement of individual protocol steps by both participants of the transaction, which makes it possible to match up the work pace of the initiator and target devices. Handshaking is widely employed in parallel interfaces (in the LPT port, expansion buses), where special interface lines are used for this purpose.

Flow control is the receiver's informing the data source (transmitter) of its capability to receive data: If the receiver cannot keep up servicing the incoming data, it requests the transmitter to suspend the transmission for a certain period of time or until special notification.

Interfaces that employ handshaking, as a rule, do not require a separate flow control mechanism (handshaking also takes care of the pace matching). In general, serial interfaces require some form of flow control; the COM port even has two versions of a flow control protocol.

Timing and Synchronization Concepts

In terms of requirements of timing and speed, data transfer transactions can be asynchronous, synchronous, and isochronous. It must be noted right away that there is no rigid correlation between the interface type and the type of transfers it conducts.

In *asynchronous* data transfers and interfaces, the participants do not have any particular responsibilities to each other in terms of timing: the initiator can begin a transaction at any time, while the target device, as a rule, can suspend it in case it is not ready to continue servicing it. Asynchronous transmission can be employed with all devices not working in real time: printers, scanners, data storage devices, etc.

In *synchronous interfaces,* the transaction parties are rigidly connected in terms of timing. They have a constant synchronization clocking signal to which all interface events are tied: bit transfer in serial and byte (word) transfer in parallel interfaces. As a rule, the clocking signal is of a precisely sustained and constant frequency, although there are some interfaces in which the clock signal period varies (the SPI, for example). Both the transmitter and the receiver have the synchronization signal; either a separate interface line is used to transmit the clocking signal, or it is packed into the common signal together with the transmitted data with the help of self-synchronizing codes. A separate clocking line makes an interface more expensive; moreover, at high speeds the old skew problem, limiting the parallel interface speed, appears. Modern serial interfaces (USB, FireWire), as in data transfer networks, use various synchronization coding methods, which makes it possible to achieve high speeds at long distances. Synchronous interfaces allow both synchronous and asynchronous data transfers; asynchronous interfaces cannot be used for synchronous transfers.

Synchronous data transfer is a *constant instantaneous speed* transfer. This type of transfer is needed, for example, for multimedia data transfers, in particular for transmitting digitized sound using pulse-code modulation (PCM): transmitting signal samplings at equal time intervals. In telephony, 8-bit samplings are transmitted at 8 KHz (overall speed 64 Kbps), while for transmitting high fidelity CD quality audio recording, 16-bit samplings are transmitted at 44.1 KHz over each stereo channel (overall speed 1.4 Mbps). Loss of synchronization causes data loss: distortion, interference, and temporary loss of sound altogether. A separate dedicated synchronous interface (or complex multiplexing schemes) is needed for each connected device for synchronous data transfers.

Isochronous data transfer transmits data with *constant average speed:* over a certain (fixed) time interval, a certain amount of data must be transferred, but the instantaneous data transfer speed is not specified. Of course, the instantaneous speed must be at least not lower than the average speed. Usually, an instantaneous speed (the throughput capacity or bandwidth of the interface) much higher than the necessary average speed is selected. This makes it possible to use one interface for connecting multiple devices and organize multiple concurrent isochronous transfer channels (their total speed will be somewhat lower than the throughput capacity (bandwidth) of the interface). Isochronous and asynchronous transfers can easily be conducted over the same interface. The issue of the interface bandwidth allocation is handled by the *isochronous resource controller* a separate software support function.

Isochronous transfers are used by multimedia devices, such as audio and video equipment. Devices are equipped with buffer memory, into which the incoming isochronous transfer packets are stored; these data are used up (to reproduce sound, for example) by the device at constant instantaneous speed (the reverse transfer is done the same way). Isochronous transfers also are suitable for use in variable-speed multimedia applications (when data are compressed, their incoming speed may vary, but, of course, its upper limit is known). Isochronous transfers are supported by the USB, FireWire, Bluetooth, and the new interfaces AGP 3.0 and PCI Express. Regular data transfer networks with high enough speeds can also carry isochronous traffic. The ideal synchronization of isochronous devices has its peculiarities: Devices have to use their own (very precise) clock generators, because there is no direct synchronization between them (as in synchronous interfaces). One method used for solving the synchronization issue is feedback mechanisms, which make it possible for the devices to correct their clock drifts.

