CHAPTER 6

Digital Data Communication Techniques

6.1 Asynchronous and Synchronous Transmission

6.2 Types of Errors

6.3 Error Detection

6.4 Error Correction

6.5 Line Configurations

6.6 Recommended Reading

6.7 Key Terms, Review Questions, and Problems

A conversation forms a two-way communication link; there is a measure of symmetry between the two parties, and messages pass to and fro. There is a continual stimulus-response, cyclic action; remarks call up other remarks, and the behavior of the two individuals becomes concerted, co-operative, and directed toward some goal. This is true communication.

—On Human Communication, Colin Cherry

KEY POINTS

- The transmission of a stream of bits from one device to another across a transmission link involves a great deal of cooperation and agreement between the two sides. One of the most fundamental requirements is **synchronization**. The receiver must know the rate at which bits are being received so that it can sample the line at appropriate intervals to determine the value of each received bit. Two techniques are in common use for this purpose. In **asynchronous transmission**, each character of data is treated independently. Each character begins with a start bit that alerts the receiver that a character is arriving. The receiver samples each bit in the character and then looks for the beginning of the next character. This technique would not work well for long blocks of data because the receiver's clock might eventually drift out of synchronization with the transmitter's clock. However, sending data in large blocks is more efficient than sending data one character at a time. For large blocks, **synchronous transmission** is used. Each block of data is formatted as a frame that includes a starting and an ending flag. Some form of synchronization, such as the use of Manchester encoding, is employed.
- **Error detection** is performed by calculating an error-detecting code that is a function of the bits being transmitted. The code is appended to the transmitted bits. The receiver calculates the code based on the incoming bits and compares it to the incoming code to check for errors.
- **Error correction** operates in a fashion similar to error detection but is capable of correcting certain errors in a transmitted bit stream.

The preceding three chapters have been concerned primarily with the attributes of data transmission, such as the characteristics of data signals and transmission media, the encoding of signals, and transmission performance. In this chapter, we shift our emphasis from data transmission to data communications.

For two devices linked by a transmission medium to exchange data, a high degree of cooperation is required. Typically, data are transmitted one bit at a time over the medium. The timing (rate, duration, spacing) of these bits must be the same for transmitter and receiver. Two common techniques for controlling this timing—asynchronous and synchronous—are explored in Section 6.1. Next, we look at the problem of bit errors. As we have seen, data transmission is not an error-free process, and some means of accounting for these errors is needed. After a brief discussion of the distinction between single-bit errors and burst errors, the chapter turns to two approaches to dealing with errors: error detection and error correction.

Next, the chapter provides an overview of the types of line configurations in common use. To supplement the material in this chapter, Appendix G looks at the physical interface between data transmitting devices and the transmission line. Typically, digital data devices do not attach to and signal across the medium directly. Instead, this process is mediated through a standardized interface that provides considerable control over the interaction between the transmitting/receiving devices and the transmission line.

6.1 ASYNCHRONOUS AND SYNCHRONOUS TRANSMISSION

In this book, we are primarily concerned with serial transmission of data; that is, data are transferred over a single signal path rather than a parallel set of lines, as is common with I/O devices and internal computer signal paths. With serial transmission, signaling elements are sent down the line one at a time. Each signaling element may be

- **Less than one bit:** This is the case, for example, with Manchester coding.
- **One bit:** NRZ-L and FSK are digital and analog examples, respectively.
- **More than one bit:** QPSK is an example.

For simplicity in the following discussion, we assume one bit per signaling element unless otherwise stated. The discussion is not materially affected by this simplification.

Recall from Figure 3.16 that the reception of digital data involves sampling the incoming signal once per bit time to determine the binary value. One of the difficulties encountered in such a process is that various transmission impairments will corrupt the signal so that occasional errors will occur. This problem is compounded by a timing difficulty: In order for the receiver to sample the incoming bits properly, it must know the arrival time and duration of each bit that it receives.

Suppose that the sender simply transmits a stream of data bits. The sender has a clock that governs the timing of the transmitted bits. For example, if data are to be transmitted at one million bits per second (1 Mbps), then one bit will be transmitted every $1/10^6 = 1$ microsecond (μs), as measured by the sender's clock. Typically, the receiver will attempt to sample the medium at the center of each bit time. The receiver will time its samples at intervals of one bit time. In our example, the

sampling would occur once every 1 μs. If the receiver times its samples based on its own clock, then there will be a problem if the transmitter's and receiver's clocks are not precisely aligned. If there is a drift of 1% (the receiver's clock is 1% faster or slower than the transmitter's clock), then the first sampling will be 0.01 of a bit time (0.01 μs) away from the center of the bit (center of bit is 0.5 μs from beginning and end of bit). After 50 or more samples, the receiver may be in error because it is sampling in the wrong bit time ($50 \times .01 = 0.5$ μs). For smaller timing differences, the error would occur later, but eventually the receiver will be out of step with the transmitter if the transmitter sends a sufficiently long stream of bits and if no steps are taken to synchronize the transmitter and receiver.

Asynchronous Transmission

Two approaches are common for achieving the desired synchronization. The first is called, oddly enough, asynchronous transmission. The strategy with this scheme is to avoid the timing problem by not sending long, uninterrupted streams of bits. Instead, data are transmitted one character at a time, where each character is five to eight bits in length.[1] Timing or synchronization must only be maintained within each character; the receiver has the opportunity to resynchronize at the beginning of each new character.

Figure 6.1 illustrates this technique. When no character is being transmitted, the line between transmitter and receiver is in an *idle* state. The definition of *idle* is equivalent to the signaling element for binary 1. Thus, for NRZ-L signaling (see Figure 5.2), which is common for asynchronous transmission, idle would be the presence of a negative voltage on the line. The beginning of a character is signaled by a *start bit* with a value of binary 0. This is followed by the 5 to 8 bits that actually make up the character. The bits of the character are transmitted beginning with the least significant bit. For example, for IRA characters, the data bits are usually followed by a parity bit, which therefore is in the most significant bit position. The parity bit is set by the transmitter such that the total number of ones in the character, including the parity bit, is even (even parity) or odd (odd parity), depending on the convention being used. The receiver uses this bit for error detection, as discussed in Section 6.3. The final element is a *stop element*, which is a binary 1. A minimum length for the stop element is specified, and this is usually 1, 1.5, or 2 times the duration of an ordinary bit. No maximum value is specified. Because the stop element is the same as the idle state, the transmitter will continue to transmit the stop element until it is ready to send the next character.

The timing requirements for this scheme are modest. For example, IRA characters are typically sent as 8-bit units, including the parity bit. If the receiver is 5% slower or faster than the transmitter, the sampling of the eighth character bit will be displaced by 45% and still be correctly sampled.

---

[1]The number of bits that comprise a character depends on the code used. We have already described one common example, the IRA code, which uses seven bits per character. Another common code is the Extended Binary Coded Decimal Interchange Code (EBCDIC), which is an 8-bit character code used on IBM mainframes.

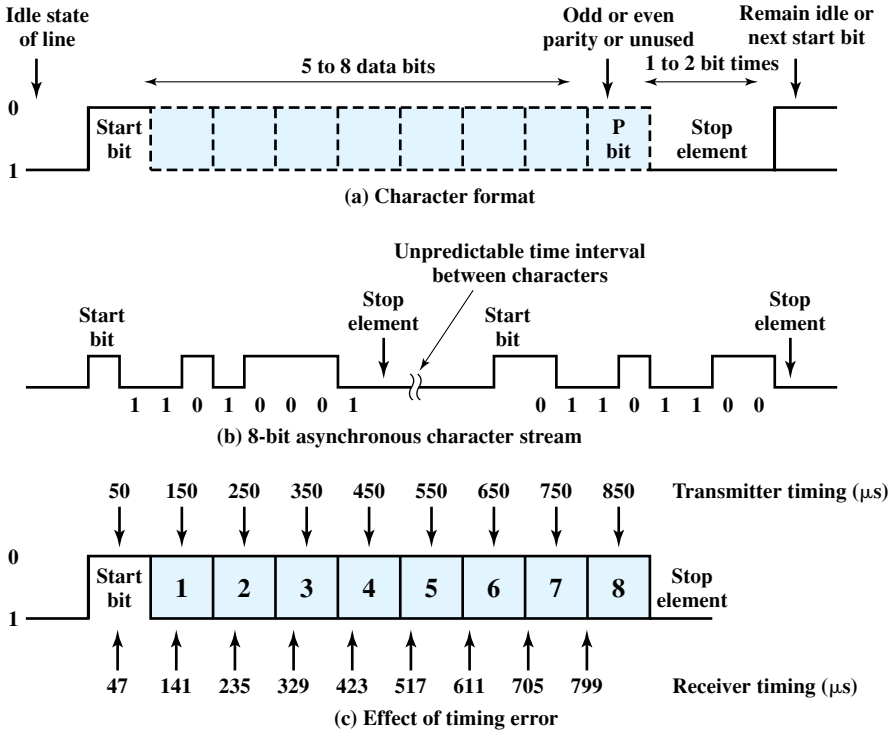


Figure 6.1 Asynchronous Transmission

EXAMPLE 6.1 Figure 6.1c shows the effects of a timing error of sufficient magnitude to cause an error in reception. In this example we assume a data rate of 10,000 bits per second (10 kbps); therefore, each bit is of 0.1 millisecond (ms), or 100 μs, duration. Assume that the receiver is fast by 6%, or 6 μs per bit time. Thus, the receiver samples the incoming character every 94 μs (based on the transmitter's clock). As can be seen, the last sample is erroneous.

