called a *data transmission* [illegible] works.
people and machines to communicate with machines over data [illegible]

In essence, binary codes add meaning to bits. Hex codes are simply shorthand notations for binary sequences and by themselves add no meaning to the data. The hex code 31 represents the binary sequence 0011 0001, which could mean different things in different situations. For example, 31 hex represents the digit 1 in ASCII and the number 39 in BCD and in decimal carries a binary weighting equivalent to 49.

In addition to sending data transmission codes, additional information may be added to the code to perform error control (i.e., error detection and/or correction [EDC]). Error control enables receivers to determine whether errors have occurred during transmission and, when they do occur, to correct them. Additional data may also be added to the original information to perform specific network functions, such as indicating the beginning and end of a character (*character framing*) or at the beginning and end of a message (*message framing*). Data added to the original information is called *overhead*, and sometimes a message contains more overhead than user information.

13-2 DATA COMMUNICATIONS CHARACTER CODES

Data communications codes are often used to represent characters and symbols, such as letters, digits, and punctuation marks. Therefore, these types of data communications codes are called *character codes, character sets, symbol codes*, or *character languages*. In essence, there are only three types of characters used with character codes: data-link control characters, alphanumeric characters, and graphic control characters. *Data-link control characters* are used to facilitate the orderly flow of data from source to destination. *Graphic control characters* involve the syntax or presentation of the data at the receiver, and *alphanumeric characters* represent the various symbols used for letters, numbers, and punctuation. Data-link control characters include special function characters for framing messages, such as STX (start of text) and ETX (end of text). Graphic control characters include special characters for displaying text on printers and monitors, such as VT (vertical tab), HT (horizontal tab), and NL (new line). Data-link and graphic control characters are described in more detail in a later chapter of this book.

The first character set was the Morse code. However, today the three most common character sets are the Baudot code, the American Standard Code for Information Interchange (ASCII), and the Extended Binary Decimal Interchange Code (EBCDIC).

13-2-1 Morse Code

The first character code that saw widespread usage was the *Morse code*, developed in 1840 by Samuel F. B. Morse. The Morse code was first used in 1844 when Morse established the world's first digital communications link between Washington, D.C., and Baltimore,

Table 13-1 International Morse Code

Letter	Code	Letter	Code	Letter	Code
A	•–	N	–•	1	•————
B	–•••	O	———	2	••–––
C	–•–•	P	•––•	3	•••––
D	–••	Q	––•–	4	••••–
E	•	R	•–•	5	•••••
F	••–•	S	•••	6	–••••
G	––•	T	–	7	––•••
H	••••	U	••–	8	–––••
I	••	V	•••–	9	———•
J	•–––	W	•––	0	–––––
K	–•–	X	–••–		
L	•–••	Y	–•––		
M	––	Z	––••		
Period	•–•–•–	Fraction bar (slash)		–••–•	
Comma	––••––	Wait		•–•••	
?	••––••	End of message		•–•–•	
Error	••••••••	Invitation to transmit		–•–	
:	———•••	End of transmission		•••–•–	
;	–•–•–•	Double dash (break)		–•••–	
()	–•––•–				

SOS (International distress signal: "save our ship") •••———•••

13-3 BAR CODES

Bar codes are those omnipresent black and white striped stickers that seem to appear on every consumer item found in every store in the United States and most of the rest of the world. Although bar codes were developed in the early 1970s, they were not used extensively until the mid-1980s. A bar code is a series of vertical black bars separated by vertical white bars (called spaces). The widths of the bars and spaces, along with their reflective abilities, represent binary 1s and 0s that identify a specific item. In addition, bar codes may contain information regarding cost, inventory management and control, security access, shipping and receiving, production counting, document and order processing, automatic billing, and many other applications. A typical bar code is shown in Figure 13-1a.

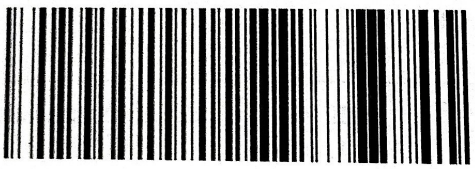
Figure 13-1b shows the layout of the fields found on a typical bar code. The start field consists of a unique sequence of bars and spaces used to identify the beginning of the data field. The data characters correspond to the bar code symbology or format used. Serial data encoded in the data character field is extracted from the card with an optical scanner. The scanner reproduces logic conditions that correspond to the difference in reflectivity of the printed bars and underlying white spaces. To read the information, simply scan over the printed bar with a smooth, uniform motion. A photodetector in the scanner senses the reflected light and converts it to electrical signals for decoding.

There are several standard bar code formats used today in industry. The format is selected on the basis of the type of data being stored, how the data is being stored, system performance, and which format is most popular with business and industry. Bar codes are generally classified as being discrete, continuous, or 2D.