[1]One of the ternary digit names.

[1]The 10-gigabyte Ethernet mode has a parallel/serial optical version.

Peripheral Equipment Connection Evolution

Having considered the general issues of interface organization, it is time for a brief survey of the evolution of peripheral equipment.

System-Level Interfaces

A system-level interface is an interface that provides its clients with direct access to the computer's system resources: addressed memory and I/O spaces, as well as hardware interrupts. Clients of this interface type—controllers of proper peripheral devices, adapters and controllers of peripheral device interfaces—by the nature of their activity must have their I/O register and their local memory (if they have them) mapped onto the common memory and I/O space. This is required in order to provide efficient (high-speed and high-performance) software interaction with these devices by the central processor (CPU). The overall efficiency and productivity of the computer can be increased by relieving the CPU of routine I/O operations, which is achieved by the following two complementary approaches:

- Use specialized I/O coprocessors
- Allow controllers of the peripheral devices (or of their interfaces) to access system resources themselves

An *Input-Output Processor* is a processor with a reduced command set customized to perform I/O control operations. This processor can both work with the common address space or/and have its separate address space for the I/O subsystem it controls. The standard direct memory access (DMA) controller employed for the ISA bus and its successors' clients (of which, only COM and LPT ports and primitive audio adapters built into the motherboards have survived) can be considered, by a great stretch of imagination, to be a primitive processor of this type. Usage of I/O processors is also meant by the I2O (Intelligent Input-Output) abbreviation when talking about powerful computer server platforms.

Ordinary computers usually content themselves with *bus mastering*, which allows peripheral device controllers (or interface controllers) themselves to access the system resources, most often the main memory areas, when conducting data or control information exchanges. The peripheral device controller must temporarily assume the role of the exchange initiator on the interface that links the device with the computer core (mostly, with memory). Because traditionally this interface is of the bus type, such an active controller is called a bus master even if it connects to a dedicated two-point interface (such as AGP). The simplest bus

master only exchanges data with memory in the direct access mode, although it makes the driver's job easier and more efficient than a standard DMA controller. For example, all modern IDE (ATA/ATAPI) controllers can do scatter write and gather read operations, which makes it possible to place a continuous data block into different physical memory pages. This in turn makes it possible to coordinate physical memory allocation used by the controller with logical memory allocation used by the CPU software that works in the virtual address space. More complex versions of memory interaction are used, for example, in communications controllers, such as LAN adapters and USB controllers. Here, the driver (a CPU program) forms control data structures in the memory that define the controller's job assignments (including the data location description that can be scattered over different pages). The controller works following these assignments, recording the execution results in these structures. Later, the driver relays the results of the work done to the concerned programs.

The history of system-level IBM PC interfaces began with the ISA bus, which was the central connecting interface in the IBM PC/XT: the CPU, the main memory, and the I/O devices all connected to this bus. Because of its central role in the PC/XT, the ISA bus was (and still mistakenly is) called a system bus[1]. With the IBM PC/AT, the ISA bus firmly took its present place in the bus hierarchy as an I/O expansion bus. The ISA bus provides an asynchronous parallel interface for connecting all kinds of peripheral adapters and controllers. Dedicated address, data, and control signal lines provide a very simple and inexpensive to implement device connection protocol. In the PC/XT, the ISA data bus was 8 bits wide and could address only 1 MB of memory with 20 address lines. With the aim of making the connected peripheral devices cheaper, only 10 bits out of the total available 16 I/O space address bits were actually used. The ISA bus has come to the present day in the 16-bit data and 24-bit address version. It now has bus-mastering capability, although only for no more than three devices. The issue of automatic configuration of ISA devices was acceptably solved with a more than 10-year delay when the ISA PnP specification came out in 1994.

