ERROR-CONTROL CODING

This chapter is the natural sequel to the preceding chapter on Shannon's information theory. In particular, in this chapter we present error-control coding techniques that provide different ways of implementing Shannon's channel-coding theorem. Each error-control coding technique involves the use of a channel encoder in the transmitter and a decoding algorithm in the receiver.

The error-control coding techniques described herein include the following important classes of codes:

- *Linear block codes.*
- *Cyclic codes.*
- *Convolutional codes.*
- *Compound codes exemplified by turbo codes and low-density parity-check codes, and their irregular variants.*

10.1 *Introduction*

The task facing the designer of a digital communication system is that of providing a cost-effective facility for transmitting information from one end of the system at a rate and a level of reliability and quality that are acceptable to a user at the other end. The two key system parameters available to the designer are transmitted signal power and channel bandwidth. These two parameters, together with the power spectral density of receiver noise, determine the signal energy per bit-to-noise power spectral density ratio E_b/N_0. In Chapter 6, we showed that this ratio uniquely determines the bit error rate for a particular modulation scheme. Practical considerations usually place a limit on the value that we can assign to E_b/N_0. Accordingly, in practice, we often arrive at a modulation scheme and find that it is not possible to provide acceptable data quality (i.e., low enough error performance). For a fixed E_b/N_0, the only practical option available for changing data quality from problematic to acceptable is to use *error-control coding*.

Another practical motivation for the use of coding is to reduce the required E_b/N_0 for a fixed bit error rate. This reduction in E_b/N_0 may, in turn, be exploited to reduce the required transmitted power or reduce the hardware costs by requiring a smaller antenna size in the case of radio communications.

Error control[1] for data integrity may be exercised by means of *forward error correction* (FEC). Figure 10.1*a* shows the model of a digital communication system using such an approach. The discrete source generates information in the form of binary symbols. The *channel encoder* in the transmitter accepts message bits and adds *redundancy* according to a prescribed rule, thereby producing encoded data at a higher bit rate. The *channel*

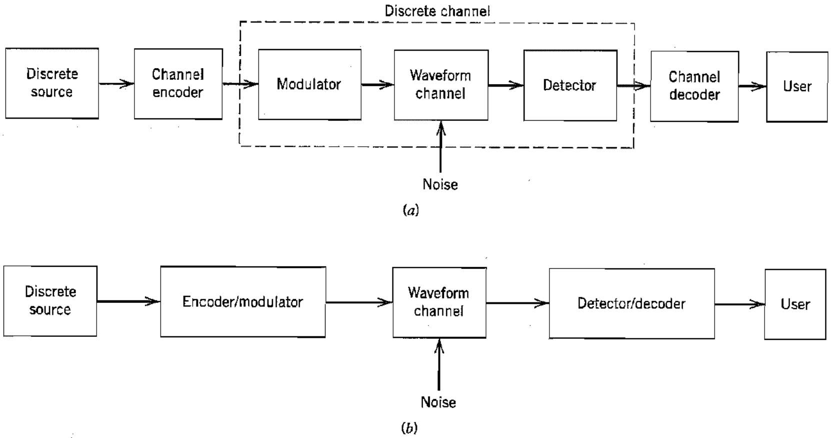


FIGURE 10.1 Simplified models of digital communication system. (*a*) Coding and modulation performed separately. (*b*) Coding and modulation combined.

decoder in the receiver exploits the redundancy to decide which message bits were actually transmitted. The combined goal of the channel encoder and decoder is to minimize the effect of channel noise. That is, the number of errors between the channel encoder input (derived from the source) and the channel decoder output (delivered to the user) is minimized.

For a fixed modulation scheme, the addition of redundancy in the coded messages implies the need for increased transmission bandwidth. Moreover, the use of error-control coding adds *complexity* to the system, especially for the implementation of decoding operations in the receiver. Thus, the design trade-offs in the use of error-control coding to achieve acceptable error performance include considerations of bandwidth and system complexity.

There are many different error-correcting codes (with roots in diverse mathematical disciplines) that we can use. Historically, these codes have been classified into *block codes* and *convolutional codes*. The distinguishing feature for this particular classification is the presence or absence of *memory* in the encoders for the two codes.

To generate an (n, k) block code, the channel encoder accepts information in successive k-bit *blocks*; for each block, it adds $n - k$ redundant bits that are algebraically related to the k message bits, thereby producing an overall encoded block of n bits, where $n > k$. The n-bit block is called a *code word*, and n is called the *block length* of the code. The channel encoder produces bits at the rate $R_0 = (n/k)R_s$, where R_s is the bit rate of the information source. The dimensionless ratio $r = k/n$ is called the *code rate*, where $0 < r < 1$. The bit rate R_0, coming out of the encoder, is called the *channel data rate*. Thus, the code rate is a dimensionless ratio, whereas the data rate produced by the source and the channel data rate are both measured in bits per second.

In a convolutional code, the encoding operation may be viewed as the *discrete-time convolution* of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a "sliding window" equal in duration to its own memory. This, in turn, means that in a convolutional code, unlike a block code, the channel encoder accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.

In the model depicted in Figure 10.1*a*, the operations of channel coding and modulation are performed separately in the transmitter; likewise for the operations of detection and decoding in the receiver. When, however, bandwidth efficiency is of major concern, the most effective method of implementing forward error-control correction coding is to combine it with modulation as a single function, as shown in Figure 10.1*b*. In such an approach, coding is redefined as a process of imposing certain patterns on the transmitted signal.

▪ AUTOMATIC-REPEAT REQUEST

Feed-forward error correction (FEC) relies on the controlled use of redundancy in the transmitted code word for both the *detection and correction* of errors incurred during the course of transmission over a noisy channel. Irrespective of whether the decoding of the received code word is successful, no further processing is performed at the receiver. Accordingly, channel coding techniques suitable for FEC require only a *one-way link* between the transmitter and receiver.

There is another approach known as *automatic-repeat request (ARQ)*[2] for solving the error-control problem. The underlying philosophy of ARQ is quite different from that of FEC. Specifically, ARQ uses redundancy merely for the purpose of *error detection*. Upon the detection of an error in a transmitted code word, the receiver requests a repeat transmission of the corrupted code word, which necessitates the use of a *return path* (i.e., a feedback channel). As such, ARQ can be used only on *half-duplex* or *full-duplex links*. In a half-duplex link, data transmission over the link can be made in either direction but *not* simultaneously. On the other hand, in a full-duplex link, it is possible for data transmission to proceed over the link in both directions simultaneously.

A half-duplex link uses the simplest ARQ scheme known as the *stop-and-wait strategy*. In this approach, a block of message bits is encoded into a code word and transmitted over the channel. The transmitter then stops and waits for feedback from the receiver. The feedback signal can be acknowledgment of a correct receipt of the code word or a request for transmission of the code word because of an error in its decoding. In the latter case, the transmitter resends the code word in question before moving onto the next block of message bits.

The idling problem in stop-and-wait ARQ results in reduced data throughput, which is alleviated in another type of ARQ known as *continuous ARQ with pullback*. This second strategy uses a full-duplex link, thereby permitting the receiver to send a feedback signal while the transmitter is engaged in sending code words over the forward channel. Specifically, the transmitter continues to send a succession of code words until it receives a request from the receiver (on the feedback channel) for a retransmission. At that point, the transmitter stops, pulls back to the particular code word that was not decoded correctly by the receiver, and retransmits the complete sequence of code words starting with the corrupted one.

In a refined version of continuous ARQ known as the *continuous ARQ with selective repeat*, data throughout is improved further by only retransmitting the code word that was received with detected errors. In other words, the need for retransmitting the successfully received code words following the corrupted code word is eliminated.

The three types of ARQ described here offer trade-offs of their own between the need for a half-duplex or full-duplex link and the requirement for efficient use of communication resources. In any event, they all rely on two premises:

- Error detection, which makes the design of the decoder relatively simple.
- Noiseless feedback channel, which is not a severe restriction because the rate of information flow over the feedback channel is typically quite low.

For these reasons, ARQ is widely used in computer-communication systems.

Nevertheless, the fact that FEC requires only one-way links for its operation makes the FEC much wider in application than ARQ. Moreover, the increased decoding complexity of FEC due to the combined need for error detection and correction is no longer a pressing practical issue because the decoder usually lends itself to microprocessor or VLSI implementation in a cost-effective manner.

10.2 *Discrete-Memoryless Channels*

Returning to the model of Figure 10.1*a*, the waveform channel is said to be memoryless if the detector output in a given interval depends only on the signal transmitted in that interval, and not on any previous transmission. Under this condition, we may model the combination of the modulator, the waveform channel, and the detector as a *discrete memoryless channel.* Such a channel is completely described by the set of transition probabilities $p(j|i)$, where i denotes a modulator input symbol, j denotes a demodulator output symbol, and $p(j|i)$ denotes the probability of receiving symbol j, given that symbol i was sent. (Discrete memoryless channels were described previously at some length in Section 9.5.)

The simplest discrete memoryless channel results from the use of binary input and binary output symbols. When binary coding is used, the modulator has only the binary symbols 0 and 1 as inputs. Likewise, the decoder has only binary inputs if binary quantization of the demodulator output is used, that is, a *hard decision* is made on the demodulator output as to which symbol was actually transmitted. In this situation, we have a *binary symmetric channel* (BSC) with a *transition probability diagram* as shown in Figure 10.2. The binary symmetric channel, assuming a channel noise modeled as additive white Gaussian noise (AWGN) channel, is completely described by the *transition probability p.* The majority of coded digital communication systems employ binary coding with hard-decision decoding, due to the simplicity of implementation offered by such an approach. *Hard-decision decoders*, or *algebraic decoders*, take advantage of the special algebraic

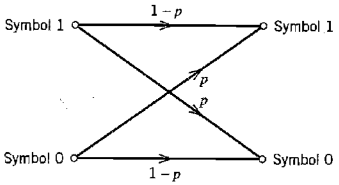
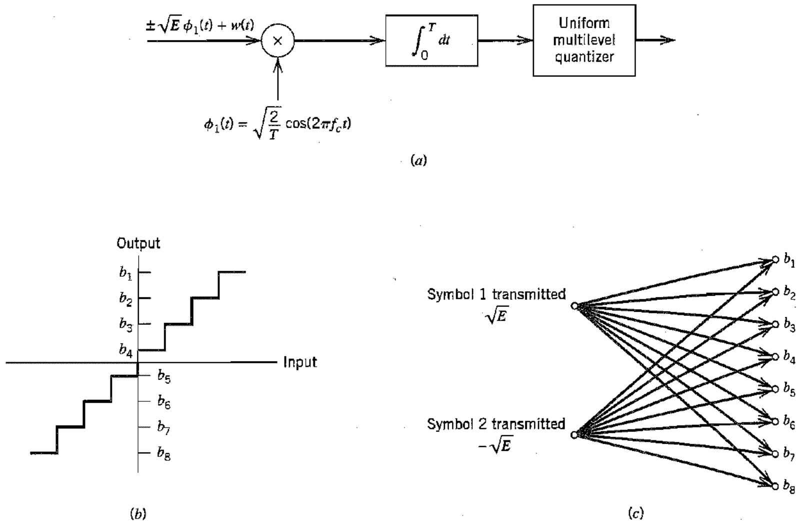


FIGURE 10.2 Transition probability diagram of binary symmetric channel.



$\pm\sqrt{E}\phi_1(t) + w(t)$
$\int_0^T dt$
Uniform multilevel quantizer
$\phi_1(t) = \sqrt{\frac{2}{T}}\cos(2\pi f_c t)$
(a)
Output
Input
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
(b)
Symbol 1 transmitted $\sqrt{E}$
Symbol 2 transmitted $-\sqrt{E}$
(c)

FIGURE 10.3 Binary input Q-ary output discrete memoryless channel. (*a*) Receiver for binary phase-shift keying. (*b*) Transfer characteristic of multilevel quantizer. (*c*) Channel transition probability diagram. Parts (*b*) and (*c*) are illustrated for eight levels of quantization.

structure that is built into the design of channel codes to make the decoding relatively easy to perform.

The use of hard decisions prior to decoding causes an irreversible loss of information in the receiver. To reduce this loss, *soft-decision* coding is used. This is achieved by including a multilevel quantizer at the demodulator output, as illustrated in Figure 10.3*a* for the case of binary PSK signals. The input–output characteristic of the quantizer is shown in Figure 10.3*b*. The modulator has only the binary symbols 0 and 1 as inputs, but the demodulator output now has an alphabet with Q symbols. Assuming the use of the quantizer as described in Figure 10.3*b*, we have $Q = 8$. Such a channel is called a *binary input Q-ary output discrete memoryless channel.* The corresponding channel transition probability diagram is shown in Figure 10.3*c*. The form of this distribution, and consequently the decoder performance, depends on the location of the representation levels of the quantizer, which, in turn, depends on the signal level and noise variance. Accordingly, the demodulator must incorporate automatic gain control if an effective multilevel quantizer is to be realized. Moreover, the use of soft decisions complicates the implementation of the decoder. Nevertheless, soft-decision decoding offers significant improvement in performance over hard-decision decoding by taking a probabilistic rather than an algebraic approach. It is for this reason that soft-decision decoders are also referred to as *probabilistic decoders.*

■ CHANNEL CODING THEOREM REVISITED

In Chapter 9, we established the concept of *channel capacity*, which, for a discrete memoryless channel, represents the maximum amount of information transmitted per channel use. The *channel coding theorem* states that if a discrete memoryless channel has capacity C and a source generates information at a rate less than C, then there exists a coding technique such that the output of the source may be transmitted over the channel with an arbitrarily low probability of symbol error. For the special case of a binary symmetric channel, the theorem tells us that if the code rate r is less than the channel capacity C, then it is possible to find a code that achieves error-free transmission over the channel. Conversely, it is not possible to find such a code if the code rate r is greater than the channel capacity C.

The channel coding theorem thus specifies the channel capacity C as a *fundamental limit* on the rate at which the transmission of reliable (error-free) messages can take place over a discrete memoryless channel. The issue that matters is not the signal-to-noise ratio, so long as it is large enough, but how the channel input is encoded.

The most unsatisfactory feature of the channel coding theorem, however, is its nonconstructive nature. The theorem asserts the existence of good codes but does not tell us how to find them. By *good codes* we mean families of channel codes that are capable of providing reliable transmission of information (i.e., at arbitrarily small probability of symbol error) over a noisy channel of interest at bit rates up to a maximum value less than the capacity of that channel. The error-control coding techniques described in this chapter provide different methods of designing good codes.

■ Notation

The codes described in this chapter are *binary codes*, for which the alphabet consists only of symbols 0 and 1. In such a code, the encoding and decoding functions involve the binary arithmetic operations of *modulo-2 addition and multiplication* performed on code words in the code.

Throughout this chapter, we use an ordinary plus sign (+) to denote modulo-2 addition. The use of this terminology will not lead to confusion because the whole chapter relies on binary arithmetic. In so doing, we avoid the use of a special symbol $\oplus$, as we did in preceding chapters. Thus, according to the notation used in this chapter, the rules for modulo-2 addition are as follows:

$$\begin{aligned} 0 + 0 &= 0 \\ 1 + 0 &= 1 \\ 0 + 1 &= 1 \\ 1 + 1 &= 0 \end{aligned}$$

Because $1 + 1 = 0$, it follows that $1 = -1$. Hence, in binary arithmetic, subtraction is the same as addition. The rules for modulo-2 multiplication are as follows:

$$\begin{aligned} 0 \times 0 &= 0 \\ 1 \times 0 &= 0 \\ 0 \times 1 &= 0 \\ 1 \times 1 &= 1 \end{aligned}$$

Division is trivial in that we have

$$\begin{aligned} 1 \div 1 &= 1 \\ 0 \div 1 &= 0 \end{aligned}$$

and division by 0 is not permitted. Modulo-2 addition is the EXCLUSIVE-OR operation in logic, and modulo-2 multiplication is the AND operation.

10.3 *Linear Block Codes*

A code is said to be *linear if any two code words in the code can be added in modulo-2 arithmetic to produce a third code word in the code.* Consider then an (n, k) linear block code, in which k bits of the n code bits are always identical to the message sequence to be transmitted. The $n - k$ bits in the remaining portion are computed from the message bits in accordance with a prescribed encoding rule that determines the mathematical structure of the code. Accordingly, these $n - k$ bits are referred to as *generalized parity check bits* or simply *parity bits.* Block codes in which the message bits are transmitted in unaltered form are called *systematic codes.* For applications requiring *both* error detection and error correction, the use of systematic block codes simplifies implementation of the decoder.

Let $m_0, m_1, \ldots, m_{k-1}$ constitute a block of k arbitrary message bits. Thus we have 2^k distinct message blocks. Let this sequence of message bits be applied to a linear block encoder, producing an n-bit code word whose elements are denoted by $c_0, c_1, \ldots, c_{n-1}$. Let $b_0, b_1, \ldots, b_{n-k-1}$ denote the $(n - k)$ parity bits in the code word. For the code to possess a systematic structure, a code word is divided into two parts, one of which is occupied by the message bits and the other by the parity bits. Clearly, we have the option of sending the message bits of a code word before the parity bits, or vice versa. The former option is illustrated in Figure 10.4, and its use is assumed in the sequel.

According to the representation of Figure 10.4, the $(n - k)$ left-most bits of a code word are identical to the corresponding parity bits, and the k right-most bits of the code word are identical to the corresponding message bits. We may therefore write

$$c_i = \begin{cases} b_i, & i = 0, 1, \ldots, n - k - 1 \\ m_{i+k-n}, & i = n - k, n - k + 1, \ldots, n - 1 \end{cases} \tag{10.1}$$

The $(n - k)$ parity bits are *linear sums* of the k message bits, as shown by the generalized relation

$$b_i = p_{0i}m_0 + p_{1i}m_1 + \cdots + p_{k-1,i}m_{k-1} \tag{10.2}$$

where the coefficients are defined as follows:

$$p_{ij} = \begin{cases} 1 & \text{if } b_i \text{ depends on } m_j \\ 0 & \text{otherwise} \end{cases} \tag{10.3}$$

The coefficients p_{ij} are chosen in such a way that the rows of the generator matrix are linearly independent and the parity equations are *unique.*

The system of Equations (10.1) and (10.2) defines the mathematical structure of the (n, k) linear block code. This system of equations may be rewritten in a compact form

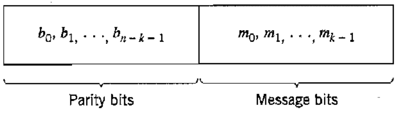


FIGURE 10.4 Structure of systematic code word.

using matrix notation. To proceed with this reformulation, we define the 1-by-k *message vector*, or *information vector*, $\mathbf{m}$, the 1-by-$(n-k)$ parity vector $\mathbf{b}$, and the 1-by-n code vector $\mathbf{c}$ as follows:

$$\mathbf{m} = [m_0, m_1, \ldots, m_{k-1}] \tag{10.4}$$

$$\mathbf{b} = [b_0, b_1, \ldots, b_{n-k-1}] \tag{10.5}$$

$$\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}] \tag{10.6}$$

Note that all three vectors are *row vectors*. The use of row vectors is adopted in this chapter for the sake of being consistent with the notation commonly used in the coding literature. We may thus rewrite the set of simultaneous equations defining the parity bits in the compact matrix form:

$$\mathbf{b} = \mathbf{mP} \tag{10.7}$$

where $\mathbf{P}$ is the k-by-$(n-k)$ *coefficient matrix* defined by

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \tag{10.8}$$

where p_{ij} is 0 or 1.

From the definitions given in Equations (10.4)–(10.6), we see that $\mathbf{c}$ may be expressed as a partitioned row vector in terms of the vectors $\mathbf{m}$ and $\mathbf{b}$ as follows:

$$\mathbf{c} = [\mathbf{b} \,\vdots\, \mathbf{m}] \tag{10.9}$$

Hence, substituting Equation (10.7) into Equation (10.9) and factoring out the common message vector $\mathbf{m}$, we get

$$\mathbf{c} = \mathbf{m}[\mathbf{P} \,\vdots\, \mathbf{I}_k] \tag{10.10}$$

where $\mathbf{I}_k$ is the k-by-k *identity matrix*:

$$\mathbf{I}_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{10.11}$$

Define the k-by-n *generator matrix*

$$\mathbf{G} = [\mathbf{P} \,\vdots\, \mathbf{I}_k] \tag{10.12}$$

The generator matrix $\mathbf{G}$ of Equation (10.12) is said to be in the *canonical form* in that its k rows are linearly independent; that is, it is not possible to express any row of the matrix $\mathbf{G}$ as a linear combination of the remaining rows. Using the definition of the generator matrix $\mathbf{G}$, we may simplify Equation (10.10) as

$$\mathbf{c} = \mathbf{mG} \tag{10.13}$$

The full set of code words, referred to simply as *the code*, is generated in accordance with Equation (10.13) by letting the message vector $\mathbf{m}$ range through the set of all 2^k binary k-tuples (1-by-k vectors). Moreover, the sum of any two code words is another code word. This basic property of linear block codes is called *closure*. To prove its validity, consider a pair of code vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ corresponding to a pair of message vectors $\mathbf{m}_i$ and $\mathbf{m}_j$, respectively. Using Equation (10.13) we may express the sum of $\mathbf{c}_i$ and $\mathbf{c}_j$ as

$$\begin{aligned}\mathbf{c}_i + \mathbf{c}_j &= \mathbf{m}_i\mathbf{G} + \mathbf{m}_j\mathbf{G}\\ &= (\mathbf{m}_i + \mathbf{m}_j)\mathbf{G}\end{aligned}$$

The modulo-2 sum of $\mathbf{m}_i$ and $\mathbf{m}_j$ represents a new message vector. Correspondingly, the modulo-2 sum of $\mathbf{c}_i$ and $\mathbf{c}_j$ represents a new code vector.

There is another way of expressing the relationship between the message bits and parity-check bits of a linear block code. Let $\mathbf{H}$ denote an $(n-k)$-by-n matrix, defined as

$$\mathbf{H} = [\mathbf{I}_{n-k} \vdots \mathbf{P}^T] \tag{10.14}$$

where $\mathbf{P}^T$ is an $(n-k)$-by-k matrix, representing the transpose of the coefficient matrix $\mathbf{P}$, and $\mathbf{I}_{n-k}$ is the $(n-k)$-by-$(n-k)$ identity matrix. Accordingly, we may perform the following multiplication of partitioned matrices:

$$\begin{aligned}\mathbf{H}\mathbf{G}^T &= [\mathbf{I}_{n-k} \vdots \mathbf{P}^T]\begin{bmatrix}\mathbf{P}^T\\ \cdots\\ \mathbf{I}_k\end{bmatrix}\\ &= \mathbf{P}^T + \mathbf{P}^T\end{aligned}$$

where we have used the fact that multiplication of a rectangular matrix by an identity matrix of compatible dimensions leaves the matrix unchanged. In modulo-2 arithmetic, we have $\mathbf{P}^T + \mathbf{P}^T = \mathbf{0}$, where $\mathbf{0}$ denotes an $(n-k)$-by-k null matrix (i.e., a matrix that has zeros for all of its elements). Hence,

$$\mathbf{H}\mathbf{G}^T = \mathbf{0} \tag{10.15}$$

Equivalently, we have $\mathbf{G}\mathbf{H}^T = \mathbf{0}$, where $\mathbf{0}$ is a new null matrix. Postmultiplying both sides of Equation (10.13) by $\mathbf{H}^T$, the transpose of $\mathbf{H}$, and then using Equation (10.15), we get

$$\begin{aligned}\mathbf{c}\mathbf{H}^T &= \mathbf{m}\mathbf{G}\mathbf{H}^T\\ &= \mathbf{0}\end{aligned} \tag{10.16}$$

The matrix $\mathbf{H}$ is called the *parity-check matrix* of the code, and the set of equations specified by Equation (10.16) are called *parity-check equations*.

The generator equation (10.13) and the parity-check detector equation (10.16) are basic to the description and operation of a linear block code. These two equations are depicted in the form of block diagrams in Figure 10.5*a* and 10.5*b*, respectively.

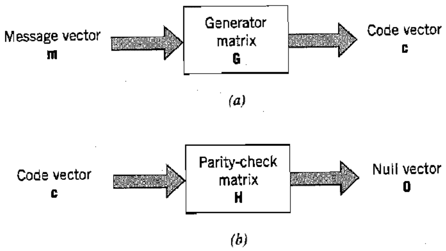


FIGURE 10.5 Block diagram representations of the generator equation (10.13) and the parity-check equation (10.16).

▶ Example 10.1 Repetition Codes

Repetition codes represent the simplest type of linear block codes. In particular, a single message bit is encoded into a block of n identical bits, producing an $(n, 1)$ block code. Such a code allows provision for a variable amount of redundancy. There are only two code words in the code: an all-zero code word and an all-one code word.