An error such as just described actually results in two errors. First, the last sampled bit is incorrectly received. Second, the bit count may now be out of alignment. If bit 7 is a 1 and bit 8 is a 0, bit 8 could be mistaken for a start bit. This condition is termed a *framing error*, as the character plus start bit and stop element are sometimes referred to as a frame. A framing error can also occur if some noise condition causes the false appearance of a start bit during the idle state.

Asynchronous transmission is simple and cheap but requires an overhead of two to three bits per character. For example, for an 8-bit character with no parity bit, using a 1-bit-long stop element, two out of every ten bits convey no information but are there merely for synchronization; thus the overhead is 20%. Of

course, the percentage overhead could be reduced by sending larger blocks of bits between the start bit and stop element. However, as Figure 6.1c indicates, the larger the block of bits, the greater the cumulative timing error. To achieve greater efficiency, a different form of synchronization, known as synchronous transmission, is used.

Synchronous Transmission

With synchronous transmission, a block of bits is transmitted in a steady stream without start and stop codes. The block may be many bits in length. To prevent timing drift between transmitter and receiver, their clocks must somehow be synchronized. One possibility is to provide a separate clock line between transmitter and receiver. One side (transmitter or receiver) pulses the line regularly with one short pulse per bit time. The other side uses these regular pulses as a clock. This technique works well over short distances, but over longer distances the clock pulses are subject to the same impairments as the data signal, and timing errors can occur. The other alternative is to embed the clocking information in the data signal. For digital signals, this can be accomplished with Manchester or differential Manchester encoding. For analog signals, a number of techniques can be used; for example, the carrier frequency itself can be used to synchronize the receiver based on the phase of the carrier.

With synchronous transmission, there is another level of synchronization required, to allow the receiver to determine the beginning and end of a block of data. To achieve this, each block begins with a *preamble* bit pattern and generally ends with a *postamble* bit pattern. In addition, other bits are added to the block that convey control information used in the data link control procedures discussed in Chapter 7. The data plus preamble, postamble, and control information are called a **frame**. The exact format of the frame depends on which data link control procedure is being used.

Figure 6.2 shows, in general terms, a typical frame format for synchronous transmission. Typically, the frame starts with a preamble called a flag, which is 8 bits long. The same flag is used as a postamble. The receiver looks for the occurrence of the flag pattern to signal the start of a frame. This is followed by some number of control fields (containing data link control protocol information), then a data field (variable length for most protocols), more control fields, and finally the flag is repeated.

For sizable blocks of data, synchronous transmission is far more efficient than asynchronous. Asynchronous transmission requires 20% or more overhead. The control information, preamble, and postamble in synchronous transmission are typically less than 100 bits.

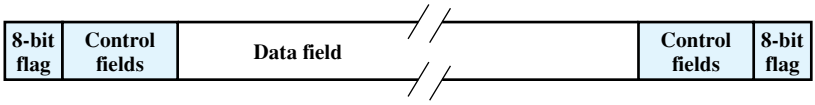


Figure 6.2 Synchronous Frame Format

EXAMPLE 6.2 One of the more common schemes, HDLC (described in Chapter 7), contains 48 bits of control, preamble, and postamble. Thus, for a 1000-character block of data, each frame consists of 48 bits of overhead and $1000 \times 8 = 8{,}000$ bits of data, for a percentage overhead of only $48/8048 \times 100\% = 0.6\%$.

6.2 TYPES OF ERRORS

In digital transmission systems, an error occurs when a bit is altered between transmission and reception; that is, a binary 1 is transmitted and a binary 0 is received, or a binary 0 is transmitted and a binary 1 is received. Two general types of errors can occur: single-bit errors and burst errors. A single-bit error is an isolated error condition that alters one bit but does not affect nearby bits. A burst error of length B is a contiguous sequence of B bits in which the first and last bits and any number of intermediate bits are received in error. More precisely, IEEE Std 100 and ITU-T Recommendation Q.9 both define an error burst as follows:

Error burst: A group of bits in which two successive erroneous bits are always separated by less than a given number x of correct bits. The last erroneous bit in the burst and the first erroneous bit in the following burst are accordingly separated by x correct bits or more.

Thus, in an error burst, there is a cluster of bits in which a number of errors occur, although not necessarily all of the bits in the cluster suffer an error.

A single-bit error can occur in the presence of white noise, when a slight random deterioration of the signal-to-noise ratio is sufficient to confuse the receiver's decision of a single bit. Burst errors are more common and more difficult to deal with. Burst errors can be caused by impulse noise, which was described in Chapter 3. Another cause is fading in a mobile wireless environment; fading is described in Chapter 14.

Note that the effects of burst errors are greater at higher data rates.

EXAMPLE 6.3 An impulse noise event or a fading event of 1 μs occurs. At a data rate of 10 Mbps, there is a resulting error burst of 10 bits. At a data rate of 100 Mbps, there is an error burst of 100 bits.

6.3 ERROR DETECTION

Regardless of the design of the transmission system, there will be errors, resulting in the change of one or more bits in a transmitted frame. In what follows, we

assume that data are transmitted as one or more contiguous sequences of bits, called frames. We define these probabilities with respect to errors in transmitted frames:

P_b: Probability that a bit is received in error; also known as the bit error rate (BER)

P_1: Probability that a frame arrives with no bit errors

P_2: Probability that, with an error-detecting algorithm in use, a frame arrives with one or more undetected errors

P_3: Probability that, with an error-detecting algorithm in use, a frame arrives with one or more detected bit errors but no undetected bit errors

First consider the case in which no means are taken to detect errors. Then the probability of detected errors (P_3) is zero. To express the remaining probabilities, assume the probability that any bit is in error (P_b) is constant and independent for each bit. Then we have

$$P_1 = (1 - P_b)^F$$
$$P_2 = 1 - P_1$$

where F is the number of bits per frame. In words, the probability that a frame arrives with no bit errors decreases when the probability of a single bit error increases, as you would expect. Also, the probability that a frame arrives with no bit errors decreases with increasing frame length; the longer the frame, the more bits it has and the higher the probability that one of these is in error.

EXAMPLE 6.4 A defined objective for ISDN (integrated services digital network) connections is that the BER on a 64-kbps channel should be less than 10^{-6} on at least 90% of observed 1-minute intervals. Suppose now that we have the rather modest user requirement that on average one frame with an undetected bit error should occur per day on a continuously used 64-kbps channel, and let us assume a frame length of 1000 bits. The number of frames that can be transmitted in a day comes out to 5.529×10^6, which yields a desired frame error rate of $P_2 = 1/(5.529 \times 10^6) = 0.18 \times 10^{-6}$. But if we assume a value of P_b of 10^{-6}, then $P_1 = (0.999999)^{1000} = 0.999$ and therefore $P_2 = 10^{-3}$, which is about three orders of magnitude too large to meet our requirement.