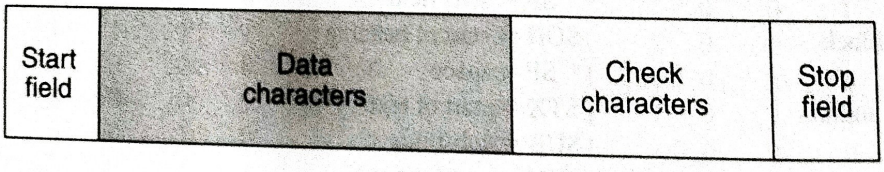
Discrete code. A discrete bar code has spaces or gaps between characters. Therefore, each character within the bar code is independent of every other character. Code 39 is an example of a discrete bar code.

Continuous code. A continuous bar code does not include spaces between characters. An example of a continuous bar code is the Universal Product Code (UPC).

2D code. A 2D (two-dimensional) bar code stores data in two dimensions instead of in conventional linear bar codes, which store data along only one axis. A 2D bar code has a larger storage capacity than one-dimensional bar codes (typically 1 kilobyte or more per data symbol).



(a)



(b)

FIGURE 13-1 (a) Bar code; (b) bar code layout

Table 13-5 Code 39 Character Set

Character	Binary Code b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Bars $b_8b_6b_4b_2b_0$	Spaces $b_7b_5b_3b_1$	Check Sum Value
0	0	0	0	1	1	0	1	0	0	00110	0100	0
1	1	0	0	1	0	0	0	0	1	10001	0100	1
2	0	0	1	1	0	0	0	0	1	01001	0100	2
3	1	0	1	1	0	0	0	0	0	11000	0100	3
4	0	0	0	1	1	0	0	0	1	00101	0100	4
5	1	0	0	1	1	0	0	0	0	10100	0100	5
6	0	0	1	1	1	0	0	0	0	01100	0100	6
7	0	0	0	1	0	0	1	0	1	00011	0100	7
8	1	0	0	1	0	0	1	0	0	10010	0100	8
9	0	0	1	1	0	0	1	0	0	01010	0100	9
A	1	0	0	0	0	1	0	0	1	10001	0010	10
B	0	0	1	0	0	1	0	0	1	01001	0010	11
C	1	0	1	0	0	1	0	0	0	11000	0010	12
D	0	0	0	0	1	1	0	0	1	00101	0010	13
E	1	0	0	0	1	1	0	0	0	10100	0010	14
F	0	0	1	0	1	1	0	0	0	01100	0010	15
G	0	0	0	0	0	1	1	0	1	00011	0010	16
H	1	0	0	0	0	1	1	0	0	10010	0010	17
I	0	0	1	0	0	1	1	0	0	01010	0010	18
J	0	0	0	0	1	1	1	0	0	00110	0010	19
K	1	0	0	0	0	0	0	1	1	10001	0001	20
L	0	0	1	0	0	0	0	1	1	01001	0001	21
M	1	0	1	0	0	0	0	1	0	11000	0001	22
N	0	0	0	0	1	0	0	1	1	00101	0001	23
O	1	0	0	0	1	0	0	1	0	10100	0001	24
P	0	0	1	0	1	0	0	1	0	01100	0001	25
Q	0	0	0	0	0	0	1	1	1	00011	0001	26
R	1	0	0	0	0	0	1	1	0	10010	0001	27
S	0	0	1	0	0	0	1	1	0	01010	0001	28
T	0	0	0	0	1	0	1	1	0	00110	0001	29
U	1	1	0	0	0	0	0	0	1	10001	1000	30
V	0	1	1	0	0	0	0	0	1	01001	1000	31
W	1	1	1	0	0	0	0	0	0	11000	1000	32
X	0	1	0	0	1	0	0	0	1	00101	1000	33
Y	1	1	0	0	1	0	0	0	0	10100	1000	34
Z	0	1	1	0	1	0	0	0	0	01100	1000	35
–	0	1	0	0	0	0	1	0	1	00011	1000	36
.	1	1	0	0	0	0	1	0	0	10010	1000	37
space	0	1	1	0	0	0	1	0	0	01010	1000	38
*	0	1	0	0	1	0	1	0	0	00110	1000	—
$	0	1	0	1	0	1	0	0	0	00000	1110	39
/	0	1	0	1	0	0	0	1	0	00000	1101	40
+	0	1	0	0	0	1	0	1	0	00000	1011	41
%	0	0	0	1	0	1	0	1	0	00000	0111	42

13-3-1 Code 39

There are numerous bar code formats available using codes that vary from numeric symbols only to full ASCII code. One of the most common codes was developed in 1974 and is called *Code 39*. Because Code 39 was one of the first of the bar codes used for postal routing, it is sometimes called the USS Code 39. Code 39 (also called *Code 3 of 9* or *3 of 9 Code*) uses an alphanumeric code similar to ASCII and is shown in Table 13-5. Code 39 consists of 36 unique codes representing the 10 digits and 26 uppercase letters. There are seven additional codes used for special characters and an exclusive start/stop character coded as an asterisk (*). Code 39 bar codes are ideally suited for making labels, such as name badges.