Consider, for example, the case of a repetition code with $k = 1$ and $n = 5$. In this case, we have four parity bits that are the same as the message bit. Hence, the identity matrix $\mathbf{I}_k = 1$, and the coefficient matrix $\mathbf{P}$ consists of a 1-by-4 vector that has 1 for all of its elements. Correspondingly, the generator matrix equals a row vector of all 1s, as shown by

$$\mathbf{G} = [1 \quad 1 \quad 1 \quad 1 \,\vdots\, 1]$$

The transpose of the coefficient matrix $\mathbf{P}$, namely, matrix $\mathbf{P}^T$, consists of a 4-by-1 vector that has 1 for all of its elements. The identity matrix $\mathbf{I}_{n-k}$ consists of a 4-by-4 matrix. Hence, the parity-check matrix equals

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & 1 \\ 0 & 1 & 0 & 0 & \vdots & 1 \\ 0 & 0 & 1 & 0 & \vdots & 1 \\ 0 & 0 & 0 & 1 & \vdots & 1 \end{bmatrix}$$

Since the message vector consists of a single binary symbol, 0 or 1, it follows from Equation (10.13) that there are only two code words: 00000 and 11111 in the (5, 1) repetition code, as expected. Note also that $\mathbf{H}\mathbf{G}^T = \mathbf{0}$, modulo-2, in accordance with Equation (10.15). ◀

■ Syndrome: Definition and Properties

The generator matrix $\mathbf{G}$ is used in the encoding operation at the transmitter. On the other hand, the parity-check matrix $\mathbf{H}$ is used in the decoding operation at the receiver. In the context of the latter operation, let $\mathbf{r}$ denote the 1-by-n *received vector* that results from sending the code vector $\mathbf{c}$ over a noisy channel. We express the vector $\mathbf{r}$ as the sum of the original code vector $\mathbf{c}$ and a vector $\mathbf{e}$, as shown by

$$\mathbf{r} = \mathbf{c} + \mathbf{e} \tag{10.17}$$

The vector $\mathbf{e}$ is called the *error vector* or *error pattern*. The ith element of $\mathbf{e}$ equals 0 if the corresponding element of $\mathbf{r}$ is the same as that of $\mathbf{c}$. On the other hand, the ith element of $\mathbf{e}$ equals 1 if the corresponding element of $\mathbf{r}$ is different from that of $\mathbf{c}$, in which case an error is said to have occurred in the ith location. That is, for $i = 1, 2, \ldots, n$, we have

$$e_i = \begin{cases} 1 & \text{if an error has occurred in the } i\text{th location} \\ 0 & \text{otherwise} \end{cases} \tag{10.18}$$

The receiver has the task of decoding the code vector $\mathbf{c}$ from the received vector $\mathbf{r}$. The algorithm commonly used to perform this decoding operation starts with the computation of a 1-by-$(n - k)$ vector called the *error-syndrome vector* or simply the *syndrome*.[3] The importance of the syndrome lies in the fact that it depends only upon the error pattern.

Given a 1-by-n received vector $\mathbf{r}$, the corresponding syndrome is formally defined as

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T \tag{10.19}$$

Accordingly, the syndrome has the following important properties.

Property 1

The syndrome depends only on the error pattern, and not on the transmitted code word.

To prove this property, we first use Equations (10.17) and (10.19), and then Equation (10.16) to obtain

$$\begin{aligned} \mathbf{s} &= (\mathbf{c} + \mathbf{e})\mathbf{H}^T \\ &= \mathbf{c}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T \end{aligned} \tag{10.20}$$

Hence, the parity-check matrix H of a code permits us to compute the syndrome s, which depends only upon the error pattern e.

Property 2

All error patterns that differ by a code word have the same syndrome.

For k message bits, there are 2^k distinct code vectors denoted as $\mathbf{c}_i$, $i = 0, 1, \ldots, 2^k - 1$. Correspondingly, for any error pattern e, we define the 2^k distinct vectors $\mathbf{e}_i$ as

$$\mathbf{e}_i = \mathbf{e} + \mathbf{c}_i, \qquad i = 0, 1, \ldots, 2^k - 1 \tag{10.21}$$

The set of vectors $\{\mathbf{e}_i, i = 0, 1, \ldots, 2^k - 1\}$ so defined is called a *coset* of the code. In other words, a coset has exactly 2^k elements that differ at most by a code vector. Thus, an (n, k) linear block code has 2^{n-k} possible cosets. In any event, multiplying both sides of Equation (10.21) by the matrix $\mathbf{H}^T$, we get

$$\begin{aligned} \mathbf{e}_i\mathbf{H}^T &= \mathbf{e}\mathbf{H}^T + \mathbf{c}_i\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T \end{aligned} \tag{10.22}$$

which is independent of the index i. Accordingly, we may state that each coset of the code is characterized by a unique syndrome.

We may put Properties 1 and 2 in perspective by expanding Equation (10.20). Specifically, with the matrix **H** having the systematic form given in Equation (10.14), where the matrix **P** is itself defined by Equation (10.8), we find from Equation (10.20) that the $(n - k)$ elements of the syndrome s are linear combinations of the n elements of the error pattern e, as shown by

$$\begin{aligned} s_0 &= e_0 + e_{n-k}p_{00} + e_{n-k+1}p_{10} + \cdots + e_{n-1}p_{k-1,0} \\ s_1 &= e_1 + e_{n-k}p_{01} + e_{n-k+1}p_{11} + \cdots + e_{n-1}p_{k-1,1} \\ &\vdots \\ s_{n-k-1} &= e_{n-k-1} + e_{n-k}p_{0,n-k-1} + \cdots + e_{n-1}p_{k-1,n-k-1} \end{aligned} \tag{10.23}$$

This set of $(n - k)$ linear equations clearly shows that the syndrome contains information about the error pattern and may therefore be used for error detection. However, it should be noted that the set of equations is *underdetermined* in that we have more unknowns than equations. Accordingly, there is *no* unique solution for the error pattern. Rather, there are 2^n error patterns that satisfy Equation (10.23) and therefore result in the same syndrome, in accordance with Property 2 and Equation (10.22). In particular, with 2^{n-k} possible syndrome vectors, the information contained in the syndrome s about the error pattern e is *not* enough for the decoder to compute the exact value of the transmitted code vector. Nevertheless, knowledge of the syndrome s reduces the search for the true error pattern e from 2^n to 2^{n-k} possibilities. Given these possibilities, the decoder has the task of making the best selection from the cosets corresponding to s.

Minimum Distance Considerations

Consider a pair of code vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ that have the same number of elements. The *Hamming distance* $d(\mathbf{c}_1, \mathbf{c}_2)$ between such a pair of code vectors is defined as the number of locations in which their respective elements differ.

The *Hamming weight* $w(\mathbf{c})$ of a code vector c is defined as the number of nonzero elements in the code vector. Equivalently, we may state that the Hamming weight of a code vector is the distance between the code vector and the all-zero code vector.

The *minimum distance* $d_{\min}$ of a linear block code is defined as the smallest Hamming distance between any pair of code vectors in the code. That is, the minimum distance is the same as the smallest Hamming weight of the difference between any pair of code vectors. From the closure property of linear block codes, the sum (or difference) of two code vectors is another code vector. Accordingly, we may state that *the minimum distance of a linear block code is the smallest Hamming weight of the nonzero code vectors in the code.*

The minimum distance $d_{\min}$ is related to the structure of the parity-check matrix H of the code in a fundamental way. From Equation (10.16) we know that a linear block code is defined by the set of all code vectors for which $\mathbf{c}\mathbf{H}^T = \mathbf{0}$, where $\mathbf{H}^T$ is the transpose of the parity-check matrix H. Let the matrix H be expressed in terms of its columns as follows:

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n] \tag{10.24}$$

Then, for a code vector c to satisfy the condition $\mathbf{c}\mathbf{H}^T = \mathbf{0}$, the vector c must have 1s in such positions that the corresponding rows of $\mathbf{H}^T$ sum to the zero vector 0. However, by definition, the number of 1s in a code vector is the Hamming weight of the code vector. Moreover, the smallest Hamming weight of the nonzero code vectors in a linear block code equals the minimum distance of the code. Hence, *the minimum distance of a linear block code is defined by the minimum number of rows of the matrix* $\mathbf{H}^T$ whose sum is equal to the zero vector.

The minimum distance of a linear block code, $d_{\min}$, is an important parameter of the code. Specifically, it determines the error-correcting capability of the code. Suppose an (n, k) linear block code is required to detect and correct all error patterns (over a binary symmetric channel), and whose Hamming weight is less than or equal to t. That is, if a code vector $\mathbf{c}_i$ in the code is transmitted and the received vector is $\mathbf{r} = \mathbf{c}_i + \mathbf{e}$, we require that the decoder output $\hat{\mathbf{c}} = \mathbf{c}_i$, whenever the error pattern e has a Hamming weight $w(\mathbf{e}) \le t$. We assume that the 2^k code vectors in the code are transmitted with equal probability. The best strategy for the decoder then is to pick the code vector closest to the received vector r, that is, the one for which the Hamming distance $d(\mathbf{c}_i, \mathbf{r})$ is the smallest. With such a strategy, the decoder will be able to detect and correct all error patterns of Hamming weight $w(\mathbf{e}) \le t$, provided that the minimum distance of the code is equal to or greater than $2t + 1$. We may demonstrate the validity of this requirement by adopting a geometric interpretation of the problem. In particular, the 1-by-n code vectors and the 1-by-n received vector are represented as points in an n-dimensional space. Suppose that we construct two spheres, each of radius t, around the points that represent code vectors $\mathbf{c}_i$ and $\mathbf{c}_j$. Let these two spheres be disjoint, as depicted in Figure 10.6*a*. For this condition to be satisfied, we require that $d(\mathbf{c}_i, \mathbf{c}_j) \ge 2t + 1$. If then the code vector $\mathbf{c}_i$ is transmitted and the Hamming distance $d(\mathbf{c}_i, \mathbf{r}) \le t$, it is clear that the decoder will pick $\mathbf{c}_i$ as it is the

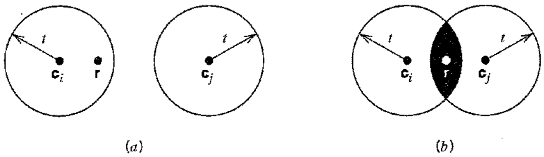


FIGURE 10.6 (*a*) Hamming distance $d(c_i, c_j) \geq 2t + 1$. (*b*) Hamming distance $d(c_i, c_j) < 2t$. The received vector is denoted by r.

code vector closest to the received vector r. If, on the other hand, the Hamming distance $d(c_i, c_j) \leq 2t$, the two spheres around c_i and c_j intersect, as depicted in Figure 10.6*b*. Here we see that if c_i is transmitted, there exists a received vector r such that the Hamming distance $d(c_i, r) \leq t$, and yet r is as close to c_j as it is to c_i. Clearly, there is now the possibility of the decoder picking the vector c_j, which is wrong. We thus conclude that *an* (*n*, *k*) *linear block code has the power to correct all error patterns of weight t or less if, and only if,*

$$d(c_i, c_j) \geq 2t + 1 \qquad \text{for all } c_i \text{ and } c_j$$

By definition, however, the smallest distance between any pair of code vectors in a code is the minimum distance of the code, $d_{\min}$. We may therefore state that *an* (*n*, *k*) *linear block code of minimum distance* $d_{\min}$ *can correct up to t errors if, and only if,*

$$t \leq \left\lfloor \tfrac{1}{2}(d_{\min} - 1) \right\rfloor \tag{10.25}$$

where $\lfloor\ \rfloor$ *denotes the largest integer* less than or equal to the enclosed quantity. Equation (10.25) gives the error-correcting capability of a linear block code a quantitative meaning.

▪ SYNDROME DECODING

We are now ready to describe a syndrome-based decoding scheme for linear block codes. Let $c_1, c_2, \ldots, c_{2^k}$ denote the 2^k code vectors of an (*n*, *k*) linear block code. Let r denote the received vector, which may have one of 2^n possible values. The receiver has the task of partitioning the 2^n possible received vectors into 2^k disjoint subsets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{2^k}$ in such a way that the *i*th subset $\mathcal{D}_i$ corresponds to code vector c_i for $1 \leq i \leq 2^k$. The received vector r is decoded into c_i if it is in the *i*th subset. For the decoding to be correct, r must be in the subset that belongs to the code vector c_i that was actually sent.

The 2^k subsets described herein constitute a *standard array* of the linear block code. To construct it, we may exploit the linear structure of the code by proceeding as follows:

1. The 2^k code vectors are placed in a row with the all-zero code vector c_1 as the left-most element.
2. An error pattern e_2 is picked and placed under c_1, and a second row is formed by adding e_2 to each of the remaining code vectors in the first row; it is important that the error pattern chosen as the first element in a row not have previously appeared in the standard array.
3. Step 2 is repeated until all the possible error patterns have been accounted for.

Figure 10.7 illustrates the structure of the standard array so constructed. The 2^k columns of this array represent the disjoint subsets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{2^k}$. The 2^{n-k} rows of the array

$$\begin{array}{ccccccc}
\mathbf{c}_1 = \mathbf{0} & \mathbf{c}_2 & \mathbf{c}_3 & \cdots & \mathbf{c}_i & \cdots & \mathbf{c}_{2^k} \\
\mathbf{e}_2 & \mathbf{c}_2 + \mathbf{e}_2 & \mathbf{c}_3 + \mathbf{e}_2 & \cdots & \mathbf{c}_i + \mathbf{e}_2 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_2 \\
\mathbf{e}_3 & \mathbf{c}_2 + \mathbf{e}_3 & \mathbf{c}_3 + \mathbf{e}_3 & \cdots & \mathbf{c}_i + \mathbf{e}_3 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_3 \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
\mathbf{e}_j & \mathbf{c}_2 + \mathbf{e}_j & \mathbf{c}_3 + \mathbf{e}_j & \cdots & \mathbf{c}_i + \mathbf{e}_j & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_j \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
\mathbf{e}_{2^{n-k}} & \mathbf{c}_2 + \mathbf{e}_{2^{n-k}} & \mathbf{c}_3 + \mathbf{e}_{2^{n-k}} & & \mathbf{c}_i + \mathbf{e}_{2^{n-k}} & & \mathbf{c}_{2^k} + \mathbf{e}_{2^{n-k}}
\end{array}$$

FIGURE 10.7 Standard array for an (n, k) block code.

represent the cosets of the code, and their first elements $\mathbf{e}_2, \ldots, \mathbf{e}_{2^{n-k}}$ are called *coset leaders*.

For a given channel, the probability of decoding error is minimized when the most likely error patterns (i.e., those with the largest probability of occurrence) are chosen as the coset leaders. In the case of a binary symmetric channel, the smaller the Hamming weight of an error pattern the more likely it is to occur. Accordingly, the standard array should be constructed with each coset leader having the minimum Hamming weight in its coset.

We may now describe a decoding procedure for a linear block code:

1. For the received vector $\mathbf{r}$, compute the syndrome $\mathbf{s} = \mathbf{r}\mathbf{H}^T$.
2. Within the coset characterized by the syndrome $\mathbf{s}$, identify the coset leader (i.e., the error pattern with the largest probability of occurrence); call it $\mathbf{e}_0$.
3. Compute the code vector

$$\mathbf{c} = \mathbf{r} + \mathbf{e}_0 \qquad (10.26)$$

as the decoded version of the received vector $\mathbf{r}$.

This procedure is called *syndrome decoding*.

▶ EXAMPLE 10.2 Hamming Codes[4]

Consider a family of (n, k) linear block codes that have the following parameters:

Block length:	$n = 2^m - 1$
Number of message bits:	$k = 2^m - m - 1$
Number of parity bits:	$n - k = m$

where $m \geq 3$. These are the so-called Hamming codes.

Consider, for example, the (7, 4) Hamming code with $n = 7$ and $k = 4$, corresponding to $m = 3$. The generator matrix of the code must have a structure that conforms to Equation (10.12). The following matrix represents an appropriate generator matrix for the (7, 4) Hamming code:

$$\mathbf{G} = \left[\begin{array}{ccc:cccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1
\end{array}\right]$$
$$\underbrace{\qquad\qquad}_{\mathbf{P}} \qquad \underbrace{\qquad\qquad\qquad}_{\mathbf{I}_k}$$

TABLE 10.1 *Code words of a (7, 4) Hamming code*

Message Word	*Code Word*	*Weight of Code Word*	*Message Word*	*Code Word*	*Weight of Code Word*
0 0 0 0	0 0 0 0 0 0 0	0	1 0 0 0	1 1 0 1 0 0 0	3
0 0 0 1	1 0 1 0 0 0 1	3	1 0 0 1	0 1 1 1 0 0 1	4
0 0 1 0	1 1 1 0 0 1 0	4	1 0 1 0	0 0 1 1 0 1 0	3
0 0 1 1	0 1 0 0 0 1 1	3	1 0 1 1	1 0 0 1 0 1 1	4
0 1 0 0	0 1 1 0 1 0 0	3	1 1 0 0	1 0 1 1 1 0 0	4
0 1 0 1	1 1 0 0 1 0 1	4	1 1 0 1	0 0 0 1 1 0 1	3
0 1 1 0	1 0 0 0 1 1 0	3	1 1 1 0	0 1 0 1 1 1 0	4
0 1 1 1	0 0 1 0 1 1 1	4	1 1 1 1	1 1 1 1 1 1 1	7

The corresponding parity-check matrix is given by

$$\mathbf{H} = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ \underbrace{0 \quad 0 \quad 1}_{\mathbf{I}_{n-k}} & & & \underbrace{0 \quad 1 \quad 1 \quad 1}_{\mathbf{P}^T} \end{array}\right]$$

With $k = 4$, there are $2^k = 16$ distinct message words, which are listed in Table 10.1. For a given message word, the corresponding code word is obtained by using Equation (10.13). Thus, the application of this equation results in the 16 code words listed in Table 10.1.

In Table 10.1, we have also listed the Hamming weights of the individual code words in the (7, 4) Hamming code. Since the smallest of the Hamming weights for the nonzero code words is 3, it follows that the minimum distance of the code is 3. Indeed, Hamming codes have the property that the minimum distance $d_{\min} = 3$, independent of the value assigned to the number of parity bits m.

To illustrate the relation between the minimum distance $d_{\min}$ and the structure of the parity-check matrix H, consider the code word 0110100. In the matrix multiplication defined by Equation (10.16), the nonzero elements of this code word "sift" out the second, third, and fifth columns of the matrix H yielding

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We may perform similar calculations for the remaining 14 nonzero code words. We thus find that the smallest number of columns in H that sums to zero is 3, confirming the earlier statement that $d_{\min} = 3$.

An important property of Hamming codes is that they satisfy the condition of Equation (10.25) with the equality sign, assuming that $t = 1$. This means that Hamming codes are *single-error correcting binary perfect codes.*

Assuming single-error patterns, we may formulate the seven coset leaders listed in the right-hand column of Table 10.2. The corresponding 2^3 syndromes, listed in the left-hand column, are calculated in accordance with Equation (10.20). The zero syndrome signifies no transmission errors.

Suppose, for example, the code vector [1110010] is sent, and the received vector is

TABLE 10.2 *Decoding table for the (7, 4) Hamming code defined in Table 10.1*

Syndrome	*Error Pattern*
0 0 0	0 0 0 0 0 0 0
1 0 0	1 0 0 0 0 0 0
0 1 0	0 1 0 0 0 0 0
0 0 1	0 0 1 0 0 0 0
1 1 0	0 0 0 1 0 0 0
0 1 1	0 0 0 0 1 0 0
1 1 1	0 0 0 0 0 1 0
1 0 1	0 0 0 0 0 0 1

[1100010] with an error in the third bit. Using Equation (10.19), the syndrome is calculated to be

$$\mathbf{s} = [1100010]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$
$$= [0 \quad 0 \quad 1]$$

From Table 10.2 the corresponding coset leader (i.e., error pattern with the highest probability of occurrence) is found to be [0010000], indicating correctly that the third bit of the received vector is erroneous. Thus, adding this error pattern to the received vector, in accordance with Equation (10.26), yields the correct code vector actually sent. ◀

DUAL CODE

Given a linear block code, we may define its *dual* as follows. Taking the transpose of both sides of Equation (10.15), we have

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}$$

where $\mathbf{H}^T$ is the transpose of the parity-check matrix of the code, and $\mathbf{0}$ is a new zero matrix. This equation suggests that every (n, k) linear block code with generator matrix $\mathbf{G}$ and parity-check matrix $\mathbf{H}$ has a *dual code* with parameters $(n, n - k)$, generator matrix $\mathbf{H}$ and parity-check matrix $\mathbf{G}$.

10.4 Cyclic Codes

Cyclic codes form a subclass of linear block codes. Indeed, many of the important linear block codes discovered to date are either cyclic codes or closely related to cyclic codes. An advantage of cyclic codes over most other types of codes is that they are easy to encode. Furthermore, cyclic codes possess a well-defined mathematical structure, which has led to the development of very efficient decoding schemes for them.

A binary code is said to be a *cyclic code* if it exhibits two fundamental properties:

1. *Linearity property:* The sum of any two code words in the code is also a code word.
2. *Cyclic property:* Any cyclic shift of a code word in the code is also a code word.

Property 1 restates the fact that a cyclic code is a linear block code (i.e., it can be described as a parity-check code). To restate Property 2 in mathematical terms, let the n-tuple $(c_0, c_1, \ldots, c_{n-1})$ denote a code word of an (n, k) linear block code. The code is a cyclic code if the n-tuples

$$\begin{gathered}(c_{n-1}, c_0, \ldots, c_{n-2}),\\ (c_{n-2}, c_{n-1}, \ldots, c_{n-3}),\\ \vdots\\ (c_1, c_2, \ldots, c_{n-1}, c_0)\end{gathered}$$

are all code words in the code.

To develop the algebraic properties of cyclic codes, we use the elements $c_0, c_1, \ldots, c_{n-1}$ of a code word to define the *code polynomial*

$$c(X) = c_0 + c_1X + c_2X^2 + \cdots + c_{n-1}X^{n-1} \tag{10.27}$$

where X is an indeterminate. Naturally, for binary codes, the coefficients are 1s and 0s. Each power of X in the polynomial $c(X)$ represents a one-bit *shift* in time. Hence, multiplication of the polynomial $c(X)$ by X may be viewed as a shift to the right. The key question is: How do we make such a shift *cyclic*? The answer to this question is addressed next.

Let the code polynomial $c(X)$ be multiplied by X^i, yielding

$$\begin{aligned}X^ic(X) &= X^i(c_0 + c_1X + \cdots + c_{n-i-1}X^{n-i-1} + c_{n-i}X^{n-i}\\ &\quad + \cdots + c_{n-1}X^{n-1})\\ &= c_0X^i + c_1X^{i+1} + \cdots + c_{n-i-1}X^{n-1} + c_{n-i}X^n\\ &\quad + \cdots + c_{n-1}X^{n+i-1}\\ &= c_{n-i}X^n + \cdots + c_{n-1}X^{n+i-1} + c_0X^i + c_1X^{i+1}\\ &\quad + \cdots + c_{n-i-1}X^{n-1}\end{aligned} \tag{10.28}$$

where, in the last line, we have merely rearranged terms. Recognizing, for example, that $c_{n-i} + c_{n-i} = 0$ in modulo-2 addition, we may manipulate the first i terms of Equation (10.28) as follows:

$$\begin{aligned}X^ic(X) &= c_{n-i} + \cdots + c_{n-1}X^{i-1} + c_0X^i + c_1X^{i+1} + \cdots + c_{n-i-1}X^{n-1}\\ &\quad + c_{n-i}(X^n + 1) + \cdots + c_{n-1}X^{i-1}(X^n + 1)\end{aligned} \tag{10.29}$$

Next, we introduce the following definitions:

$$\begin{aligned}c^{(i)}(X) &= c_{n-i} + \cdots + c_{n-1}X^{i-1} + c_0X^i + c_1X^{i+1}\\ &\quad + \cdots + c_{n-i-1}X^{n-1}\end{aligned} \tag{10.30}$$

$$q(X) = c_{n-i} + c_{n-i+1}X + \cdots + c_{n-1}X^{i-1} \tag{10.31}$$

Accordingly, Equation (10.29) is reformulated in the compact form

$$X^i c(X) = q(X)(X^n + 1) + c^{(i)}(X) \tag{10.32}$$

The polynomial $c^{(i)}(X)$ is recognized as the code polynomial of the code word $(c_{n-i}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-i-1})$ obtained by applying i cyclic shifts to the code word $(c_0, c_1, \ldots, c_{n-i-1}, c_{n-i}, \ldots, c_{n-1})$. Moreover, from Equation (10.32) we readily see that $c^{(i)}(X)$ is the remainder that results from dividing $X^i c(X)$ by $(X^n + 1)$. We may thus formally state the cyclic property in polynomial notation as follows: *If $c(X)$ is a code polynomial, then the polynomial*

$$c^{(i)}(X) = X^i c(X) \bmod (X^n + 1) \tag{10.33}$$

is also a code polynomial for any cyclic shift i; the term *mod* is the abbreviation for *modulo*. The special form of polynomial multiplication described in Equation (10.33) is referred to as *multiplication modulo* $X^n + 1$. In effect, the multiplication is subject to the constraint $X^n = 1$, the application of which restores the polynomial $X^i c(X)$ to order $n - 1$ for all $i < n$. (Note that in modulo-2 arithmetic, $X^n + 1$ has the same value as $X^n - 1$.)

Generator Polynomial

The polynomial $X^n + 1$ and its factors play a major role in the generation of cyclic codes. Let $g(X)$ be a polynomial of degree $n - k$ that is a factor of $X^n + 1$; as such, $g(X)$ is *the polynomial of least degree in the code.* In general, $g(X)$ may be expanded as follows:

$$g(X) = 1 + \sum_{i=1}^{n-k-1} g_i X^i + X^{n-k} \tag{10.34}$$

where the coefficient g_i is equal to 0 or 1. According to this expansion, the polynomial $g(X)$ has two terms with coefficient 1 separated by $n - k - 1$ terms. The polynomial $g(X)$ is called the *generator polynomial* of a cyclic code. A cyclic code is uniquely determined by the generator polynomial $g(X)$ in that each code polynomial in the code can be expressed in the form of a polynomial product as follows:

$$c(X) = a(X)g(X) \tag{10.35}$$

where $a(X)$ is a polynomial in X with degree $k - 1$. The $c(X)$ so formed satisfies the condition of Equation (10.33) since $g(X)$ is a factor of $X^n + 1$.