This is the kind of result that motivates the use of error-detecting techniques. All of these techniques operate on the following principle (Figure 6.3). For a given frame of bits, additional bits that constitute an **error-detecting code** are added by the transmitter. This code is calculated as a function of the other transmitted bits. Typically, for a data block of k bits, the error-detecting algorithm yields an error-detecting code of $n - k$ bits, where $(n - k) < k$. The error-detecting code, also referred to as the **check bits**, is appended to the data block to produce a frame of n bits, which is then

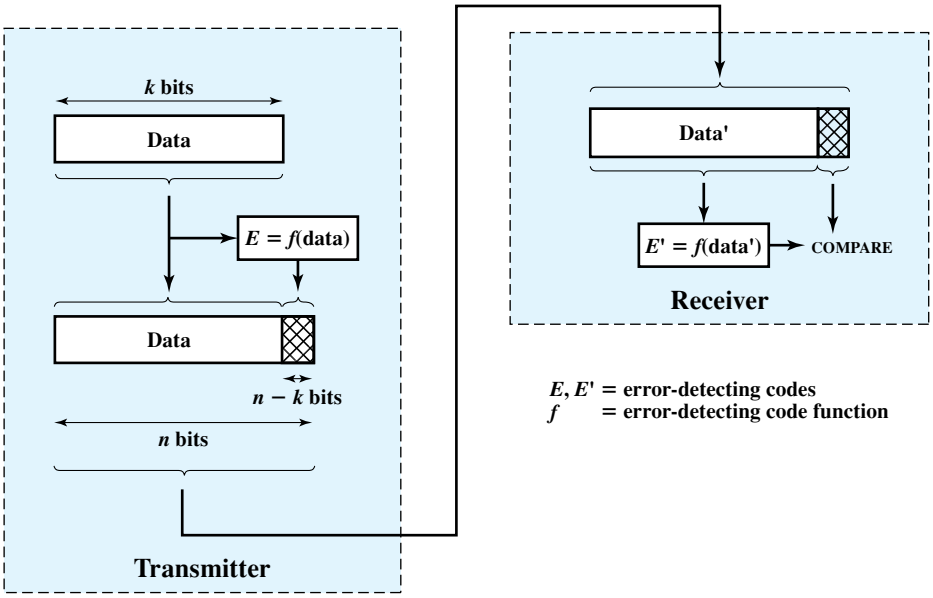


Figure 6.3 Error Detection Process

transmitted. The receiver separates the incoming frame into the k bits of data and $(n - k)$ bits of the error-detecting code. The receiver performs the same error-detecting calculation on the data bits and compares this value with the value of the incoming error-detecting code. A detected error occurs if and only if there is a mismatch. Thus P_3 is the probability that a frame contains errors and that the error-detecting scheme will detect that fact. P_2 is known as the residual error rate and is the probability that an error will be undetected despite the use of an error-detecting scheme.

Parity Check

The simplest error-detecting scheme is to append a parity bit to the end of a block of data. A typical example is character transmission, in which a parity bit is attached to each 7-bit IRA character. The value of this bit is selected so that the character has an even number of 1s (even parity) or an odd number of 1s (odd parity).

EXAMPLE 6.5 If the transmitter is transmitting an IRA character G (1110001) and using odd parity, it will append a 1 and transmit 11110001.[2] The receiver examines the received character and, if the total number of 1s is odd, assumes that no error has occurred. If one bit (or any odd number of bits) is erroneously inverted during transmission (for example, 11100001), then the receiver will detect an error.

[2]Recall from our discussion in Section 5.1 that the least significant bit of a character is transmitted first and that the parity bit is the most significant bit.

Note, however, that if two (or any even number) of bits are inverted due to error, an undetected error occurs. Typically, even parity is used for synchronous transmission and odd parity for asynchronous transmission.

The use of the parity bit is not foolproof, as noise impulses are often long enough to destroy more than one bit, particularly at high data rates.

Cyclic Redundancy Check (CRC)

One of the most common, and one of the most powerful, error-detecting codes is the cyclic redundancy check (CRC), which can be described as follows. Given a k-bit block of bits, or message, the transmitter generates an $(n - k)$-bit sequence, known as a frame check sequence (FCS), such that the resulting frame, consisting of n bits, is exactly divisible by some predetermined number. The receiver then divides the incoming frame by that number and, if there is no remainder, assumes there was no error.[3]

To clarify this, we present the procedure in three equivalent ways: modulo 2 arithmetic, polynomials, and digital logic.

Modulo 2 Arithmetic Modulo 2 arithmetic uses binary addition with no carries, which is just the exclusive-OR (XOR) operation. Binary subtraction with no carries is also interpreted as the XOR operation: For example,

$$\begin{array}{r} 1111 \\ +1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 1111 \\ -0101 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 11001 \\ \times\ 11 \\ \hline 11001 \\ 11001 \\ \hline 101011 \end{array}$$

Now define

T = n-bit frame to be transmitted
D = k-bit block of data, or message, the first k bits of T
F = $(n - k)$-bit FCS, the last $(n - k)$ bits of T
P = pattern of $n - k + 1$ bits; this is the predetermined divisor

We would like T/P to have no remainder. It should be clear that

$$T = 2^{n-k}D + F$$

That is, by multiplying D by 2^{n-k}, we have in effect shifted it to the left by $n - k$ bits and padded out the result with zeroes. Adding F yields the concatenation of D and

[3]This procedure is slightly different from that of Figure 6.3. As shall be seen, the CRC process could be implemented as follows. The receiver could perform a division operation on the incoming k data bits and compare the result to the incoming $(n - k)$ check bits.

F, which is T. We want T to be exactly divisible by P. Suppose that we divide $2^{n-k}D$ by P:

$$\frac{2^{n-k}D}{P} = Q + \frac{R}{P} \tag{6.1}$$

There is a quotient and a remainder. Because division is modulo 2, the remainder is always at least one bit shorter than the divisor. We will use this remainder as our FCS. Then

$$T = 2^{n-k}D + R \tag{6.2}$$

Does this R satisfy our condition that T/P have no remainder? To see that it does, consider

$$\frac{T}{P} = \frac{2^{n-k}D + R}{P} = \frac{2^{n-k}D}{P} + \frac{R}{P}$$

Substituting Equation (6.1), we have

$$\frac{T}{P} = Q + \frac{R}{P} + \frac{R}{P}$$

However, any binary number added to itself modulo 2 yields zero. Thus

$$\frac{T}{P} = Q + \frac{R + R}{P} = Q$$

There is no remainder, and therefore T is exactly divisible by P. Thus, the FCS is easily generated: Simply divide $2^{n-k}D$ by P and use the $(n - k)$-bit remainder as the FCS. On reception, the receiver will divide T by P and will get no remainder if there have been no errors.

EXAMPLE 6.6

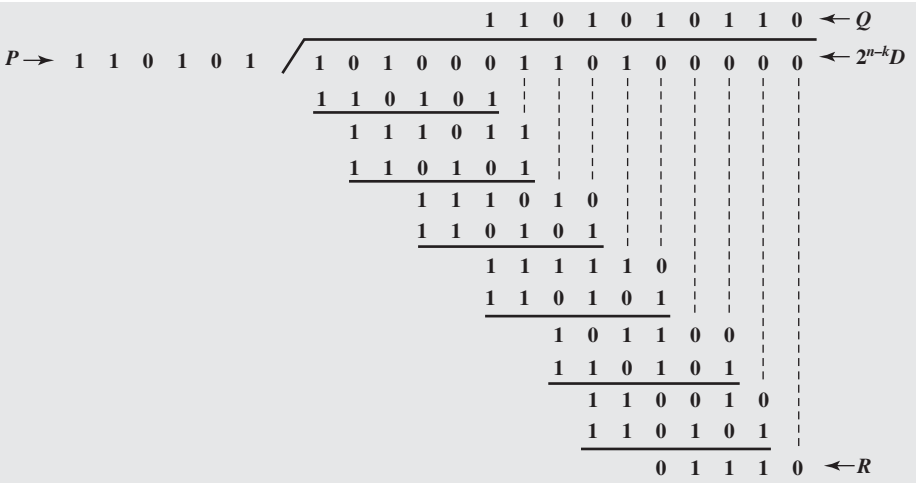
1. Given

$$\begin{aligned} \text{Message } D &= 1010001101 \text{ (10 bits)} \\ \text{Pattern } P &= 110101 \text{ (6 bits)} \\ \text{FCS } R &= \text{to be calculated (5 bits)} \end{aligned}$$

Thus, $n = 15$, $k = 10$, and $(n - k) = 5$.

