In this chapter, we give an overview of selected aspects of computer input–output (I/O) and communication between the CPU and external I/O devices. Because of the wide variety of different I/O devices and the quest for faster handling of programs and data, I/O is one of the most complex areas of computer design. As a consequence, we are able to present only selected pieces of the I/O puzzle. We illustrate in detail just three devices: a keyboard, a hard drive, and an LCD screen. We then introduce the I/O bus and the I/O interfaces that connect to I/O devices. We look at the Universal Serial Bus (USB), one of many solutions to the problem of accessing I/O devices. Finally, we discuss three modes for performing data transfers: program-controlled transfer, interrupt-initiated transfer, and direct memory access.

In terms of the generic computer at the beginning of Chapter 1, it is apparent that I/O involves a very large part of the computer. Only the processor, external cache, and RAM are not as highly involved, although they, too, are used extensively in directing and performing I/O transfers. Even the generic computer, which has fewer I/O devices than most PC systems, has a diverse set of such devices requiring significant digital electronic hardware for support.

12-1 COMPUTER I/O

The input and output subsystem of a computer provides an efficient mode of communication between the CPU and the outside environment. Programs and data must be entered into the memory for processing, and results obtained from computations must be recorded or displayed. Among the input and output devices commonly found in computer systems are keyboards, displays, printers, magnetic drives, and compact disk read-only memory (CD-ROM) drives. Other input and
output devices frequently encountered are modems or other communication interfaces, scanners, and sound cards with speakers and microphones. Significant numbers of computers, such as those used in automobiles, have analog-to-digital converters, digital-to-analog converters, and other data-acquisition and control components.

The I/O facility of a computer is a function of its intended application. This results in a wide diversity of attached devices and corresponding differences in the needs for interacting with them. Since each device behaves differently, it would be time consuming to dwell on the detailed interconnections needed between the computer and each peripheral. We will, therefore, examine just three peripherals that appear in most computers. In addition, we present some of the common characteristics found in the I/O subsystem of computers, as well as the various techniques available for transferring data either in parallel, using many conducting paths, or serially, through communication lines.

12-2 Sample Peripherals

Devices that the CPU controls directly are said to be connected online. These devices communicate directly with the CPU or transfer binary information into or out of the memory upon command from the CPU. Input or output devices attached to the computer online are called peripherals. In this section, we examine three peripheral devices: a keyboard, a hard drive, and a graphics display. We also use the keyboard as an example to illustrate I/O concepts in a later section. We introduce the hard drive both to motivate the need for direct memory access and to provide background for the role of the device in Chapter 13 as a component in a memory hierarchy. We include the graphics display to illustrate the very high potential transfer-rate requirements of contemporary applications.

Keyboard

The keyboard is among the simplest of the electromechanical devices attached to the typical computer. Since it is manually controlled, it has one of the slowest data rates of any peripheral.

The keyboard consists of a collection of keys that can be depressed by the user. It is necessary to detect which of the keys have been depressed. To do this, a scan matrix that lies beneath the keys is used, as shown in Figure 12-1. This two-dimensional matrix is conceptually similar to the matrix used in RAM. The matrix shown in the figure is \(8 \times 16\), giving 128 intersections, so it can handle up to 128 keys. A decoder drives the \(X\) lines of the matrix, which are analogous to the word lines of a RAM. A multiplexer is attached to the \(Y\) lines of the matrix, which are analogous to the bit lines of a RAM. The decoder and the multiplexer are controlled by a microcontroller, a tiny computer that contains RAM, ROM, a timer, and simple I/O interfaces.