FIGURE 13-7 POSTNET bar code for the nine-digit ZIP+4 code for the ZIP code 85281–4220

2, and 1, and the least-significant bit (b_0) has a weighting of 0. All digits follow the weighting scheme except the digit 0, which is assigned the code 11000. For example, the binary code for the digits 8 is determined as follows:

weighting	7	4	2	1	0
bit position	b_4	b_3	b_2	b_1	b_0
digit 8	1	0	0	1	0
	7	+		1	= 8

All POSTNET bar codes begin with a tall start bar and end with a tall stop bar. The bits in the bar codes themselves are arranged in contiguous order from left to right to complete the numeric portion of the code. A special error-detection character called a check character is appended to the end of each bar code. The check character is one digit long (five bits) and, when added to the sum of the bar code digits, causes the sum of all the digits (including the check character) to equal the next-highest multiple of 10.

Example 13-3

Determine the check character for the nine-digit ZIP+4 number 85281–4220.

Solution The sum of all the digits in the number 85281–4220 is 32. The next-highest multiple of 10 is 40, so the checksum is 8 (32 + 8 = 40). The POSTNET bar code for the number 85281–4220 is shown in Figure 13-7.

13-4 ERROR CONTROL

A data communications circuit can be as short as a few feet or as long as several thousand miles, and the transmission medium can be as simple as a pair of wires or as complex as a microwave, satellite, or optical fiber communications system. Therefore, because of the nonideal transmission characteristics associated with any communications system, it is inevitable that errors will occur, and it is necessary to develop and implement procedures for error control. Transmission errors are caused by electrical interference from natural sources, such as lightning, as well as from man-made sources, such as motors, generators, power lines, and fluorescent lights.

Data communications errors can be generally classified as *single bit, multiple bit,* or *burst.* Single-bit errors are when only one bit within a given data string is in error. Single-bit errors affect only one character within a message. A multiple-bit error is when two or more nonconsecutive bits within a given data string are in error. Multiple-bit errors can affect one or more characters within a message. A burst error is when two or more consecutive bits within a given data string are in error. Burst errors can affect one or more characters within a message.

3-6 ERROR CORRECTION

Although detecting errors is an important aspect of data communications, determining what to do with data that contains errors is another consideration. There are two basic types of errors: *lost message* and *damaged message*. A lost message is one that never arrives at the destination or one that arrives but is damaged to the extent that it is unrecognizable. A damaged message is one that is recognized at the destination but contains one or more transmission error.

Data communications network designers have developed two basic strategies for handling transmission errors: *error-detecting codes* and *error-correcting codes*. Error-detecting codes include enough redundant information with each transmitted message to enable the receiver to determine when an error has occurred. Parity bits, block check characters, and cyclic redundancy characters are examples of error-detecting codes. Error-correcting codes include sufficient extraneous information along with each message to enable the receiver to determine when an error has occurred and which bit is in error.

Transmission errors can occur as single-bit errors or as bursts of errors, depending on the physical processes that caused them. Having errors occur in bursts is an advantage when data is transmitted in blocks or frames containing many bits. For example, if a typical block size is 10,000 bits and the system has a probability of error of 10^{-4} (one bit error in every 10,000 bits transmitted), independent bit errors would most likely produce an error in every block. However, if errors occur in bursts of 1000, only one or two blocks out of every 1000 transmitted would contain errors. The disadvantage of bursts of errors is they are more difficult to detect and even more difficult to correct than isolated single-bit errors.

In the early days of data communications, virtually all communications took place between human operators sitting in front of terminals communicating with mainframe computers. Because of the low bit rates available at the time, about the most effective means of providing error correction was a technique called *symbol substitution*. Symbol substitution was designed to be used in a human environment—when there is a human being at a terminal to analyze the received data and make decisions on its integrity. With symbol substitution, if a character is received in error, rather than revert to a higher level of error correction or display the incorrect character, a unique character that is undefined by the character code, such as a reverse question mark, is substituted for the bad character. If the operator cannot discern the flawed character, a retransmission is called for (i.e., symbol substitution is a form of *selective retransmission*). For example, if the message involved alpha characters, the operator could probably figure out what the incorrect character is. However, if the message involved numbers, the operator would probably request a retransmission.

In the modern world of data communications, there are two primary methods used for error correction: *retransmission* and *forward error correction*.