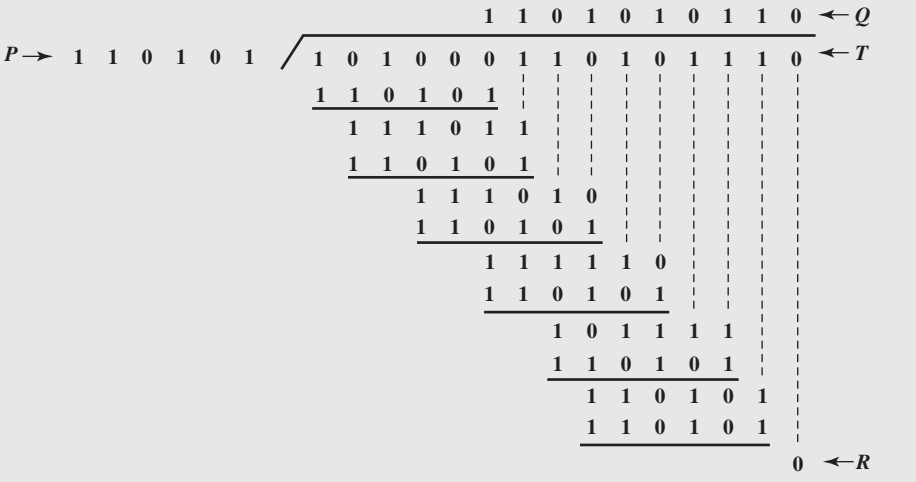
2. The message is multiplied by 2^5, yielding 101000110100000.

3. This product is divided by P:



4. The remainder is added to 2^5D to give $T = 101000110101110$, which is transmitted.

5. If there are no errors, the receiver receives T intact. The received frame is divided by P:



Because there is no remainder, it is assumed that there have been no errors.

The pattern P is chosen to be one bit longer than the desired FCS, and the exact bit pattern chosen depends on the type of errors expected. At minimum, both the high- and low-order bits of P must be 1.

There is a concise method for specifying the occurrence of one or more errors. An error results in the reversal of a bit. This is equivalent to taking the XOR of the

bit and 1 (modulo 2 addition of 1 to the bit): $0 + 1 = 1; 1 + 1 = 0$. Thus, the errors in an n-bit frame can be represented by an n-bit field with 1s in each error position. The resulting frame T_r can be expressed as

$$T_r = T \oplus E$$

where

T = transmitted frame
E = error pattern with 1s in positions where errors occur
T_r = received frame
$\oplus$ = bitwise exclusive-OR(XOR)

If there is an error ($E \neq 0$), the receiver will fail to detect the error if and only if T_r is divisible by P, which is equivalent to E divisible by P. Intuitively, this seems an unlikely occurrence.

Polynomials A second way of viewing the CRC process is to express all values as polynomials in a dummy variable X, with binary coefficients. The coefficients correspond to the bits in the binary number. Thus, for $D = 110011$, we have $D(X) = X^5 + X^4 + X + 1$, and for $P = 11001$, we have $P(X) = X^4 + X^3 + 1$. Arithmetic operations are again modulo 2. The CRC process can now be described as

$$\frac{X^{n-k}D(X)}{P(X)} = Q(X) + \frac{R(X)}{P(X)}$$
$$T(X) = X^{n-k}D(X) + R(X)$$

Compare these equations with Equations (6.1) and (6.2).

EXAMPLE 6.7 Using the preceding example, for $D = 1010001101$, we have $D(X) = X^9 + X^7 + X^3 + X^2 + 1$, and for $P = 110101$, we have $P(X) = X^5 + X^4 + X^2 + 1$. We should end up with $R = 01110$, which corresponds to $R(X) = X^3 + X^2 + X$. Figure 6.4 shows the polynomial division that corresponds to the binary division in the preceding example.

An error $E(X)$ will only be undetectable if it is divisible by $P(X)$. It can be shown [PETE 61, RAMA88] that all of the following errors are not divisible by a suitably chosen $P(X)$ and hence are detectable:

- All single-bit errors, if $P(X)$ has more than one nonzero term
- All double-bit errors, as long as $P(X)$ is a special type of polynomial, called a primitive polynomial, with maximum exponent L, and the frame length is less than $2^L - 1$.

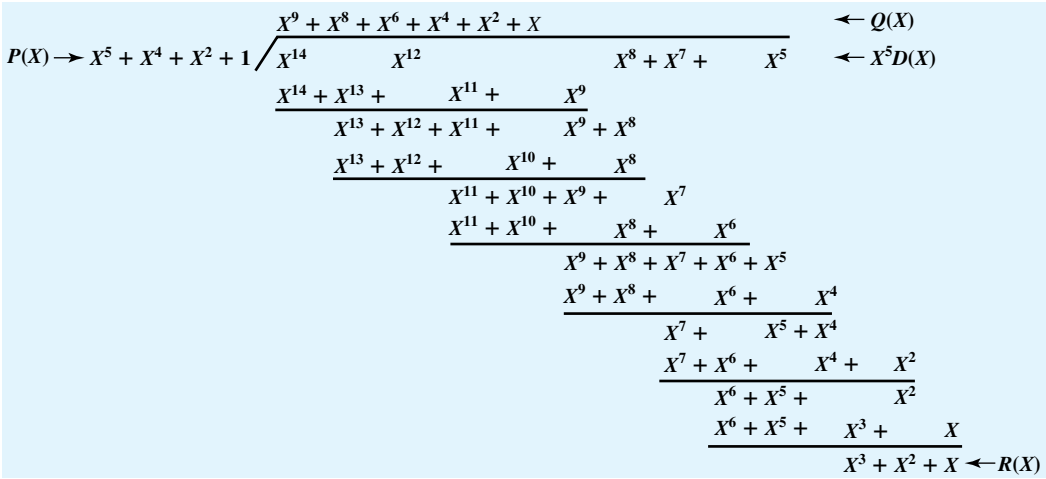


Figure 6.4 Example of Polynomial Division

- Any odd number of errors, as long as $P(X)$ contains a factor $(X + 1)$
- Any burst error for which the length of the burst is less than or equal to $n - k$; that is, less than or equal to the length of the FCS
- A fraction of error bursts of length $n - k + 1$; the fraction equals $1 - 2^{-(n-k-1)}$
- A fraction of error bursts of length greater than $n - k + 1$; the fraction equals $1 - 2^{-(n-k)}$

In addition, it can be shown that if all error patterns are considered equally likely, then for a burst error of length $r + 1$, the probability of an undetected error ($E(X)$ is divisible by $P(X)$) is $1/2^{r-1}$, and for a longer burst, the probability is $1/2^r$, where r is the length of the FCS.

Four versions of $P(X)$ are widely used:

$$\begin{aligned}
\text{CRC-12} &= X^{12} + X^{11} + X^3 + X^2 + X + 1 \\
\text{CRC-16} &= X^{16} + X^{15} + X^2 + 1 \\
\text{CRC-CCITT} &= X^{16} + X^{12} + X^5 + 1 \\
\text{CRC-32} &= X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} \\
&\quad + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1
\end{aligned}$$

The CRC-12 system is used for transmission of streams of 6-bit characters and generates a 12-bit FCS. Both CRC-16 and CRC-CCITT are popular for 8-bit characters, in the United States and Europe, respectively, and both result in a 16-bit FCS. This would seem adequate for most applications, although CRC-32 is specified as an option in some point-to-point synchronous transmission standards and is used in IEEE 802 LAN standards.

Digital Logic The CRC process can be represented by, and indeed implemented as, a dividing circuit consisting of XOR gates and a shift register. The shift register is a string of 1-bit storage devices. Each device has an output line, which indicates the value currently stored, and an input line. At discrete time instants, known as clock times, the value in the storage device is replaced by the value indicated by its input line. The entire register is clocked simultaneously, causing a 1-bit shift along the entire register. The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.

EXAMPLE 6.8 The architecture of a CRC circuit is best explained by first considering an example, which is illustrated in Figure 6.5. In this example, we use

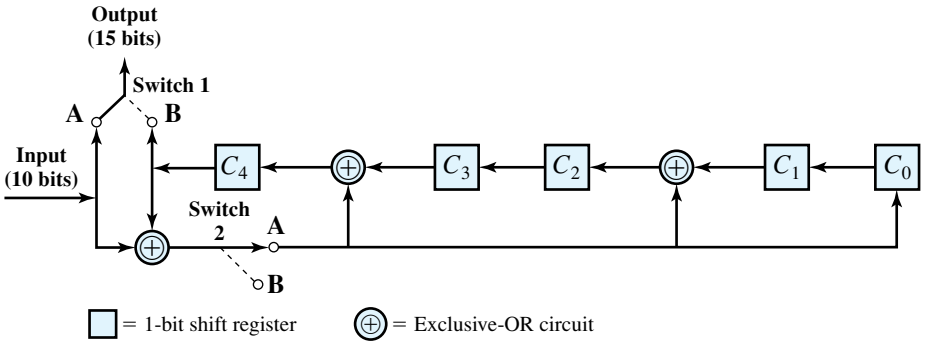
$$\text{Data } D = 1010001101; \qquad D(X) = X^9 + X^7 + X^3 + X^2 + 1$$
$$\text{Divisor } P = 110101; \qquad P(X) = X^5 + X^4 + X^2 + 1$$

which were used earlier in the discussion.