The microcontroller is programmed to periodically scan all intersections in the matrix by manipulating the control inputs of the decoder and multiplexer. If
the key is depressed at an intersection, a signal path is closed from an output of the \(X\) decoder to an input of the \(Y\) multiplexer. The existence of this path is sensed at an input to the microcontroller. The 7-bit control code applied to the decoder and multiplexer at the time identifies the key. To allow for "rollover" in typing, in which multiple keys are depressed before any of them is released, the microcontroller actually identifies the depressing and release of the keys. Whether a key is depressed or released, the control code at the time of the event is sensed and is translated by the microcontroller into a \(K\)-scan code. When a key is depressed, a \textit{make code} is produced; when a key is released, a \textit{break code} is produced. Thus, there are two codes for each key, one for when the key is depressed and one for when it is released. Note that the scanning of the entire keyboard occurs hundreds of times per second, so there is no danger of missing any depression or release of a key.

After presenting a number of I/O interface concepts, we will revisit the keyboard to see what happens to the \(K\)-scan codes before they are finally translated to ASCII characters.

\textbf{Hard Drive}

The hard drive is the primary intermediate-speed, nonvolatile, writable storage medium for most computers. The typical hard drive stores information serially on a nonremovable disk, as shown in the upper right of the generic computer at the beginning of Chapter 1. Each platter is magnetizable on one or both surfaces. There are one or more read/write \textit{heads} per recording surface. Each disk is divided into concentric \textit{tracks}, as illustrated in Figure 12-2. The set of tracks that are at the same distance from the center of all disk surfaces is referred to as a \textit{cylinder}. Each track is divided into \textit{sectors} containing a fixed number of bytes. The number of bytes per sector typically ranges from 256 to 4K. The typical byte address includes the cylinder number, head number, sector number, and word offset within the sector. The addressing assumes that the number of sectors per track is fixed. In modern, high-capacity drives, more sectors are included in the longer outer tracks than in the shorter inner tracks. In addition, a number of spare sectors are reserved to take the
any interactive output device. The display is shown as a simplified schematic. Two parallel laser beams are shown to create a moving focus on the liquid crystal, which is used to show information on the screen. The liquid crystal is shown in two layers, each of which is rotated by a different amount. The pixels are shown as being on or off, depending on the orientation of the liquid crystal. The liquid crystal is shown as being in two layers, each of which is rotated by a different amount.

The image shows a simplified schematic of a liquid crystal display. The liquid crystal is shown as being in two layers, each of which is rotated by a different amount. The pixels are shown as being on or off, depending on the orientation of the liquid crystal. The liquid crystal is shown as being in two layers, each of which is rotated by a different amount.
pixel on the surface of the rear substrate facing the liquid crystal. The transistor, conductors and so on, are separated from the liquid crystal by coating layers including the final one with the fine grooves in it.

In terms of operation, the circuitry behaves much like a DRAM. To write the lower row of elements, the voltage values to be applied are placed on Data Lines \( m, m + 1, m + 2 \), and so on, a high voltage is placed on Gate Line \( n + 3 \), and 0 volts is placed on all other Gate Lines. The voltage values are placed on the storage capacitor \( C \) and on the upper surface of the subpixel. For technical reasons, the applied voltages are inverted each time a row is written. When Gate Line 3 is returned to 0 volts, the transistor turns off and the voltage is stored on capacitance \( C \). The rows of the LCD are successively written one at a time, with a full panel write taking less than a sixtieth of a second.

The inputs to the Data Lines and Gate Lines are provided by the driver circuitry for the LCD panel. In addition, there is a display controller that may be combined with the driver circuitry. The display driver may be driven by digital inputs or analog RGB inputs as used for the older cathode ray tube display technology.

### I/O Transfer Rates

An indicated earlier, the three peripheral devices discussed in this section give a sense of the range of peak I/O transfer rates. The keyboard data transfer rate is less than 10 bytes/s. For the hard drive, while the drive controller is capturing the data arriving rapidly from the disc in the sector buffer, the transfer of data from the buffer to main memory is impossible. Thus, in the case in which the next sector is to be read immediately, all of the data from the sector buffer needs to be stored in main memory during the time the gap on the disc between the sectors passes under the disc head. For 63 sectors and a rotational speed of 5400 rpm, this time is about 25 \( \mu \)s. Thus, the peak transfer rate required is 512 B/25 \( \mu \)s = 20 MB/s. For a 1280 \( \times \) 800 display with 256 colors, if a screen is to be changed entirely every sixtieth of a second, 3 MB of data must be delivered to the video RAM from the CPU in that amount of time. This requires a data rate of 3 MB \( \times \) 60 = 180 MB/s. Based on the preceding, we can conclude that the peak data rates required by the particular peripherals we have considered have a wide range. The bus system must be designed to handle the highest transfer rates between peripherals and memory.