13-6-1 Retransmission

Retransmission, as the name implies, is when a receive station requests the transmit station to resend a message (or a portion of a message) when the message is received in error. Because the receive terminal automatically calls for a retransmission of the entire message, retransmission is often called ARQ, which is an old two-way radio term that means *automatic repeat request* or *automatic retransmission request*. ARQ is probably the most reliable method of error correction, although it is not always the most efficient. Impairments on transmission media often occur in

Error performance is the rate in which errors occur, which can be described as either an expected value or an empirical value. The theoretical (mathematical) expectation of the rate at which errors will occur is called *probability of error* (*P*[*e*]), whereas the actual historical record of a system's error performance is called *bit error rate* (BER). For example, if a system has a $P(e)$ of 10^{-5}, this means that mathematically the system can expect to experience one bit error for every 100,000 bits transported through the system ($10^{-5} = 1/10^5 = 1/100{,}000$). If a system has a BER of 10^{-5}, this means that in the past there was one bit error for every 100,000 bits transported. Typically, a BER is measured and then compared with the probability of error to evaluate a system's performance.

Error control can be divided into two general categories: *error detection* and *error correction*.

13-5 ERROR DETECTION

Error detection is the process of monitoring data and determining when transmission errors have occurred. Error-detection techniques neither correct errors nor identify which bits are in error; they indicate only when an error has occurred. The purpose of error detection is not to prevent errors from occurring but to prevent undetected errors from occurring. How data communications systems react to transmission errors is system dependent and varies considerably.

The most common error-detection techniques used for data communications networks are redundancy, echoplex, exact-count coding, and redundancy checking, which includes vertical redundancy checking, checksum, longitudinal redundancy checking, and cyclic redundancy checking. Generally speaking, the higher the transmission bit rate, the more complicated the form of error detection used.

13-5-1 Redundancy

Redundancy is a form of error detection where each data unit is sent multiple times, usually twice. At the receive end, the two units are compared, and if they are the same, it is assumed that no transmission errors have occurred. When the data unit is a single character, it is called *character redundancy*, whereas if the data unit is the entire message, it is called *message redundancy*. Character redundancy is the most common form of redundancy. If the exact same character is not received twice in succession, a transmission error must have occurred. With message redundancy, if the exact same sequence of characters is not received twice in succession, in exactly the same order, a transmission error must have occurred.

Another type of redundancy used with short messages is to transmit the same message several times. At the receive end, if a given number of the messages are the same, it is assumed to be a successful transmission. For example, a message might be sent 10 times, and if seven or more of the received messages are the same, it is considered a successful transmission.

13-5-2 Echoplex

Echoplex (sometimes called *echo checking*) is a relatively simple form of error-detection scheme used almost exclusively with data communications systems involving human operators working in real time at computer terminals or PCs. With echoplex, receiving devices retransmit received data back to the transmitting device; therefore, echoplex requires full-duplex operation. Each character is transmitted immediately after it has been typed on the keyboard. At the receive end, once a character has been received, it is immediately transmitted back to the originating terminal, where it appears on that terminal's screen. When the character appears on the screen, the operator has verification that the character has been received at the destination terminal. If a transmission error occurs, the wrong character will be displayed on the transmit terminal's screen. When this happens, the operator can send a backspace and remove the erroneous character and then type and resend the correct character.

receiver, to determine the bit position in error, extract the Hamming bits, and XOR them with the binary code for each data bit position that contains a logic 1:

Bit position	Binary number	
Hamming bits	10110	
2	00010	
XOR	10100	
6	00110	
XOR	10010	
12	01100	
XOR	11110	
16	10000	
XOR	01110	= 14

Therefore, bit position 14 contains an error.

CHARACTER SYNCHRONIZATION

In essence, *synchronize* means to harmonize, coincide, or agree in time. There are several levels of synchronization necessary for successful data communications to occur, including modem (carrier) synchronization, clock synchronization, character synchronization, and message synchronization. *Modem synchronization* involves recovering the carrier reference (both frequency and phase), which is necessary for coherent demodulation. *Clock synchronization* is simply duplicating the transmit clock in the receiver. *Message synchronization* is identifying the beginning and end of the actual message, which may be embedded within a more complex data transmission. *Character synchronization* involves identifying the beginning and end of a character within a message. The purpose of this section is to describe how character synchronization is achieved. The three other forms of synchronization are described in more detail in subsequent chapters of this book.

Clock synchronization ensures that the transmitter and receiver agree on a precise time slot for the occurrence of a bit and a precise rate at which bits will be transported through the system. When a continuous string of data is received, it is also necessary to identify which bits belong to which characters and which bits are the MSBs and LSBs of the character. In essence, this is *character synchronization:* identifying the beginning and end of a character code. In data communications circuits, there are two formats commonly used to achieve character synchronization: asynchronous and synchronous.