Figure 6.5a shows the shift register implementation. The process begins with the shift register cleared (all zeros). The message, or dividend, is then entered, one bit at a time, starting with the most significant bit. Figure 6.5b is a table that shows the step-by-step operation as the input is applied one bit at a time. Each row of the table shows the values currently stored in the five shift-register elements. In addition, the row shows the values that appear at the outputs of the three XOR circuits. Finally, the row shows the value of the next input bit, which is available for the operation of the next step.

Note that the XOR operation affects C_4, C_2, and C_0 on the next shift. This is identical to the binary long division process illustrated earlier. The process continues through all the bits of the message. To produce the proper output, two switches are used. The input data bits are fed in with both switches in the A position. As a result, for the first 10 steps, the input bits are fed into the shift register and also used as output bits. After the last data bit is processed, the shift register contains the remainder (FCS) (shown shaded). As soon as the last data bit is provided to the shift register, both switches are set to the B position. This has two effects: (1) All of the XOR gates become simple pass-throughs; no bits are changed, and (2) as the shifting process continues, the 5 CRC bits are output.

At the receiver, the same logic is used. As each bit of M arrives, it is inserted into the shift register. If there have been no errors, the shift register should contain the bit pattern for R at the conclusion of M. The transmitted bits of R now begin to arrive, and the effect is to zero out the register so that, at the conclusion of reception, the register contains all 0s.



(a) Shift-register implementation

	C_4	C_3	C_2	C_1	C_0	$C_4 \oplus C_3 \oplus I$	$C_4 \oplus C_1 \oplus I$	$C_4 \oplus I$	I = input	
Initial	0	0	0	0	0	1	1	1	1	
Step 1	1	0	1	0	1	1	1	1	0	
Step 2	1	1	1	1	1	1	1	0	1	
Step 3	1	1	1	1	0	0	0	1	0	
Step 4	0	1	0	0	1	1	0	0	0	Message to be sent
Step 5	1	0	0	1	0	1	0	1	0	
Step 6	1	0	0	0	1	0	0	0	1	
Step 7	0	0	0	1	0	1	0	1	1	
Step 8	1	0	0	0	1	1	1	1	0	
Step 9	1	0	1	1	1	0	1	0	1	
Step 10	0	1	1	1	0					

(b) Example with input of 1010001101

Figure 6.5 Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$

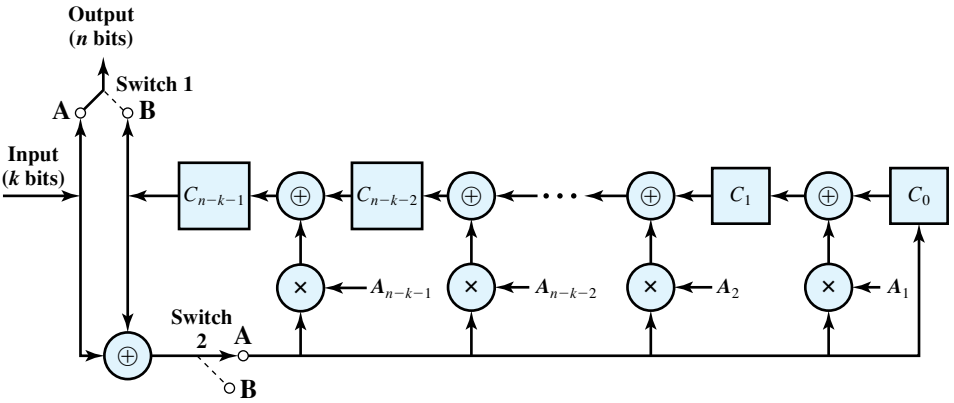


Figure 6.6 General CRC Architecture to Implement Divisor $(1 + A_1X + A_2X^2 + \cdots + A_{n-1}X^{n-k-1} + X^{n-k})$

Figure 6.6 indicates the general architecture of the shift register implementation of a CRC for the polynomial $P(X) = \sum_{i=0}^{n-k} A_i X^i$, where $A_0 = A_{n-k} = 1$ and all other A_i equal either 0 or 1.[4]

6.4 ERROR CORRECTION

Error detection is a useful technique, found in data link control protocols, such as HDLC, and in transport protocols, such as TCP. However, correction of errors using an error-detecting code, requires that block of data be retransmitted, as explained in Chapter 7. For wireless applications this approach is inadequate for two reasons.

1. The bit error rate on a wireless link can be quite high, which would result in a large number of retransmissions.
2. In some cases, especially satellite links, the propagation delay is very long compared to the transmission time of a single frame. The result is a very inefficient system. As is discussed in Chapter 7, the common approach to retransmission is to retransmit the frame in error plus all subsequent frames. With a long data link, an error in a single frame necessitates retransmitting many frames.

Instead, it would be desirable to enable the receiver to correct errors in an incoming transmission on the basis of the bits in that transmission. Figure 6.7 shows in general how this is done. On the transmission end, each k-bit block of data is

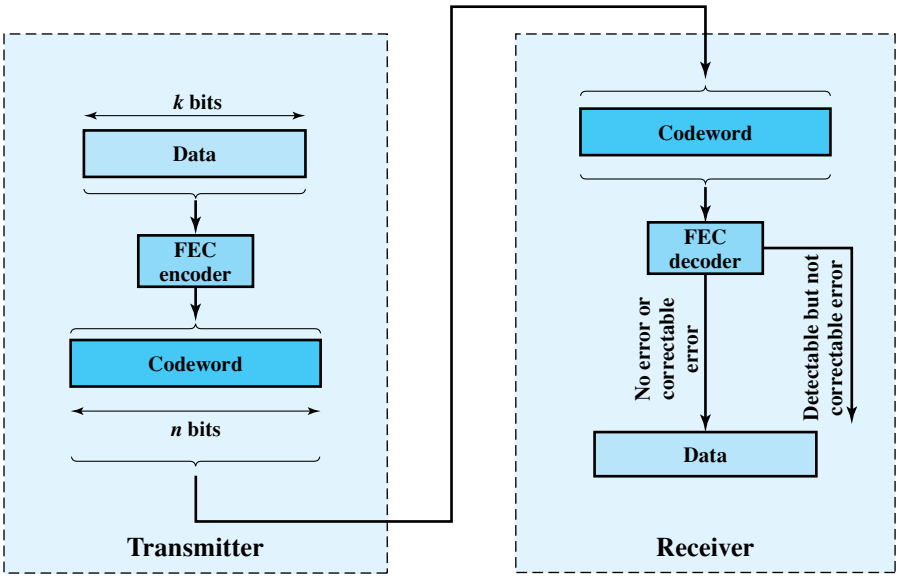


Figure 6.7 Error Correction Process

[4]It is common for the CRC register to be shown shifting to the right, which is the reverse of the analogy to binary division. Because binary numbers are usually shown with the most significant bit on the left, a left-shifting register, as is used here, is more appropriate.

mapped into an n-bit block ($n > k$) called a **codeword**, using an FEC (forward error correction) encoder. The codeword is then transmitted. During transmission, the signal is subject to impairments, which may produce bit errors in the signal. At the receiver, the incoming signal is demodulated to produce a bit string that is similar to the original codeword but may contain errors. This block is passed through an FEC decoder, with one of four possible outcomes:

1. If there are no bit errors, the input to the FEC decoder is identical to the original codeword, and the decoder produces the original data block as output.
2. For certain error patterns, it is possible for the decoder to detect and correct those errors. Thus, even though the incoming data block differs from the transmitted codeword, the FEC decoder is able to map this block into the original data block.
3. For certain error patterns, the decoder can detect but not correct the errors. In this case, the decode simply reports an uncorrectable error.
4. For certain, typically rare, error patterns, the decoder does not detect that any errors have occurred and maps the incoming n-bit data block into a k-bit block that differs from the original k-bit block.

How is it possible for the decoder to correct bit errors? In essence, error correction works by adding redundancy to the transmitted message. The redundancy makes it possible for the receiver to deduce what the original message was, even in the face of a certain level of error rate. In this section we look at a widely used form of error-correcting code known as a block error-correcting code. Our discussion only deals with basic principles; a discussion of specific error-correcting codes is beyond our scope.

