DIGITAL COMMUNICATIONS

Fundamentals and Applications

BERNARD SKLAR

The Aerospace Corporation, El Segundo, California
and
University of California, Los Angeles

PRENTICE HALL

Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

SKLAR, BERNARD (date)
Digital communications.

Bibliography: p.
Includes index.
1. Digital communications. I. Title.
TK5103.7.S55 1988 621.38'0413 87-1316
ISBN 0-13-211939-0

Editorial/production supervision and
interior design: Reynold Rieger
Cover design: Wanda Lubelska Design
Manufacturing buyers: Gordon Osbourne and Paula Benevento

A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

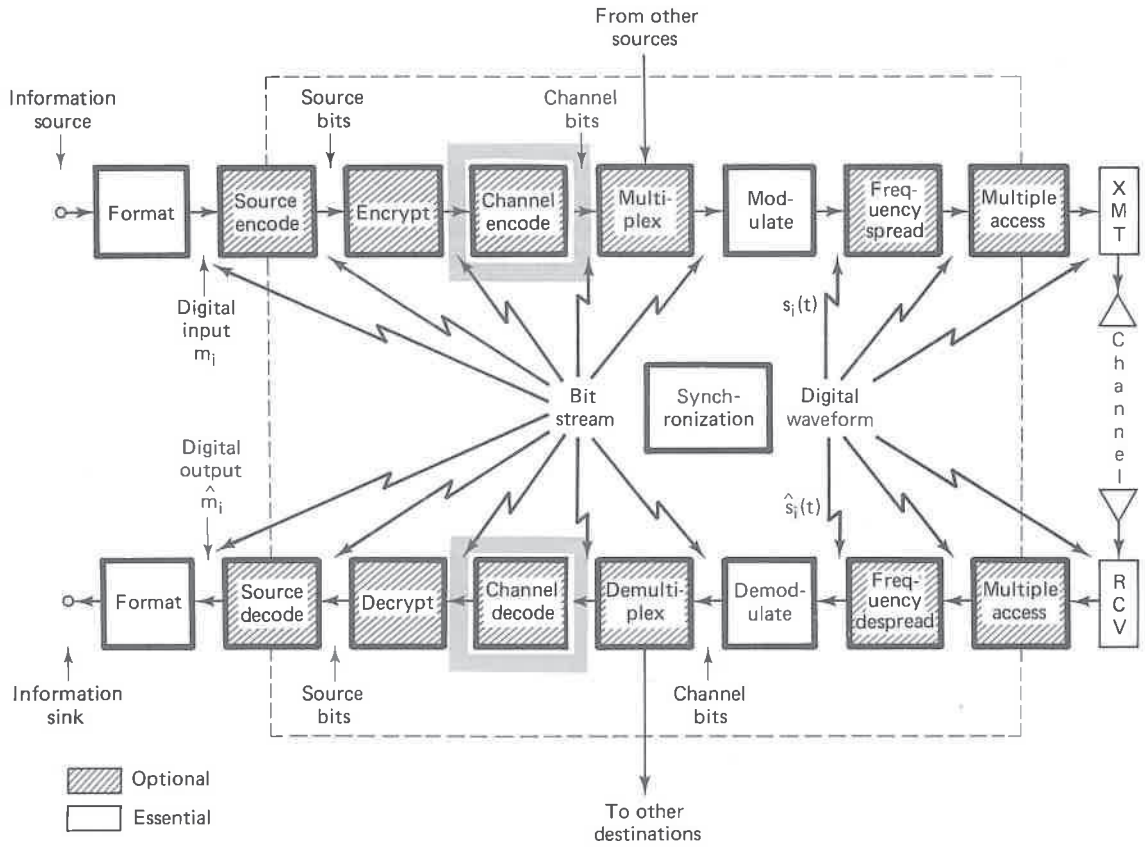
ISBN 0-13-211939-0 025

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Contents

CHAPTER 6

Channel Coding: Part 2



There are two major categories of channel codes: block and convolutional. Chapter 5 deals mainly with block coding. This chapter deals mainly with convolutional coding. A *linear block code* is described by two integers, n and k, and a generator matrix or polynomial. The integer k is the number of data bits that form an input to a block encoder. The integer n is the total number of bits in the associated codeword out of the encoder. A characteristic of linear block codes is that each codeword n-tuple is uniquely determined by the input message k-tuple. The ratio k/n is called the *rate* of the code—a measure of the amount of added redundancy. A *convolutional code* is described by three integers, n, k, and K, where the ratio k/n has the same code rate significance (information per coded bit) that it has for block codes; however, n does *not* define a block or codeword length as it does for block codes. The integer K is a parameter known as the *constraint length*; it represents the number of k-tuple stages in the encoding shift register. An important characteristic of convolutional codes, different from block codes, is that the encoder has memory—the n-tuple emitted by the convolutional encoding procedure is not only a function of an input k-tuple, but is also a function of the previous $K - 1$ input k-tuples. In practice, n and k are small integers and K is varied to control the redundancy.