13-7-1 Asynchronous Serial Data

The term *asynchronous* literally means "without synchronism," which in data communications terminology means "without a specific time reference." *Asynchronous serial data* is typically used with so called *dumb terminals*, which are basically inexpensive, limited-capability computer terminals attached to relatively low-speed data networks. Dumb terminals can receive data but usually cannot modify the data once it is displayed on the screen. Dumb terminals generally send data in real time. Therefore, unless the operator is Superman (or Superwoman), there is always a slight pause (sometimes called *idle time*) between characters, and the pause is not always the same length of time. Consequently, it is necessary for a receiver to resynchronize to each character. Asynchronous data transmis-

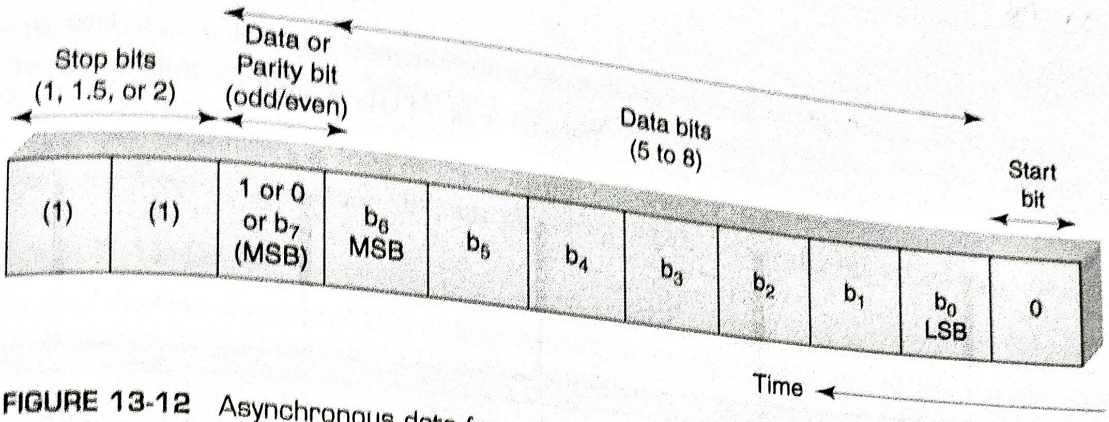


FIGURE 13-12 Asynchronous data format

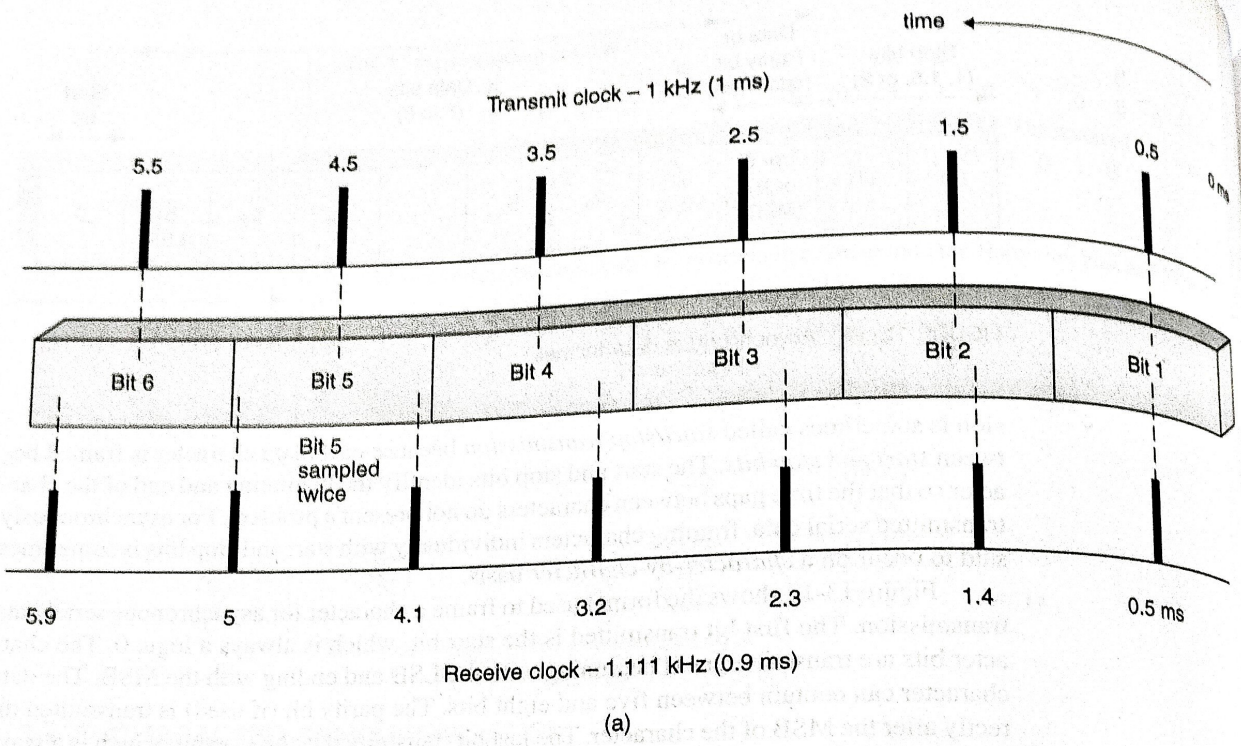
sion is sometimes called *start/stop transmission* because each data character is framed between *start* and *stop bits*. The start and stop bits identify the beginning and end of the character so that the time gaps between characters do not present a problem. For asynchronously transmitted serial data, framing characters individually with start and stop bits is sometimes said to occur on a *character-by-character* basis.