Before proceeding, we note that in many cases, the error-correcting code follows the same general layout as shown for error-detecting codes in Figure 6.3. That is, the FEC algorithm takes as input a k-bit block and adds $(n - k)$ check bits to that block to produce an n-bit block; all of the bits in the original k-bit block show up in the n-bit block. For some FEC algorithms, the FEC algorithm maps the k-bit input into an n-bit codeword in such a way that the original k bits do not appear in the codeword.

Block Code Principles

To begin, we define a term that shall be of use to us. The **Hamming distance** $d(\mathbf{v}_1, \mathbf{v}_2)$ between two n-bit binary sequences $\mathbf{v}_1$ and $\mathbf{v}_2$ is the number of bits in which $\mathbf{v}_1$ and $\mathbf{v}_2$ disagree. For example, if

$$\mathbf{v}_1 = 011011, \qquad \mathbf{v}_2 = 110001$$

then

$$d(\mathbf{v}_1, \mathbf{v}_2) = 3$$

Now let us consider the block code technique for error correction. Suppose we wish to transmit blocks of data of length k bits. Instead of transmitting each block as k bits, we map each k-bit sequence into a unique n-bit codeword.

EXAMPLE 6.9

For $k = 2$ and $n = 5$, we can make the following assignment:

Data Block	Codeword
00	00000
01	00111
10	11001
11	11110

Now, suppose that a codeword block is received with the bit pattern 00100. This is not a valid codeword, and so the receiver has detected an error. Can the error be corrected? We cannot be sure which data block was sent because 1, 2, 3, 4, or even all 5 of the bits that were transmitted may have been corrupted by noise. However, notice that it would require only a single bit change to transform the valid codeword 00000 into 00100. It would take two bit changes to transform 00111 to 00100, three bit changes to transform 11110 to 00100, and it would take four bit changes to transform 11001 into 00100. Thus, we can deduce that the most likely codeword that was sent was 00000 and that therefore the desired data block is 00. This is error correction. In terms of Hamming distances, we have

$$d(00000, 00100) = 1; \quad d(00111, 00100) = 2;$$
$$d(11001, 00100) = 4; \quad d(11110, 00100) = 3$$

So the rule we would like to impose is that if an invalid codeword is received, then the valid codeword that is closest to it (minimum distance) is selected. This will only work if there is a unique valid codeword at a minimum distance from each invalid codeword.

For our example, it is not true that for every invalid codeword there is one and only one valid codeword at a minimum distance. There are $2^5 = 32$ possible codewords of which 4 are valid, leaving 28 invalid codewords. For the invalid codewords, we have the following:

Invalid Codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	00000 or 11110
00100	1	00000	10011	2	00111 or 11001
00101	1	00111	10100	2	00000 or 11110
00110	1	00111	10101	2	00111 or 11001
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	00000 or 11110	11000	1	11001
01011	2	00111 or 11001	11010	1	11110

01100	2	00000 or 11110	11011	1	11001
01101	2	00111 or 11001	11100	1	11110
01110	1	11110	11101	1	11001
01111	1	00111	11111	1	11110

There are eight cases in which an invalid codeword is at a distance 2 from two different valid codewords. Thus, if one such invalid codeword is received, an error in 2 bits could have caused it and the receiver has no way to choose between the two alternatives. An error is detected but cannot be corrected. However, in every case in which a single bit error occurs, the resulting codeword is of distance 1 from only one valid codeword and the decision can be made. This code is therefore capable of correcting all single-bit errors but cannot correct double bit errors. Another way to see this is to look at the pairwise distances between valid codewords:

$$d(00000, 00111) = 3; \quad d(00000, 11001) = 3; \quad d(00000, 11110) = 4;$$
$$d(00111, 11001) = 4; \quad d(00111, 11110) = 3; \quad d(11001, 11110) = 3;$$

The minimum distance between valid codewords is 3. Therefore, a single bit error will result in an invalid codeword that is a distance 1 from the original valid codeword but a distance at least 2 from all other valid codewords. As a result, the code can always correct a single-bit error. Note that the code also will always detect a double-bit error.

The preceding example illustrates the essential properties of a block error-correcting code. An (n, k) block code encodes k data bits into n-bit codewords. Typically, each valid codeword reproduces the original k data bits and adds to them $(n - k)$ check bits to form the n-bit codeword. Thus the design of a block code is equivalent to the design of a function of the form $\mathbf{v_c} = f(\mathbf{v_d})$, where $\mathbf{v_d}$ is a vector of k data bits and $\mathbf{v_c}$ is a vector of n codeword bits.

With an (n, k) block code, there are 2^k valid codewords out of a total of 2^n possible codewords. The ratio of redundant bits to data bits, $(n - k)/k$, is called the **redundancy** of the code, and the ratio of data bits to total bits, k/n, is called the **code rate**. The code rate is a measure of how much additional bandwidth is required to carry data at the same data rate as without the code. For example, a code rate of 1/2 requires double the transmission capacity of an uncoded system to maintain the same data rate. Our example has a code rate of 2/5 and so requires 2.5 times the capacity of an uncoded system. For example, if the data rate input to the encoder is 1 Mbps, then the output from the encoder must be at a rate of 2.5 Mbps to keep up.

For a code consisting of the codewords $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_s$, where $s = 2^n$, the minimum distance $d_{\min}$ of the code is defined as

$$d_{\min} = \min_{i \neq j} [d(\mathbf{w}_i, \mathbf{w}_j)]$$

It can be shown that the following conditions hold. For a given positive integer t, if a code satisfies $d_{\min} \geq (2t + 1)$, then the code can correct all bit errors up to and including errors of t bits. If $d_{\min} \geq 2t$, then all errors $\leq (t - 1)$ bits can be corrected and errors of t bits can be detected but not, in general, corrected. Conversely, any code for which all errors of magnitude $\leq t$ are corrected must satisfy $d_{\min} \geq (2t + 1)$, and any code for which all errors of magnitude $\leq (t - 1)$ are corrected and all errors of magnitude t are detected must satisfy $d_{\min} \geq 2t$.

Another way of putting the relationship between $d_{\min}$ and t is to say that the maximum number of guaranteed correctable errors per codeword satisfies

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

where $\lfloor x \rfloor$ means the largest integer not to exceed x (e.g., $\lfloor 6.3 \rfloor = 6$). Furthermore, if we are concerned only with error detection and not error correction, then the number of errors, t, that can be detected satisfies

$$t = d_{\min} - 1$$

To see this, consider that if $d_{\min}$ errors occur, this could change one valid codeword into another. Any number of errors less than $d_{\min}$ can not result in another valid codeword.

The design of a block code involves a number of considerations.

1. For given values of n and k, we would like the largest possible value of $d_{\min}$.
2. The code should be relatively easy to encode and decode, requiring minimal memory and processing time.
3. We would like the number of extra bits, $(n - k)$, to be small, to reduce bandwidth.
4. We would like the number of extra bits, $(n - k)$, to be large, to reduce error rate.

Clearly, the last two objectives are in conflict, and tradeoffs must be made.

It is instructive to examine Figure 6.8, based on [LEBO98]. The literature on error-correcting codes frequently includes graphs of this sort to demonstrate the effectiveness of various encoding schemes. Recall from Chapter 5 that coding can be used to reduce the required E_b/N_0 value to achieve a given bit error rate.[5] The coding discussed in Chapter 5 has to do with the definition of signal elements to represent bits. The coding discussed in this chapter also has an effect on E_b/N_0. In Figure 6.8, the curve on the right is for an uncoded modulation system; the shaded region represents the area in which improvement can be achieved. In this region, a smaller BER (bit error rate) is achieved for a given E_b/N_0, and conversely, for a given BER, a smaller E_b/N_0 is required. The other curve is a typical result of a code rate of one-half (equal number of data and check bits). Note that at an error rate of 10^{-6}, the use of coding allows a reduction in E_b/N_0 of 2.77 dB. This reduction is referred to as

[5] E_b/N_0 is the ratio of signal energy per bit to noise power density per Hertz; it is defined and discussed in Chapter 3.