The EISA bus was founded on more progressive ideas: It was a 32-bit synchronous bus supporting packet transfers, which allowed higher bandwidth. Moreover, the basis for automatic device configuration—the possibility to selectively access bus slots externally to the regular addressing and to define the standard configuration description structures (used addresses, interrupt requests, DMA channels, etc.)—was initially made in the EISA. A device in any EISA slot can request and receive direct bus control (become a bus master). The bus protocol provides fail-safe transfers (transactions into nothing are impossible in EISA). With all its progressive features, the EISA bus took root only in the powerful servers of its time: Both the motherboard and devices for EISA turned out to be too expensive. The EISA bus can be considered a second-generation I/O expansion bus.

There was also the Micro-Channel Architecture (MCA) bus with characteristics similar to EISA, but it passed into oblivion along with the poignantly famous IBM PS/2 computer series (of which only the PS/2 mouse and the small keyboard connector have been left). The VESA Local Bus (VLB), which disappeared almost as soon as it appeared, was only the system bus of the x486 processors routed to peripheral device slots: It was doomed by the low stability of this type of connection and the switch to Pentium and higher processors, which have entirely different interfaces. The idea of the VLB lay in bringing certain controllers (graphics and disk drive) closer to the memory and the processor in order to increase the throughput.

Connecting peripheral devices needs a bus with a long life span, because by nature, peripheral devices are more conservative than processors, which swiftly replace one another. Moreover, it must provide high bandwidth and bus mastering by an unlimited number of connected devices. The PCI bus with its modern versions has become such a long-life bus.

The main features of the PCI bus are as follows:

- *Parallel interfaces with a protocol that provide reliable exchange.* Address and data bus width is 32 bits with possible expansion to 64 bits. Data validity is achieved by parity checking, while the transfer reliability is provided by a mandatory acknowledgment from the target device, with the result that the transaction initiator always knows its outcome.

- *Synchronous interface and high-speed packet transfers.* 33 MHz at 32-bit data width allows bandwidth of 133 MBps to be achieved. The frequency can be raised to 66 MHz, which at 64-bit data width gives peak bandwidth of up to 532 MBps.
- *Automatic device configuration built into the bus specification.* The set of standard functions allows information about devices' needs in system resources (address space and interrupts) to be gathered as well as to control devices (enabling them as either transaction initiator or target devices).
- A *command set* to access resources of different address spaces, including the commands for optimized access to an entire cache memory line.

The declared high bus bandwidth is almost reached only by active bus devices (Bus Masters), as only they are capable of generating long packet transactions. Input/output over the PCI bus performed by the CPU gives much more modest real bandwidth values.

A distinctive feature of the PCI bus is that it can handle only a limited number of devices (no more than five or six) connected to the bus (earlier buses did not have such a rigid limitation). To connect more devices, *PCI bridges* are used, which form additional buses. However, the buffering mechanisms used in PCI bridges insert significant delays into transaction execution time across a bridge.

The PCI protocol's weak spot is read operations, especially if they are addressed to not very fast memory. These operations can tie the bus up for a long time by unproductive waiting for data.

The traditional way of signaling PCI interrupts also draws complaints: Only four interrupt lines for all devices on all buses condemn them to shared usage. Although there are no electrical contraindications for sharing interrupt lines in the PCI (as there are in the ISA), the absence of a unified interrupt request indicator makes identification of the device requesting interrupt difficult and response to this interrupt slow. This standardized interrupt request indicator appeared only in the PCI 2.3 after many years of active bus use. Although, starting from version 2.2, a new interrupt notification mechanism, the MSI, has been used, which takes care of all problems of the traditional notification.

The PCI specification was the starting point for creating a dedicated graphics accelerator interface, the *Accelerated Graphics Port* (AGP). This is a synchronous 32-bit parallel point-to-point interface. At a clock rate of 66 MHz, it can reach speeds of 264 MBps (mode 1x), 533 MBps (mode 2x, double synchronization), 1,066 MBps (4x, synchronization from data source), and in the latest version, AGP8x, speeds of 2,132 MBps are achieved. In addition to raising the peak speed, the AGP employs a more sophisticated protocol that allows memory request queuing. Thank to this feature, the bus does not idle while waiting for data, producing a positive effect on the efficiency. Another distinctive feature of AGP is hardware memory address translation, which makes it possible to coordinate the physical memory model that the accelerator employs with the virtual memory model, with which the graphics software of the central processor works. The AGP is a system-level specialized interface: Only the graphics accelerator (video card) is connected to it.