Suppose we are given the generator polynomial $g(X)$ and the requirement is to encode the message sequence $(m_0, m_1, \ldots, m_{k-1})$ into an (n, k) *systematic* cyclic code. That is, the message bits are transmitted in unaltered form, as shown by the following structure for a code word (see Figure 10.4):

$$(\underbrace{b_0, b_1, \ldots, b_{n-k-1}}_{n-k \text{ parity bits}}, \underbrace{m_0, m_1, \ldots, m_{k-1}}_{k \text{ message bits}})$$

Let the *message polynomial* be defined by

$$m(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1} \tag{10.36}$$

and let

$$b(X) = b_0 + b_1 X + \cdots + b_{n-k-1} X^{n-k-1} \tag{10.37}$$

According to Equation (10.1), we want the code polynomial to be in the form

$$c(X) = b(X) + X^{n-k}m(X) \tag{10.38}$$

Hence, the use of Equations (10.35) and (10.38) yields

$$a(X)g(X) = b(X) + X^{n-k}m(X)$$

Equivalently, in light of modulo-2 addition, we may write

$$\frac{X^{n-k}m(X)}{g(X)} = a(X) + \frac{b(X)}{g(X)} \tag{10.39}$$

Equation (10.39) states that the polynomial $b(X)$ is the *remainder* left over after dividing $X^{n-k}m(X)$ by $g(X)$.

We may now summarize the steps involved in the encoding procedure for an (n, k) cyclic code assured of a systematic structure. Specifically, we proceed as follows:

1. Multiply the message polynomial $m(X)$ by X^{n-k}.
2. Divide $X^{n-k}m(X)$ by the generator polynomial $g(X)$, obtaining the remainder $b(X)$.
3. Add $b(X)$ to $X^{n-k}m(X)$, obtaining the code polynomial $c(X)$.

■ PARITY-CHECK POLYNOMIAL

An (n, k) cyclic code is uniquely specified by its generator polynomial $g(X)$ of order $(n - k)$. Such a code is also uniquely specified by another polynomial of degree k, which is called the *parity-check polynomial*, defined by

$$h(X) = 1 + \sum_{i=1}^{k-1} h_i X^i + X^k \tag{10.40}$$

where the coefficients h_i are 0 or 1. The parity-check polynomial $h(X)$ has a form similar to the generator polynomial in that there are two terms with coefficient 1, but separated by $k - 1$ terms.

The generator polynomial $g(X)$ is equivalent to the generator matrix **G** as a description of the code. Correspondingly, the parity-check polynomial, denoted by $h(X)$, is an equivalent representation of the parity-check matrix **H**. We thus find that the matrix relation $\mathbf{HG}^T = 0$ presented in Equation (10.15) for linear block codes corresponds to the relationship

$$g(X)h(X) \bmod(X^n + 1) = 0 \tag{10.41}$$

Accordingly, we may state that *the generator polynomial* $g(X)$ *and the parity-check polynomial* $h(X)$ *are factors of the polynomial* $X^n + 1$, as shown by

$$g(X)h(X) = X^n + 1 \tag{10.42}$$

This property provides the basis for selecting the generator or parity-check polynomial of a cyclic code. In particular, we may state that if $g(X)$ is a polynomial of degree $(n- k)$ and it is also a factor of $X^n+ 1$, then $g(X)$ is the generator polynomial of an (n, k) cyclic code. Equivalently, we may state that if $h(X)$ is a polynomial of degree k and it is also a factor of $X^n + 1$, then $h(X)$ is the parity-check polynomial of an (n, k) cyclic code.

A final comment is in order. Any factor of $X^n + 1$ with degree $(n - k)$, the number of parity bits, can be used as a generator polynomial. For large values of n, the polynomial $X^n + 1$ may have many factors of degree $n - k$. Some of these polynomial factors generate good cyclic codes, whereas some of them generate bad cyclic codes. The issue of how to select generator polynomials that produce good cyclic codes is very difficult to resolve. Indeed, coding theorists have expended much effort in the search for good cyclic codes.

Generator and Parity-Check Matrices

Given the generator polynomial $g(X)$ of an (n, k) cyclic code, we may construct the generator matrix G of the code by noting that the k polynomials $g(X), Xg(X), \dots, X^{k-1}g(X)$ span the code. Hence, the n-tuples corresponding to these polynomials may be used as rows of the k-by-n generator matrix G.

However, the construction of the parity-check matrix H of the cyclic code from the parity-check polynomial $h(X)$ requires special attention, as described here. Multiplying Equation (10.42) by $a(x)$ and then using Equation (10.35), we obtain

$$c(X)h(X) = a(X) + X^n a(X) \tag{10.43}$$

The polynomials $c(X)$ and $h(X)$ are themselves defined by Equations (10.27) and (10.40), respectively, which means that their product on the left-hand side of Equation (10.43) contains terms with powers extending up to $n + k - 1$. On the other hand, the polynomial $a(X)$ has degree $k - 1$ or less, the implication of which is that the powers of $X^k, X^{k+1}, \dots, X^{n-1}$ do *not* appear in the polynomial on the right-hand side of Equation (10.43). Thus, setting the coefficients of $X^k, X^{k-1}, \dots, X^{n-1}$ in the expansion of the product polynomial $c(X)h(X)$ equal to zero, we obtain the following set of $n - k$ equations:

$$\sum_{i=j}^{j+k} c_i h_{k+j-i} = 0 \qquad \text{for } 0 \le j \le n - k - 1 \tag{10.44}$$

Comparing Equation (10.44) with the corresponding relation of Equation (10.16), we may make the following important observation: The coefficients of the parity-check polynomial $h(X)$ involved in the polynomial multiplication described in Equation (10.44) are arranged in *reversed* order with respect to the coefficients of the parity-check matrix H involved in forming the inner product of vectors described in Equation (10.16). This observation suggests that we define the *reciprocal of the parity-check polynomial* as follows:

$$\begin{aligned} X^k h(X^{-1}) &= X^k\left(1 + \sum_{i=1}^{k-1} h_i X^{-i} + X^{-k}\right) \\ &= 1 + \sum_{i=1}^{k-1} h_{k-i} X^i + X^k \end{aligned} \tag{10.45}$$

which is also a factor of $X^n + 1$. The n-tuples pertaining to the $(n - k)$ polynomials $X^k h(X^{-1}), X^{k+1}h(X^{-1}), \dots, X^{n-1}h(X^{-1})$ may now be used in rows of the $(n - k)$-by-n parity-check matrix H.

In general, the generator matrix G and the parity-check matrix H constructed in the manner described here are not in their systematic forms. They can be put into their systematic forms by performing simple operations on their respective rows, as illustrated in Example 10.3.

Encoder for Cyclic Codes

Earlier we showed that the encoding procedure for an (n, k) cyclic code in systematic form involves three steps: (1) multiplication of the message polynomial $m(X)$ by X^{n-k}, (2) di-

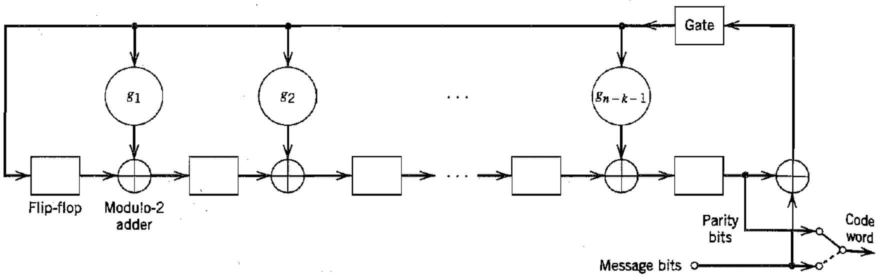


FIGURE 10.8 Encoder for an (n, k) cyclic code.

vision of $X^{n-k}m(X)$ by the generator polynomial $g(X)$ to obtain the remainder $b(X)$, and (3) addition of $b(X)$ to $X^{n-k}m(X)$ to form the desired code polynomial. These three steps can be implemented by means of the encoder shown in Figure 10.8, consisting of a *linear feedback shift register* with $(n - k)$ stages.

The boxes in Figure 10.8 represent *flip-flops*, or *unit-delay elements*. The flip-flop is a device that resides in one of two possible states denoted by 0 and 1. An *external clock* (not shown in Figure 10.8) controls the operation of all the flip-flops. Every time the clock ticks, the contents of the flip-flops (initially set to the state 0) are shifted out in the direction of the arrows. In addition to the flip-flops, the encoder of Figure 10.8 includes a second set of logic elements, namely, *adders*, which compute the modulo-2 sums of their respective inputs. Finally, the *multipliers* multiply their respective inputs by the associated coefficients. In particular, if the coefficient $g_i = 1$, the multiplier is just a direct "connection." If, on the other hand, the coefficient $g_i = 0$, the multiplier is "no connection."

The operation of the encoder shown in Figure 10.8 proceeds as follows:

1. The gate is switched on. Hence, the k message bits are shifted into the channel. As soon as the k message bits have entered the shift register, the resulting $(n - k)$ bits in the register form the parity bits [recall that the parity bits are the same as the coefficients of the remainder $b(X)$].
2. The gate is switched off, thereby breaking the feedback connections.
3. The contents of the shift register are read out into the channel.

■ CALCULATION OF THE SYNDROME

Suppose the code word $(c_0, c_1, \ldots, c_{n-1})$ is transmitted over a noisy channel, resulting in the received word $(r_0, r_1, \ldots, r_{n-1})$. From Section 10.3, we recall that the first step in the decoding of a linear block code is to calculate the syndrome for the received word. If the syndrome is zero, there are no transmission errors in the received word. If, on the other hand, the syndrome is nonzero, the received word contains transmission errors that require correction.

In the case of a cyclic code in systematic form, the syndrome can be calculated easily. Let the received word be represented by a polynomial of degree $n - 1$ or less, as shown by

$$r(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1} \tag{10.46}$$

Let $q(X)$ denote the quotient and $s(X)$ denote the remainder, which are the results of dividing $r(X)$ by the generator polynomial $g(X)$. We may therefore express $r(X)$ as follows:

$$r(X) = q(X)g(X) + s(X) \tag{10.47}$$

The remainder $s(X)$ is a polynomial of degree $n - k - 1$ or less, which is the result of interest. It is called the *syndrome polynomial* because its coefficients make up the $(n - k)$-by-1 syndrome $\mathbf{s}$.

Figure 10.9 shows a *syndrome calculator* that is identical to the encoder of Figure 10.8 except for the fact that the received bits are fed into the $(n - k)$ stages of the feedback shift register from the left. As soon as all the received bits have been shifted into the shift register, its contents define the syndrome $\mathbf{s}$.

The syndrome polynomial $s(X)$ has the following useful properties that follow from the definition given in Equation (10.47).

1. *The syndrome of a received word polynomial is also the syndrome of the corresponding error polynomial.*

Given that a cyclic code with polynomial $c(X)$ is sent over a noisy channel, the received word polynomial is defined by

$$r(X) = c(X) + e(X) \tag{10.48}$$

where $e(X)$ is the *error polynomial.* Equivalently, we may write

$$e(X) = r(X) + c(X) \tag{10.49}$$

Hence, substituting Equations (10.35) and (10.47) into (10.49), we get

$$e(X) = u(X)g(X) + s(X) \tag{10.50}$$

where the quotient is $u(X) = a(X) + q(X)$. Equation (10.50) shows that $s(X)$ is also the syndrome of the error polynomial $e(X)$. The implication of this property is that when the syndrome polynomial $s(X)$ is nonzero, the presence of transmission errors in the received word is detected.

2. *Let $s(X)$ be the syndrome of a received word polynomial $r(X)$. Then, the syndrome of $Xr(X)$, a cyclic shift of $r(X)$, is $Xs(X)$.*

Applying a cyclic shift to both sides of Equation (10.47), we get

$$Xr(X) = Xq(X)g(X) + Xs(X) \tag{10.51}$$

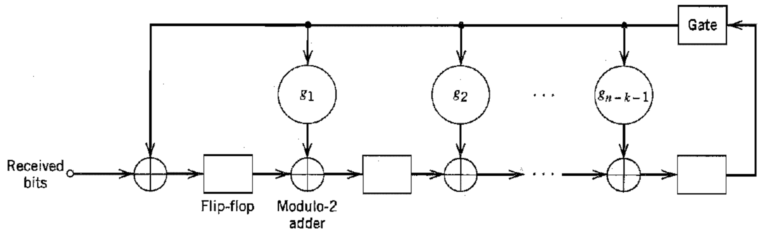


FIGURE 10.9 Syndrome calculator for (n, k) cyclic code.

from which we readily see that $Xs(X)$ is the remainder of the division of $Xr(X)$ by $g(X)$. Hence, the syndrome of $Xr(X)$ is $Xs(X)$ as stated. We may generalize this result by stating that if $s(X)$ is the syndrome of $r(X)$, then $X^i s(X)$ is the syndrome of $X^i r(X)$.

3. *The syndrome polynomial $s(X)$ is identical to the error polynomial $e(X)$, assuming that the errors are confined to the $(n - k)$ parity-check bits of the received word polynomial $r(X)$.*

The assumption made here is another way of saying that the degree of the error polynomial $e(X)$ is less than or equal to $(n - k - 1)$. Since the generator polynomial $g(X)$ is of degree $(n - k)$, by definition, it follows that Equation (10.50) can only be satisfied if the quotient $u(X)$ is zero. In other words, the error polynomial $e(X)$ and the syndrome polynomial $s(X)$ are one and the same. The implication of Property 3 is that, under the aforementioned conditions, error correction can be accomplished simply by adding the syndrome polynomial $s(X)$ to the received word polynomial $r(X)$.

▶ EXAMPLE 10.3 Hamming Codes Revisited

To illustrate the issues relating to the polynomial representation of cyclic codes, we consider the generation of a (7, 4) cyclic code. With the block length $n = 7$, we start by factorizing $X^7 + 1$ into three *irreducible polynomials*:

$$X^7 + 1 = (1 + X)(1 + X^2 + X^3)(1 + X + X^3)$$

By an "irreducible polynomial" we mean a polynomial that cannot be factored using only polynomials with coefficients from the binary field. An irreducible polynomial of degree m is said to be *primitive* if the smallest positive integer n for which the polynomial divides $X^n + 1$ is $n = 2^m - 1$. For the example at hand, the two polynomials $(1 + X^2 + X^3)$ and $(1 + X + X^3)$ are primitive. Let us take

$$g(X) = 1 + X + X^3$$

as the generator polynomial, whose degree equals the number of parity bits. This means that the parity-check polynomial is given by

$$\begin{aligned} h(X) &= (1 + X)(1 + X^2 + X^3) \\ &= 1 + X + X^2 + X^4 \end{aligned}$$

whose degree equals the number of message bits $k = 4$.

Next, we illustrate the procedure for the construction of a code word by using this generator polynomial to encode the message sequence 1001. The corresponding message polynomial is given by

$$m(X) = 1 + X^3$$

Hence, multiplying $m(X)$ by $X^{n-k} = X^3$, we get

$$X^{n-k}m(X) = X^3 + X^6$$

The second step is to divide $X^{n-k}m(X)$ by $g(X)$, the details of which (for the example at hand) are given below:

$$\begin{array}{r@{}lllll}
 & X^3 & +X & & & \\ \cline{2-6}
X^3+X+1\,\big) & X^6 & & +X^3 & & \\
 & X^6 & +X^4 & +X^3 & & \\ \cline{2-6}
 & & X^4 & & & \\
 & & X^4 & & +X^2 & +X \\ \cline{3-6}
 & & & & X^2 & +X
\end{array}$$

Note that in this long division we have treated subtraction the same as addition, since we are operating in modulo-2 arithmetic. We may thus write

$$\frac{X^3 + X^6}{1 + X + X^3} = X + X^3 + \frac{X + X^2}{1 + X + X^3}$$

That is, the quotient $a(X)$ and remainder $b(X)$ are as follows, respectively:

$$a(X) = X + X^3$$
$$b(X) = X + X^2$$

Hence, from Equation (10.38) we find that the desired code polynomial is

$$\begin{aligned} c(X) &= b(X) + X^{n-k}m(X) \\ &= X + X^2 + X^3 + X^6 \end{aligned}$$

The code word is therefore 0111001. The four right-most bits, 1001, are the specified message bits. The three left-most bits, 011, are the parity-check bits. The code word thus generated is exactly the same as the corresponding one shown in Table 10.1 for a (7, 4) Hamming code.

We may generalize this result by stating that *any cyclic code generated by a primitive polynomial is a Hamming code of minimum distance 3.*

We next show that the generator polynomial $g(X)$ and the parity-check polynomial $h(X)$ uniquely specify the generator matrix **G** and the parity-check matrix **H**, respectively.

To construct the 4-by-7 generator matrix **G**, we start with four polynomials represented by $g(X)$ and three cyclic-shifted versions of it, as shown by

$$\begin{aligned} g(X) &= 1 + X + X^3 \\ Xg(X) &= X + X^2 + X^4 \\ X^2g(X) &= X^2 + X^3 + X^5 \\ X^3g(X) &= X^3 + X^4 + X^6 \end{aligned}$$

The polynomials $g(X)$, $Xg(X)$, $X^2g(X)$, and $X^3g(X)$ represent code polynomials in the (7, 4) Hamming code. If the coefficients of these polynomials are used as the elements of the rows of a 4-by-7 matrix, we get the following generator matrix:

$$\mathbf{G}' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Clearly, the generator matrix **G**′ so constructed is not in systematic form. We can put it into a systematic form by adding the first row to the third row, and adding the sum of the first two rows to the fourth row. These manipulations result in the desired generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

which is exactly the same as that in Example 10.2.

We next show how to construct the 3-by-7 parity-check matrix **H** from the parity-check polynomial $h(X)$. To do this, we first take the *reciprocal* of $h(X)$, namely, $X^4h(X^{-1})$. For the problem at hand, we form three polynomials represented by $X^4h(X^{-1})$ and two shifted versions of it, as shown by

$$\begin{aligned} X^4h(X^{-1}) &= 1 + X^2 + X^3 + X^4 \\ X^5h(X^{-1}) &= X + X^3 + X^4 + X^5 \\ X^6h(X^{-1}) &= X^2 + X^4 + X^5 + X^6 \end{aligned}$$

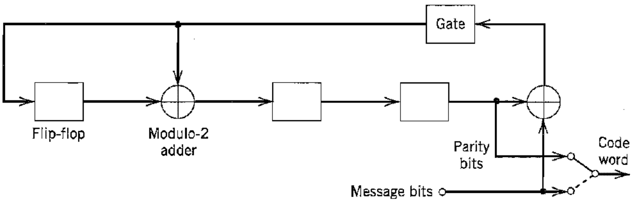


FIGURE 10.10 Encoder for the (7, 4) cyclic code generated by $g(X) = 1 + X + X^3$.

Using the coefficients of these three polynomials as the elements of the rows of the 3-by-7 parity-check matrix, we get

$$\mathbf{H}' = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Here again we see that the matrix $\mathbf{H}'$ is not in systematic form. To put it into a systematic form, we add the third row to the first row to obtain

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

which is exactly the same as that of Example 10.2.

Figure 10.10 shows the encoder for the (7, 4) cyclic Hamming code generated by the polynomial $g(X) = 1 + X + X^3$. To illustrate the operation of this encoder, consider the message sequence (1001). The contents of the shift register are modified by the incoming message bits as in Table 10.3. After four shifts, the contents of the shift register, and therefore the parity bits, are (011). Accordingly, appending these parity bits to the message bits (1001), we get the code word (0111001); this result is exactly the same as that determined earlier in the example.

Figure 10.11 shows the corresponding syndrome calculator for the (7, 4) Hamming code. Let the transmitted code word be (0111001) and the received word be (0110001); that is, the middle bit is in error. As the received bits are fed into the shift register, initially set to zero, its contents are modified as in Table 10.4. At the end of the seventh shift, the syndrome is identified from the contents of the shift register as 110. Since the syndrome is nonzero, the received word is in error. Moreover, from Table 10.2, we see that the error pattern corresponding to this syndrome is 0001000. This indicates that the error is in the middle bit of the received word, which is indeed the case. ◀

TABLE 10.3 ***Contents of the shift register in the encoder of Figure 10.10 for message sequence (1001)***

Shift	*Input*	*Register Contents*
		0 0 0 (initial state)
1	1	1 1 0
2	0	0 1 1
3	0	1 1 1
4	1	0 1 1

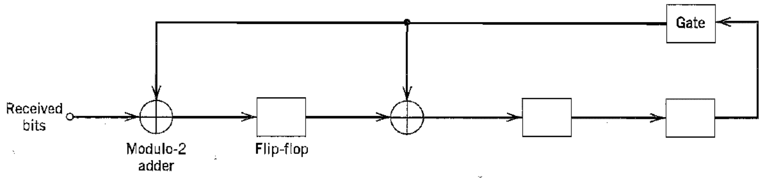


FIGURE 10.11 Syndrome calculator for the (7, 4) cyclic code generated by the polynomial $g(X) = 1 + X + X^3$.

▶ EXAMPLE 10.4 Maximal-Length Codes

For any positive integer $m \geq 3$, there exists a *maximal-length code* with the following parameters:

Block length:	$n = 2^m - 1$
Number of message bits:	$k = m$
Minimum distance:	$d_{\min} = 2^{m-1}$

Maximal-length codes are generated by polynomials of the form

$$g(X) = \frac{1 + X^n}{h(X)} \tag{10.52}$$

where $h(X)$ is any primitive polynomial of degree m. Earlier we stated that any cyclic code generated by a primitive polynomial is a Hamming code of minimum distance 3 (see Example 10.3). It follows therefore that maximal-length codes are the *dual* of Hamming codes.

The polynomial $h(X)$ defines the feedback connections of the encoder. The generator polynomial $g(X)$ defines one period of the maximal-length code, assuming that the encoder is in the initial state 00 . . . 01. To illustrate these points, consider the example of a (7, 3) maximal-length code, which is the dual of the (7, 4) Hamming code described in Example 10.3. Thus, choosing

$$h(X) = 1 + X + X^3$$

we find that the generator polynomial of the (7, 3) maximal-length code is

$$g(X) = 1 + X + X^2 + X^4$$

TABLE 10.4 ***Contents of the syndrome calculator in Figure 10.11 for the received word 0110001***

Shift	*Input Bit*	*Contents of Shift Register*
		0 0 0 (initial state)
1	1	1 0 0
2	0	0 1 0
3	0	0 0 1
4	0	1 1 0
5	1	1 1 1
6	1	0 0 1
7	0	1 1 0

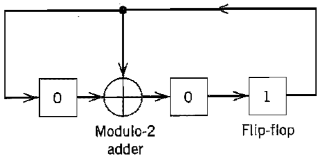


FIGURE 10.12 Encoder for the (7, 3) maximal-length code; the initial state of the encoder is shown in the figure.

Figure 10.12 shows the encoder for the (7, 3) maximal-length code, the feedback connections of which are exactly the same as those shown in Figure 8.2 in Chapter 8. The period of the code is $n = 7$. Thus, assuming that the encoder is in the initial state 001, as indicated in Figure 10.12, we find the output sequence is described by

$$\underbrace{1\ \ 0\ \ 0}_{\substack{\text{initial}\\ \text{state}}}\qquad \underbrace{1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0}_{g(X)\,=\,1+X+X^2+X^4}$$

This result may be readily validated by cycling through the encoder of Figure 10.12.

Note that if we were to choose the other primitive polynomial

$$h(X) = 1 + X^2 + X^3$$

for the (7, 3) maximal-length code, we would simply get the "image" of the code described above, and the output sequence would be "reversed" in time. ◀

▪ OTHER CYCLIC CODES

We conclude the discussion of cyclic codes by presenting the characteristics of three other important classes of cyclic codes.

Cyclic Redundancy Check Codes

Cyclic codes are extremely well-suited for *error detection*. We make this statement for two reasons. First, they can be designed to detect many combinations of likely errors. Second, the implementation of both encoding and error-detecting circuits is practical. It is for these reasons that many of the error-detecting codes used in practice are of the cyclic-code type. A cyclic code used for error-detection is referred to as *cyclic redundancy check (CRC) code.*

We define an *error burst* of length B in an n-bit received word as a contiguous sequence of B bits in which the first and last bits or any number of intermediate bits are received in error. Binary (n, k) CRC codes are capable of detecting the following error patterns:

1. All error bursts of length $n - k$ or less.
2. A fraction of error bursts of length equal to $n - k + 1$; the fraction equals $1 - 2^{-(n-k-1)}$.
3. A fraction of error bursts of length greater than $n - k + 1$; the fraction equals $1 - 2^{-(n-k-1)}$.
4. All combinations of $d_{\min} - 1$ (or fewer) errors.
5. All error patterns with an odd number of errors if the generator polynomial $g(X)$ for the code has an even number of nonzero coefficients.

TABLE 10.5 *CRC codes*

Code	*Generator Polynomial,* g(X)	n − k
CRC-12 code	$1 + X + X^2 + X^3 + X^{11} + X^{12}$	12
CRC-16 code (USA)	$1 + X^2 + X^{15} + X^{16}$	16
CRC-ITU code	$1 + X^5 + X^{12} + X^{16}$	16

Table 10.5 presents the generator polynomials of three CRC codes that have become international standards. All three codes contain $1 + X$ as a prime factor. The CRC-12 code is used for 6-bit characters, and the other two codes are used for 8-bit characters. CRC codes provide a powerful method of error detection for use in automatic-repeat request (ARQ) strategies discussed in Section 10.1, and digital subscriber lines discussed in Chapter 4.