6.1 CONVOLUTIONAL ENCODING

In Figure 1.2 we presented a typical block diagram of a digital communication system. A version of this functional diagram, focusing primarily on the convolutional encode/decode and modulate/demodulate portions of the communication

link, is shown in Figure 6.1. The input message source is denoted by the sequence $\mathbf{m} = m_1, m_2, \ldots, m_i, \ldots$, where each m_i represents a binary digit (bit). We shall assume that each m_i is equally likely to be a one or a zero, and independent from digit to digit. Being independent, the bit sequence lacks any redundancy; that is, knowledge about bit m_i gives no information about m_j $(i \neq j)$. The encoder transforms each sequence $\mathbf{m}$ into a unique codeword sequence $\mathbf{U} = G(\mathbf{m})$. Even though the sequence $\mathbf{m}$ uniquely defines the sequence $\mathbf{U}$, a key feature of convolutional codes is that a given k-tuple within $\mathbf{m}$ does *not* uniquely define its associated n-tuple within $\mathbf{U}$ since the encoding of each k-tuple is *not only* a function of that k-tuple but is also a function of the $K - 1$ input k-tuples that precede it. The sequence $\mathbf{U}$ can be partitioned into a sequence of branch words: $\mathbf{U} = U_1, U_2, \ldots, U_i, \ldots$. Each branch word U_i is made up of binary *code symbols*, often called *channel symbols*, *channel bits*, or *coded bits*; unlike the input message bits the code symbols are not independent.

In a typical communication application, the codeword sequence $\mathbf{U}$ modulates a waveform $s(t)$. During transmission, the waveform $s(t)$ is corrupted by noise, resulting in a received waveform $\hat{s}(t)$ and a demodulated sequence $\mathbf{Z} = Z_1, Z_2, \ldots, Z_i, \ldots$, as indicated in Figure 6.1. The task of the decoder is to produce an estimate $\hat{\mathbf{m}} = \hat{m}_1, \hat{m}_2, \ldots, \hat{m}_i, \ldots$, of the original message sequence, using the received sequence $\mathbf{Z}$ together with a priori knowledge of the encoding procedure.

A general convolutional encoder, shown in Figure 6.2, is mechanized with a kK-stage shift register and n modulo-2 adders, where K is the constraint length. The constraint length represents the number of k-bit shifts over which a single information bit can influence the encoder output. At each unit of time, k bits are shifted into the first k stages of the register; all bits in the register are shifted k stages to the right, and the outputs of the n adders are sequentially sampled to

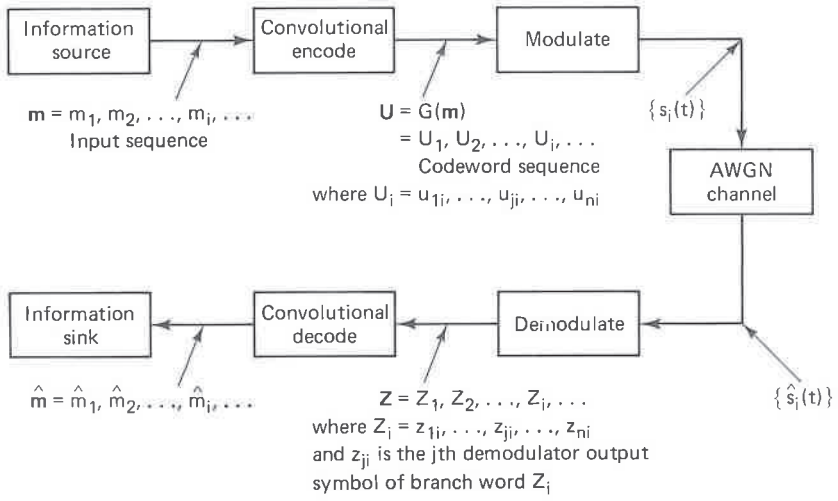


Figure 6.1 Encode/decode and modulate/demodulate portions of a communication link.

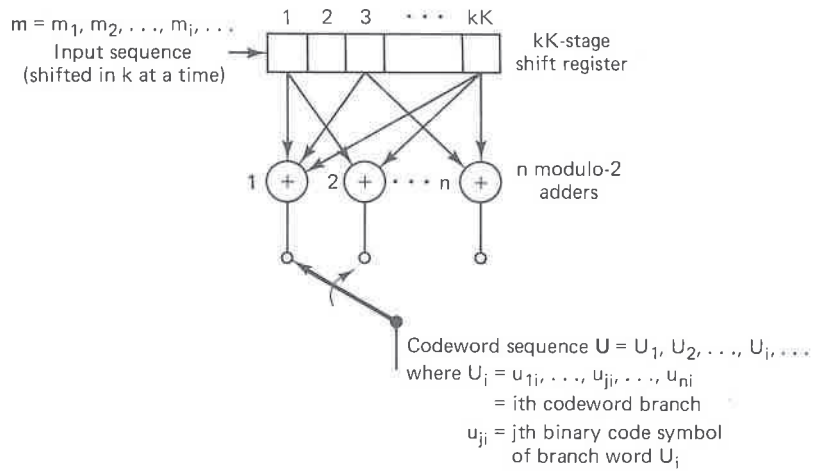


Figure 6.2 Convolutional encoder with constraint length K and rate k/n.

yield the binary code symbols or coded bits. These code symbols are then used by the modulator to specify the waveforms to be transmitted over the channel. Since there are n coded bits for each input group of k message bits, the code rate is k/n message bit per coded bit, where $k < n$.