The PCI bus' use as a general-purpose bus was developed in the PCI-X standard, whose second version has already come out. In this standard, the peak speed has also been raised: The bus clock rate can be raised up to 133 MHz (PCI-X66, PC-X100, and PCI-X133). A new mode, Mode 2, appeared in version 2.0 with synchronization from the data source; memory write (and this operation only) speeds can increase two- or four-fold (PCI-X266 and PCI-X533, respectively). Consequently, the peak memory write speed in the 64-bit version of the bus can reach 4 GBps (reading speeds reach up to 1 GBps). Version 2.0 of the interface also defines a 16-bit version of the bus.

Bus protocol was changed in PCI-X: An additional transaction attribute transmission stage was added, in which the exchange initiator makes its identifier available (in PCI, the target device has no idea which device requests the transaction). Moreover, the number of bytes to be transmitted is also declared, allowing the exchange participants to plan their actions. The main change in the protocol is the split transaction capability: If the target device cannot respond quickly, it forces the initiator to release the bus and organizes data delivery later itself. Because of this, bus usage efficiency is significantly raised: bus idling while waiting for data is

eliminated. Splitting transaction also raises efficiency of bridge operations: Their transaction costs become less wasteful.

A new way of interdevice communications was also introduced in PCI-X: Messages are addressed to device *identifiers* (bus, device, and function numbers) and not to resource (memory or I/O) addresses. In the previous versions of the bus, this capability was absent; this communication method helps to raise the intelligence level of the I/O subsystem.

Of interest is the fact that, with all its innovations, PCI-X remains compatible with PCI devices and buses: Each bus will work in the mode that the weakest participant can support, from the 33 MHz PCI to the PCI-X533. Bridges allow different system segments (PCI buses) to work in different modes, so it is possible to arrive to an effective configuration, in which new bandwidth-critical devices will get along with no problem with old devices.

Presently, third generation technology I/O interfaces, such as PCI Express and Hyper Transport, are beginning to be employed. A transition from bus to point-to-point serial link types is characteristic of third generation interfaces. As a matter of fact, third generation interfaces approach the characteristics of local networks (within the confines of the motherboard). Here, packets containing both the control information (commands) and the data proper are transferred between nodes. Each interface is made up of a pair of oncoming one-way links working independently of each other (no handshaking of any kind is provided for in transfers). Bandwidths of one interface can be different for different directions: a bandwidth is chosen taking into account the requirements of the given application. New capabilities appeared in the I/O architecture, including control over the quality of service (QoS), power consumption, and link resource budgets. Node connection topologies and link properties are different, but both the Hyper Transport and PCI Express, just like the previous generation buses, do not allow loops in the topology.

HyperTransport is a point-to-point connection between microchips within the card limits (connectors and expansion cards are not provided for). The topology's main version is a chain of device tunnels, each with two interfaces and transferring all packets through. Bridges and switches can also be employed for packet routing. Links can be 2, 4, 4, 16, or 32 bits wide; a clock rate of 200 MHz-800 MHz at double-edge latching provides a peak data transfer rate of up to 6.4 GBps (32-bit link). Because packets can be sent concurrently in both directions, the interface's bandwidth can be said to be 12.8 GBps. The common configuration principles and supported transaction types make PCI to PCI-X integration easy by using bridges.

PCI Express, or 3-Generation Input-Output (3GIO), is a point-to-point peripheral component interconnect on motherboards as well as on expansion cards. The 1-, 2-, 4-, 8-, 12-, 16-, or 32-bit interface provides transfer speeds of up to 8 GBps. It preserves the software features on the PCI bus (configuration, transaction types), which makes for a smooth migration from the PCI to PCI Express.