Figure 13-12 shows the format used to frame a character for asynchronous serial data transmission. The first bit transmitted is the start bit, which is always a logic 0. The character bits are transmitted next beginning with the LSB and ending with the MSB. The data character can contain between five and eight bits. The parity bit (if used) is transmitted directly after the MSB of the character. The last bit transmitted is the stop bit, which is always a logic 1, and there can be either one, one and a half, or two stop bits. Therefore, a data character may be comprised of between seven and 11 bits.

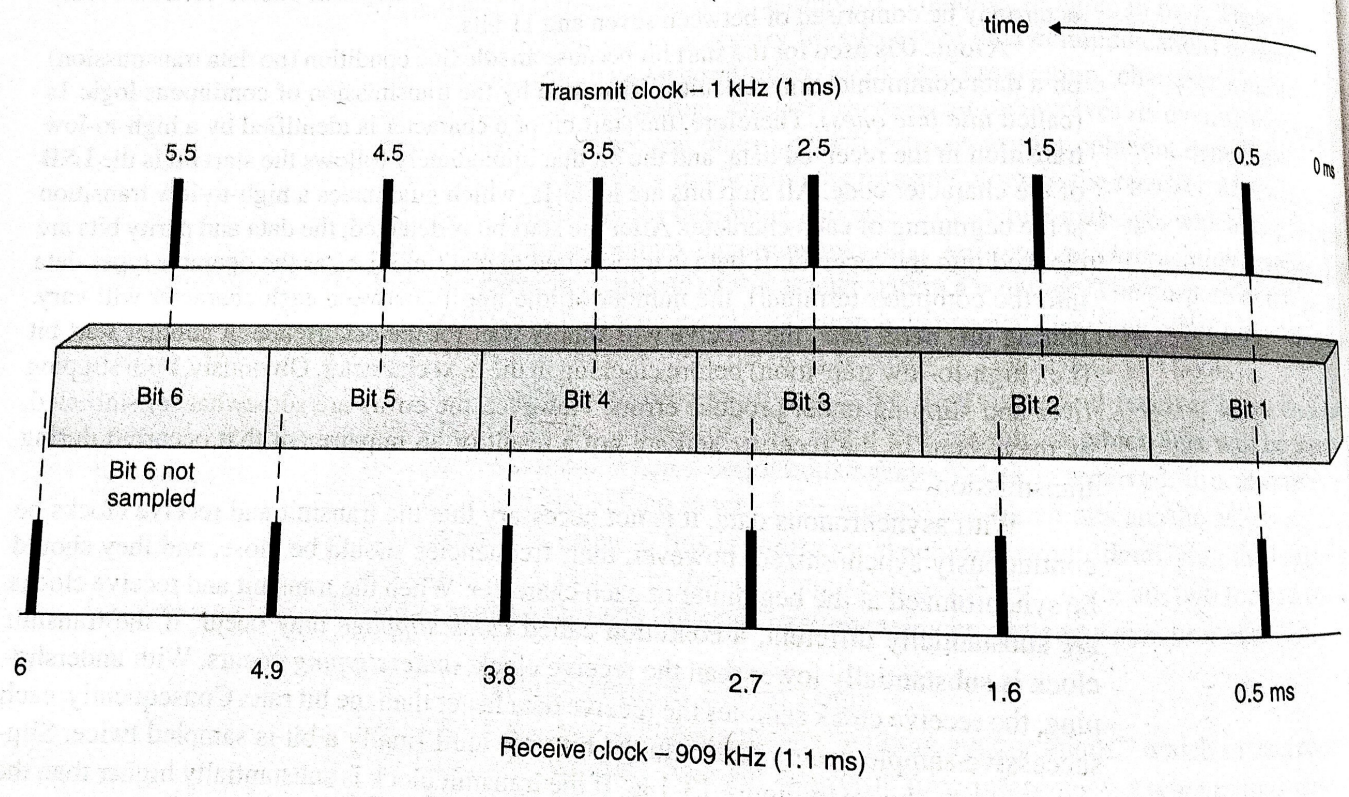
A logic 0 is used for the start bit because an idle line condition (no data transmission) on a data communications circuit is identified by the transmission of continuous logic 1s (called *idle line ones*). Therefore, the start bit of a character is identified by a high-to-low transition in the received data, and the bit that immediately follows the start bit is the LSB of the character code. All stop bits are logic 1s, which guarantees a high-to-low transition at the beginning of each character. After the start bit is detected, the data and parity bits are clocked into the receiver. If data is transmitted in real time (i.e., as the operator types data into the computer terminal), the number of idle line 1s between each character will vary. During this *dead time*, the receive will simply wait for the occurrence of another start bit (i.e., high-to-low transition) before clocking in the next character. Obviously, both slipping over and slipping under produce errors. However, the errors are somewhat self-inflicted, as they occur in the receiver and are not a result of an impairment that occurred during transmission.