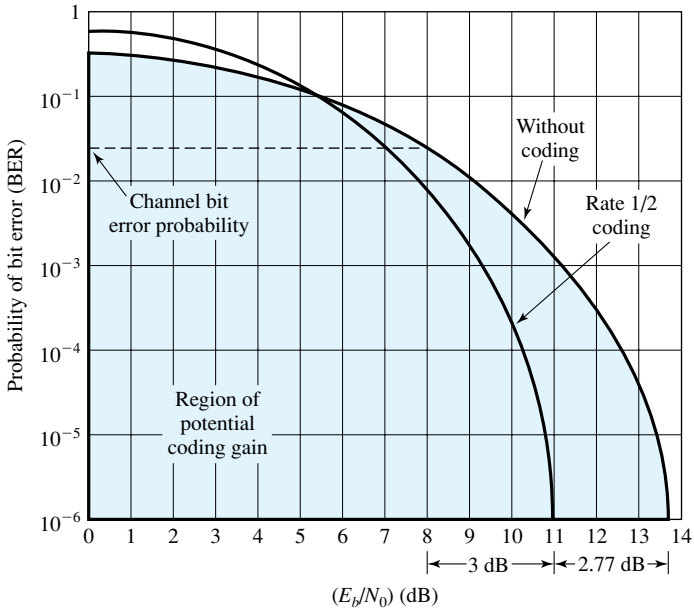


Figure 6.8 How Coding Improves System Performance

the **coding gain**, which is defined as the reduction, in decibels, in the required E_b/N_0 to achieve a specified BER of an error-correcting coded system compared to an uncoded system using the same modulation.

It is important to realize that the BER for the second rate 1/2 curve refers to the rate of uncorrected errors and that the E_b value refers to the energy per data bit. Because the rate is 1/2, there are two bits on the channel for each data bit, and the energy per coded bit is half that of the energy per data bit, or a reduction of 3 dB to a value of 8 dB. If we look at the energy per coded bit for this system, then we see that the channel bit error rate is about 2.4×10^{-2}, or 0.024.

Finally, note that below a certain threshold of E_b/N_0, the coding scheme actually degrades performance. In our example of Figure 6.8, the threshold occurs at about 5.4 dB. Below the threshold, the extra check bits add overhead to the system that reduces the energy per data bit causing increased errors. Above the threshold, the error-correcting power of the code more than compensates for the reduced E_b, resulting in a coding gain.

6.5 LINE CONFIGURATIONS

Two characteristics that distinguish various data link configurations are topology and whether the link is half duplex or full duplex.

Topology

The topology of a data link refers to the physical arrangement of stations on a transmission medium. If there are only two stations (e.g., a terminal and a computer or

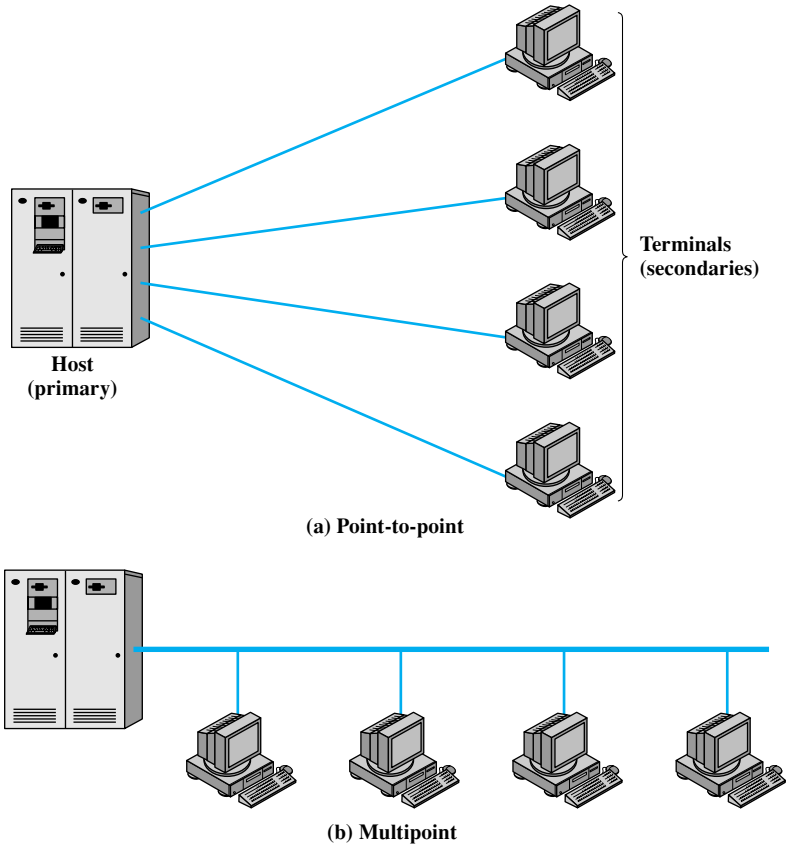


Figure 6.9 Traditional Computer/Terminal Configurations

two computers), the link is point to point. If there are more than two stations, then it is a multipoint topology. Traditionally, a multipoint link has been used in the case of a computer (primary station) and a set of terminals (secondary stations). In today's environments, the multipoint topology is found in local area networks.

Traditional multipoint topologies are made possible when the terminals are only transmitting a fraction of the time. Figure 6.9 illustrates the advantages of the multipoint configuration. If each terminal has a point-to-point link to its computer, then the computer must have one I/O port for each terminal. Also there is a separate transmission line from the computer to each terminal. In a multipoint configuration, the computer needs only a single I/O port and a single transmission line, which saves costs.

Full Duplex and Half Duplex

Data exchanges over a transmission line can be classified as full duplex or half duplex. With **half-duplex transmission**, only one of two stations on a point-to-point link may transmit at a time. This mode is also referred to as *two-way alternate*, suggestive of the fact that two stations must alternate in transmitting. This can be

compared to a one-lane, two-way bridge. This form of transmission is often used for terminal-to-computer interaction. While a user is entering and transmitting data, the computer is prevented from sending data to the terminal, which would appear on the terminal screen and cause confusion.

For **full-duplex transmission**, two stations can simultaneously send and receive data from each other. Thus, this mode is known as *two-way simultaneous* and may be compared to a two-lane, two-way bridge. For computer-to-computer data exchange, this form of transmission is more efficient than half-duplex transmission.

With digital signaling, which requires guided transmission, full-duplex operation usually requires two separate transmission paths (e.g., two twisted pairs), while half duplex requires only one. For analog signaling, it depends on frequency: If a station transmits and receives on the same frequency, it must operate in half-duplex mode for wireless transmission, although it may operate in full-duplex mode for guided transmission using two separate transmission lines. If a station transmits on one frequency and receives on another, it may operate in full-duplex mode for wireless transmission and in full-duplex mode with a single line for guided transmission.

It is possible to transmit digital signals simultaneously in both directions on a single transmission line using a technique called echo cancellation. This is a signal processing technique whose explanation is beyond the scope of this book.

6.6 RECOMMENDED READING

The classic treatment of error detecting codes and CRC is [PETE61]. [RAMA88] is an excellent tutorial on CRC.

[STAL05] discusses most of the widely used error-correcting codes. [ADAM91] provides comprehensive treatment of error-correcting codes. [SKLA01] contains a clear, well-written section on the subject. Two useful survey articles are [BERL87] and [BHAR83]. A quite readable theoretical and mathematical treatment of error-correcting codes is [ASH90].

[FREE98] provides good coverage of many physical layer interface standards.

ADAM91 Adamek, J. *Foundations of Coding.* New York: Wiley, 1991.

ASH90 Ash, R. *Information Theory.* New York: Dover, 1990.

BERL87 Berlekamp, E.; Peile, R.; and Pope, S. "The Application of Error Control to Communications." *IEEE Communications Magazine*, April 1987.

BHAR83 Bhargava, V. "Forward Error Correction Schemes for Digital Communications." *IEEE Communications Magazine*, January 1983.

FREE98 Freeman, R. *Telecommunication Transmission Handbook.* New York: Wiley, 1998.

PETE61 Peterson, W., and Brown, D. "Cyclic Codes for Error Detection." *Proceedings of the IEEE*, January 1961.

RAMA88 Ramabadran, T., and Gaitonde, S. "A Tutorial on CRC Computations." *IEEE Micro*, August 1988.

SKLA01 Sklar, B. *Digital Communications: Fundamentals and Applications.* Upper Saddle River, NJ: Prentice Hall, 2001.

STAL05 Stallings, W. *Wireless Communications and Networks, Second Edition.* Upper Saddle River, NJ: Prentice Hall, 2005.