We shall consider only the most commonly used binary convolutional encoders for which $k = 1$, that is, those encoders in which the message bits are shifted into the encoder one bit at a time, although generalization to higher-order alphabets is straightforward [1, 2]. For the $k = 1$ encoder, at the ith unit of time, message bit m_i is shifted into the first shift register stage; all previous bits in the register are shifted one stage to the right, and as in the more general case, the outputs of the n adders are sequentially sampled and transmitted. Since there are n coded bits for each message bit, the code rate is $1/n$. The n code symbols occurring at time t_i comprise the ith branch word, $U_i = u_{1i}, u_{2i}, \ldots, u_{ni}$, where u_{ji} $(j = 1, 2, \ldots, n)$ is the jth code symbol belonging to the ith branch word. Note that for the rate $1/n$ encoder, the kK-stage shift register can be referred to simply as a K-stage register, and the constraint length K, which was expressed in units of k-tuple stages, can be referred to as constraint length in units of bits.

6.2 CONVOLUTIONAL ENCODER REPRESENTATION

To describe a convolutional code, one needs to characterize the encoding function $G(\mathbf{m})$, so that given an input sequence $\mathbf{m}$, one can readily compute the output sequence $\mathbf{U}$. Several methods are used for representing a convolutional encoder, the most popular being the *connection pictorial, connection vectors or polynomials*, the *state diagram*, the *tree diagram*, and the *trellis diagram*. They are each described below.

6.2.1 Connection Representation

We shall use the convolutional encoder, shown in Figure 6.3, as a model for discussing convolutional encoders. The figure illustrates a (2, 1) convolutional encoder with constraint length $K = 3$. There are $n = 2$ modulo-2 adders; thus the code rate k/n is $\frac{1}{2}$. At each input bit time, a bit is shifted into the leftmost stage and the bits in the register are shifted one position to the right. Next, the output switch samples the output of each modulo-2 adder (i.e., first the upper adder, then the lower adder), thus forming the code symbol pair making up the branch word associated with the bit just inputted. The sampling is repeated for each inputted bit. The choice of connections between the adders and the stages of the register gives rise to the characteristics of the code. Any change in the choice of connections results in a different code. The connections are, of course, *not* chosen or changed arbitrarily. The problem of choosing connections to yield good distance properties is complicated and has not been solved in general; however, good codes have been found by computer search for all constraint lengths less than about 20 [3–5].

Unlike a block code that has a fixed word length n, a convolutional code has no particular block size. However, convolutional codes are often forced into a block structure by *periodic truncation*. This requires a number of zero bits to be appended to the end of the input data sequence, for the purpose of clearing or *flushing* the encoding shift register of the data bits. Since the added zeros carry no information, the *effective code rate* falls below k/n. To keep the code rate close to k/n, the truncation period is generally made as long as practical.

One way to represent the encoder is to specify a set of n *connection vectors*, one for each of the n modulo-2 adders. Each vector has dimension K and describes the connection of the encoding shift register to that modulo-2 adder. A one in the ith position of the vector indicates that the corresponding stage in the shift register is connected to the modulo-2 adder, and a zero in a given position indicates that no connection exists between the stage and the modulo-2 adder. For the encoder example in Figure 6.3, we can write the connection vector $\mathbf{g}_1$ for the upper connections and $\mathbf{g}_2$ for the lower connections as follows:

$$\mathbf{g}_1 = 1\ 1\ 1$$

$$\mathbf{g}_2 = 1\ 0\ 1$$

Consider that a message vector $\mathbf{m} = 1\ 0\ 1$ is convolutionally encoded with the encoder shown in Figure 6.3. The three message bits are inputted, one at a time, at times t_1, t_2, and t_3, as shown in Figure 6.4. Subsequently, $(K - 1) = 2$ zeros are inputted at times t_4 and t_5 to flush the register and thus ensure that the tail end of the message is shifted the full length of the register. The output sequence is seen to be 1 1 1 0 0 0 1 0 1 1, where the leftmost symbol represents the earliest transmission. The entire output sequence, including the code symbols as a result of flushing, are needed to decode the message. To flush the message from the encoder requires one less zero than the number of stages in the register, or $K - 1$ flush bits. Another zero input is shown at time t_6, for the reader to verify that the corresponding branch word output is then 00.