Bose–Chaudhuri–Hocquenghem (BCH) Codes[5]

One of the most important and powerful classes of linear-block codes are *BCH codes*, which are cyclic codes with a wide variety of parameters. The most common binary BCH codes, known as *primitive BCH codes*, are characterized for any positive integers m (equal to or greater than 3) and t [less than $(2^m - 1)/2$] by the following parameters:

Block length:	$n = 2^m - 1$
Number of message bits:	$k \geq n - mt$
Minimum distance:	$d_{\min} \geq 2t + 1$

Each BCH code is a *t-error correcting code* in that it can detect and correct up to t random errors per code word. The Hamming single-error correcting codes can be described as BCH codes. The BCH codes offer flexibility in the choice of code parameters, namely, block length and code rate. Furthermore, for block lengths of a few hundred bits or less, the BCH codes are among the best known codes of the same block length and code rate.

A detailed treatment of the construction of BCH codes is beyond the scope of our present discussion. To provide a feel for their capability, we present in Table 10.6, the code parameters and generator polynomials for binary block BCH codes of length up to $2^5 - 1$. For example, suppose we wish to construct the generator polynomial for (15, 7)

TABLE 10.6 *Binary BCH codes of length up to $2^5 - 1$*

n	k	t	*Generator Polynomial*								
7	4	1								1	011
15	11	1								10	011
15	7	2							111	010	001
15	5	3						10	100	110	111
31	26	1								100	101
31	21	2						11	101	101	001
31	16	3				1	000	111	110	101	111
31	11	5			101	100	010	011	011	010	101
31	6	7	11	001	011	011	110	101	000	100	111

Notation: n = block length
k = number of message bits
t = maximum number of detectable errors

The high-order coefficients of the generator polynomial g(X) are at the left.

BCH code. From Table 10.6 we have (111 010 001) for the coefficients of the generator polynomial; hence, we write

$$g(X) = X^8 + X^7 + X^6 + X^4 + 1$$

Reed–Solomon Codes[6]

The *Reed–Solomon codes* are an important subclass of *nonbinary* BCH codes; they are often abbreviated as RS codes. The encoder for an RS code differs from a binary encoder in that it operates on multiple bits rather than individual bits. Specifically, an RS (n, k) code is used to encode m-bit symbols into blocks consisting of $n = 2^m - 1$ symbols, that is, $m(2^m - 1)$ bits, where $m \geq 1$. Thus, the encoding algorithm expands a block of k symbols to n symbols by adding $n - k$ redundant symbols. When m is an integer power of two, the m-bit symbols are called *bytes*. A popular value of m is 8; indeed, 8-bit RS codes are extremely powerful.

A t-error-correcting RS code has the following parameters:

Block length:	$n = 2^m - 1$ symbols
Message size:	k symbols
Parity-check size:	$n - k = 2t$ symbols
Minimum distance:	$d_{\min} = 2t + 1$ symbols

The block length of the RS code is one less than the size of a code symbol, and the minimum distance is one greater than the number of parity-check symbols. The RS codes make highly efficient use of redundancy, and block lengths and symbol sizes can be adjusted readily to accommodate a wide range of message sizes. Moreover, the RS codes provide a wide range of code rates that can be chosen to optimize performance. Finally, efficient decoding techniques are available for use with RS codes, which is one more reason for their wide application (e.g., compact disc digital audio systems).

10.5 *Convolutional Codes*[7]

In block coding, the encoder accepts a k-bit message block and generates an n-bit code word. Thus, code words are produced on a block-by-block basis. Clearly, provision must be made in the encoder to buffer an entire message block before generating the associated code word. There are applications, however, where the message bits come in *serially* rather than in large blocks, in which case the use of a buffer may be undesirable. In such situations, the use of *convolutional coding* may be the preferred method. A convolutional coder generates redundant bits by using *modulo-2 convolutions*, hence the name.

The encoder of a binary convolutional code with rate $1/n$, measured in bits per symbol, may be viewed as a *finite-state machine* that consists of an M-stage shift register with prescribed connections to n modulo-2 adders, and a multiplexer that serializes the outputs of the adders. An L-bit message sequence produces a coded output sequence of length $n(L + M)$ bits. The *code rate* is therefore given by

$$r = \frac{L}{n(L + M)} \quad \text{bits/symbol} \tag{10.53}$$

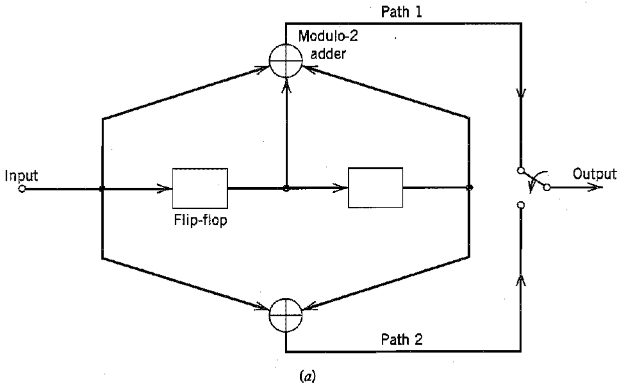
Typically, we have $L \gg M$. Hence, the code rate simplifies to

$$r \simeq \frac{1}{n} \quad \text{bits/symbol} \tag{10.54}$$

The *constraint length* of a convolutional code, expressed in terms of message bits, is defined as the number of shifts over which a single message bit can influence the encoder output. In an encoder with an M-stage shift register, the *memory* of the encoder equals M message bits, and $K = M + 1$ shifts are required for a message bit to enter the shift register and finally come out. Hence, the constraint length of the encoder is K.

Figure 10.13*a* shows a convolutional encoder with $n = 2$ and $K = 3$. Hence, the code rate of this encoder is 1/2. The encoder of Figure 10.13*a* operates on the incoming message sequence, one bit at a time.

We may generate a binary convolutional code with rate k/n by using k separate shift registers with prescribed connections to n modulo-2 adders, an input multiplexer and



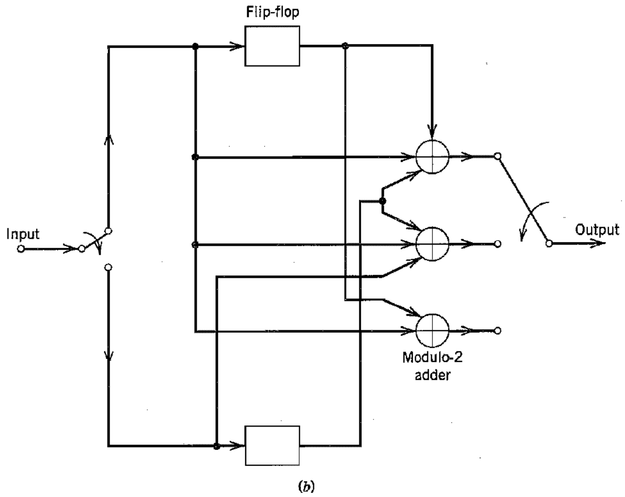


FIGURE 10.13 (*a*) Constraint length-3, rate-$\frac{1}{2}$ convolutional encoder. (*b*) Constraint length-2, rate-$\frac{2}{3}$ convolutional encoder.

an output multiplexer. An example of such an encoder is shown in Figure 10.13*b*, where $k = 2$, $n = 3$, and the two shift registers have $K = 2$ each. The code rate is 2/3. In this second example, the encoder processes the incoming message sequence two bits at a time.

The convolutional codes generated by the encoders of Figure 10.13 are *nonsystematic* codes. Unlike block coding, the use of nonsystematic codes is ordinarily preferred over systematic codes in convolutional coding.

Each path connecting the output to the input of a convolutional encoder may be characterized in terms of its *impulse response*, defined as the response of that path to a symbol 1 applied to its input, with each flip-flop in the encoder set initially in the zero state. Equivalently, we may characterize each path in terms of a *generator polynomial*, defined as the *unit-delay transform* of the impulse response. To be specific, let the *generator sequence* $(g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)})$ denote the impulse response of the ith path, where the coefficients $g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)}$ equal 0 or 1. Correspondingly, the *generator polynomial* of the ith path is defined by

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)}D + g_2^{(i)}D^2 + \cdots + g_M^{(i)}D^M \tag{10.55}$$

where D denotes the unit-delay variable. The complete convolutional encoder is described by the set of generator polynomials $\{g^{(1)}(D), g^{(2)}(D), \ldots, g^{(n)}(D)\}$. Traditionally, different variables are used for the description of convolutional and cyclic codes, with D being commonly used for convolutional codes and X for cyclic codes.

▶ EXAMPLE 10.5

Consider the convolutional encoder of Figure 10.13*a*, which has two paths numbered 1 and 2 for convenience of reference. The impulse response of path 1 (i.e., upper path) is (1, 1, 1). Hence, the corresponding generator polynomial is given by

$$g^{(1)}(D) = 1 + D + D^2$$

The impulse response of path 2 (i.e., lower path) is (1, 0, 1). Hence, the corresponding generator polynomial is given by

$$g^{(2)}(D) = 1 + D^2$$

For the message sequence (10011), say, we have the polynomial representation

$$m(D) = 1 + D^3 + D^4$$

As with Fourier transformation, convolution in the time domain is transformed into multiplication in the D-domain. Hence, the output polynomial of path 1 is given by

$$\begin{aligned} c^{(1)}(D) &= g^{(1)}(D)m(D) \\ &= (1 + D + D^2)(1 + D^3 + D^4) \\ &= 1 + D + D^2 + D^3 + D^6 \end{aligned}$$

From this we immediately deduce that the output sequence of path 1 is (1111001). Similarly, the output polynomial of path 2 is given by

$$\begin{aligned} c^{(2)}(D) &= g^{(2)}(D)m(D) \\ &= (1 + D^2)(1 + D^3 + D^4) \\ &= 1 + D^2 + D^3 + D^4 + D^5 + D^6 \end{aligned}$$

The output sequence of path 2 is therefore (1011111). Finally, multiplexing the two output sequences of paths 1 and 2, we get the encoded sequence

$$\mathbf{c} = (11, 10, 11, 11, 01, 01, 11)$$

Note that the message sequence of length $L = 5$ bits produces an encoded sequence of length $n(L + K - 1) = 14$ bits. Note also that for the shift register to be restored to its zero initial state, a terminating sequence of $K - 1 = 2$ zeros is appended to the last input bit of the message sequence. The terminating sequence of $K - 1$ zeros is called the *tail of the message.* ◀

Code Tree, Trellis, and State Diagram

Traditionally, the structural properties of a convolutional encoder are portrayed in graphical form by using any one of three equivalent diagrams: code tree, trellis, and state diagram. We will use the convolutional encoder of Figure 10.13*a* as a running example to illustrate the insights that each one of these three diagrams can provide.

We begin the discussion with the *code tree* of Figure 10.14. Each branch of the tree represents an input symbol, with the corresponding pair of output binary symbols indicated on the branch. The convention used to distinguish the input binary symbols 0 and 1 is as follows. An input 0 specifies the upper branch of a bifurcation, whereas input 1 specifies the lower branch. A specific *path* in the tree is traced from left to right in accordance with the input (message) sequence. The corresponding coded symbols on the branches of that path constitute the input (message) sequence. Consider, for example, the message sequence (10011) applied to the input of the encoder of Figure 10.13*a*. Following the procedure just described, we find that the corresponding encoded sequence is (11, 10, 11, 11, 01), which agrees with the first 5 pairs of bits in the encoded sequence $\{c_i\}$ derived in Example 10.5.

From the diagram of Figure 10.14, we observe that the tree becomes *repetitive* after the first three branches. Indeed, beyond the third branch, the two nodes labeled a are identical, and so are all the other node pairs that are identically labeled. We may establish this repetitive property of the tree by examining the associated encoder of Figure 10.13*a*. The encoder has memory $M = K - 1 = 2$ message bits. Hence, when the third message bit enters the encoder, the first message bit is shifted out of the register. Consequently, after the third branch, the message sequences ($100\, m_3 m_4 \ldots$) and ($000\, m_3 m_4 \ldots$) generate the same code symbols, and the pair of nodes labeled a may be joined together. The same reasoning applies to other nodes. Accordingly, we may collapse the code tree of Figure 10.14 into the new form shown in Figure 10.15, which is called a *trellis*.[8] It is so called since a trellis is a treelike structure with remerging branches. The convention used in Figure 10.15 to distinguish between input symbols 0 and 1 is as follows. A code branch produced by an input 0 is drawn as a solid line, whereas a code branch produced by an input 1 is drawn as a dashed line. As before, each input (message) sequence corresponds to a specific path through the trellis. For example, we readily see from Figure 10.15 that the message sequence (10011) produces the encoded output sequence (11, 10, 11, 11, 01), which agrees with our previous result.

A trellis is more instructive than a tree in that it brings out explicitly the fact that the associated convolutional encoder is a *finite-state machine.* We define the *state* of a convolutional encoder of rate $1/n$ as the $(K - 1)$ message bits stored in the encoder's shift register. At time j, the portion of the message sequence containing the most recent K bits is written as $(m_{j-K+1}, \ldots, m_{j-1}, m_j)$, where m_j is the *current* bit. The $(K - 1)$-bit state of the encoder at time j is therefore written simply as $(m_{j-1}, \ldots, m_{j-K+2}, m_{j-K+1})$. In the

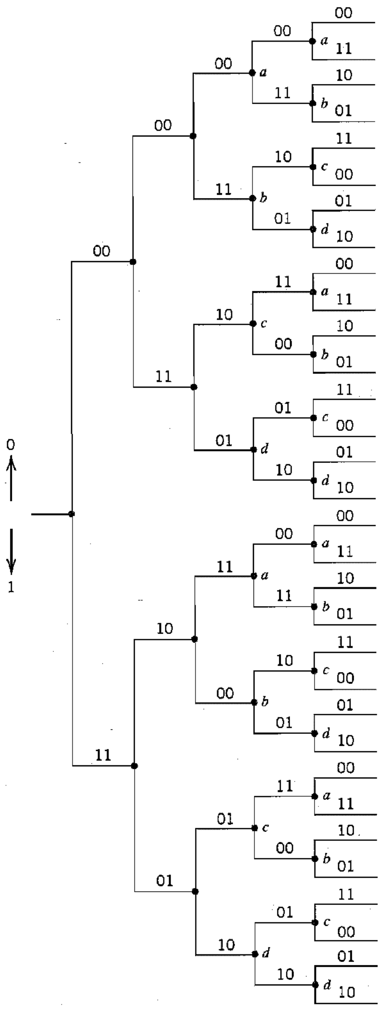


FIGURE 10.14 Code tree for the convolutional encoder of Figure 10.13*a*.

case of the simple convolutional encoder of Figure 10.13*a* we have $(K - 1) = 2$. Hence, the state of this encoder can assume any one of four possible values, as described in Table 10.7. The trellis contains $(L + K)$ *levels*, where L is the length of the incoming message sequence, and K is the constraint length of the code. The levels of the trellis are labeled as $j = 0, 1, \ldots, L + K - 1$ in Figure 10.15 for $K = 3$. Level j is also referred to as *depth* j; both terms are used interchangeably. The first $(K - 1)$ levels correspond to the encoder's departure from the initial state a, and the last $(K - 1)$ levels correspond to the encoder's

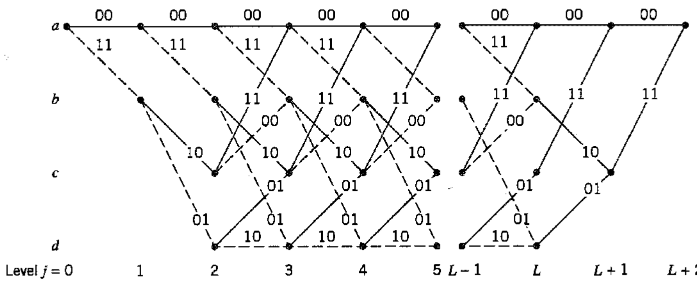


FIGURE 10.15 Trellis for the convolutional encoder of Figure 10.13*a*.

return to the state *a*. Clearly, not all the states can be reached in these two portions of the trellis. However, in the central portion of the trellis, for which the level *j* lies in the range $K - 1 \leq j \leq L$, all the states of the encoder are reachable. Note also that the central portion of the trellis exhibits a fixed periodic structure.

Consider next a portion of the trellis corresponding to times *j* and $j + 1$. We assume that $j \geq 2$ for the example at hand, so that it is possible for the current state of the encoder to be *a*, *b*, *c*, or *d*. For convenience of presentation, we have reproduced this portion of the trellis in Figure 10.16*a*. The left nodes represent the four possible current states of the encoder, whereas the right nodes represent the next states. Clearly, we may coalesce the left and right nodes. By so doing, we obtain the *state diagram* of the encoder, shown in Figure 10.16*b*. The nodes of the figure represent the four possible states of the encoder, with each node having two incoming branches and two outgoing branches. A transition from one state to another in response to input 0 is represented by a solid branch, whereas a transition in response to input 1 is represented by a dashed branch. The binary label on each branch represents the encoder's output as it moves from one state to another. Suppose, for example, the current state of the encoder is (01), which is represented by node *c*. The application of input 1 to the encoder of Figure 10.13*a* results in the state (10) and the encoded output (00). Accordingly, with the help of this state diagram, we may readily determine the output of the encoder of Figure 10.13*a* for any incoming message sequence. We simply start at state *a*, the all-zero initial state, and walk through the state diagram in accordance with the message sequence. We follow a solid branch if the input is a 0 and a dashed branch if it is a 1. As each branch is traversed, we output the corresponding binary label on the branch. Consider, for example, the message sequence (10011). For this input we follow the path *abcabd*, and therefore output the sequence (11, 10, 11, 11, 01), which

TABLE 10.7 ***State table for the convolutional encoder of Figure 10.13*a**

State	*Binary Description*
a	00
b	10
c	01
d	11

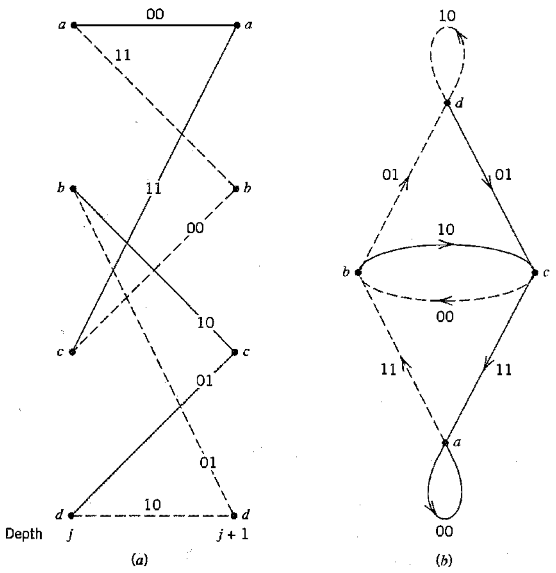


FIGURE 10.16 (*a*) A portion of the central part of the trellis for the encoder of Figure 10.13*a* (*b*) State diagram of the convolutional encoder of Figure 10.13*a*.

agrees exactly with our previous result. Thus, the input–output relation of a convolutional encoder is also completely described by its state diagram.

10.6 *Maximum Likelihood Decoding of Convolutional Codes*

Now that we understand the operation of a convolutional encoder, the next issue to be considered is the decoding of a convolutional code. In this section we first describe the underlying theory of maximum likelihood decoding, and then present an efficient algorithm for its practical implementation.

Let $\mathbf{m}$ denote a *message vector*, and $\mathbf{c}$ denote the corresponding *code vector* applied by the encoder to the input of a discrete memoryless channel. Let $\mathbf{r}$ denote the *received vector*, which may differ from the transmitted code vector due to channel noise. Given the received vector $\mathbf{r}$, the decoder is required to make an *estimate* $\hat{\mathbf{m}}$ of the message vector. Since there is a one-to-one correspondence between the message vector $\mathbf{m}$ and the code vector $\mathbf{c}$, the decoder may equivalently produce an estimate $\hat{\mathbf{c}}$ of the code vector. We may then put $\hat{\mathbf{m}} = \mathbf{m}$ if and only if $\hat{\mathbf{c}} = \mathbf{c}$. Otherwise, a *decoding error* is committed in the receiver. The *decoding rule* for choosing the estimate $\hat{\mathbf{c}}$, given the received vector $\mathbf{r}$, is said to be optimum when the *probability of decoding error* is minimized. From the material presented in Chapter 6, we may state that for equiprobable messages, the probability of decoding error is minimized if the estimate $\hat{\mathbf{c}}$ is chosen to maximize the *log-likelihood function*. Let $p(\mathbf{r}\,|\,\mathbf{c})$ denote the conditional probability of receiving $\mathbf{r}$, given that $\mathbf{c}$ was sent.

The log-likelihood function equals $\log p(\mathbf{r}|\mathbf{c})$. The *maximum likelihood decoder* or decision rule is described as follows:

$$\text{Choose the estimate } \hat{\mathbf{c}} \text{ for which the log-likelihood function } \log p(\mathbf{r}|\mathbf{c}) \text{ is maximum.} \tag{10.56}$$

Consider now the special case of a binary symmetric channel. In this case, both the transmitted code vector c and the received vector r represent binary sequences of length N, say. Naturally, these two sequences may differ from each other in some locations because of errors due to channel noise. Let c_i and r_i denote the ith elements of c and r, respectively. We then have

$$p(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{N} p(r_i|c_i) \tag{10.57}$$

Correspondingly, the log-likelihood is

$$\log p(\mathbf{r}|\mathbf{c}) = \sum_{i=1}^{N} \log p(r_i|c_i) \tag{10.58}$$

Let the transition probability $p(r_i|c_i)$ be defined as

$$p(r_i|c_i) = \begin{cases} p, & \text{if } r_i \neq c_i \\ 1 - p, & \text{if } r_i = c_i \end{cases} \tag{10.59}$$

Suppose also that the received vector r differs from the transmitted code vector c in exactly d positions. The number d is the *Hamming distance* between vectors r and c. Then, we may rewrite the log-likelihood function in Equation (10.58) as

$$\begin{aligned} \log p(\mathbf{r}|\mathbf{c}) &= d \log p + (N - d) \log(1 - p) \\ &= d \log\left(\frac{p}{1 - p}\right) + N \log(1 - p) \end{aligned} \tag{10.60}$$

In general, the probability of an error occurring is low enough for us to assume $p < 1/2$. We also recognize that $N \log(1 - p)$ is a constant for all c. Accordingly, we may restate the maximum-likelihood decoding rule for the binary symmetric channel as follows:

$$\text{Choose the estimate } \hat{\mathbf{c}} \text{ that minimizes the Hamming distance between the received vector } \mathbf{r} \text{ and the transmitted vector } \mathbf{c}. \tag{10.61}$$

That is, for the binary symmetric channel, the maximum-likelihood decoder reduces to a *minimum distance decoder*. In such a decoder, the received vector r is compared with each possible transmitted code vector c, and the particular one closest to r is chosen as the correct transmitted code vector. The term "closest" is used in the sense of minimum number of differing binary symbols (i.e., Hamming distance) between the code vectors under investigation.

THE VITERBI ALGORITHM[9]

The equivalence between maximum likelihood decoding and minimum distance decoding for a binary symmetric channel implies that we may decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places. Since a code tree is equivalent to a trellis, we may equally limit our choice to the possible paths in the trellis representation of the code. The reason for preferring the trellis over the tree is that the number of nodes at any level of the trellis does not continue to grow as the number of incoming message bits increases; rather, it remains constant at 2^{K-1}, where K is the constraint length of the code.

Consider, for example, the trellis diagram of Figure 10.15 for a convolutional code with rate $r = 1/2$ and constraint length $K = 3$. We observe that at level $j = 3$, there are two paths entering any of the four nodes in the trellis. Moreover, these two paths will be identical onward from that point. Clearly, a minimum distance decoder may make a decision at that point as to which of those two paths to retain, without any loss of performance. A similar decision may be made at level $j = 4$, and so on. This sequence of decisions is exactly what the *Viterbi algorithm* does as it walks through the trellis. The algorithm operates by computing a *metric* or discrepancy for every possible path in the trellis. The metric for a particular path is defined as the Hamming distance between the coded sequence represented by that path and the received sequence. Thus, for each node (state) in the trellis of Figure 10.15 the algorithm compares the two paths entering the node. The path with the lower metric is retained, and the other path is discarded. This computation is repeated for every level j of the trellis in the range $M \leq j \leq L$, where $M = K - 1$ is the encoder's memory and L is the length of the incoming message sequence. The paths that are retained by the algorithm are called *survivor* or *active paths*. For a convolutional code of constraint length $K= 3$, for example, no more than $2^{K-1} = 4$ survivor paths and their metrics will ever be stored. This list of 2^{K-1} paths is always guaranteed to contain the maximum-likelihood choice.

A difficulty that may arise in the application of the Viterbi algorithm is the possibility that when the paths entering a state are compared, their metrics are found to be identical. In such a situation, we make the choice by flipping a fair coin (i.e., simply make a guess).

In summary, the Viterbi algorithm is a maximum-likelihood decoder, which is optimum for an AWGN channel. It proceeds in a step-by-step fashion as follows:

Initialization

Label the left-most state of the trellis (i.e., the all-zero state at level 0) as 0, since there is no discrepancy at this point in the computation.

Computation step j + 1

Let $j = 0, 1, 2, \ldots$, and suppose that at the previous step j we have done two things:

- All survivor paths are identified.
- The survivor path and its metric for each state of the trellis are stored.

Then, at level (clock time) $j + 1$, compute the metric for all the paths entering each state of the trellis by adding the metric of the incoming branches to the metric of the connecting survivor path from level j. Hence, for each state, identify the path with the lowest metric as the survivor of step $j + 1$, thereby updating the computation.