With asynchronous data, it is not necessary that the transmit and receive clocks be continuously synchronized; however, their frequencies should be close, and they should be synchronized at the beginning of each character. When the transmit and receive clocks are substantially different, a condition called *clock slippage* may occur. If the transmit clock is substantially lower than the receive clock, *underslipping* occurs. With underslipping, the receive clock samples the receive data faster than the bit rate. Consequently, each successive sample occurs earlier in the bit time until finally a bit is sampled twice. Slipping under is shown in Figure 13-13a. If the transmit clock is substantially higher than the receive clock, a condition called *overslipping* occurs. With overslipping, the receive clock samples the receive data slower than the bit rate. Consequently, each successive sample occurs later in the bit time until finally a bit is completely skipped. Slipping over is shown in Figure 13-13b.

...illustrated in Figures 13-13a and b, the difference in the transmit and ...ted to illustrate the concept of clock slippage.



(a)



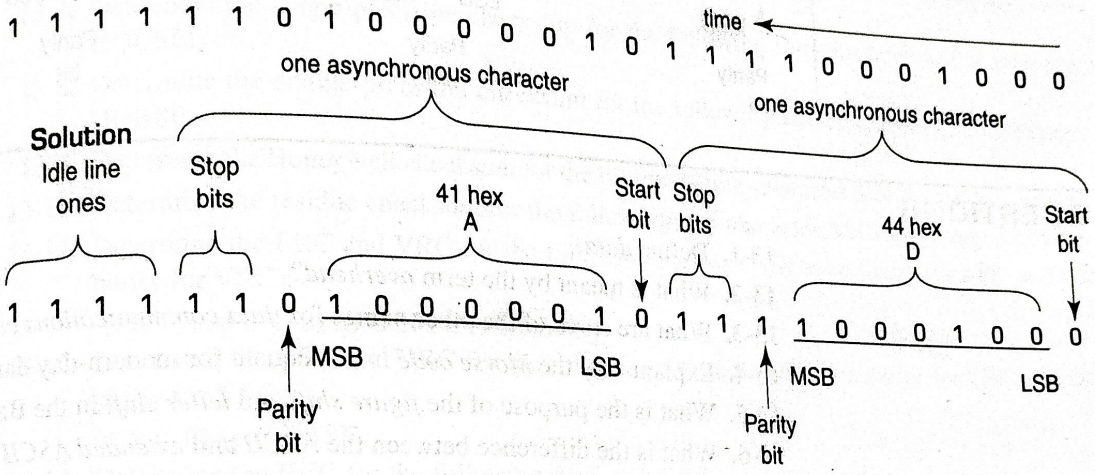
(b)

FIGURE 13-13 Clock slippage: (a) slipping under; (b) slipping over

In an actual asynchronous transmission system, it may take several hundred clock cycles before clock slippage occurs. When start and stop bits are used to frame each data character, the receiver essentially resynchronizes at the beginning of each character when the start bit is detected. Therefore, as long as the transmit and receive clock frequencies are reasonably close, the possibility of clock slippage occurring with an 11-bit character is remote.