### 12-3 I/O Interfaces

Peripherals connected to a computer need special communication links to interface them with the CPU. The purpose of these links is to resolve the differences in the properties of the CPU and memory and the properties of each peripheral. The major differences are as follows:

1. Peripherals are often electromechanical devices whose manner of operation is different from that of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data-transfer rate of peripherals is usually different from the clock rate of the CPU. Consequently, a synchronization mechanism may be needed.

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.

4. The operating modes of peripherals differ from each other, and each must be controlled in a way that does not disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and the peripherals to supervise and synchronize all input and output transfers. These components are called interface units, because they interface between the bus from the CPU and the peripheral device. In addition, each device has its own controller to supervise the operations of the particular mechanism of that peripheral. For example, the controller in a printer attached to a computer controls the motion of the paper, the timing of the printing, and the selection of the characters to be printed.

I/O Bus and Interface Unit

A typical communication structure between the CPU and several peripherals is shown in Figure 12-5. Each peripheral has an interface unit associated with it. The common bus from the CPU is attached to all peripheral interfaces. To communicate with a particular device, the CPU places a device address on the address bus. Each interface attached to the common bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals with addresses that do not correspond to the address on the bus ignore the bus activity. At the same time that the address is made available on the address bus, the CPU provides a function code on.
the control lines. The selected interface responds to the function code and proceeds to execute it. If data must be transferred, the interface communicates with both the device and the CPU data bus to synchronize the transfer.

In addition to communicating with the I/O devices, the CPU of a computer must communicate with the memory unit through an address and data bus. There are two ways that external computer buses communicate with memory and I/O. One method uses common data, address, and control buses for both memory and I/O. We have referred to this configuration as memory-mapped I/O. The common address space is shared between the interface units and memory words, each having distinct addresses. Computers that adopt the memory-mapped scheme read and write from interface units as if they were assigned memory addresses by using the same instructions that read from and write to memory.

The second alternative is to share a common address bus and data bus, but use different control lines for memory and I/O. Such computers have separate read and write lines for memory and I/O. To read or write from memory, the CPU activates the memory read or memory write control. To perform input to or output from an interface, the CPU activates the read I/O or write I/O control, using special instructions. In this way, the addresses assigned to memory and I/O interface units are independent from each other and are distinguished by separate control lines. This method is referred to as the isolated I/O configuration.

Example of I/O Interface

A typical I/O interface unit is shown in block diagram form in Figure 12-6. It consists of two data registers called ports, a control register, a status register, a bidirectional data bus, and timing and control circuits. The function of the interface is to translate the signals between the CPU buses and the I/O device and to provide the needed hardware to satisfy the two sets of timing constraints.

The I/O data from the device can be transferred into either port A or port B. The interface may operate with an output device, with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data; if it services a scanner, it will only input data. A hard drive transfers data in both directions, but not at the same time; so the interface needs only one set of I/O bidirectional data lines.

The control register receives control information from the CPU. By loading appropriate bits into this register, the interface and the device can be placed in a variety of operating modes. For example, a printer may be set in a mode that permits cartridges to be changed. The bits in the status register are used for status conditions and for recording errors that may occur during data transfer. For example, a status bit may indicate that port A has received a new data item from the device, while another bit in the status register may indicate that a parity error has occurred during the transfer.

The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select input and the two register select inputs. A circuit (usually a decoder or a gate) detects the address assigned to the interface registers. This circuit sets the chip select (CS)
input when the interface is selected by the address bus. The two register select inputs R51 and R50 are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface, as specified in the table accompanying the diagram in Figure 12-6. The contents of the selected register are transferred into the CPU via the data bus when the I/O read signal is set. The CPU transfers binary information into the selected register via the data bus when the I/O write input is set.