Final Step

Continue the computation until the algorithm completes its forward search through the trellis and therefore reaches the termination node (i.e., all-zero state), at which time it makes a decision on the maximum likelihood path. Then, like a block decoder, the sequence of symbols associated with that path is released to the destination as the decoded version of the received sequence. In this sense, it is therefore more correct to refer to the Viterbi algorithm as a *maximum likelihood sequence estimator.*

However, when the received sequence is very long (near infinite), the storage requirement of the Viterbi algorithm becomes too high, and some compromises must be made.

The approach usually taken is to "truncate" the path memory of the decoder as described here. A *decoding window* of length ℓ is specified, and the algorithm operates on a corresponding frame of the received sequence, always stopping after ℓ steps. A decision is then made on the "best" path and the symbol associated with the first branch on that path is released to the user. The symbol associated with the last branch of the path is dropped. Next, the decoding window is moved forward one time interval, and a decision on the next code frame is made, and so on. The decoding decisions made in this way are no longer truly maximum likelihood, but they can be made almost as good provided that the decoding window is long enough. Experience and analysis have shown that satisfactory results are obtained if the decoding window length ℓ is on the order of 5 times the constraint length K of the convolutional code or more.

▶ Example 10.6 Correct Decoding of Received All-Zero Sequence

Suppose that the encoder of Figure 10.13*a* generates an all-zero sequence that is sent over a binary symmetric channel, and that the received sequence is (0100010000 . . .). There are two errors in the received sequence due to noise in the channel: one in the second bit and the other in the sixth bit. We wish to show that this double-error pattern is correctable through the application of the Viterbi decoding algorithm.

In Figure 10.17, we show the results of applying the algorithm for level $j = 1, 2, 3, 4, 5$. We see that for $j = 2$ there are (for the first time) four paths, one for each of the four states of the encoder. The figure also includes the metric of each path for each level in the computation.

In the left side of Figure 10.17, for $j = 3$ we show the paths entering each of the states, together with their individual metrics. In the right side of the figure, we show the four survivors that result from application of the algorithm for level $j = 3, 4, 5$.

Examining the four survivors in Figure 10.17 for $j = 5$, we see that the all-zero path has the smallest metric and will remain the path of smallest metric from this point forward. This clearly shows that the all-zero sequence is the maximum likelihood choice of the Viterbi decoding algorithm, which agrees exactly with the transmitted sequence. ◀

▶ Example 10.7 Incorrect Decoding of Received All-Zero Sequence

Suppose next that the received sequence is (1100010000 . . .), which contains three errors compared to the transmitted all-zero sequence.

In Figure 10.18, we show the results of applying the Viterbi decoding algorithm for $j = 1, 2, 3, 4$. We see that in this example the correct path has been eliminated by level $j = 3$. Clearly, a triple-error pattern is uncorrectable by the Viterbi algorithm when applied to a convolutional code of rate 1/2 and constraint length $K = 3$. The exception to this rule is a triple-error pattern spread over a time span longer than one constraint length, in which case it is very likely to be correctable. ◀

Free Distance of a Convolutional Code

The performance of a convolutional code depends not only on the decoding algorithm used but also on the distance properties of the code. In this context, the most important single measure of a convolutional code's ability to combat channel noise is the free distance, denoted by d_{free}. The *free distance* of a convolutional code is defined as *the minimum Hamming distance between any two code words in the code*. A convolutional code with free distance d_{free} can correct t errors if and only if d_{free} is greater than $2t$.

The free distance can be obtained quite simply from the state diagram of the convolutional encoder. Consider, for example, Figure 10.16*b*, which shows the state diagram

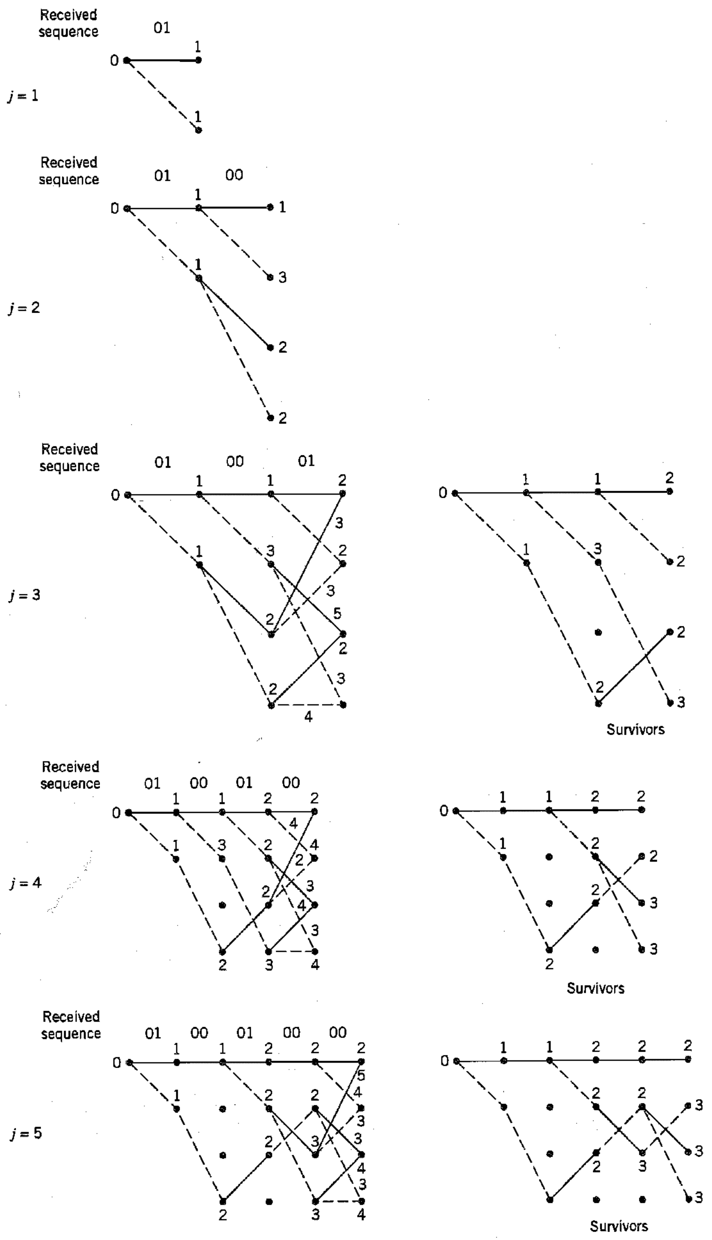


FIGURE 10.17 Illustrating steps in the Viterbi algorithm for Example 10.6.

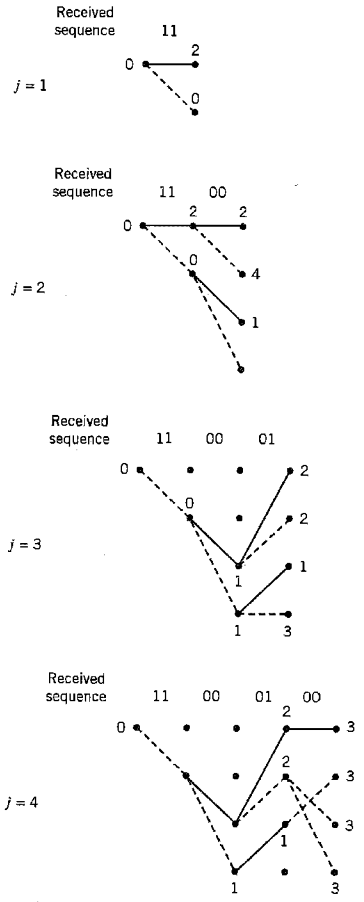


FIGURE 10.18 Illustrating breakdown of the Viterbi algorithm in Example 10.7.

of the encoder of Figure 10.13*a*. Any nonzero code sequence corresponds to a complete path beginning and ending at the 00 state (i.e., node *a*). We thus find it useful to split this node in the manner shown in the modified state diagram of Figure 10.19, which may be viewed as a *signal-flow graph* with a single input and a single output. A signal-flow graph consists of *nodes* and directed *branches*; it operates by the following rules:

1. A branch multiplies the signal at its input node by the *transmittance* characterizing that branch.
2. A node with incoming branches *sums* the signals produced by all of those branches.
3. The signal at a node is applied equally to all the branches outgoing from that node.
4. The *transfer function* of the graph is the ratio of the output signal to the input signal.

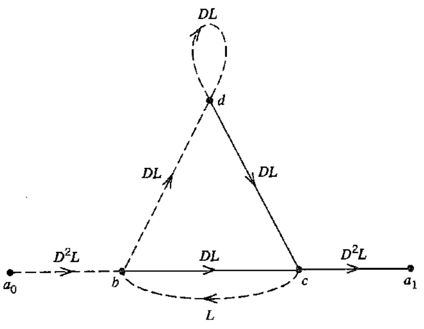


FIGURE 10.19 Modified state diagram of convolutional encoder.

Returning to the signal-flow graph of Figure 10.19, we note that the exponent of D on a branch in this graph describes the Hamming weight of the encoder output corresponding to that branch. The exponent of L is always equal to one, since the length of each branch is one. Let $T(D, L)$ denote the transfer function of the signal-flow graph, with D and L playing the role of dummy variables. For the example of Figure 10.19, we may readily use rules 1, 2, and 3 to obtain the following input-output relations:

$$\left.\begin{aligned} b &= D^2La_0 + Lc \\ c &= DLb + DLd \\ d &= DLb + DLd \\ a_1 &= D^2Lc \end{aligned}\right\} \tag{10.62}$$

where a_0, b, c, d, and a_1 denote the node signals of the graph. Solving the set of Equations (10.62) for the ratio a_1/a_0, we find that the transfer function of the graph in Figure 10.19 is given by

$$T(D, L) = \frac{D^5L^3}{1 - DL(1 + L)} \tag{10.63}$$

Using the binomial expansion, we may equivalently write

$$T(D, L) = D^5L^3 \sum_{i=0}^{\infty} (DL(1 + L))^i \tag{10.64}$$

Setting $L = 1$ in Equation (10.64), we thus get the *distance transfer function* expressed in the form of a power series:

$$T(D, 1) = D^5 + 2D^6 + 4D^7 + \cdots \tag{10.65}$$

Since the free distance is the minimum Hamming distance between any two code words in the code and the distance transfer function $T(D, 1)$ enumerates the number of code words that are a given distance apart, it follows that the exponent of the first term in the expansion of $T(D, 1)$ defines the free distance. Thus, on the basis of Equation (10.65), the convolutional code of Figure 10.13a has a free distance $d_{\text{free}} = 5$.

This result indicates that up to two errors in the received sequence are correctable, for two or fewer transmission errors will cause the received sequence to be at most at a Hamming distance of 2 from the transmitted sequence but at least at a Hamming distance of 3 from any other code sequence in the code. In other words, in spite of the presence of

TABLE 10.8 *Maximum free distances attainable with systematic and nonsystematic convolutional codes of rate 1/2*

Constraint Length K	*Systematic*	*Nonsystematic*
2	3	3
3	4	5
4	4	6
5	5	7
6	6	8
7	6	10
8	7	10

any pair of transmission errors, the received sequence remains closer to the transmitted sequence than any other possible code sequence. However, this statement is no longer true if there are three or more *closely spaced* transmission errors in the received sequence. These observations confirm the results reported earlier in Examples 10.6 and 10.7.

In using the distance transfer function $T(D, 1)$ to calculate the free distance of a convolutional code, it is assumed that the power series in the unit-delay variable D representing $T(D, 1)$ is *convergent* (i.e., its sum has a "finite" value). This assumption is required to justify the expansion given in Equation (10.65) for the convolutional code of Figure 10.13*a*. However, there is no guarantee that $T(D, 1)$ is always convergent. When $T(D, 1)$ is nonconvergent, an infinite number of decoding errors are caused by a finite number of transmission errors; the convolutional code is then subject to catastrophic error propagation, and the code is called a *catastrophic code.*[10] In this context it is noteworthy that a *systematic* convolutional code cannot be catastrophic. Unfortunately, for a prescribed constraint length K, the free distances that can be attained with systematic convolutional codes using schemes such as those shown in Figure 10.13 are usually smaller than for the case of nonsystematic convolutional codes, as indicated in Table 10.8.

ASYMPTOTIC CODING GAIN[11]

The transfer function of the encoder state diagram, modified in a manner similar to that illustrated in Figure 10.19, may be used to evaluate a *bound on the bit error rate* for a given decoding scheme; details of this evaluation are, however, beyond the scope of our present discussion. Here we simply summarize the results for two special channels, namely, the binary symmetric channel and the binary-input additive white Gaussian noise (AWGN) channel, assuming the use of binary phase-shift keying (PSK) with coherent detection.

1. *Binary symmetric channel.* The binary symmetric channel may be modeled as an additive white Gaussian noise channel with binary phase-shift keying (PSK) as the modulation and with hard-decision demodulation. The transition probability p of the binary symmetric channel is then equal to the bit error rate (BER) for the uncoded binary PSK system. From Chapter 6 we recall that for large values of E_b/N_0, the ratio of signal energy per bit-to-noise power spectral density, the bit error rate for binary PSK without coding is dominated by the exponential factor $\exp(-E_b/N_0)$. On the other hand, the bit error rate for the same modulation scheme with convolutional coding is dominated by the exponential

factor $\exp(-d_{\text{free}} r E_b/2N_0)$, where r is the code rate and d_{free} is the free distance of the convolutional code. Therefore, as a figure of merit for measuring the improvement in error performance made by the use of coding with hard-decision decoding, we may use the exponents to define the *asymptotic coding gain* (in decibels) as follows:

$$G_a = 10 \log_{10}\left(\frac{d_{\text{free}} r}{2}\right) \text{ dB} \tag{10.66}$$

2. *Binary-input AWGN channel.* Consider next the case of a memoryless binary-input AWGN channel with no output quantization [i.e., the output amplitude lies in the interval $(-\infty, \infty)$]. For this channel, theory shows that for large values of E_b/N_0 the bit error rate for binary PSK with convolutional coding is dominated by the exponential factor $\exp(-d_{\text{free}} r E_b/N_0)$, where the parameters are as previously defined. Accordingly, in this case, we find that the asymptotic coding gain is defined by

$$G_a = 10 \log_{10}(d_{\text{free}} r) \text{ dB} \tag{10.67}$$

From Equations (10.66) and (10.67) we see that the asymptotic coding gain for the binary-input AWGN channel is greater than that for the binary symmetric channel by 3 dB. In other words, for large E_b/N_0, the transmitter for a binary symmetric channel must generate an additional 3 dB of signal energy (or power) over that for a binary-input AWGN channel if we are to achieve the same error performance. Clearly, there is an advantage to be gained by permitting an unquantized demodulator output instead of making hard decisions. This improvement in performance, however, is attained at the cost of increased decoder complexity due to the requirement for accepting analog inputs.

The asymptotic coding gain for a binary-input AWGN channel is approximated to within about 0.25 dB by a binary input Q-ary output discrete memoryless channel with the number of representation levels $Q = 8$. This means that we may avoid the need for an analog decoder by using a soft-decision decoder that performs finite output quantization (typically, $Q = 8$), and yet realize a performance close to the optimum.

10.7 Trellis-Coded Modulation[12]

In the traditional approach to channel coding described in the preceding sections of the chapter, encoding is performed separately from modulation in the transmitter; likewise for decoding and detection in the receiver. Moreover, error control is provided by transmitting additional redundant bits in the code, which has the effect of lowering the information bit rate per channel bandwidth. That is, bandwidth efficiency is traded for increased power efficiency.

To attain a more effective utilization of the available bandwidth and power, coding and modulation have to be treated as a single entity. We may deal with this new situation by redefining coding as *the process of imposing certain patterns on the transmitted signal.* Indeed, this definition includes the traditional idea of parity coding.

Trellis codes for band-limited channels result from the treatment of modulation and coding as a *combined* entity rather than as two separate operations. The combination itself is referred to as *trellis-coded modulation* (TCM). This form of signaling has three basic features:

1. The number of signal points in the constellation used is larger than what is required for the modulation format of interest with the same data rate; the additional points allow redundancy for forward error-control coding without sacrificing bandwidth.

2. Convolutional coding is used to introduce a certain dependency between successive signal points, such that only certain *patterns* or *sequences of signal points* are permitted.
3. Soft-decision decoding is performed in the receiver, in which the permissible sequence of signals is modeled as a trellis structure; hence, the name "trellis codes."

This latter requirement is the result of using an enlarged signal constellation. By increasing the size of the constellation, the probability of symbol error increases for a fixed signal-to-noise ratio. Hence, with hard-decision demodulation we would face a performance loss before we begin. Performing soft-decision decoding on the combined code and modulation trellis ameliorates this problem.

In the presence of AWGN, maximum likelihood decoding of trellis codes consists of finding that particular path through the trellis with *minimum squared Euclidean distance* to the received sequence. Thus, in the design of trellis codes, the emphasis is on maximizing the Euclidean distance between code vectors (or, equivalently, code words) rather than maximizing the Hamming distance of an error-correcting code. The reason for this approach is that, except for conventional coding with binary PSK and QPSK, maximizing the Hamming distance is not the same as maximizing the squared Euclidean distance. Accordingly, in what follows, the Euclidean distance is adopted as the distance measure of interest. Moreover, while a more general treatment is possible, the discussion is (by choice) confined to the case of *two-dimensional constellations of signal points*. The implication of such a choice is to restrict the development of trellis codes to multilevel amplitude and/or phase modulation schemes such as M-ary PSK and M-ary QAM.

The approach used to design this type of trellis codes involves partitioning an M-ary constellation of interest successively into 2, 4, 8, . . . subsets with size $M/2, M/4, M/8, \ldots$, and having progressively larger increasing minimum Euclidean distance between their respective signal points. Such a design approach by *set partitioning* represents the "key idea" in the construction of efficient coded modulation techniques for band-limited channels.

In Figure 10.20, we illustrate the partitioning procedure by considering a circular constellation that corresponds to 8-PSK. The figure depicts the constellation itself and the 2 and 4 subsets resulting from two levels of partitioning. These subsets share the common

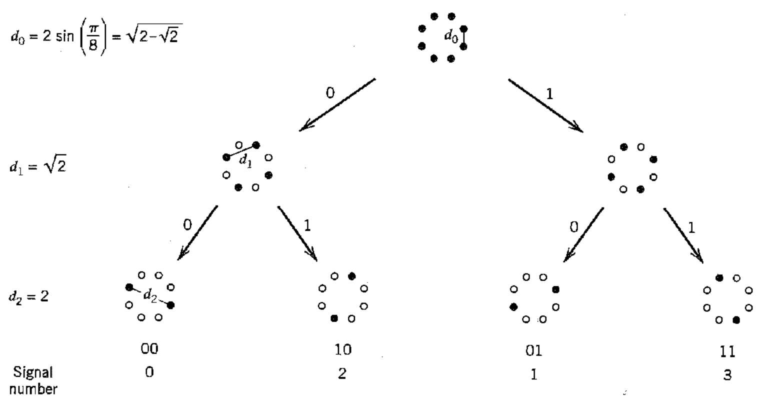


FIGURE 10.20 Partitioning of 8-PSK constellation, which shows that $d_0 < d_1 < d_2$.

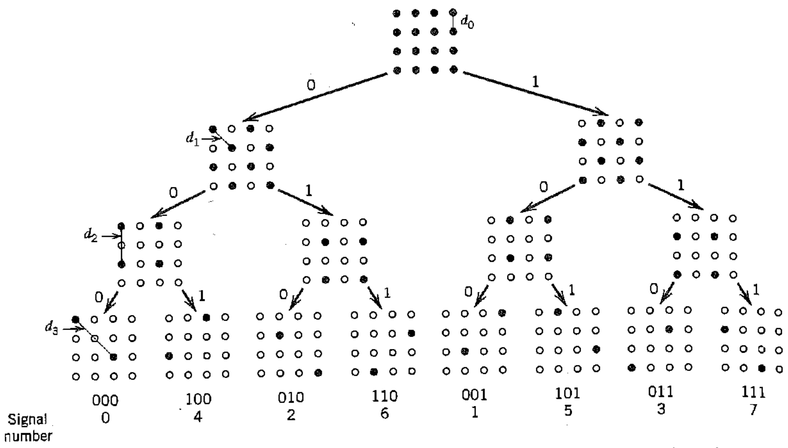


FIGURE 10.21 Partitioning of 16-QAM constellation, which shows that $d_0 < d_1 < d_2 < d_3$.

property that the minimum Euclidean distances between their individual points follow an increasing pattern: $d_0 < d_1 < d_2$.

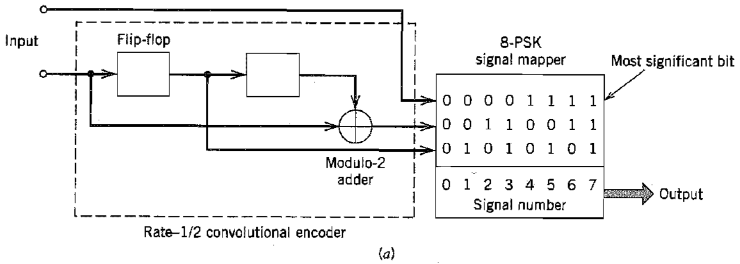
Figure 10.21 illustrates the partitioning of a rectangular constellation corresponding to 16-QAM. Here again we see that the subsets have increasing within-subset Euclidean distances: $d_0 < d_1 < d_2 < d_3$.

Based on the subsets resulting from successive partitioning of a two-dimensional constellation, we may devise relatively simple and yet highly effective coding schemes. Specifically, to send n bits/symbol with *quadrature modulation* (i.e., one that has in-phase and quadrature components), we start with a two-dimensional constellation of 2^{n+1} signal points appropriate for the modulation format of interest; a circular grid is used for M-ary PSK, and a rectangular one for M-ary QAM. In any event, the constellation is partitioned into 4 or 8 subsets. One or two incoming bits per symbol enter a rate-1/2 or rate-2/3 binary convolutional encoder, respectively; the resulting two or three coded bits per symbol determine the selection of a particular subset. The remaining uncoded data bits determine which particular point from the selected subset is to be signaled. This class of trellis codes is known as *Ungerboeck codes*.

Since the modulator has memory, we may use the Viterbi algorithm to perform maximum likelihood sequence estimation at the receiver. Each branch in the trellis of the Ungerboeck code corresponds to a subset rather than an individual signal point. The first step in the detection is to determine the signal point within each subset that is closest to the received signal point in the Euclidean sense. The signal point so determined and its metric (i.e., the squared Euclidean distance between it and the received point) may be used thereafter for the branch in question, and the Viterbi algorithm may then proceed in the usual manner.

■ UNGERBOECK CODES FOR 8-PSK

The scheme of Figure 10.22*a* depicts the simplest Ungerboeck 8-PSK code for the transmission of 2 bits/symbol. The scheme uses a rate-1/2 convolutional encoder; the corre-



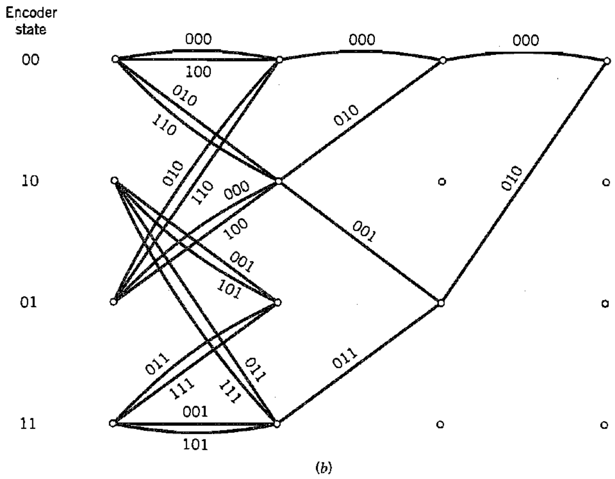
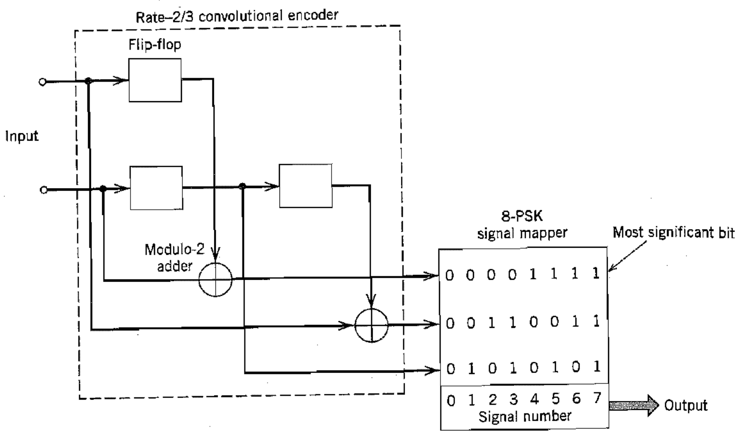


Figure 10.22 (*a*) Four-state Ungerboeck code for 8-PSK; the mapper follows Figure 10.20. (*b*) Trellis of the code.

sponding trellis of the code is shown in Figure 10.22*b*, which has four states. Note that the most significant bit of the incoming binary word is left uncoded. Therefore, each branch of the trellis may correspond to two different output values of the 8-PSK modulator or, equivalently, to one of the four 2-point subsets shown in Figure 10.20. The trellis of Figure 10.22*b* also includes the minimum distance path.

The scheme of Figure 10.23*a* depicts another Ungerboeck 8-PSK code for transmitting 2 bits/sample; it is next in the level of complexity. This second scheme uses a rate-2/3 convolutional encoder. Therefore, the corresponding trellis of the code has eight states, as shown in Figure 10.23*b*. In this case, both bits of the incoming binary word are encoded. Hence, each branch of the trellis corresponds to a specific output value of the 8-PSK modulator. The trellis of Figure 10.23*b* also includes the minimum distance path.