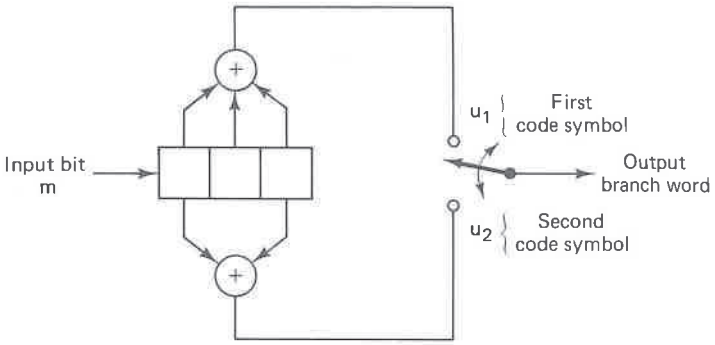


Figure 6.3 Convolutional encoder (rate $\frac{1}{2}$, $K = 3$).

6.2.1.1 Impulse Response of the Encoder

We can approach the encoder in terms of its *impulse response*—that is, the response of the encoder to a single "one" bit that moves through it. Consider the contents of the register in Figure 6.3 as a one moves through it.

Register contents	Branch word	
	u_1	u_2
1 0 0	1	1
0 1 0	1	0
0 0 1	1	1

Input sequence: 1 0 0
Output sequence: 1 1 1 0 1 1

The output sequence for the input "one" is called the impulse response of the encoder. Then for the input sequence $\mathbf{m} = 1\ 0\ 1$, the output may be found by the *superposition* or the *linear addition* of the time-shifted input "impulses" as follows:

Input **m**	Output				
1	1 1	1 0	1 1		
0		0 0	0 0	0 0	
1			1 1	1 0	1 1
Modulo-2 sum:	1 1	1 0	0 0	1 0	1 1

Observe that this is the same output as that obtained in Figure 6.4, demonstrating that *convolutional codes are linear*—just as the linear block codes of Chapter 5. It is from this property of generating the output by the linear addition of time-shifted impulses, or the convolution of the input sequence with the impulse response of the encoder, that we derive the name *convolutional encoder*. Often,

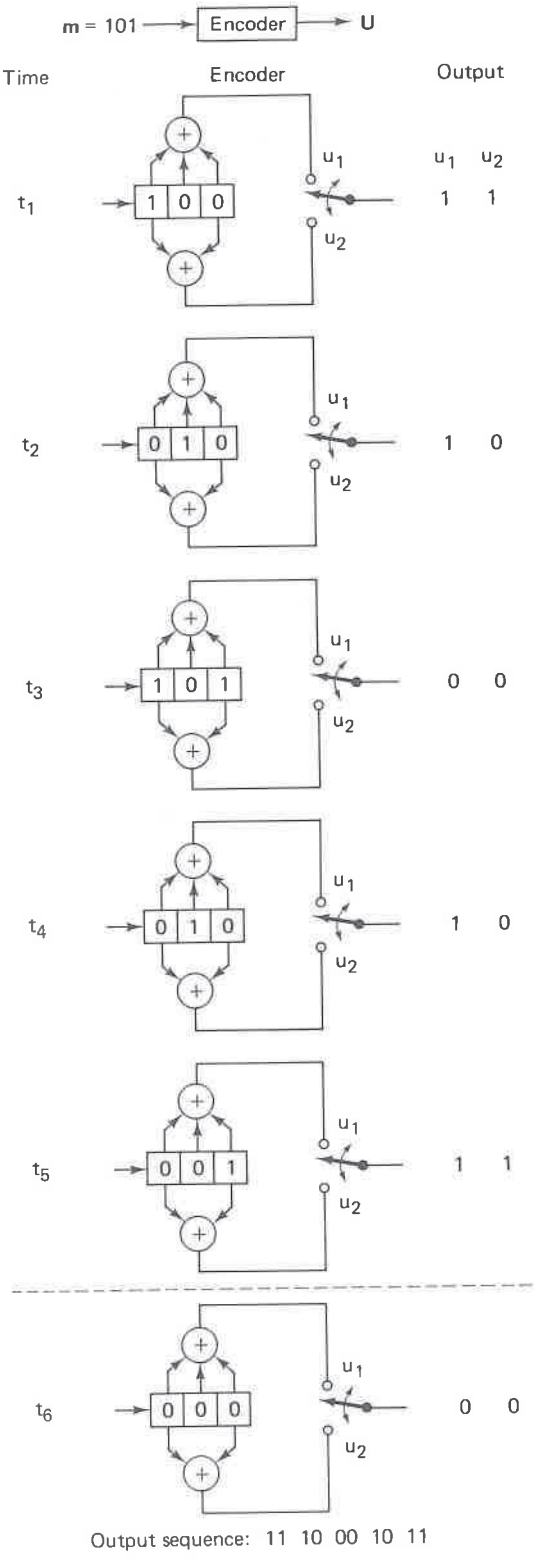


Figure 6.4 Convolutionally encoding a message sequence with a rate $\frac{1}{2}$, $K = 3$ encoder.

this encoder characterization is presented in terms of an infinite-order generator matrix [6].