The CPU, interface, and I/O device are likely to have different clocks that are not synchronized with each other. Thus, these units are said to be asynchronous with respect to each other. Asynchronous data transfer between two independent units requires that control signals be transmitted between the units to indicate the time at which data is being transmitted. In the case of CPU-to-interface communication, control signals must also indicate the time at which the address is valid. We will look at two methods for performing this timing: strobing, as it is called, and handshaking. Initially, we will consider generic cases in which no addresses are involved; subsequently, we will add addressing. The communicating units for the generic case will be referred to as the source unit and destination unit.
Strobing

Data transfers using strobing are shown in Figure 12-7. The data bus between the two units is assumed to be made bidirectional by the use of three-state buffers.

The transfer in Figure 12-7(a) is initiated by the destination unit. In the shaded area of the data signal, the data is invalid. Also, a change in Strobe at the tail of each arrow causes a change on the data bus at the head of the arrow. The destination unit changes Strobe from 0 to 1. When the value 1 on Strobe reaches the source unit, the unit responds by placing the data on the data bus. The destination unit expects the data to be available, at worst, a fixed amount of time after Strobe goes to 1. At that time, the destination unit captures the data in a register and changes Strobe from 1 to 0. In response to the 0 value on Strobe, the source unit removes the data from the bus.

The transfer in Figure 12-7(b) is initiated by the source unit. In this case, the source unit places the data on the data bus. After a short time required for the data to settle on the bus, the source unit changes Strobe from 0 to 1. In response to Strobe equal to 1, the destination unit sets up the transfer to one of its registers. The source then changes Strobe from 1 to 0, which triggers the transfer into the register at the destination. Finally, after a short time required to ensure that the register transfer is done, the source removes the data from the data bus, completing the transfer.
Although simple, the strobe method of transferring data has several disadvantages. First, when the source unit initiates the transfer, there is no indication to it that the data was ever captured by the destination unit. It is possible, due to a hardware failure, that the destination unit did not receive the change in Strobe. Second, when the destination unit performs the transfer, there is no indication to it that the source has actually placed the data on the bus. Thus, the destination unit could be reading arbitrary values from the bus rather than actual data. Finally, the speeds at which the various units respond may vary. If there are multiple units, the unit initiating a transfer must wait for the delay of the slowest of the attached communicating units before changing Strobe to 0. Thus, the time taken for every transfer is determined by the slowest unit with which a given unit initiates transfers.

**Handshaking**

The *handshaking* method uses two control signals to deal with the timing of transfers. In addition to the signal from the unit initiating the transfer, there is a second control signal from the other unit involved in the transfer.

The basic principle of a two-signal handshaking procedure for data transfer is as follows. One control line from the initiating unit is used to request a response from the other unit. The second control line from the other unit is used to reply to the initiating unit that the response is occurring. In this way, each unit informs the other of its status, and the result is an orderly transfer through the bus.

Figure 12-8 shows data transfer procedures using handshaking. In Figure 12-8(a), the transfer is initiated by the destination unit. The two handshaking lines are called Request and Reply. In the initial state both Request and Reply are reset and in the 00 state. Subsequent states are 10, 11, and 01. The destination unit initiates the transfer by enabling Request. The source unit responds by placing the data on the bus. After a short time for settling of the data on the bus, the source unit activates Reply, to signal the presence of the data. In response to Reply, the destination unit captures the data in a register and resets Request. The source unit then resets Reply, and the system goes to the initial state. The destination unit may not make another request until the source unit has shown its readiness to provide new data by disabling Reply. Figure 12-8(b) represents handshaking for the source-initiated transfer. In this case, the source controls the interval between when the data is applied and when Request changes to 1 and between when Request changes to 0 and when the data is removed.