Figures 10.22*b* and 10.23*b* also include the encoder states. In Figure 10.22, the state of the encoder is defined by the contents of the two-stage shift register. On the other hand, in Figure 10.23, it is defined by the content of the single-stage (top) shift register followed by that of the two-stage (bottom) shift register.



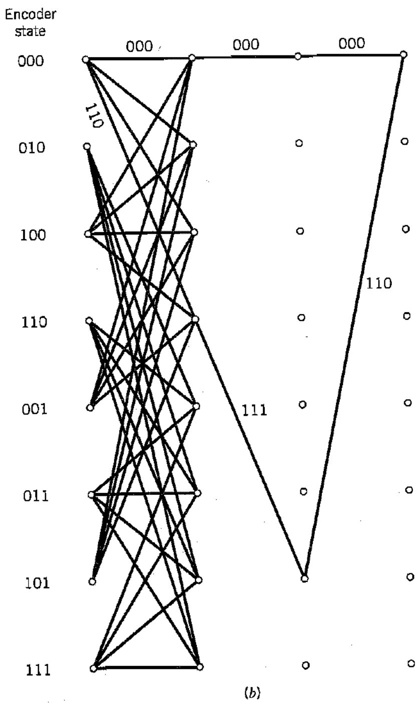


FIGURE 10.23 (*a*) Eight-state Ungerboeck code for 8-PSK; the mapper follows Figure 10.20. (*b*) Trellis of the code with only some of the branches shown.

▨ Asymptotic Coding Gain

Following the discussion in Section 10.6, we define the *asymptotic coding gain* of Ungerboeck codes as

$$G_a = 10 \log_{10}\left(\frac{d_{\text{free}}^2}{d_{\text{ref}}^2}\right) \tag{10.68}$$

where d_{free} is the *free Euclidean distance* of the code and d_{ref} is the minimum Euclidean distance of an uncoded modulation scheme operating with the same signal energy per bit. For example, by using the Ungerboeck 8-PSK code of Figure 10.22*a*, the signal constellation has 8 message points, and we send 2 message bits per point. Hence, uncoded transmission requires a signal constellation with 4 message points. We may therefore regard uncoded 4-PSK as the reference for the Ungerboeck 8-PSK code of Figure 10.22*a*.

The Ungerboeck 8-PSK code of Figure 10.22*a* achieves an asymptotic coding gain of 3 dB, calculated as follows:

1. Each branch of the trellis in Figure 10.22*b* corresponds to a subset of two antipodal signal points. Hence, the free Euclidean distance d_{free} of the code can be no larger than the Euclidean distance d_2 between the antipodal signal points of such a subset. We may therefore write

$$d_{\text{free}} = d_2 = 2$$

   where the distance d_2 is defined in Figure 10.24*a*; see also Figure 10.20.
2. The minimum Euclidean distance of an uncoded QPSK, viewed as a reference operating with the same signal energy per bit, equals (see Figure 10.24*b*)

$$d_{\text{ref}} = \sqrt{2}$$

Hence, as previously stated, the use of Equation (10.68) yields an asymptotic coding gain of $10 \log_{10} 2 = 3$ dB.

The asymptotic coding gain achievable with Ungerboeck codes increases with the number of states in the convolutional encoder. Table 10.9 presents the asymptotic coding gain (in dB) for Ungerboeck 8-PSK codes for increasing number of states, expressed with

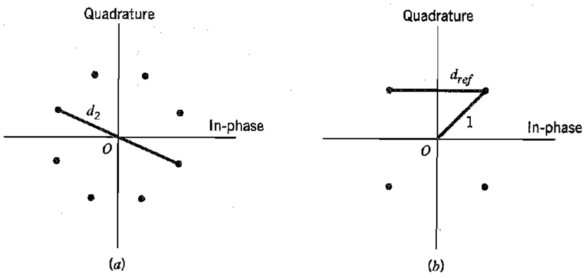


Figure 10.24 Signal-space diagrams for calculation of asymptotic coding gain of Ungerboeck 8-PSK code. (*a*) Definition of distance d_2. (*b*) Definition of reference distance d_{ref}.

TABLE 10.9 *Asymptotic coding gain of Ungerboeck 8-PSK codes, with respect to uncoded 4-PSK*

Number of states	4	8	16	32	64	128	256	512
Coding gain (dB)	3	3.6	4.1	4.6	4.8	5	5.4	5.7

respect to uncoded 4-PSK. Note that improvements on the order of 6 dB require codes with a very large number of states.

10.8 *Turbo Codes*[13]

Traditionally, the design of good codes has been tackled by constructing codes with a great deal of algebraic structure, for which there are feasible decoding schemes. Such an approach is exemplified by the linear block codes and convolutional codes discussed in preceding sections. The difficulty with these traditional codes is that, in an effort to approach the theoretical limit for Shannon's channel capacity, we need to increase the code-word length of a linear block code or the constraint length of a convolutional code, which, in turn, causes the computational complexity of a maximum likelihood decoder to increase exponentially. Ultimately, we reach a point where complexity of the decoder is so high that it becomes physically unrealizable.

Various approaches have been proposed for the construction of powerful codes with large "equivalent" block lengths structured in such a way that the decoding can be split into a number of manageable steps. Building on these previous approaches, the development of *turbo codes* and *low-density parity-check codes* has been by far most successful. Indeed, this development has opened a brand new and exciting way of constructing good codes and decoding them with feasible complexity. Turbo codes are discussed in this section and low-density parity-check codes are discussed in Section 10.10.

■ TURBO CODING

In its most basic form, the encoder of a turbo code consists of two *constituent* systematic encoders joined together by means of an interleaver, as illustrated in Figure 10.25.

An *interleaver* is an input-output mapping device that *permutes* the ordering of a sequence of symbols from a fixed alphabet in a completely deterministic manner; that is, it takes the symbols at the input and produces identical symbols at the output but in a different temporal order. The interleaver can be of many types, of which the periodic and pseudo-random are two. Turbo codes use a pseudo-random interleaver, which operates

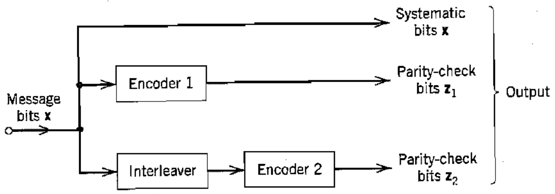


FIGURE 10.25 Block diagram of turbo encoder.

only on the systematic bits. There are two reasons for the use of an interleaver in a turbo code:

- To tie together errors that are easily made in one half of the turbo code to errors that are exceptionally unlikely to occur in the other half. This is indeed the main reason why the turbo code performs better than a traditional code.
- To provide robust performance with respect to mismatched decoding, which is a problem that arises when the channel statistics are not known or have been incorrectly specified.

Typically, but not necessarily, the same code is used for both constituent encoders in Figure 10.25. The constituent codes recommended for turbo codes are short constraint-length *recursive systematic convolutional (RSC) codes.* The reason for making the convolutional codes recursive (i.e., feeding one or more of the tap outputs in the shift register back to the input) is to make the internal state of the shift register depend on past outputs. This affects the behavior of the error patterns (a single error in the systematic bits produces an infinite number of parity errors), with the result that a better performance of the overall coding strategy is attained.

EXAMPLE 10.8 Eight-state RSC Encoder

Figure 10.26 shows an example eight-state RSC encoder. The generator matrix for this recursive convolutional code is

$$g(D) = \left[1, \frac{1 + D + D^2 + D^3}{1 + D + D^3}\right] \tag{10.69}$$

where D is the delay variable. The second entry of the matrix $g(D)$ is the transfer function of the feedback shift register, defined as the transform of the output divided by the transform of the input. Let $M(D)$ denote the transform of the message sequence $\{m_i\}_{i=1}^{k}$ and $B(D)$ denote the transform of the parity sequence $\{b_i\}_{i=1}^{n-k}$. By definition, we have

$$\frac{B(D)}{M(D)} = \frac{1 + D + D^2 + D^3}{1 + D + D^3}$$

Cross-multiplying, we get:

$$(1 + D + D^2 + D^3)M(D) = (1 + D + D^3)B(D)$$

which, on inversion into the time domain, yields

$$m_i + m_{i-1} + m_{i-2} + m_{i-3} + b_i + b_{i-1} + b_{i-3} = 0 \tag{10.70}$$

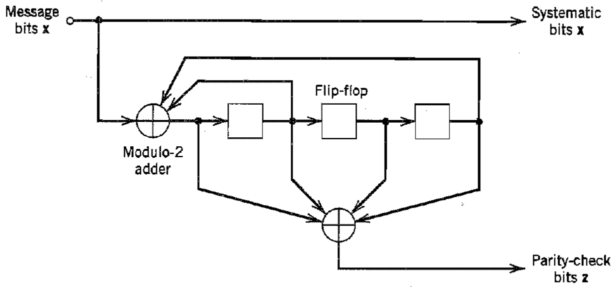


FIGURE 10.26 Example eight-state recursive systematic convolutional (RSC) encoder.

where the addition is modulo-2. Equation (10.70) is the parity-check equation, which the convolutional encoder of Figure 10.26 satisfies at each time step i. ◀

In Figure 10.25 the input data stream is applied directly to encoder 1, and the pseudo-randomly reordered version of the same data stream is applied to encoder 2. The systematic bits (i.e., original message bits) and the two sets of parity-check bits generated by the two encoders constitute the output of the turbo encoder. Although the constituent codes are convolutional, in reality turbo codes are block codes with the block size being determined by the size of the interleaver. Moreover, since both RSC encoders in Figure 10.25 are linear, we may describe turbo codes as *linear block codes.*

The block nature of the turbo code raises a practical issue: How do we know the beginning and the end of a code word? The common practice is to initialize the encoder to the *all-zero state* and then encode the data. After encoding a certain number of data bits a number of tail bits are added so as to make the encoder return to the all-zero state at the end of each block; thereafter the cycle is repeated. The termination approaches of turbo codes include the following:

- A simple approach is to terminate the first RSC code in the encoder and leave the second one unterminated. A drawback of this approach is that the bits at the end of the block due to the second RSC code are more vulnerable to noise than the other bits. Experimental work has shown that turbo codes exhibit a leveling off in performance as the SNR increases. This behavior is not like an error floor, but it has the appearance of an error floor compared to the steep drop in error performance at low SNR. This *error floor* is affected by a number of factors, the dominant one of which is the choice of interleaver.
- A more refined approach[14] is to terminate both constituent codes in the encoder in a symmetric manner. Through the combined use of a good interleaver and dual termination, the error floor can be reduced by an order of magnitude compared to the simple termination approach.

In the original version of the turbo encoder, the parity-check bits generated by the two encoders in Figure 10.25 were punctured prior to data transmission over the channel to maintain the rate at 1/2. A *punctured code* is constructed by deleting certain parity check bits, thereby increasing the data rate. Puncturing is the inverse of extending a code. It should, however, be emphasized that the use of a puncture map is not a necessary requirement for the generation of turbo codes.

The novelty of the parallel encoding scheme of Figure 10.25 is in the use of recursive systematic convolutional (RSC) codes and the introduction of a pseudo-random interleaver between the two encoders. Thus a turbo code appears essentially *random* to the channel by virtue of the pseudo-random interleaver, yet it possesses sufficient structure for the decoding to be physically realizable. Coding theory asserts that a code chosen at random is capable of approaching Shannon's channel capacity, provided that the block size is sufficiently large.[15] This is indeed the reason behind the impressive performance of turbo codes, as discussed next.

▪ PERFORMANCE OF TURBO CODES

Figure 10.27 shows the error performance of a 1/2 rate, turbo code with a large block size for binary data transmission over an AWGN channel.[16] The code uses an interleaver of

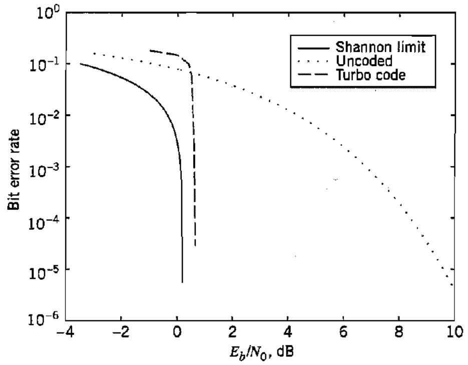


FIGURE 10.27 Noise performances of 1/2 rate, turbo code and uncoded transmission for AWGN channel; the figure also includes Shannon's theoretical limit on channel capacity for code rate $r = 1/2$.

size 65,536 and a BCJR-based decoder; details of this decoder are presented later in the section. Eighteen iterations of turbo decoding were used in the computation.

For the purpose of comparison, Figure 10.27 also includes two other curves for the same AWGN channel:

- Uncoded transmission (i.e., code rate $r = 1$).
- Shannon's theoretical limit for code rate 1/2, which follows from Figure 9.18*b*.

From Figure 10.27, we may draw two important conclusions:

1. Although the bit error rate for the turbo-coded transmission is significantly higher than that for uncoded transmission at low E_b/N_0, the bit error rate for the turbo-coded transmission drops very rapidly once a critical value of E_b/N_0 has been reached.
2. At a bit error rate of 10^{-5}, the turbo code is less than 0.5 dB from Shannon's theoretical limit.

Note, however, attaining this highly impressive performance requires that the size of the interleaver, or, equivalently, the block length of the turbo code, be large. Also, the large number of iterations needed to improve performance increases the decoder latency. This drawback is due to the fact that the digital processing of information does not lend itself readily to the application of feedback, which is a distinctive feature of the turbo decoder.

Now that we have an appreciation for the impressive performance of turbo codes, the stage is set for a discussion of how turbo decoding is actually performed.

TURBO DECODING

Turbo codes derive their distinctive name from analogy of the decoding algorithm to the "turbo engine" principle. Figure 10.28*a* shows the basic structure of the turbo decoder. It operates on noisy versions of the systematic bits and the two sets of parity-check bits in two decoding stages to produce an estimate of the original message bits.

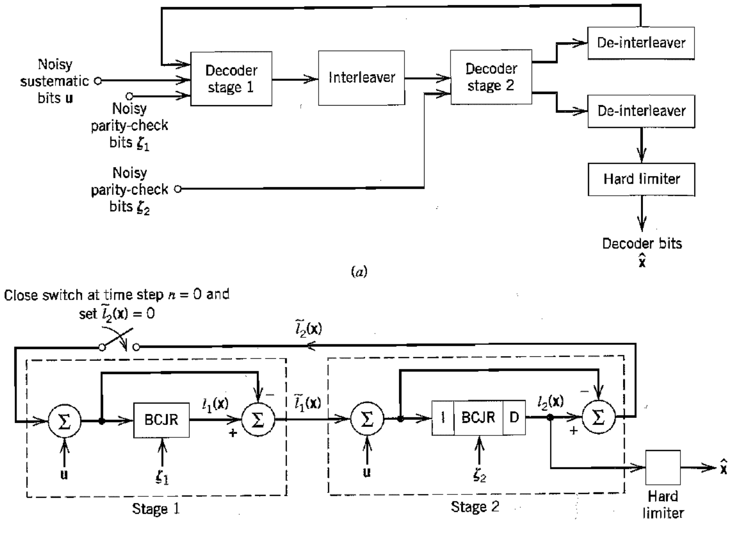


FIGURE 10.28 (*a*) Block diagram of turbo decoder. (*b*) Extrinsic form of turbo decoder, where I stands for interleaver, D for de-interleaver, and BCJR for BCJR algorithm for log-MAP decoding.

Each of the two decoding stages uses a *BCJR algorithm*,[17] which was originally invented by Bahl, Cocke, Jelinek, and Raviv (hence the name) to solve a *maximum a posteriori probability (MAP) detection* problem. The BCJR algorithm differs from the Viterbi algorithm in two fundamental respects:

1. The BCJR algorithm is a *soft input–soft output* decoding algorithm with two recursions, one forward and the other backward, both of which involve soft decisions. In contrast, the Viterbi algorithm is a *soft input–hard output* decoding algorithm, with a single forward recursion involving soft decisions; the recursion ends with a hard decision, whereby a particular survivor path among several ones is retained. In computational terms, the BCJR algorithm is therefore more complex than the Viterbi algorithm because of the backward recursion.
2. The BCJR algorithm is a MAP decoder in that it minimizes the bit errors by estimating the *a posteriori* probabilities of the individual bits in a code word; to reconstruct the original data sequence, the soft outputs of the BCJR algorithm are hard-limited. On the other hand, the Viterbi algorithm is a maximum likelihood sequence estimator in that it maximizes the likelihood function for the whole sequence, not each bit. As such, the average bit error rate of the BCJR algorithm can be slightly better than the Viterbi algorithm; it is never worse.

Most important, formulation of the BCJR algorithm rests on the fundamental assumptions that (1) the channel encoding, namely, the convolutional encoding performed in the transmitter, is modeled as a *Markov process*, and (2) the channel is memoryless. In the context of our present discussion, the Markovian assumption means that if a code can be represented as a trellis, then the present state of the trellis depends only on the past state and the input bit. (A mathematical treatment of the BCJR algorithm is given later in this section.)

Before proceeding to describe the operation of the two-stage turbo decoder in Figure 10.28*a*, we find it desirable to introduce the notion of extrinsic information. The most convenient representation for this concept is as a log-likelihood ratio, in which case extrinsic information is computed as the difference between two log-likelihood ratios as depicted in Figure 10.29. Formally, *extrinsic information*, generated by a decoding stage for a set of systematic (message) bits, is defined as the difference between the log-likelihood ratio computed at the output of that decoding stage and the *intrinsic information* represented by a log-likelihood ratio fed back to the input of the decoding stage. In effect, extrinsic information is the incremental information gained by exploiting the dependencies that exist between a message bit of interest and incoming raw data bits processed by the decoder.

On this basis, we may depict the flow of information in the two-stage turbo decoder of Figure 10.28*a* in a *symmetric extrinsic* manner as shown in Figure 10.28*b*. The first decoding stage uses the BCJR algorithm to produce a soft estimate of systematic bit x_j, expressed as the log-likelihood ratio

$$l_1(x_j) = \log\left(\frac{P(x_j = 1 \mid \mathbf{u}, \zeta_1, \tilde{l}_2(\mathbf{x}))}{P(x_j = 0 \mid \mathbf{u}, \zeta_1, \tilde{l}_2(\mathbf{x}))}\right), \qquad j = 1, 2, \ldots, k \tag{10.71}$$

where $\mathbf{u}$ is the set of noisy systematic bits, ζ_1 is the set of noisy parity-check bits generated by encoder 1, and $\tilde{l}_2(\mathbf{x})$ is the extrinsic information about the set of message bits $\mathbf{x}$ derived from the second decoding stage and fed back to the first stage. Assuming that the k message bits are statistically independent, the total log-likelihood ratio at the output of the first decoding stage is therefore

$$l_1(\mathbf{x}) = \sum_{j=1}^{k} l_1(x_j) \tag{10.72}$$

Hence, the extrinsic information about the message bits derived from the first decoding stage is

$$\tilde{l}_1(\mathbf{x}) = l_1(\mathbf{x}) - \tilde{l}_2(\mathbf{x}) \tag{10.73}$$

where $\tilde{l}_2(\mathbf{x})$ is to be defined.

Before application to the second decoding stage, the extrinsic information $\tilde{l}_1(\mathbf{x})$ is reordered to compensate for the psuedo-random interleaving introduced in the turbo encoder. In addition, the noisy parity-check bits ζ_2 generated by encoder 2 are used as input. Thus by using the BCJR algorithm, the second decoding stage produces a more refined

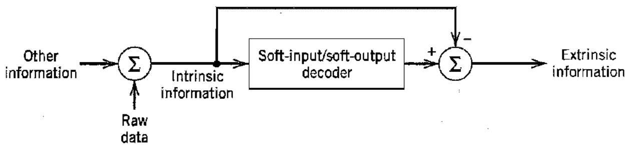


FIGURE 10.29 Illustrating the concept of extrinsic information.

soft estimate of the message bits **x**. This estimate is re-interleaved to produce the total log-likelihood ratio $l_2(\mathbf{x})$. The extrinsic information $\tilde{l}_2(\mathbf{x})$ fed back to the first decoding stage is therefore

$$\tilde{l}_2(\mathbf{x}) = l_2(\mathbf{x}) - \tilde{l}_1(\mathbf{x}) \tag{10.74}$$

where $\tilde{l}_1(\mathbf{x})$ is itself defined by Equation (10.73), and $l_2(\mathbf{x})$ is the log-likelihood ratio computed by the second stage. Specifically, for the jth element of the vector **x**, we have

$$l_2(x_j) = \log_2\left(\frac{P(x_j = 1 \mid \mathbf{u}, \boldsymbol{\zeta}_2, \tilde{l}_1(\mathbf{x}))}{P(x_j = 0 \mid \mathbf{u}, \boldsymbol{\zeta}_2, \tilde{l}_1(\mathbf{x}))}\right), \qquad j = 1, 2, \ldots, k \tag{10.75}$$

Through the application of $\tilde{l}_2(\mathbf{x})$ to the first stage, the feedback loop around the pair of decoding stages is thereby closed. Note that although in actual fact the set of noisy systematic bits **u** is only applied to the first decoding stage as in Figure 10.28*a*, by formulating the information flow in the symmetric extrinsic manner depicted in Figure 10.28*b* we find that **u** is, in effect, also applied to the second decoding stage.

An estimate of the message bits **x** is computed by hard-limiting the log-likelihood ratio $l_2(\mathbf{x})$ at the output of the second stage, as shown by

$$\hat{\mathbf{x}} = \operatorname{sgn}(l_2(\mathbf{x})) \tag{10.76}$$

where the signum function operates on each element of $l_2(\mathbf{x})$ individually.

To initiate the turbo decoding algorithm, we simply set $\tilde{l}_2(\mathbf{x}) = 0$ on the first iteration of the algorithm; see Figure 10.28*b*.

The motivation for feeding only extrinsic information from one stage to the next in the turbo decoder of Figure 10.28 is to maintain as much statistical independence between the bits as possible from one iteration to the next. The feedback decoding strategy described herein implicitly relies on this assumption. If this assumption of statistical independence is strictly true, it can be shown that the estimate $\hat{\mathbf{x}}$ defined in Equation (10.76) approaches the MAP solution as the number of iterations approaches infinity.[18] The assumption of statistical independence appears to be close to the truth in the vast majority of cases encountered in practice.

▪ THE BCJR ALGORITHM

For a discussion of turbo decoding to be complete, a mathematical exposition of the BCJR algorithm for MAP estimation is in order.

Let $x(t)$ be the input to a trellis encoder at time t. Let $y(t)$ be the corresponding output observed at the receiver. Note that $y(t)$ may include more than one observation; for example, a rate $1/n$ code produces n bits for each input bit, in which case we have an n-dimensional observation vector. Let the observation vector be denoted by

$$\mathbf{y}_{(1,t)} = [y(1), y(2), \ldots, y(t)]$$

Let $\lambda_m(t)$ denote the probability that a state $\mathrm{s}(t)$ of the trellis encoder equals m, where $m = 1, 2, \ldots, M$. We may then write

$$\boldsymbol{\lambda}(t) = P[\mathbf{s}(t) \mid \mathbf{y}] \tag{10.77}$$

where $\mathbf{s}(t)$ and $\boldsymbol{\lambda}(t)$ are both M-by-1 vectors. Then, for a rate $1/n$ linear convolutional code with feedback as in the RSC code, the probability that a symbol "1" was the message bit is given by

$$P(x(t) = 1 \mid \mathbf{y}) = \sum_{s \in \mathcal{F}_A} \lambda_s(t) \tag{10.78}$$

where $\mathcal{F}_A$ is the set of transitions that correspond to a symbol "1" at the input, and $\lambda_s(t)$ is the s-component of $\boldsymbol{\lambda}(t)$.

Define the *forward estimation* of state probabilities as the M-by-1 vector

$$\boldsymbol{\alpha}(t) = P(\mathbf{s}(t) \mid \mathbf{y}_{(1,t)}) \tag{10.79}$$

where the observation vector $\mathbf{y}_{(1,t)}$ is defined above. Also define the *backward estimation* of state probabilities as the M-by-1 vector

$$\boldsymbol{\beta}(t) = P(\mathbf{s}(t) \mid \mathbf{y}_{(t,k)}) \tag{10.80}$$

where

$$\mathbf{y}_{(t,k)} = [y(t), y(t+1), \ldots, y(k)]$$

The vectors $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$ are estimates of the state probabilities at time t based on the past and future data, respectively. We may then formulate the *separability theorem* as follows:

The state probabilities at time t are related to the forward estimator $\boldsymbol{\alpha}(t)$ and backward estimator $\boldsymbol{\beta}(t)$ by the vector

$$\boldsymbol{\lambda}(t) = \frac{\boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t)}{\| \boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t) \|_1} \tag{10.81}$$

where $\boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t)$ is the vector product of $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$, and $\| \boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t) \|_1$ is the L_1 norm of this vector product.

The *vector product* $\boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t)$ (not to be confused with the inner product) is defined in terms of the individual elements of $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$ by

$$\boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t) = \begin{bmatrix} \alpha_1(t)\beta_1(t) \\ \alpha_2(t)\beta_2(t) \\ \vdots \\ \alpha_M(t)\beta_M(t) \end{bmatrix} \tag{10.82}$$

and the L_1 *norm* of $\boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t)$ is defined by

$$\| \boldsymbol{\alpha}(t) \cdot \boldsymbol{\beta}(t) \|_1 = \sum_{m=1}^{M} \alpha_m(t)\beta_m(t) \tag{10.83}$$

The separability theorem says that the state distribution at time t given the past is independent of the state distribution at time t given the future, which is intuitively satisfying recalling the Markovian assumption for channel encoding, which is basic to the BCJR algorithm. Moreover, this theorem provides the basis of a simple way of combining the forward and backward estimates to obtain a complete description of the state probabilities.