Notice that the *effective code rate* for the foregoing example with 3-bit input sequence and 10-bit output sequence is $k/n = \frac{3}{10}$—quite a bit less than the rate $\frac{1}{2}$ that might have been expected from the knowledge that each input data bit yields a pair of output channel bits. The reason for the disparity is that the final data bit into the encoder needs to be shifted through the encoder. All of the output channel bits are needed in the decoding process. If the message had been longer, say 300 bits, the output codeword sequence would contain 604 bits, resulting in a code rate of 300/604—much closer to $\frac{1}{2}$.

6.2.1.2 Polynomial Representation

Sometimes, the encoder connections are characterized by *generator polynomials*, similar to those used in Chapter 5 for describing the feedback shift register implementation of cyclic codes. We can represent a convolutional encoder with a set of n generator polynomials, one for each of the n modulo-2 adders. Each polynomial is of degree $K - 1$ or less and describes the connection of the encoding shift register to that modulo-2 adder, much the same way that a connection vector does. The coefficient of each term in the $(K - 1)$-degree polynomial is either 1 or 0, depending on whether a connection exists or does not exist between the shift register and the modulo-2 adder in question. For the encoder example in Figure 6.3, we can write the generator polynomial $\mathbf{g}_1(X)$ for the upper connections and $\mathbf{g}_2(X)$ for the lower connections as follows:

$$\mathbf{g}_1(X) = 1 + X + X^2$$

$$\mathbf{g}_2(X) = 1 + X^2$$

where the lowest-order term in the polynomial corresponds to the input stage of the register. The output sequence is found as follows:

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}_1(X) \text{ interlaced with } \mathbf{m}(X)\mathbf{g}_2(X)$$

First, express the message vector $\mathbf{m} = 1\ 0\ 1$ as a polynomial—that is, $\mathbf{m}(X) = 1 + X^2$. We shall again assume the use of zeros following the message bits, to flush the register. Then the output polynomial, $\mathbf{U}(X)$, or the output sequence, $\mathbf{U}$, of the Figure 6.3 encoder can be found for the input message $\mathbf{m}$ as follows:

$$\mathbf{m}(X)\mathbf{g}_1(X) = (1 + X^2)(1 + X + X^2) = 1 + X + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = (1 + X^2)(1 + X^2) = 1 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = 1 + X + 0X^2 + X^3 + X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = 1 + 0X + 0X^2 + 0X^3 + X^4$$

$$\mathbf{U}(X) = (1, 1) + (1, 0)X + (0, 0)X^2 + (1, 0)X^3 + (1, 1)X^4$$

$$\mathbf{U} = 1\ 1 \quad 1\ 0 \quad 0\ 0 \quad 1\ 0 \quad 1\ 1$$

In this example we started with another point of view—that the convolutional encoder can be treated as a set of *cyclic code shift registers*. We represented the encoder with *polynomial generators* as used for describing cyclic codes. However, we arrived at the same output sequence as in Figure 6.4, and the same output sequence as the impulse response treatment of the preceding section. For a good presentation of convolutional code structure in the context of linear sequential circuits, see Reference [7].

6.2.2 State Representation and the State Diagram

The state of a rate $1/n$ convolutional encoder is defined as the contents of the rightmost $K - 1$ stages (see Figure 6.3). Knowledge of the state together with knowledge of the next input is necessary and sufficient to determine the next output. Let the state of the encoder at time, t_i, be defined as $X_i = m_{i-1}, m_{i-2}, \ldots, m_{i-K+1}$. The ith codeword branch, U_i, is completely determined by state X_i and the present input bit m_i; thus the state X_i represents the past history of the encoder in determining the encoder output. The encoder state is said to be *Markov*, in the sense that the probability, $P(X_{i+1}|X_i, X_{i-1}, \ldots, X_0)$, of being in state X_{i+1}, given all previous states, depends only on the most recent state, X_i; that is, the probability is equal to $P(X_{i+1}|X_i)$.