InfiniBand is a new server interconnect architecture based on the idea of constructing a centralized I/O fabric. InfiniBand nodes are connected with the help of the switching structure that allows redundant links. Link redundancy is welcomed: It allows the total throughput capacity to be raised. Redundancy is provided by both the internal elements of the switching structure and the presence of several interfaces and devices that allow connections to multiple points of the switching structure fabric. There are three levels on link throughput capacities in the InfiniBand: 0.25, 1, and 3 GBps. Each InfiniBand Link consists of one, four, or 12 oncoming one-way lanes working in half-duplex mode at transmission speeds of 2.5 Gbps each.

Links are made employing both intercard connectors and cable connections: copper up to 17 meters, and optical up 10 kilometers. In copper links, each lane is a differential signal wire pair; in optical links, each lane is a single fiber. The architecture allows up to tens of thousands of nodes to be linked into one subnet. Depending on the function, nodes are equipped with one of the two types of InfiniBand adapters: Host Channel Adapter (HCA) for the transaction initiating processor nodes, and Terminal Channel Adapter (TCA) for I/O nodes (storage device interface and network controllers, graphics controllers, I/O racks with traditional expansion buses, etc.).

Peripheral Interfaces

The traditional approach to organizing peripheral device connection consists of introducing a separate individualized interface for each type of device (for a group of similar type devices in the best case). This interface's controller (adapter) is connected to the I/O expansion bus and requires its resources: address ranges in the memory and I/O spaces, interrupt request lines, centralized DMA channels. Consequently, a computer with an extensive collection of peripheral devices has a great number of specific peripheral interface controllers that are not connected among themselves. Specific interfaces were introduced historically, but they often lost their technical justification as time went by.

The necessity for a special *monitor interface* is hard to argue against: Probably, no other device needs such a huge information stream to be output to it. However, tying the interface just to the CRT monitor turned out to be inconvenient for matrix LCD displays, and it began to change with time.

For the first *mass storage devices* (diskette drives), the individual interface allowed the actual peripheral device to be simplified to the highest degree. Inexpensive streamers also used this interface, but its capabilities are very limited. Even related devices—hard disk drives—required a new, albeit similar, interface to be introduced. Binding an interface to a specific type of hard disk drive caused the problem of controller/hard disk drive compatibility. Shortly afterward, the development of the mass storage devices required the utmost physical proximity of drive to its controller. For the sake of software compatibility, the traditional hard disk drive controller was placed onto the drive itself: The Integrated Drive Electronics (IDE) interface came into existence. A section of the ISA bus was used as the connection interface, with a ribbon cable connecting the controller to it via buffered transceivers. In this way, the specific ATA interface was born: still the most inexpensive and popular internal mass storage device interface, and not only of the disk drive type. It has been developing mainly in terms of increasing the data transfer speed: up to 133 MBps in the parallel version, and 150 MBps and up in the serial version (Serial ATA). Storage devices with an integrated controller do not need a specific interface: Only a fast means of data transport between the main memory and the device is needed.

The LPT port appeared as an adapter for the specialized Centronics Data Corporation *printer interface:* perhaps, the first affordable personal printer. The desire to connect different devices to the hardware on hand led to a chain of LPT port modernizations that culminated in adopting a standard for quite unified and symmetrical interface: the IEEE 1284, to which several devices can now be connected.

Connecting a *keyboard* (and PS/2 mouse) was done in an amazingly original way: To organize a serial bidirectional interface when a too high data transfer speed was not required, a separate microcontroller was installed on the PC/AT motherboard. The resulting interface was not compatible with any other standard interface and is held sacred to this time.

The *COM port* with its adapter was the first external PC interface not oriented at any specific device from the start: The subset of the RS-232C interface implemented in the COM port is used not only in telecommunications (where it originated), but also in many other peripheral devices. However, this interface is too slow, and only one device can be connected to it.

The first high efficiency universal shared peripheral interface in PCs was the *SCSI bus,* which was taken from mini-computers. Nominally, devices of the most varied type can be equipped with the SCSI bus: printers, scanners, mass storage devices, communications equipment, processing devices, etc. In practice, the SCSI is mainly used to connect mass storage devices and scanners: In other areas, it has more inexpensive (perhaps even specialized) alternatives. The SCSI bus has come a long way, from a slow 8-bit bus to the high-speed 16-bit Ultra-320 bus, and now the time of its serial version has arrived.