To proceed further, let the state transition probability at time t be defined by

$$\gamma_{m',m}(t) = P(s(t) = m, \mathbf{y}(t) \mid s(t-1) = m') \tag{10.84}$$

and denote the M-by-M matrix of transition probabilities as

$$\mathbf{\Gamma}(t) = \{\gamma_{m',m}(t)\} \tag{10.85}$$

We may then formulate the *recursion theorem* as follows:

The forward estimate $\boldsymbol{\alpha}(t)$ and backward estimate $\boldsymbol{\beta}(t)$ are computed recursively as

$$\boldsymbol{\alpha}^T(t) = \frac{\boldsymbol{\alpha}^T(t-1)\mathbf{\Gamma}(t)}{\| \boldsymbol{\alpha}^T(t-1)\mathbf{\Gamma}(t) \|_1} \tag{10.86}$$

and

$$\boldsymbol{\beta}(t) = \frac{\mathbf{\Gamma}(t+1)\boldsymbol{\beta}(t+1)}{\| \mathbf{\Gamma}(t+1)\boldsymbol{\beta}(t+1) \|_1} \tag{10.87}$$

where the superscript T denotes matrix transposition.

The separability and recursion theorems together define the BCJR algorithm for the computation of *a posteriori* probabilities of the states and transitions of a code trellis, given the observation vector. Using these estimates, the likelihood ratios needed for turbo decoding may then be computed by performing summations over selected subsets of states as required.

10.9 *Computer Experiment: Turbo Decoding*

Two properties constitute the hallmark of turbo codes:

Property 1:

The error performance of the turbo decoder improves with the number of iterations of the decoding algorithm. This is achieved by feeding extrinsic information from the output of the first decoding stage to the input of the second decoding stage in the forward path and feeding extrinsic information from the output of the second stage to the input of the first stage in the backward path, and then permitting the iterative decoding process to take its natural course in response to the received noisy message and parity bits.

Property 2

The turbo decoder is capable of approaching the Shannon theoretical limit of channel capacity in a computationally feasible manner; this property has been demonstrated experimentally but not yet proven theoretically.

Property 2 requires that the block length of the turbo code be large. Unfortunately, a demonstration of this property requires the use of sophisticated implementations of the turbo decoding algorithm that are beyond the scope of this book. Accordingly, we focus our attention on a demonstration of Property 1 in this computer experiment.

So, as the primary objective of this computer experiment, we wish to use the log-MAP implementation of the BCJR algorithm to demonstrate Property 1 of turbo decoding.

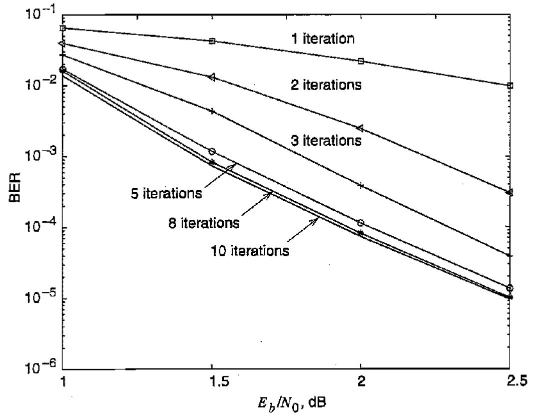


FIGURE 10.30 Results of the computer experiment on turbo decoding, for increasing number of iterations.

The only channel impairment assumed in the experiment is additive white Gaussian noise. Details of the turbo encoder and decoder are as follows:

Turbo Encoder (described in Figure 10.25):

Encoder 1: convolutional encoder [1, 1, 1]
Encoder 2: convolutional encoder [1, 0, 1]
Block (i.e., interleaver) length: 1,200 bits

Turbo Decoder (described in Figure 10.28):

The BCJR algorithm for log-MAP decoding.

The experiment was carried out for $E_b/N_0 = 1$, 1.5, 2, and 2.5 dB, with varying number of iterations at each E_b/N_0. For each trial of the experiment, the number of bit errors was calculated after accumulating a total of 20 blocks of data (each 1,200 bits long) that were noise-corrupted. The probability of error was then evaluated as the ratio of bit errors to the total number of encoded bits. Note that in this calculation, many of the blocks of encoded bits were correctly decoded.

The results of the experiment are plotted in Figure 10.30. The following observations can be made from this figure:

1. For a given E_b/N_0, the probability of error decreases with increasing number of iterations, confirming Property 1 of turbo decoding.
2. After eight iterations, there is no significant improvement in decoding performance.
3. For a fixed number of iterations, the probability of error decreases with increasing E_b/N_0, which is to be expected.

10.10 *Low-Density Parity-Check Codes*[19]

Turbo codes, discussed in Section 10.8, and low-density parity-check (LDPC) codes, discussed in this section, belong to a broad family of error-control coding techniques called *compound codes*. The two most important advantages of LDPC codes over turbo codes are:

- Absence of low-weight code words.
- Iterative decoding of lower complexity.

With regard to the issue of low-weight code words, we usually find that a small number of code words in a turbo code are undesirably close to the given code word. Due to this closeness in weights, once in a while the channel noise causes the transmitted code word to be mistaken for a nearby code word. Indeed, it is this behavior that is responsible for the error floor (typically around a bit error rate of 10^{-5} to 10^{-6}) that was mentioned earlier. In contrast, LDPC codes can be easily constructed so that they do not have such low-weight code words, and they can therefore achieve *vanishingly small* bit error rates. The error-floor problem in turbo codes can be alleviated by careful design of the interleaver.

Turning next to the issue of decoding complexity, we note that the computational complexity of a turbo decoder is dominated by the BCJR algorithm, which operates on the trellis for the convolutional code used in the encoder. The number of computations in each recursion of the BCJR algorithm scales linearly with the number of states in the trellis. Commonly used turbo codes employ trellises with 16 states or more. In contrast, LDPC codes use a simple parity-check trellis that has just two states. Consequently, the decoders for LDPC codes are significantly simpler than those for turbo decoders. Moreover, being parallelizable, LDPC decoding may be performed at greater speeds than turbo decoding.

However, a practical objection to the use of LDPC codes is that for large block lengths, their encoding complexity is high compared to turbo codes.

▪ CONSTRUCTION OF LDPC CODES

LDPC codes are specified by a parity-check matrix denoted by **A**, which is *sparse*; that is, it consists mainly of 0s and a small number of 1s. In particular, we speak of (n, t_c, t_r) LDPC codes, where n denotes the block length, t_c denotes the weight (i.e., number of 1s) in each column of the matrix **A**, and t_r denotes the weight of each row with $t_r > t_c$. The rate of such a LDPC code is

$$r = 1 - \frac{t_c}{t_r} \tag{10.88}$$

whose validity may be justified as follows. Let ρ denote the *density* of 1s in the parity-check matrix **A**. Then, following the terminology introduced in Section 10.3, we may set

$$t_c = \rho(n - k)$$

and

$$t_r = \rho n$$

where $(n - k)$ is the number of rows in **A** and n is the number of columns (i.e., the block length). Therefore, dividing t_c by t_r, we get

$$\frac{t_c}{t_r} = 1 - \frac{k}{n}$$

By definition, the code rate of a block code is k/n, hence the result of Equation (10.88) follows. For this result to hold, however, the rows of **A** must be *linearly independent.*

The structure of LDPC codes is well portrayed by *bipartite graphs*. Figure 10.31 shows such a graph for the example code of $n = 10$, $t_c = 3$, and $t_r = 5$. The left-hand nodes in the graph of Figure 10.31 are *variable nodes*, which correspond to elements of the code word. The right-hand nodes of the graph are *check nodes*, which correspond to the set of parity-check constraints satisfied by code words in the code. LDPC codes of the type exemplified by the graph of Figure 10.31 are said to be *regular* in that all the nodes of a similar kind have exactly the same degree. In the example graph of Figure 10.31, the degree of the variable nodes is $t_c = 3$, and the degree of the check nodes is $t_r = 5$. As the block length n approaches infinity, each check node is connected to a vanishingly small fraction of variable nodes, hence the term *low-density.*

The matrix **A** is constructed by putting 1s in **A** at *random*, subject to the *regularity constraints*:

- Each column contains a small fixed number, t_c, of 1s.
- Each row contains a small fixed number, t_r, of 1s.

In practice, these regularity constraints are often violated slightly in order to avoid having linearly dependent rows in the parity-check matrix **A**.

Unlike the linear block codes discussed in Section 10.3, the parity-check matrix **A** of LDPC codes is *not* systematic (i.e., it does not have the parity-check bits appearing in diagonal form), hence the use of a symbol different from that used in Section 10.3. Nevertheless, for coding purposes, we may derive a generator matrix **G** for LDPC codes by means of *Gaussian elimination* performed in modulo-2 arithmetic; this procedure is illustrated later in Example 10.9. Following the terminology introduced in Section 10.3, the 1-by-n code vector **c** is first partitioned as

$$\mathbf{c} = [\mathbf{b} \vdots \mathbf{m}]$$

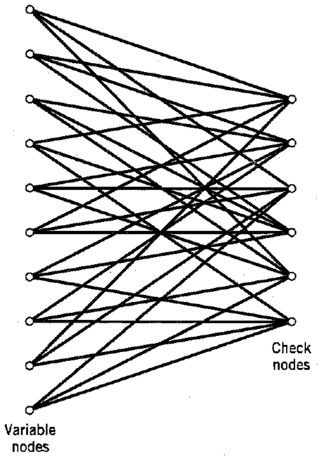


FIGURE 10.31 Bipartite graph of the (10, 3, 5) LDPC code.

where **m** is the k-by-1 message vector, and **b** is the $(n\text{-}k)$-by-1 parity vector; see Equation (10.9). Correspondingly, the parity-check matrix A is partitioned as

$$\mathbf{A}^T = \begin{bmatrix} \mathbf{A}_1 \\ \cdots\cdots \\ \mathbf{A}_2 \end{bmatrix} \tag{10.89}$$

where $\mathbf{A}_1$ is a square matrix of dimensions $(n\text{-}k) \times (n\text{-}k)$, and $\mathbf{A}_2$ is a rectangular matrix of dimensions $k \times (n\text{-}k)$; transposition symbolized by the superscript T is used in the partitioning of matrix **A** for convenience of presentation. Imposing the constraint of Equation (10.16) on the LDPC code, we may write

$$[\mathbf{b} \vdots \mathbf{m}]\begin{bmatrix} \mathbf{A}_1 \\ \cdots\cdots \\ \mathbf{A}_2 \end{bmatrix} = \mathbf{0}$$

or, equivalently,

$$\mathbf{b}\mathbf{A}_1 + \mathbf{m}\mathbf{A}_2 = \mathbf{0} \tag{10.90}$$

Recall from Equation (10.7) that the vectors **m** and **b** are related by

$$\mathbf{b} = \mathbf{m}\mathbf{P}$$

where **P** is the coefficient matrix. Hence, substituting this relation into Equation (10.90), we readily find that, for any nonzero message vector **m**, the coefficient matrix of LDPC codes satisfies the condition

$$\mathbf{P}\mathbf{A}_1 + \mathbf{A}_2 = \mathbf{0}$$

which holds for *all* nonzero message vectors and, in particular, for **m** in the form $[0 \cdots 0\ 1\ 0 \cdots 0]$ that will isolate individual rows of the generator matrix.

Solving this equation for matrix **P**, we get

$$\mathbf{P} = \mathbf{A}_2\mathbf{A}_1^{-1} \tag{10.91}$$

where $\mathbf{A}_1^{-1}$ is the *inverse* of matrix $\mathbf{A}_1$, which is naturally defined in modulo-2 arithmetic. Finally, the generator matrix of LDPC codes is defined by

$$\begin{aligned} \mathbf{G} &= [\mathbf{P} \vdots \mathbf{I}_k] \\ &= [\mathbf{A}_2\mathbf{A}_1^{-1} \vdots \mathbf{I}_k] \end{aligned} \tag{10.92}$$

where $\mathbf{I}_k$ is the k-by-k identity matrix; see Equation (10.12).

It is important to note that if we take the parity-check matrix **A** for some arbitrary LDPC code and just pick $(n - k)$ columns of **A** at random to form a square matrix $\mathbf{A}_1$, there is *no* guarantee that $\mathbf{A}_1$ will be *nonsingular* (i.e., the inverse $\mathbf{A}_1^{-1}$ will exist), even if the rows of **A** are linearly independent. In fact, for a typical LDPC code with large block length n, such a randomly selected $\mathbf{A}_1$ is highly unlikely to be nonsingular, because it is very likely that at least one row of $\mathbf{A}_1$ will be all 0s. Of course, when the rows of **A** are linearly independent, there will be *some* set of $(n - k)$ columns of **A** that will make a nonsingular $\mathbf{A}_1$, as illustrated in Example 10.9. For some construction methods for LDPC codes the first $(n - k)$ columns of **A** may be guaranteed to produce a nonsingular $\mathbf{A}_1$, or at least do so with a high probability, but that is *not* true in general.

▶ Example 10.9 (10, 3, 5) LDPC Code

Consider the bipartite graph of Figure 10.31 pertaining to a (10, 3, 5) LDPC code. The parity-check matrix of the code is defined by

$$\mathbf{A} = \left[\begin{array}{cccccc:cccc} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \underbrace{0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0}_{\mathbf{A}_1^T} & & & & & & \underbrace{1 \;\; 1 \;\; 1 \;\; 1}_{\mathbf{A}_2^T} \end{array}\right]$$

which appears to be random, while maintaining the regularity constraints: $t_c = 3$ and $t_r = 5$. Partitioning the matrix A in the manner described in Equation (10.89):

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

To derive the inverse of matrix $\mathbf{A}_1$, we first use Equation (10.90) to write

$$\underbrace{[b_0, b_1, b_2, b_3, b_4, b_5]}_{\mathbf{b}} \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}}_{\mathbf{A}_1} = \underbrace{[u_0, u_1, u_2, u_3, u_4, u_5]}_{\mathbf{u} = \mathbf{m}\mathbf{A}_2}$$

where we have introduced the vector **u** to denote the matrix product $\mathbf{m}\mathbf{A}_2$. By using Gaussian elimination, the matrix $\mathbf{A}_1$ is transformed into *lower diagonal form* (i.e., all the elements above the main diagonal are zero), as shown by

$$\mathbf{A}_1 \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

This transformation is achieved by the following modulo-2 additions performed on the columns of square matrix $\mathbf{A}_1$:

- Columns 1 and 2 are added to column 3.
- Column 2 is added to column 4.
- Columns 1 and 4 are added to column 5.
- Columns 1, 2 and 5 are added to column 6.

Correspondingly, the vector $\mathbf{u}$ is transformed as

$$\mathbf{u} \rightarrow [u_0, u_1, u_0 + u_1 + u_2, u_1 + u_3, u_0 + u_3 + u_4, u_0 + u_1 + u_4 + u_5]$$

Accordingly, premultiplying the transformed matrix $\mathbf{A}_1$ by the parity vector $\mathbf{b}$, using successive eliminations in modulo-2 arithmetic working backwards, and putting the solutions for the elements of the parity vector $\mathbf{b}$ in terms of the elements of the vector $\mathbf{u}$ in matrix form, we get

$$\underbrace{[u_0, u_1, u_2, u_3, u_4, u_5]}_{\mathbf{u}} \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}}_{\mathbf{A}_1^{-1}} = \underbrace{[b_0, b_1, b_2, b_3, b_4, b_5]}_{\mathbf{b}}$$

The inverse of matrix $\mathbf{A}_1$ is therefore

$$\mathbf{A}_1^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The matrix product $\mathbf{A}_2\mathbf{A}_1^{-1}$ is (using the given value of $\mathbf{A}_2$ and the value of $\mathbf{A}_1^{-1}$ just found)

$$\mathbf{A}_2\mathbf{A}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Finally, using Equation (10.92), the generator of the (10, 3, 5) LDPC code is

$$\mathbf{G} = \left[\begin{array}{cccccc:cccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ \underbrace{0 \;\; 1 \;\; 0 \;\; 1 \;\; 1}_{\mathbf{A}_2\mathbf{A}_1^{-1}} & & & & & 0 & \underbrace{0 \;\; 0 \;\; 0 \;\; 1}_{\mathbf{I}_k} & & & \end{array}\right]$$

It is important to recognize that the LDPC code described in this example is intended only for the purpose of illustrating the procedure involved in the generation of such a code. In practice, the block length n is orders of magnitude larger than that considered in this example. Moreover, in constructing the matrix $\mathbf{A}$, we may constrain all pairs of columns to have a *matrix overlap* (i.e., inner product of any two columns in matrix **A**) not to exceed 1; such a constraint, over and above the regularity constraints, is expected to improve the performance of LDPC codes. Unfortunately, with a small block length as that considered in this example, it is difficult to satisfy this additional requirement. ◀

MINIMUM DISTANCE OF LDPC CODES

In practice, the block length of a LDPC code is large, ranging from 10^3 to 10^6, which means that the number of code words in a particular code is correspondingly large. Consequently, the algebraic analysis of LDPC codes is rather difficult. It is much more productive to perform a *statistical analysis* on an ensemble of LDPC codes. Such an analysis permits us to make statistical statements about certain properties of member codes in the ensemble. Moreover, an LDPC code with these properties can be found with high probability by a random selection from the ensemble.

Among these properties, the minimum distance of the member codes is of particular interest. From Section 10.3 we recall that the minimum distance of a linear block code is, by definition, the smallest Hamming distance between any pair of code vectors in the code. Over an ensemble of LDPC codes, the minimum distance of a member code is naturally a random variable. Elsewhere[20] it is shown that as the block length n increases, for fixed $t_c \geq 3$ and $t_r > t_c$ the probability distribution of the minimum distance can be overbounded by a function that approaches a unit step function at a fixed fraction $\Delta_{t_ct_r}$ of the block length n. Thus, for large n, practically all the LDPC codes in the ensemble have a minimum distance of at least $n\,\Delta_{t_ct_r}$. Table 10.10 presents the rate r and $\Delta_{t_ct_r}$ of LDPC codes for different values of the weight-pair (t_c, t_r). From this table we see that for $t_c = 3$ and $t_r = 6$ the code rate r attains its highest value of 1/2 and the fraction $\Delta_{t_ct_r}$ attains its smallest value, hence the preferred choice of $t_c = 3$ and $t_r = 6$ in the design of LDPC codes.

PROBABILISTIC DECODING OF LDPC CODES

At the transmitter, a message vector **m** is encoded into a code vector **c** = **mG**, where **G** is the generator matrix for a specified weight-pair (t_c, t_r) and therefore minimum distance $d_{\min}$. The vector **c** is transmitted over a noisy channel to produce the received vector

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

where **e** is the error vector due to channel noise; see Equation (10.17). By construction, the matrix **A** is a parity matrix of the LDPC code; that is, $\mathbf{AG}^T = \mathbf{0}$. Given the received

TABLE 10.10[a] ***The rate r and fractional term $\Delta_{t_ct_r}$ of LDPC codes for varying weights t_c and t_r***

t_c	t_r	*Rate* r	$\Delta_{t_ct_r}$
5	6	0.167	0.255
4	5	0.2	0.210
3	4	0.25	0.122
4	6	0.333	0.129
3	5	0.4	0.044
3	6	0.5	0.023

[a]Adapted from Gallager (1962) with permission of the IEEE.

vector $\mathbf{r}$, the bit-by-bit decoding problem is to find the most probable vector $\hat{\mathbf{c}}$ that satisfies the condition $\hat{\mathbf{c}}\mathbf{A}^T = \mathbf{0}$.

In what follows, a bit refers to an element of the received vector $\mathbf{r}$, and a check refers to a row of matrix $\mathbf{A}$. Let $\mathcal{J}(i)$ denote the set of bits that participate in check i. Let $\mathcal{I}(j)$ denote the set of checks in which bit j participates. A set $\mathcal{J}(i)$ that excludes bit j is denoted by $\mathcal{J}(i)\backslash j$. Likewise, a set $\mathcal{I}(j)$ that excludes check i is denoted by $\mathcal{I}(j)\backslash i$.

The decoding algorithm has two alternating steps: horizontal step and vertical step, which run along the rows and columns of matrix $\mathbf{A}$, respectively. In the course of these steps, two probabilistic quantities associated with nonzero elements of matrix $\mathbf{A}$ are alternately updated. One quantity, denoted by P_{ij}^x, defines the probability that bit j is symbol x (i.e., symbol 0 or 1), given the information derived via checks performed in the horizontal step, except for check i. The second quantity, denoted by Q_{ij}^x, defines the probability that check i is satisfied, given that bit j is fixed at the value x and the other bits have the probabilities $P_{ij'} : j' \in \mathcal{J}(i)\backslash j$.

The LDPC decoding algorithm then proceeds as follows:[21]

Initialization

The variables P_{ij}^0 and P_{ij}^1 are set equal to the *a priori* probabilities p_j^0 and p_j^1 of symbols 0 and 1, respectively, with $p_j^0 + p_j^1 = 1$.

Horizontal Step

In the horizontal step of the algorithm, we run through the checks i. Define

$$\Delta P_{ij} = P_{ij}^0 - P_{ij}^1$$

For each weight-pair (i, j), compute

$$\Delta Q_{ij} = \prod_{j' \in \mathcal{J}(i)\backslash j} \Delta P_{ij'}$$

Hence, set

$$Q_{ij}^0 = \frac{1}{2}(1 + \Delta Q_{ij})$$

$$Q_{ij}^1 = \frac{1}{2}(1 - \Delta Q_{ij})$$

Vertical Step

In the vertical step of the algorithm, the values of the probabilities P_{ij}^0 and P_{ij}^1 are updated using the quantities computed in the horizontal step. In particular, for each bit j, compute

$$P_{ij}^0 = \alpha_{ij} p_j^0 \prod_{i' \in \mathcal{I}(j)\backslash i} Q_{i'j}^0$$

$$P_{ij}^1 = \alpha_{ij} p_j^1 \prod_{i' \in \mathcal{I}(j)\backslash i} Q_{i'j}^1$$

where the scaling factor α_{ij} is chosen to make

$$P_{ij}^0 + P_{ij}^1 = 1$$

In the vertical step, we may also update the *pseudo-posterior probabilities*:

$$P_j^0 = \alpha_j p_j^0 \prod_{i \in \mathcal{I}(j)} Q_{ij}^0$$

$$P_j^1 = \alpha_j p_j^1 \prod_{i \in \mathcal{I}(j)} Q_{ij}^1$$

where α_j is chosen to make

$$P_j^0 + P_j^1 = 1$$

The quantities obtained in the vertical step are used to compute a tentative estimate $\hat{\mathbf{c}}$. If the condition $\hat{\mathbf{c}}\mathbf{A}^T = \mathbf{0}$ is satisfied, the decoding algorithm is terminated. Otherwise, the algorithm goes back to the horizontal step. If after some maximum number of iterations (e.g., 100 or 200) there is no valid decoding, a decoding failure is declared. The decoding procedure described herein is a special case of the general low-complexity *sum-product algorithm.*

Simply stated, the sum-product algorithm passes probabilistic quantities between the check nodes and variable nodes of the bipartite graph. By virtue of the fact that each parity-check constraint can be represented by a simple convolutional coder with one bit of memory, we find that LDPC decoders are simpler to implement than turbo decoders, as stated earlier.

In terms of performance, however, we may say the following in light of experimental results reported in the literature: Regular LDPC codes do not appear to come as close to Shannon's limit as do their turbo code counterparts.

10.11 *Irregular Codes*

The turbo codes discussed in Section 10.8 and the LDPC codes discussed in Section 10.10 are both regular codes, each in its own individual way. The error-correcting performance of both of these codes over a noisy channel can be improved substantially by using their respective irregular forms.

In a standard turbo code with its encoder as shown in Figure 10.25, the interleaver maps each systematic bit to a unique input bit of convolutional encoder 2. In contrast, *irregular turbo codes*[22] use a special design of interleaver that maps some systematic bits to multiple input bits of the convolutional encoder. For example, each of 10 percent of the systematic bits may be mapped to eight inputs of the convolutional encoder instead of

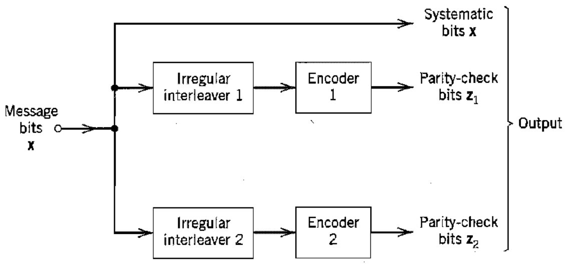


FIGURE 10.32 Block diagram of irregular turbo encoder.

a single one. As shown in Figure 10.32, similar *irregular interleavers* are used in both convolutional encoding paths to generate the parity-check bits $\mathbf{z}_1$ and $\mathbf{z}_2$ in response to the message bits $\mathbf{x}$. Irregular turbo codes are decoded in a similar fashion to regular turbo codes.

To construct an *irregular LDPC code*,[23] the degrees of the variable and check nodes in the bipartite graph are chosen according to some distribution. For example, we may have an irregular LDPC code with the following graphical representation:

- One half of the variable nodes have degree 5 and the other half of the variable nodes have degree 3.
- One half of the check nodes have degree 6 and the other half of the check nodes have degree 8.

For a given block length and a given degree sequence, we define an ensemble of codes by choosing the edges (i.e., the connections between the variable and check nodes) in a random fashion. Specifically, the edges emanating from the variable nodes are enumerated in some arbitrary order, and likewise for the edges emanating from the check nodes.