One way to represent simple encoders is with a *state diagram*; such a representation for the encoder in Figure 6.3 is shown in Figure 6.5. The states, shown in the boxes of the diagram, represent the possible contents of the rightmost $K - 1$ stages of the register, and the paths between the states represent the output branch words resulting from such state transitions. The states of the register are designated $a = 00$, $b = 10$, $c = 01$, and $d = 11$; the diagram shown in Figure 6.5 illustrates all the state transitions that are possible for the encoder in Figure

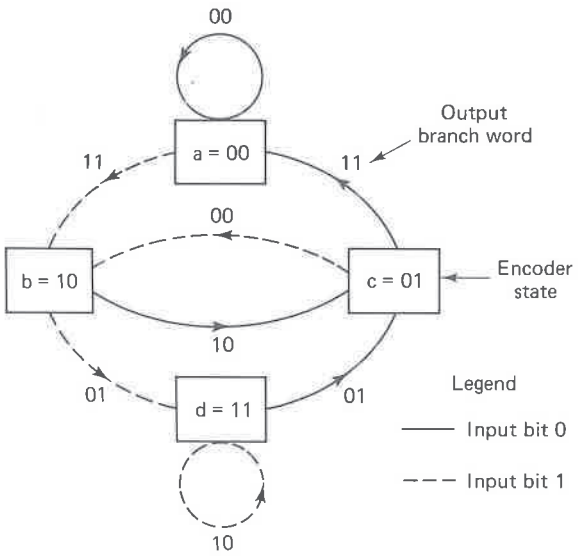


Figure 6.5 Encoder state diagram (rate $\frac{1}{2}$, $K = 3$).

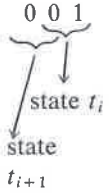
6.3. There are *only two transitions* emanating from each state, corresponding to the two possible input bits. Next to each path between states is written the output branch word associated with the state transition. In drawing the path, we use the convention that a solid line denotes a path associated with an input bit, zero, and a dashed line denotes a path associated with an input bit, one. Notice that it is *not possible* in a single transition to move from a given state to *any arbitrary state*. As a consequence of shifting-in one bit at a time, there are only two possible state transitions that the register can make at each bit time. For example, if the present encoder state is 00, the *only possibilities* for the state at the next shift are 00 or 10.

Example 6.1 Convolutional Encoding

For the encoder shown in Figure 6.3, show the state changes and the resulting output codeword sequence **U** for the message sequence $\mathbf{m} = 1\ 1\ 0\ 1\ 1$, followed by $K - 1 = 2$ zeros to flush the register. Assume that the initial contents of the register are all zeros.

Solution

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	0 0 0	0 0	0 0	–	
1	1 0 0	0 0	1 0	1	1
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1



Output sequence: $\mathbf{U} = 1\ 1 \quad 0\ 1 \quad 0\ 1 \quad 0\ 0 \quad 0\ 1 \quad 0\ 1 \quad 1\ 1$

Example 6.2 Convolutional Encoding

In Example 6.1 the initial contents of the register are all zeros. This is equivalent to the condition that the given input sequence is preceded by two zero bits (the encoding is a function of the present bit and the $K - 1$ prior bits). Repeat Example 6.1 with the assumption that the given input sequence is preceded by two one bits, and verify that now the codeword sequence **U** for input sequence $\mathbf{m} = 1\ 1\ 0\ 1\ 1$ is different than the codeword found in Example 6.1.

Solution

The entry "×" signifies "don't know."

Input bit m_i	Register contents	State at time t_i	State at time t_{i+1}	Branch word at time t_i u_1	u_2
–	1 1 ×	1 ×	1 1	–	
1	1 1 1	1 1	1 1	1	0
1	1 1 1	1 1	1 1	1	0
0	0 1 1	1 1	0 1	0	1
1	1 0 1	0 1	1 0	0	0
1	1 1 0	1 0	1 1	0	1
0	0 1 1	1 1	0 1	0	1
0	0 0 1	0 1	0 0	1	1

state t_i

state t_{i+1}

Output sequence: $\mathbf{U}$ = 1 0 1 0 0 1 0 0 0 1 0 1 1 1

By comparing this result with that of Example 6.1, we can see that each branch word of the output sequence $\mathbf{U}$ is *not only* a function of the input bit, but is also a function of the $K - 1$ prior bits.

6.2.3 The Tree Diagram

Although the state diagram completely characterizes the encoder, one cannot easily use it for tracking the encoder transitions as a function of time since the diagram cannot represent time history. The tree diagram adds the *dimension of time* to the state diagram. The tree diagram for the covolutional encoder shown in Figure 6.3 is illustrated in Figure 6.6. At each successive input bit time the encoding procedure can be described by traversing the diagram from left to right, each tree branch describing an output branch word. The branching rule for finding a codeword sequence is as follows: If the input bit is a zero, its associated branch word is found by moving to the next rightmost branch in the upward direction. If the input bit is a one, its branch word is found by moving to the next rightmost branch in the downward direction. Assuming that the initial contents of the encoder is all zeros, the diagram shows that if the first input bit is a zero, the output branch word is 00 and, if the first input bit is a one, the output branch word is 11. Similarly, if the first input bit is a one and the second input bit is a zero, the second output branch word is 10. Or, if the first input bit is a one and the second input bit is a one, the second output branch word is 01. Following this procedure we see that the input sequence 1 1 0 1 1 traces the heavy line drawn on the tree