The handshaking scheme provides a high degree of flexibility and reliability, because the successful completion of a data transfer relies on active participation by both units. If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a time-out mechanism, which produces an alarm if the data transfer is not completed within a predetermined time interval. The time-out is implemented by means of an internal clock that starts counting time when the unit sets one of its handshaking control signals. If the return handshake does not occur within a given period, the unit assumes that an error occurred. The time-out signal can be used to interrupt the CPU and execute a service routine that
takes appropriate error recovery action. Also, the timing is controlled by both units, not just the initiating unit. Within the time-out limits, the response of each unit to a change in the control signal of the other unit can take an arbitrary amount of time, and the transfer will still be successful.

The examples of transfers in Figure 12-7 and Figure 12-8 represent transfers between an interface and an I/O device and between a CPU and an interface. In the latter case, however, an address will be necessary to select the interface with which the CPU wishes to communicate and a register within the interface. In order to ensure that the CPU addresses the correct interface, the address must have settled on the address bus before the Strobe or Request signal changes from 0 to 1. Further, the address must remain stable until the change in the strobe or request from 1 to 0 has settled to 0 at the interface logic. If either of these conditions is violated, another interface may be falsely activated, causing an incorrect data transfer.
12-4 SERIAL COMMUNICATION

The transfer of data between two units may be parallel or serial. In parallel data transfer, each bit of the message has its own path, and the entire message is transmitted at one time. This means that an n-bit message is transmitted in parallel through n separate conductor paths. In serial data transmission, each bit in the message is sent in sequence, one at a time. This method requires the use of one or two signal lines. Parallel transmission is faster, because multiple signal lines operate in parallel. It is used for short distances and when speed is important. Serial transmission is slower, but less expensive, since it requires only one conductor. Serial connections are becoming increasingly important because of the ease of connecting smaller cables and because as data rates increase, signal skew from line-to-line becomes more problematic. For the serial case there are just one or two signals, so that skew is less of a problem.

One way that computers and terminals that are remote from each other are connected is via telephone lines. Since telephone lines were originally designed for voice communication, but computers communicate in terms of digital signals, some form of conversion is needed. The devices that do the conversion are called data sets or modems (modulator–demodulators). There are various modulation schemes, as well as several different grades of communication media and transmission speeds. Serial data can be transmitted between two points in three different modes: simplex, half duplex, or full duplex. A simplex line carries information in one direction only. This mode is seldom used in data communication, because the receiver cannot communicate with the transmitter to indicate whether errors have occurred. Examples of simplex transmission are radio and television broadcasting.

A half-duplex transmission system is capable of transmitting in both directions, but in only one direction at a time. A pair of wires is needed for this mode. A common situation is for one modem to act as the transmitter and the other as the receiver. When transmission in one direction is completed, the roles of the modems are reversed to enable transmission in the opposite direction. The time required to switch a half-duplex line from one direction to the other is called the turnaround time.

A full-duplex transmission system can send and receive data in both directions simultaneously. This can be achieved by means of a two-wire plus ground link, with a different wire dedicated to each direction of transmission. Alternatively, a single-wire circuit can support full-duplex communication if the frequency spectrum is subdivided into two nonoverlapping frequency bands to create separate receiving and transmitting channels in the same physical pair of wires.

The serial transmission of data can be synchronous or asynchronous. In synchronous transmission, the two units share a common clock frequency, and bits are transmitted continuously at that frequency. In long-distance serial transmission, the transmitter and receiver units are each driven by separate clocks of the same frequency. Synchronization signals are transmitted periodically between the two units to keep their clock frequencies in step with each other. In asynchronous transmission, binary information is sent only when it is available, and the line remains idle when there is no information to be transmitted. This is in contrast to synchronous
transmission, in which bits must be transmitted continuously to keep the clock frequencies in both units synchronized.

**Asynchronous Transmission** A supplement containing the deleted subsection on asynchronous transmission, a serial port protocol used less frequently in new designs, is available on the text Companion Website.