Figure 10.33 plots the error performances of the following codes:[24]

- Irregular LDPC code: $k = 50{,}000$, $n = 100{,}000$, rate $= 1/2$
- Turbo code (regular): $k = 65{,}536$, $n = 131{,}072$, and rate $= 1/2$
- Irregular turbo code: $k = 65{,}536$, $n = 131{,}072$, and rate $= 1/2$

where k is the number of message bits and n is the block length. The generator polynomials for the two convolutional encoders in the regular/irregular turbo codes are as follows:

$$\text{Encoder 1: } g(D) = 1 + D^4$$
$$\text{Encoder 2: } g(D) = 1 + D + D^2 + D^3 + D^4$$

Figure 10.33 also includes the corresponding theoretical limit on channel capacity for code rate $r = 1/2$.

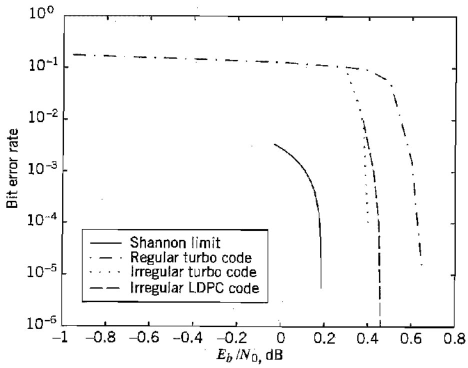


FIGURE 10.33 Noise performances of regular turbo code, irregular turbo code and irregular low-density parity-check (LDPC) code, compared to the Shannon limit for code rate $r = 1/2$.

Based on the results presented in Figure 10.33, we may make the following observations:

- The irregular LDPC code outperforms the regular turbo code in that it comes closer to Shannon's theoretical limit by 0.175 dB.
- Among the three codes displayed therein, the irregular turbo code is the best in that it is just 0.213 dB away from Shannon's theoretical limit.

10.12 *Summary and Discussion*

In this chapter, we studied error-control coding techniques that have established themselves as indispensable tools for reliable digital communication over noisy channels. The effect of errors occurring during transmission is reduced by adding redundancy to the data prior to transmission in a controlled manner. The redundancy is used to enable a decoder in the receiver to detect and correct errors.

Error-control coding techniques may be divided into two broadly defined families:

1. *Algebraic codes*, which rely on abstract algebraic structure built into the design of the codes for decoding at the receiver. Algebraic codes include Hamming codes, maximal-length codes, BCH codes, and Reed-Solomon codes. These particular codes share two properties:

   Linearity property, the sum of any two code words in the code is also a code word.

   Cyclic property, any cyclic shift of a code word is also a code word in the code.

   Reed-Solomon codes are very powerful codes, capable of combatting both random and burst errors; they find applications in difficult environments such as deep-space communications and compact discs.
2. *Probabilistic codes*, which rely on probabilistic methods for their decoding at the receiver. Probabilistic codes include trellis codes, turbo codes, and low-density parity-check codes. In particular, the decoding is based on one or the other of two basic methods, as summarized here:

   Soft input–hard output, which is exemplified by the Viterbi algorithm that performs maximum likelihood sequence estimation in the decoding of trellis codes.

   Soft input–soft output, which is exemplified by the BCJR algorithm that performs maximum *a posteriori* estimation on a bit-by-bit basis in the decoding of turbo codes, or a special form of the sum-product algorithm in the decoding of low-density parity-check codes.

Trellis codes combine linear convolutional encoding and modulation to permit significant coding gains over conventional uncoded multilevel modulation without sacrificing bandwidth efficiency. Turbo codes and low-density parity-check codes share the following properties:

- Random encoding of a linear block kind.
- Error performance within a hair's breadth of Shannon's theoretical limit on channel capacity in a physically realizable fashion.

In practical terms, turbo codes and low-density parity-check codes have made it possible to achieve coding gains on the order of 10 dB, which is unmatched previously. These coding gains may be exploited to dramatically extend the range of digital communication receivers, substantially increase the bit rates of digital communication systems, or significantly decrease the transmitted signal energy per symbol. These benefits have significant implications for the design of wireless communications and deep-space communications, just to mention two important applications of digital communications. Indeed, turbo codes have already been standardized for use on deep-space communication links and wireless communication systems.

NOTES AND REFERENCES

1. For an introductory discussion of error correction by coding, see Chapter 2 of Lucky (1989); see also the book by Adámek (1991), and the paper by Bhargava (1983). The classic book on error-control coding is Peterson and Weldon (1972). Error-control coding is also discussed in the classic book of Gallager (1968). The books of Lin and Costello (1983), Micheleson and Levesque (1985), MacWilliams and Sloane (1977), and Wilson (1998) are also devoted to error-control coding. For a collection of key papers on the development of coding theory, see the book edited by Berlekamp (1974).
2. For a survey of various ARQ schemes, see Lin, Costello, and Miller (1984).
3. In medicine, the term *syndrome* is used to describe a pattern of symptoms that aids in the diagnosis of a disease. In coding, the error pattern plays the role of the disease and parity-check failure that of a symptom. This use of *syndrome* was coined by Hagelbarger (1959).
4. The first error-correcting codes (known as Hamming codes) were invented by Hamming at about the same time as the conception of information theory by Shannon; for details, see the classic paper by Hamming (1950).
5. For a description of BCH codes and their decoding algorithms, see Lin and Costello (1983, pp. 141–183) and MacWilliams and Sloane (1977, pp. 257–293). Table 10.6 on binary BCH codes is adapted from Lin and Costello (1983).
6. The Reed-Solomon codes are named in honor of their inventors: see their classic 1960 paper. For details of Reed-Solomon codes, see MacWilliams and Sloane (1977, pp. 294–306). The book edited by Wicker and Bhargava (1994) contains an introduction to Reed-Solomon codes, a historical overview of these codes written by their inventors, Irving S. Reed and Gustave Solomon, and the applications of Reed-Solomon codes to the exploration of the solar system, the compact disc, automatic repeat-request protocols, and spread-spectrum multiple-access communications, and chapters on other related issues.
7. Convolutional codes were first introduced, as an alternative to block codes, by P. Elias (1955).
8. The term *trellis* was introduced by Forney (1973).
9. In a classic paper, Viterbi (1967) proposed a decoding algorithm for convolutional codes that has become known as the *Viterbi algorithm*. The algorithm was recognized by Forney (1972, 1973) to be a maximum likelihood decoder. Readable accounts of the Viterbi algorithm are presented in Lin and Costello (1983), Blahut (1990), and Adámek (1991).
10. Catastrophic convolutional codes are discussed in Benedetto, Biglieri, and Castellani (1987). Table 10.8 is adapted from their book.
11. For details of the evaluation of asymptotic coding gain for binary symmetric and binary-input AWGN channels, see Viterbi and Omura (1979, pp. 242–252) and Lin and Costello (1983, pp. 322–329).
12. Trellis-coded modulation was invented by G. Ungerboeck; its historical evolution is described in Ungerboeck (1982). Table 10.9 is adapted from this latter paper.

    Trellis-coded modulation may be viewed as a form of *signal-space coding*—a viewpoint discussed at an introductory level in Chapter 14 of the book by Lee and Messerschmitt (1994). For an extensive treatment of trellis-coded modulation, see the books by Biglieri, Divsalar, McLane, and Simon (1991), and Schlegel (1997).

13. Turbo codes were originated by C. Berrou and A. Glavieux. Work on these codes was motivated by two papers on error-correcting codes: Battail (1987), and Hagenauer and Hoecher (1989). The first description of turbo codes using heuristic arguments was presented at a conference paper by Berrou, Glavieux, and Thitimajshima (1993); see also Berrou and Glavieux (1996). For reflections on the early work on turbo codes and subsequent developments, see Berrou and Glavieux (1998).

    For a book on the basics of turbo codes, see Heegard and Wicker (1999). Using a procedure reminiscent of random coding (see Note 15), Benedetto and Montorosi (1996) have provided partial explanations for the impressive performance of turbo codes.

    In two independent studies reported in the papers by McEliece, MacKay, and Cheng (1998), and Kschischang and Frey (1998), it is shown that turbo decoding duplicates an algorithm in artificial intelligence due to Pearl (1982), which involves the propagation of belief. The term *belief* is another way of referring to *a posteriori* probability. These two papers have opened a new avenue of research, which links turbo decoding and learning machines. For an insightful discussion of turbo codes, see the book by Frey (1998).

    A pseudo-random interleaver is basic to the operation of turbo codes. Denenshgaran and Mondin (1999) present a systematic procedure for designing interleavers (i.e., permuters) for turbo codes.

14. The dual termination of turbo codes is discussed in Guinand and Lodge (1996).

15. Random coding is discussed in Cover and Thomas (1991), Section 8.7.

16. The plots presented in Fig. 10.27 follow those in Fig. 6.8 of the book by Frey (1998).

17. In the early 1960s, Baum and Welch derived an iterative procedure for solving the parameter estimation problem, hence the name *Baum-Welch algorithm* (Baum and Petrie (1966); Baum et al. (1970)). In the *BCJR algorithm*, named after Bahl, Cocke, Jelinek, and Raviv (1974), the Baum-Welch algorithm is applied to the problem of soft output, maximum likelihood decoding of convolutional codes.

18. The proof that the estimate $\hat{\mathbf{x}}$ in Eq. (10.76) approaches the MAP solution as the number of iterations approaches infinity is discussed in the paper by Moher and Gulliver (1998).

19. Low-density parity-check (LDPC) codes were originally discovered by Gallager (1962, 1963). They were rediscovered independently by MacKay and Neal (1995); see also MacKay (1999).

    In the 1960s and for a good while thereafter, the computers available at that time were not powerful enough to process the long block lengths that are needed to achieve excellent performance with LDPC codes, hence the lack of interest in their use for over twenty years.

20. For a detailed treatment of the statement that the probability distribution of the minimum distance of an LDPC code approaches a unit step function of the block length for certain values of weight-pair (t_c, t_r), see Gallager (1962, 1963).

21. The decoding algorithm of LDPC codes described herein follows MacKay and Neal (1996, 1997).

22. Irregular turbo codes were invented by Frey and MacKay (1999).

23. Irregular LDPC codes were invented independently by MaKay et al. (1999) and Richardson et al. (1999).

24. The codes, whose performances are plotted in Fig. 10.34, are due to the following originators:
    - Regular turbo codes: Berrou and Glavieux (1996); Berrou et al. (1995).
    - Irregular turbo codes: Frey and MacKay (1999).
    - Irregular LDPC codes: Richardson et al. (1999).

PROBLEMS

Soft-Decision Coding

10.1 Consider a binary input Q-ary output discrete memoryless channel. The channel is said to be symmetric if the channel transition probability $p(j|i)$ satisfies the condition:

$$p(j|0) = p(Q - 1 - j|1), \qquad j = 0, 1, \ldots, Q - 1$$

Suppose that the channel input symbols 0 and 1 are equally likely. Show that the channel output symbols are also equally likely; that is,

$$p(j) = \frac{1}{Q}, \qquad j = 0, 1, \ldots, Q - 1$$

10.2 Consider the quantized demodulator for binary PSK signals shown in Fig. 10.3*a*. The quantizer is a four-level quantizer, normalized as in Fig. P10.2. Evaluate the transition probabilities of the binary input-quarternary output discrete memoryless channel so characterized. Hence, show that it is a symmetric channel. Assume that the transmitted signal energy per bit is E_b, and the additive white Gaussian noise has zero mean and power spectral density $N_0/2$.

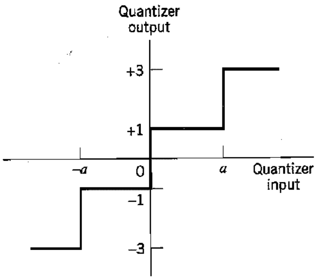


FIGURE P10.2

10.3 Consider a binary input AWGN channel, in which the binary symbols 1 and 0 are equally likely. The binary symbols are transmitted over the channel by means of phase-shift keying. The code symbol energy is E, and the AWGN has zero mean and power spectral density $N_0/2$. Show that the channel transition probability is given by

$$p(y|0) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E}{N_0}}\right)^2\right], \qquad -\infty < y < \infty$$

Linear Block and Cyclic Codes

10.4 In a *single-parity-check code*, a single parity bit is appended to a block of k message bits $(m_1, m_2, \ldots, m_k)$. The single parity bit b_1 is chosen so that the code word satisfies the *even parity rule*:

$$m_1 + m_2 + \cdots + m_k + b_1 = 0, \qquad \text{mod } 2$$

For $k = 3$, set up the 2^k possible code words in the code defined by this rule.

10.5 Compare the parity-check matrix of the (7, 4) Hamming code considered in Example 10.2 with that of a (4, 1) repetition code.

10.6 Consider the (7, 4) Hamming code of Example 10.2. The generator matrix G and the parity-check matrix H of the code are described in that example. Show that these two matrices satisfy the condition

$$\mathbf{H}\mathbf{G}^T = 0$$

10.7 (a) For the (7, 4) Hamming code described in Example 10.2, construct the eight code words in the dual code.

(b) Find the minimum distance of the dual code determined in part (a).

10.8 Consider the (5, 1) repetition code of Example 10.1. Evaluate the syndrome s for the following error patterns:

(a) All five possible single-error patterns

(b) All 10 possible double-error patterns

10.9 For an application that requires error detection *only*, we may use a *nonsystematic* code. In this problem, we explore the generation of such a cyclic code. Let $g(X)$ denote the generator polynomial, and $m(X)$ denote the message polynomial. We define the code polynomial $c(X)$ simply as

$$c(X) = m(X)g(X)$$

Hence, for a given generator polynomial, we may readily determine the code words in the code. To illustrate this procedure, consider the generator polynomial for a (7, 4) Hamming code:

$$g(X) = 1 + X + X^3$$

Determine the 16 code words in the code, and confirm the nonsystematic nature of the code.

10.10 The polynomial $1 + X^7$ has $1 + X + X^3$ and $1 + X^2 + X^3$ as primitive factors. In Example 10.3, we used $1 + X + X^3$ as the generator polynomial for a (7, 4) Hamming code. In this problem, we consider the adoption of $1 + X^2 + X^3$ as the generator polynomial. This should lead to a (7, 4) Hamming code that is different from the code analyzed in Example 10.3. Develop the encoder and syndrome calculator for the generator polynomial:

$$g(X) = 1 + X^2 + X^3$$

Compare your results with those in Example 10.3.

10.11 Consider the (7, 4) Hamming code defined by the generator polynomial

$$g(X) = 1 + X + X^3$$

The code word 0111001 is sent over a noisy channel, producing the received word 0101001 that has a single error. Determine the syndrome polynomial $s(X)$ for this received word, and show that it is identical to the error polynomial $e(X)$.

10.12 The generator polynomial of a (15, 11) Hamming code is defined by

$$g(X) = 1 + X + X^4$$

Develop the encoder and syndrome calculator for this code, using a systematic form for the code.

10.13 Consider the (15, 4) maximal-length code that is the dual of the (15, 11) Hamming code of Problem 10.12. Do the following:

(a) Find the feedback connections of the encoder, and compare your results with those of Table 7.1 on maximal-length codes presented in Chapter 7.

(b) Find the generator polynomial $g(X)$; hence, determine the output sequence assuming the initial state 0001. Confirm the validity of your result by cycling the initial state through the encoder.

10.14 Consider the (31, 15) Reed-Solomon code.
(a) How many bits are there in a symbol of the code?
(b) What is the block length in bits?
(c) What is the minimum distance of the code?
(d) How many symbols in error can the code correct?

Convolutional Codes

10.15 A convolutional encoder has a single-shift register with two stages, (i.e., constraint length $K = 3$), three modulo-2 adders, and an output multiplexer. The generator sequences of the encoder are as follows:

$$g^{(1)} = (1, 0, 1)$$
$$g^{(2)} = (1, 1, 0)$$
$$g^{(3)} = (1, 1, 1)$$

Draw the block diagram of the encoder.

Note: For Problems 10.16–10.23, *the same message sequence* 10111 . . . *is used so that we may compare the outputs of different encoders for the same input.*

10.16 Consider the rate $r = 1/2$, constraint length $K = 2$ convolutional encoder of Fig. P10.16. The code is systematic. Find the encoder output produced by the message sequence 10111. . . .

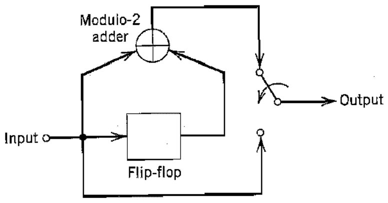


FIGURE P10.16

10.17 Figure P10.17 shows the encoder for a rate $r = 1/2$, constraint length $K = 4$ convolutional code. Determine the encoder output produced by the message sequence 10111. . . .

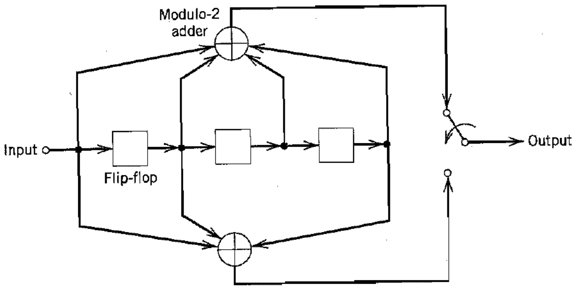


FIGURE P10.17

10.18 Consider the encoder of Fig. 10.13*b* for a rate $r = 2/3$, constraint length $K = 2$ convolutional code. Determine the code sequence produced by the message sequence 10111. . . .

10.19 Construct the code tree for the convolutional encoder of Fig. P10.16. Trace the path through the tree that corresponds to the message sequence 10111 . . . , and compare the encoder output with that determined in Problem 10.16.

10.20 Construct the code tree for the encoder of Fig. P10.17. Trace the path through the tree that corresponds to the message sequence 10111. . . . Compare the resulting encoder output with that found in Problem 10.17.

10.21 Construct the trellis diagram for the encoder of Fig. P10.17, assuming a message sequence of length 5. Trace the path through the trellis corresponding to the message sequence 10111. . . . Compare the resulting encoder output with that found in Problem 10.17.

10.22 Construct the state diagram for the encoder of Fig. P10.17. Starting with the all-zero state, trace the path that corresponds to the message sequence 10111 . . . , and compare the resulting code sequence with that determined in Problem 10.17.

10.23 Consider the encoder of Fig. 10.13*b*.

(a) Construct the state diagram for this encoder.

(b) Starting from the all-zero state, trace the path that corresponds to the message sequence 10111. . . . Compare the resulting sequence with that determined in Problem 10.18.

10.24 By viewing the minimum shift keying (MSK) scheme as a finite-state machine, construct the trellis diagram for MSK. (A description of MSK is presented in Chapter 6.)

10.25 The trellis diagram of a rate-1/2, constraint length-3 convolutional code is shown in Figure P10.25. The all-zero sequence is transmitted, and the received sequence is 100010000. . . . Using the Viterbi algorithm, compute the decoded sequence.

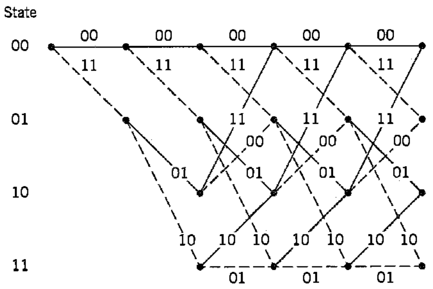


FIGURE P10.25

10.26 Consider a rate-1/2, constraint length-7 convolutional code with free distance $d_{\text{free}} = 10$. Calculate the asymptotic coding gain for the following two channels:

(a) Binary symmetric channel

(b) Binary input AWGN channel

10.27 In Section 10.6 we described the Viterbi algorithm for maximum likelihood decoding of a convolutional code. Another application of the Viterbi algorithm is for maximum likelihood demodulation of a received sequence corrupted by intersymbol interference due to a dispersive channel. Figure P10.27 shows the trellis diagram for intersymbol interference, assuming a binary data sequence. The channel is discrete, described by the finite impulse response (1, 0.1). The received sequence is (1.0, −0.3, −0.7, 0, . . .). Use the Viterbi algorithm to determine the maximum likelihood decoded version of this sequence.

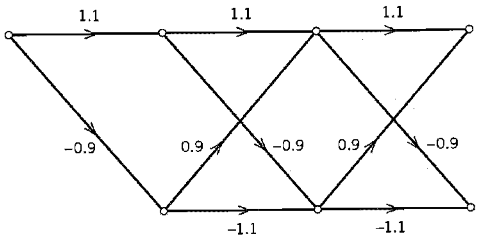


FIGURE P10.27

10.28 Figure P10.28 depicts 32-QAM cross constellation. Partition this constellation into eight subsets. At each stage of the partitioning, indicate the within-subset (shortest) Euclidean distance.

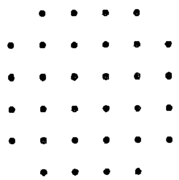


FIGURE P10.28

10.29 As explained in the Introduction to this chapter, channel coding can be used to reduce the E_b/N_0 required for a prescribed error performance or reduce the size of the receiving antenna for a prescribed E_b/N_0. In this problem we explore these two practical benefits of coding by revisiting Example 8.2 in Chapter 8 on the downlink power calculations for a domestic satellite communication system. In particular, we now assume that the design of the downlink includes the use of a coding scheme consisting of a rate-1/2 convolutional encoder with length $K = 7$ and Viterbi decoding. The coding gain of this scheme is 5.1 dB, assuming the use of soft quantization. Hence do the following:

(a) Recalculate the required E_b/N_0 ratio of the system.

(b) Assuming that the required E_b/N_0 ratio remains unchanged, calculate the reduction in the size of the receiving dish antenna that is made possible by the use of this coding scheme in the downlink.

10.30 Unlike the convolutional codes considered in this chapter, we recall from Chapter 6 that the convolutional code used in the voiceband modem V.32 modem is *nonlinear*. Figure P10.30 shows the circuit diagram of the convolutional encoder used in this modem; it uses modulo-2 multiplication and gates in addition to modulo-2 additions and delays. Explain the reason for nonlinearity of the encoder in Fig. P10.30, and use an example to illustrate your explanation.

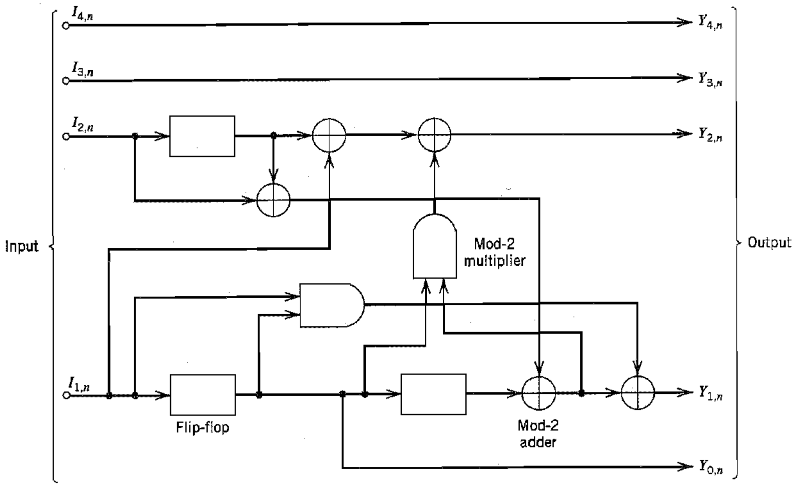


FIGURE P10.30

Turbo Codes

10.31 Let $r_c^{(1)} = p/q_1$ and $r_c^{(2)} = p/q_2$ be the code rates of RSC encoders 1 and 2 in the turbo encoder of Fig. 10.25. Find the code rate of the turbo code.

10.32 The feedback nature of the constituent codes in the turbo encoder of Fig. 10.25 has the following implication: A single bit error corresponds to an infinite sequence of channel errors. Illustrate this phenomenon by using a message sequence consisting of symbol 1 followed by an infinite number of symbols 0.

10.33 Consider the following generator matrices for rate 1/2 turbo codes:

$$\text{4-state encoder:} \quad g(D) = \left[1, \frac{1 + D + D^2}{1 + D^2}\right]$$

$$\text{8-state encoder:} \quad g(D) = \left[1, \frac{1 + D^2 + D^3}{1 + D + D^2 + D^3}\right]$$

$$\text{16-state encoder:} \quad g(D) = \left[1, \frac{1 + D^4}{1 + D + D^2 + D^3 + D^4}\right]$$

(a) Construct the block diagram for each one of these RSC encoders.

(b) Setup the parity-check equation associated with each encoder.

10.34 The turbo encoder of Fig. 10.25 involves the use of two RSC encoders.

(a) Generalize this encoder to encompass a total of M interleavers.

(b) Construct the block diagram of the turbo decoder that exploits the M sets of parity-check bits generated by such a generalization.

10.35 Turbo decoding relies on the feedback of extrinsic information. The fundamental principle adhered to in the turbo decoder is to avoid feeding a decoding stage information that stems from the stage itself. Explain the justification for this principle in conceptual terms.

10.36 Suppose a communication receiver consists of two components, a demodulator and a decoder. The demodulator is based on a Markov model of the combined modulator and channel, and the decoder is based on a Markov model of a forward error correction code. Discuss how the turbo principle may be applied to construct a joint demodulator/decoder for this system.

Computer Experiment

10.37 In this experiment we continue the investigation into turbo codes presented in Section 10.9 by evaluating the effect of block size on the noise performance of the decoder.

As before, the two convolutional encoders of the turbo encoder are as follows:

Encoder 1: [1, 1, 1]

Encoder 2: [1, 0, 1]

The transmitted E_b/N_0 is 1 dB. The block errors to termination are prescribed not to exceed 15.

With this background information, plot the bit error rate of the turbo decoder versus the number of iterations for two different block (i.e., interleaver) sizes: 200 and 400.