6.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

asynchronous transmission	error-detecting code	interchange circuits
codeword	forward error correction (FEC)	Integrated Services Digital Network (ISDN)
cyclic code	frame	modem
cyclic redundancy check (CRC)	frame check sequence (FCS)	parity bit
EIA-232	full duplex	parity check
error correction	half duplex	point-to-point
error-correcting code (ECC)	Hamming code	synchronous transmission
error detection	Hamming distance	

Review Questions

6.1. How is the transmission of a single character differentiated from the transmission of the next character in asynchronous transmission?
6.2. What is a major disadvantage of asynchronous transmission?
6.3. How is synchronization provided for synchronous transmission?
6.4. What is a parity bit?
6.5. What is the CRC?
6.6. Why would you expect a CRC to detect more errors than a parity bit?
6.7. List three different ways in which the CRC algorithm can be described.
6.8. Is it possible to design an ECC that will correct some double bit errors but not all double bit errors? Why or why not?
6.9. In an (n, k) block ECC, what do n and k represent?

Problems

6.1 Suppose a file of 10,000 bytes is to be sent over a line at 2400 bps.
- **a.** Calculate the overhead in bits and time in using asynchronous communication. Assume one start bit and a stop element of length one bit, and 8 bits to send the byte itself for each character. The 8-bit character consists of all data bits, with no parity bit.
- **b.** Calculate the overhead in bits and time using synchronous communication. Assume that the data are sent in frames. Each frame consists of 1000 characters = 8000 bits and an overhead of 48 control bits per frame.
- **c.** What would the answers to parts (a) and (b) be for a file of 100,000 characters?
- **d.** What would the answers to parts (a) and (b) be for the original file of 10,000 characters except at a data rate of 9600 bps?

6.2 A data source produces 7-bit IRA characters. Derive an expression of the maximum effective data rate (rate of IRA data bits) over an x-bps line for the following:
- **a.** Asynchronous transmission, with a 1.5-unit stop element and a parity bit.
- **b.** Synchronous transmission, with a frame consisting of 48 control bits and 128 information bits. The information field contains 8-bit (parity included) IRA characters.
- **c.** Same as part (b), except that the information field is 1024 bits.

6.3 Demonstrate by example (write down a few dozen arbitrary bit patterns; assume one start bit and a stop element of length one bit) that a receiver that suffers a framing error on asynchronous transmission will eventually become realigned.

6.4 Suppose that a sender and receiver use asynchronous transmission and agree not to use any stop elements. Could this work? If so, explain any necessary conditions.

6.5 An asynchronous transmission scheme uses 8 data bits, an even parity bit, and a stop element of length 2 bits. What percentage of clock inaccuracy can be tolerated at the receiver with respect to the framing error? Assume that the bit samples are taken at the middle of the clock period. Also assume that at the beginning of the start bit the clock and incoming bits are in phase.

6.6 Suppose that a synchronous serial data transmission is clocked by two clocks (one at the sender and one at the receiver) that each have a drift of 1 minute in one year. How long a sequence of bits can be sent before possible clock drift could cause a problem? Assume that a bit waveform will be good if it is sampled within 40% of its center and that the sender and receiver are resynchronized at the beginning of each frame. Note that the transmission rate is not a factor, as both the bit period and the absolute timing error decrease proportionately at higher transmission rates.

6.7 Would you expect that the inclusion of a parity bit with each character would change the probability of receiving a correct message?

6.8 Two communicating devices are using a single-bit even parity check for error detection. The transmitter sends the byte 10101010 and, because of channel noise, the receiver gets the byte 10011010. Will the receiver detect the error? Why or why not?

6.9 What is the purpose of using modulo 2 arithmetic rather than binary arithmetic in computing an FCS?

6.10 Consider a frame consisting of two characters of four bits each. Assume that the probability of bit error is 10^{-3} and that it is independent for each bit.

a. What is the probability that the received frame contains at least one error?
b. Now add a parity bit to each character. What is the probability?

6.11 Using the CRC-CCITT polynomial, generate the 16-bit CRC code for a message consisting of a 1 followed by 15 0s.

a. Use long division.
b. Use the shift register mechanism shown in Figure 6.6.

6.12 Explain in words why the shift register implementation of CRC will result in all 0s at the receiver if there are no errors. Demonstrate by example.

6.13 For $P = 110011$ and $M = 11100011$, find the CRC.

6.14 A CRC is constructed to generate a 4-bit FCS for an 11-bit message. The generator polynomial is $X^4 + X^3 + 1$.

a. Draw the shift register circuit that would perform this task (see Figure 6.6).
b. Encode the data bit sequence 10011011100 (leftmost bit is the least significant) using the generator polynomial and give the codeword.
c. Now assume that bit 7 (counting from the LSB) in the codeword is in error and show that the detection algorithm detects the error.

6.15 **a.** In a CRC error-detecting scheme, choose $P(x) = x^4 + x + 1$. Encode the bits 10010011011.
b. Suppose the channel introduces an error pattern 100010000000000 (i.e., a flip from 1 to 0 or from 0 to 1 in position 1 and 5). What is received? Can the error be detected?
c. Repeat part (b) with error pattern 100110000000000.

6.16 A modified CRC procedure is commonly used in communications standards. It is defined as follows:

$$\frac{X^{16}D(X) + X^k L(X)}{P(X)} = Q + \frac{R(X)}{P(X)}$$

$$\text{FCS} = L(X) + R(X)$$

where

$$L(X) = X^{15} + X^{14} + X^{13} + \cdots + X + 1$$

and k is the number of bits being checked (address, control, and information fields).

a. Describe in words the effect of this procedure.
b. Explain the potential benefits.
c. Show a shift register implementation for $P(X) = X^{16} + X^{12} + X^5 + 1$.

6.17 Calculate the Hamming pairwise distances among the following codewords:

a. 00000, 10101, 01010
b. 000000, 010101, 101010, 110110

6.18 Section 6.4 discusses block error-correcting codes that make a decision on the basis of minimum distance. That is, given a code consisting of s equally likely codewords of length n, for each received sequence $\mathbf{v}$, the receiver selects the codeword $\mathbf{w}$ for which the distance $d(\mathbf{w}, \mathbf{v})$ is a minimum. We would like to prove that this scheme is "ideal" in the sense that the receiver always selects the codeword for which the probability of $\mathbf{w}$ given $\mathbf{v}$, $p(\mathbf{w}|\mathbf{v})$, is a maximum. Because all codewords are assumed equally likely, the codeword that maximizes $p(\mathbf{w}|\mathbf{v})$ is the same as the codeword that maximizes $p(\mathbf{v}|\mathbf{w})$.

a. In order that $\mathbf{w}$ be received as $\mathbf{v}$, there must be exactly $d(\mathbf{w}, \mathbf{v})$ errors in transmission, and these errors must occur in those bits where $\mathbf{w}$ and $\mathbf{v}$ disagree. Let β be the probability that a given bit is transmitted incorrectly and n be the length of a codeword. Write an expression for $p(\mathbf{v}|\mathbf{w})$ as a function of β, $d(\mathbf{w}, \mathbf{v})$, and n. *Hint:* The number of bits in error is $d(\mathbf{w}, \mathbf{v})$ and the number of bits not in error is $n - d(\mathbf{w}, \mathbf{v})$.
b. Now compare $p(\mathbf{v}|\mathbf{w}_1)$ and $\mathrm{p}(\mathbf{v}|\mathbf{w}_2)$ for two different codewords $\mathbf{w}_1$ and $\mathbf{w}_2$ by calculating $p(\mathbf{v}|\mathbf{w}_1)/\mathrm{p}(\mathbf{v}|\mathbf{w}_2)$.
c. Assume that $0 < \beta < 0.5$ and show that $p(\mathbf{v}|\mathbf{w}_1) > p(\mathbf{v}|\mathbf{w}_2)$ if and only if $d(\mathbf{v}, \mathbf{w}_1) < \mathrm{d}(\mathbf{v}, \mathbf{w}_2)$. This proves that the codeword $\mathbf{w}$ that gives the largest value of $p(\mathbf{v}|\mathbf{w})$ is that word whose distance from $\mathbf{v}$ is a minimum.

6.19 Section 6.4 states that for a given positive integer t, if a code satisfies $d_{\min} \geq 2t + 1$, then the code can correct all bit errors up to and including errors of t bits. Prove this assertion. *Hint:* Start by observing that for a codeword $\mathbf{w}$ to be decoded as another codeword $\mathbf{w}'$, the received sequence must be at least as close to $\mathbf{w}'$ as to $\mathbf{w}$.

Note: The remaining problems concern material in Appendix G.

6.20 Draw a timing diagram showing the state of all EIA-232 leads between two DTE-DCE pairs during the course of a data call on the switched telephone network.

6.21 Explain the operation of each null modem connection in Figure G.5.

6.22 For the V.24/EIA-232 Remote Loopback circuit to function properly, what circuits must be logically connected?