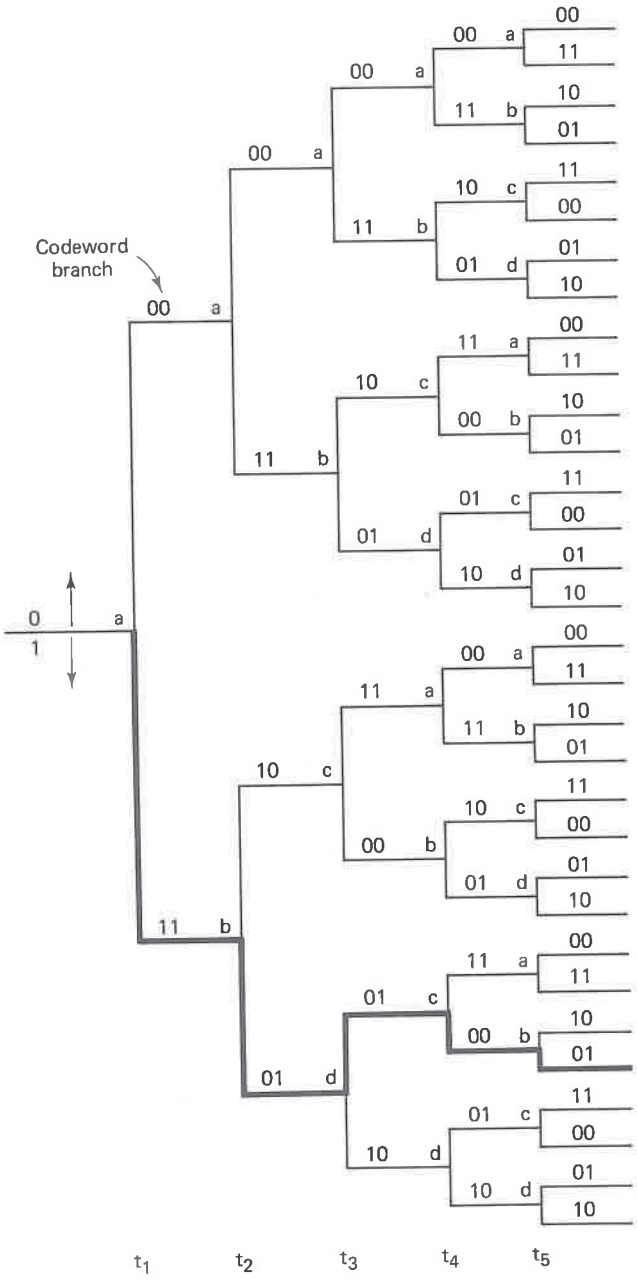


Figure 6.6 Tree representation of encoder (rate $\frac{1}{2}$, $K = 3$).

diagram in Figure 6.6. This path corresponds to the following output codeword sequence: 1 1 0 1 0 1 0 0 0 1.

The added dimension of time in the tree diagram (compared to the state diagram) allows one to dynamically describe the encoder as a function of a particular input sequence. However, can you see one problem in trying to use a tree

diagram for describing a sequence of any length? The number of branches increase as a function of 2^L, where L is the number of bits in the input sequence. You would quickly run out of paper, and patience.

6.2.4 The Trellis Diagram

Observation of the Figure 6.6 tree diagram shows that for this example, the structure repeats itself at time t_4, after the third branching (in general, the tree structure *repeats after K branchings*, where K is the constraint length). We label each node in the tree of Figure 6.6 to correspond to the four possible states in the shift register, as follows: $a = 00$, $b = 10$, $c = 01$, and $d = 11$. The first branching of the tree structure, at time t_1, produces a pair of nodes labeled a and b. At each successive branching the number of nodes double. The second branching, at time t_2, results in four nodes labeled a, b, c, and d. After the *third* branching there are a total of eight nodes; two of them are labeled a, two are labeled b, two are labeled c, and two are labeled d. We can see that all branches emanating from two nodes of the same state generate identical branch word sequences. From this point on, the upper and the lower halves of the tree are identical. The reason for this should be obvious from examination of the encoder in Figure 6.3. As the fourth input bit enters the encoder on the left, the first input bit is ejected on the right and no longer influences the output branch words. Consequently, the input sequences 1 0 0 x y . . . and 0 0 0 x y . . . , where the leftmost bit is the earliest bit, generate the same branch words after the ($K = 3$)rd branching. This means that any two nodes having the same state label, at the same time t_i, can be merged since all succeeding paths will be indistinguishable. If we do this to the tree structure of Figure 6.6, we obtain another diagram, called the trellis. The *trellis dia-*