Example 13-10

For the following sequence of bits, identify the ASCII-encoded character, the start and stop bits, and the parity bits (assume even parity and two stop bits):



13-7-2 Synchronous Serial Data

Synchronous serial data is usually associated with *smart* or *intelligent* computer terminals, which are usually PCs that possess the capacity to modify or alter information they receive. Synchronous data generally involves transporting serial data at relatively high speeds in groups of characters called *blocks* or *frames*. Therefore, synchronous data is not sent in real time. Instead, a message is composed or formulated, then the entire message is transmitted as a single entity with no time lapses between characters. With *synchronous data*, rather than frame each character independently with start and stop bits, a unique sequence of bits, sometimes called a *synchronizing* (SYN) *character*, is transmitted at the beginning of each message. For synchronously transmitted serial data, framing characters in blocks is sometimes said to occur on a *block-by-block* basis. For example, with ASCII code, the SYN character is 16 hex, and with EBCDIC the SYN character is 32 hex. The receiver disregards incoming data until it receives one or more SYN characters. Once the synchronizing sequence is detected, the receiver clocks in the next eight bits and interprets them as the first character of the message. The receiver continues clocking in bits, interpreting them in groups of eight until it receives another unique character that signifies the end of the message. The end-of-message character varies with the type of protocol being used and what type of message it is associated with. With synchronous data, the transmit and receive clocks must be synchronized because character synchronization occurs only once at the beginning of a message.

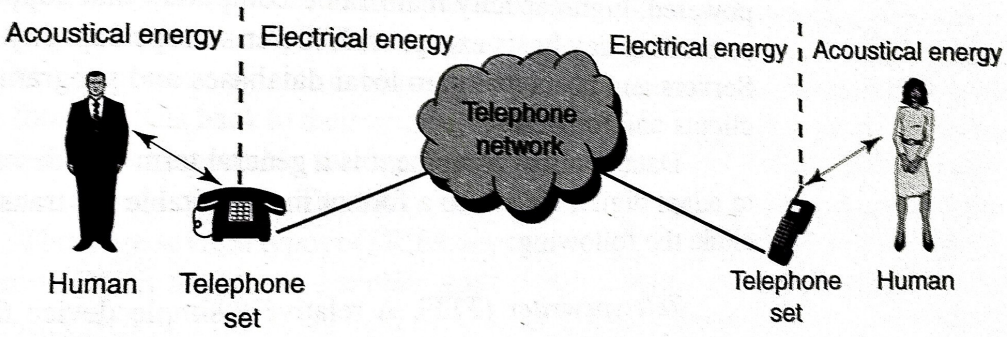
With synchronous data, each character has two or three bits added to each character (one start and either one, one and a half, or two stop bits). These bits are additional overhead and, thus, reduce the efficiency of the transmission (i.e., the ratio of information bits to total transmitted bits). Synchronous data generally has two SYN characters (16 bits of overhead) added to each message. Therefore, asynchronous data is more efficient for short messages.

works, the intent of this chapter is to describe only hardware associated with the lowest layer of the OSI protocol hierarchy: the physical layer.

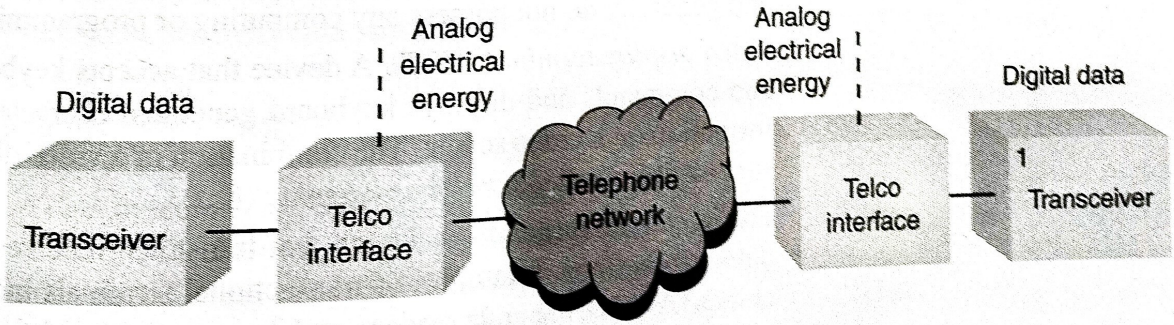
14-2 DATA COMMUNICATIONS HARDWARE

Digital information sources, such as personal computers, communicate with each other using the POTS (plain old telephone service) telephone network in a manner very similar to the way analog information sources, such as human conversations, communicate with each other using the POTS telephone network. With both digital and analog information sources, special devices are necessary to interface the sources to the telephone network.

Figure 14-1 shows a comparison between human speech (analog) communications and computer data (digital) communications using the POTS telephone network. Figure 14-1a shows how two humans communicate over the telephone network using standard analog telephone sets. The telephone sets interface human speech signals to the telephone network and vice versa. At the transmit end, the telephone set converts acoustical energy (information) to electrical energy, and at the receive end, the telephone set converts electrical energy back to acoustical energy. Figure 14-1b shows how digital data is transported over the telephone network. At the transmitting end, a telco interface converts digital data from the transceiver to analog electrical energy that is transported through the telephone network. At the receiving end, a telco interface converts the analog electrical energy received from the telephone network back to digital data.



(a)



(b)

FIGURE 14-1 Telephone communications network: (a) human communications; (b) digital data communications