The destiny of the *Universal Serial Bus* (USB) was to revolutionize peripheral interconnections. The idea of its introduction lies in creating a practical, inexpensive, universal, efficient interface for connecting a large number of peripheral devices; the interface has a built-in hot swap and autoconfiguration support. Moreover, only one controller needs to transport data among all devices and the main memory buffers. This eliminates

configuration problems when connecting multiple devices, it also means a multitude of controllers and adapters. A USB controller is a system to mass service exchanges among multiple devices. One-point service makes it possible to coordinate the pace of servicing different devices and, as far as possible, evenly to distribute the total bandwidth, leaving a guaranteed band for isochronous transfers. This task of mutual coordination is practically impossible to accomplish using isolated controllers. However, the totalitarian authority of the USB controller requires that it be sufficiently nimble. This was only partially achieved in the first version (USB 1.0): The throughput capacity that was to be shared out was too low (12 Mbps is the line speed, out of which 12:8 = 1.5 MBps cannot be used because of the overhead expenses). For example, a USB 1.0 printer works slower than an LPT printer. However, the connection convenience is unarguable, and a great many USB devices have appeared. The second version (USB 2.0) with its 480 Mbps inspires more optimism.

Long before the USB appeared, in the "parallel world" of Macintosh computers, the no less universal serial FireWire bus, which was capable of carrying video data stream and used even for data storage devices right from the start, had taken root. Now, the FireWire (IEEE 1394) has also taken firm residence in PCs, supplementing the set of the external interfaces. The functional capabilities, practicality, and bandwidths of the FireWire and USB 2.0 are quite close, but supporting both interfaces increases the range of equipment that can be connected.

Despite the USB and FireWire's strong positions, there are still a multitude of various interfaces. Take, for example, the swarm of flash memory device connection interfaces: CompactFlash, SmartMedia Card, MultiMedia Card, Secure Digital, Miniature Card, Memory Stick, etc. Each has its connector and individual protocol. A simple microcontroller with a USB interface makes it possible to bring them all to the common denominator, and to exchange data with PCs.

Interface Selection

When developing custom devices, the issue of choosing the appropriate connection interface arises. This issue should be decided based on the principle of reasonable sufficiency, giving preference to external interfaces as far as possible. It should be remembered that hardware development is closely connected with software support of the device: both the programs executed by the processor (software) and the programs executed by the embedded microcontroller (firmware), on the basis of which, as a rule, modern devices are built. There is a multitude of microcontrollers available with popular interfaces such as USB, RS-232, I2C, and others. However, in some cases, standard I/O expansion buses also have to be used.

These buses present wider possibilities for processor/equipment interaction, and are not constrained by the rigid limitations of the external interfaces. However, the versatility and efficiency of the internal expansion buses extracts the price of more intricate interface circuitry implementation and difficulties in providing compatibility with the other equipment installed in the computer. Mistakes here can cause computer malfunctions (it is luck if only temporarily). It is not without reason that respectable computer manufacturers guarantee their equipment's operability only if certified (by them or by independent laboratories) expansion cards are used with it. When external interfaces are used, troubles in case of errors most often only involve the connected device.

A significant issue to consider when selecting an interface is two aspects of hot swap capability: First of all, being able to connect/disconnect devices on the fly, without fear of damages to the devices themselves and their interface circuits, to the stored and transferred data, and, finally, to the user; secondly, being able to use the newly connected device without having to reload the system, as well as to be able to continue stable operation after disconnecting a device. Far from all external interfaces support the hot swap to the full extent; for example, a scanner with a SCSI interface must be connected to the computer and turned on prior to loading the operating system, otherwise, it will not be available to the system. Hot swap is not regularly used for internal interfaces: expansion buses, memory modules, and even most ATA and SCSI hard drives. Hot swap is supported by industrial computer expansion buses, and also in special array constructions of storage devices.