**Synchronous Transmission**

The modems employed in synchronous transmission have internal clocks that are set to the frequency at which bits are being transmitted. For proper operation, the clocks of the transmitter and receiver modems must remain synchronized at all times. The communication line, however, carries only the data bits, from which information on the clock frequency must be extracted. Frequency synchronization is achieved by the receiving modem from the signal transitions that occur in the data that is received. Any frequency shift that may occur between the transmitter and receiver clocks is continuously adjusted by maintaining the receiver clock at the frequency of the incoming bit stream. In this way, the same rate is maintained in both the transmitter and the receiver.

Contrary to asynchronous transmission, in which each character can be sent separately, synchronous transmission must send a continuous message in order to maintain synchronism. The message consists of a group of bits that form a block of data. The entire block is transmitted with special control bits at the beginning and the end, in order to frame the block into one unit of information.

**The Keyboard Revisited**

To this point, we have covered the basic nature of the I/O interface and serial transmission. With these two concepts available, we are now ready to continue with the example of the keyboard and its interface, as shown in Figure 12-9. The K-scan code produced by the keyboard microcontroller is to be transferred serially from the keyboard through the keyboard cable to the keyboard controller in the computer. In this case, however, a signal Keyboard clock is also sent through the cable. Thus, the transmission is synchronous with a transmitted clock signal, rather than asynchronous. These same signals are used to transmit control commands to the keyboard. In the keyboard controller, the microcontroller converts the K-scan code to a more standard scan code, which it then places in the Input register, at the same time sending an interrupt signal to the CPU indicating that a key has been pressed and a code is available. The interrupt-handling routine reads the scan code from the input register into a special area in memory. This area is manipulated by software stored in the Basic Input/Output System (BIOS) that can translate the scan code into an ASCII character code for use by applications.

The Output register in the interface receives data from the CPU. The data can be passed on to control the keyboard—for example, setting the repetition rate
when a key is held down. The Control register is used for commands to the keyboard controller. Finally, the Status register reports specific information on the status of the keyboard and the keyboard controller.

An interesting aspect of keyboard I/O is its high complexity. It involves two microcontrollers executing different programs, plus the main processor executing BIOS software (i.e., three different computers executing three distinct programs).

**A Packet-Based Serial I/O Bus**

Serial I/O, as described for the keyboard, uses a serial cable specifically dedicated to communicating between the computer and the keyboard. Whether parallel or serial, external I/O connections are typically dedicated. The use of these dedicated paths often requires that the computer case be opened and cards inserted with electronics and connectors specific to the particular I/O standard used for a given I/O device.

In contrast, packet-based serial I/O permits many different external I/O devices to use a shared communication structure that is attached to the computer through just one or two connectors. The types of devices supported include keyboards, mice, joysticks, printers, scanners, and speakers. The particular packet-based serial I/O we will describe here is the Universal Serial Bus (USB), which is becoming commonplace as the connection approach of choice for slow-speed to medium-speed I/O devices.

The interconnection of I/O devices by using USB is shown in Figure 12-10. The computer and attached devices can be classified as hubs, devices, or compound devices. A hub provides attachment points for USB devices and other hubs. A hub contains a USB interface for control and status handling and a repeater for transferring information through the hub.

The computer contains a USB controller and the root hub. Additional hubs may be a part of the USB I/O structure. If a hub is combined with a device such as
the keyboard shown in Figure 12-10, then the keyboard is referred to as a compound device. Aside from such compound devices, a USB device contains only one USB port to serve its function alone. The scanner is an example of a regular USB device. Without USB, the monitor, keyboard, mouse, joystick, microphone, speakers, printer, and scanner shown would all have separate I/O connections directly on the computer. The monitor, printer, scanner, microphone, and speakers might all require special cards to be inserted, as discussed previously. With USB, only two connections are required.

The USB cables contains four wires: ground, power, and two data lines (D+ and D−) used for differential signaling. The power wire is used to provide small amounts of power to devices such as keyboards so that they do not need their own power supplies. To provide immunity to signal variation and noise, 0s and 1s are transmitted by using the difference in voltage between D+ and D−. If the voltage on D+ exceeds the voltage on D− by 200 millivolts or more, then the logic value is a 1. If the voltage on D− exceeds the voltage on D+ by 200 millivolts or more, the logic value is a 0. Other voltage relationships between D+ and D− are used as special signal states as well.

The logic values used for signaling are not the actual logic values of the information being transmitted. Instead, a Non-Return-to-Zero Inverted (NRZI)
signaling convention is used. A zero in the data being transmitted is represented by a transition from 1 to 0 or 0 to 1 and a 1 is represented by a fixed value of 1 or 0. The relationship between the data being transmitted and the NRZI representation is illustrated in Figure 12-11. As is typical for I/O devices, there is no common clock serving both the computer and the device. NRZI encoding of the data provides edges that can be used to maintain synchronization between the arriving data and the time at which each bit is sampled at the receiver. If there are a large number of 1s in series in the data, there will be no transitions for some time in the NRZI encoding. To prevent loss of synchronization, a 0 is “stuffed” in before every seventh bit position in a string of 1s prior to NRZI encoding so that no more than six 1s appear in series. The receiver must be able to remove these extra zeros when converting NRZI data to normal data.
USB information is transmitted in packets. Each packet contains a specific set of fields, depending on the packet type. Logical strings of packets are used to compose USB transactions. For example, an output transaction consists of an Out packet followed by a Data packet and a Handshake packet. The Out packet comes from the USB controller in the computer and notifies the device that it is to receive data. The computer then sends the Data packet. If the Data packet is received without error, then the device responds with the Acknowledge Handshake packet. Next, we detail the information contained in each of these packets.

Figure 12-12(a) shows a general format for USB packets and the formats for each of the three packets involved in an output transaction. Note that each packet begins with a synchronization pattern SYNC. This pattern is 00000001. Because of the sequence of zeros, the corresponding NRZI pattern contains seven edges, which provides a pattern to which the receiving clock can be synchronized. Since this pattern is preceded by a specific signal voltage state referred to as Idle, the pattern also signals the beginning of a new packet.

Following the SYNC, each packet format contains eight bits called the packet identifier (PID). In the PID, the packet type is specified by four bits, with an additional four bits that are complements of the first four to provide an error check on the type. A very large class of type errors will be detected by the repetition of the type as its complement. The type is optionally followed by information specific to the packet, which varies depending upon the packet type. Optionally, a CRC field appears next. The CRC pattern consisting of five or 16 bits is a Cyclic Redundancy Check pattern. This pattern is calculated at transmission of the packet from the packet-specific data. The same calculation is performed when the data is received. If the CRC pattern does not match the newly calculated pattern, then an error has been detected. In response to the error, the packet can be ignored and retransmitted. In the last field of the packet, an End of Packet (EOP) appears. This consists of D+ and D-, both low for two bit times, followed by the Idle state for a bit time. As its name indicates, this sequence of signal states identifies the end of the current packet. It should be noted that all fields are presented least significant bit first.

Referring to Figure 12-12(b), for the Output packet, the Type and Check fields are followed by a Device Address, an Endpoint Address, and a CRC pattern. The Device Address consists of seven bits and defines the device that is to input data. The Endpoint Address consists of four bits and defines which port of the device is to receive the information in the Data packet to follow. For example, there may be a port for data and one for control on a given device.

For the Data packet, the packet-specific data consists of 0 to 1024 data bytes. Due to the length of the packet, complex errors are more likely, so the CRC pattern is increased in length to 16 bits to improve its error-detection capability.

In the Handshake packet, the packet-specific data is empty. The response to the receipt of the data packet is carried by the PID. PID 01001011 is an Acknowledge (ACK) indicating that the packet was received without any errors detected. Absence of any HANDSHAKE packet when one would normally appear is an indication of an error. PID 01011010 is a No Acknowledge, indicating that the target is temporarily unable to accept or return data. PID 01110000 is a Stall