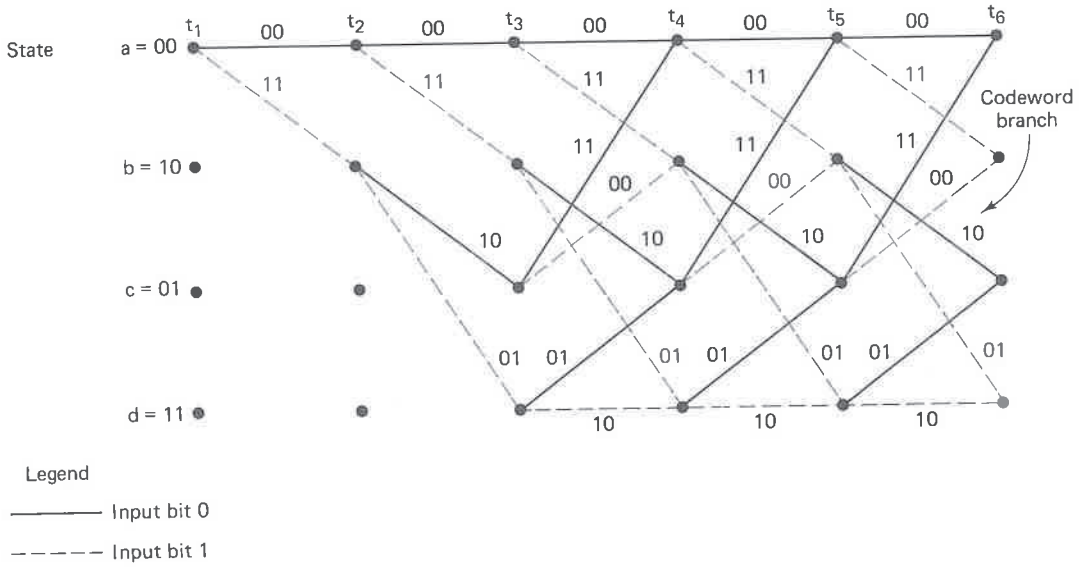


Figure 6.7 Encoder trellis diagram (rate $\frac{1}{2}$, $K = 3$).

gram, by exploiting the repetitive structure, provides a more manageable encoder description than does the tree diagram. The trellis diagram for the convolutional encoder of Figure 6.3 is shown in Figure 6.7.

In drawing the trellis diagram, we use the same convention that we introduced with the state diagram—that a solid line denotes the output generated by an input bit, zero, and a dashed line denotes the output generated by an input bit, one. The nodes of the trellis characterize the encoder states; the first row nodes correspond to the state $a = 00$, the second and subsequent rows correspond to the states $b = 10$, $c = 01$, and $d = 11$. At each unit of time the trellis requires 2^{K-1} nodes to represent the 2^{K-1} possible encoder states. The trellis in our example assumes a fixed periodic structure after trellis depth 3 is reached (at time t_4). In the general case, the fixed structure prevails after depth K is reached. After this point, each of the states can be entered from either of two preceding states. Also, each of the states can transition to one of two states. Of the two outgoing branches, one corresponds to an input bit zero and the other corresponds to an input bit one. On Figure 6.7 the output branch words corresponding to the state transitions appear as labels on the trellis branches.

6.3 FORMULATION OF THE CONVOLUTIONAL DECODING PROBLEM

6.3.1 Maximum Likelihood Decoding

If all input message sequences are equally likely, a decoder that achieves the minimum probability of error is one that compares the conditional probabilities, also called the *likelihood functions*, $P(\mathbf{Z}|\mathbf{U}^{(m)})$, where $\mathbf{Z}$ is the received sequence and $\mathbf{U}^{(m)}$ is one of the possible transmitted sequences, and chooses the maximum. The decoder chooses $\mathbf{U}^{(m')}$ if

$$P(\mathbf{Z}|\mathbf{U}^{(m')}) = \max_{\text{all } \mathbf{U}^{(m)}} P(\mathbf{Z}|\mathbf{U}^{(m)}) \tag{6.1}$$

The *maximum likelihood* concept, as stated in Equation (6.1), is a fundamental development of decision theory (see Appendix B); it is the formalization of a "common-sense" way to make decisions when there is statistical knowledge of the possibilities. In the binary demodulation treatment in Chapters 2 and 3 there were *only two* equally likely possible signals, $s_1(t)$ or $s_2(t)$, that might have been transmitted. Therefore, to make the binary maximum likelihood decision, given a received signal, meant only to decide that $s_1(t)$ was transmitted if

$$p(z|s_1) > p(z|s_2)$$

otherwise, to decide that $s_2(t)$ was transmitted. The parameter z represents $z(T)$, the receiver output at a symbol duration time $t = T$. However, when applying maximum likelihood to the convolutional decoding problem, there are typically a *multitude* of possible codeword sequences that might have been transmitted. To be specific, an L-bit codeword sequence is a member of a set of 2^L possible

sequences. Therefore, in the maximum likelihood context, we can say that the decoder chooses a particular $\mathbf{U}^{(m')}$ as the transmitted sequence if the likelihood $P(\mathbf{Z}|\mathbf{U}^{(m')})$ is greater than the likelihoods of all the other possible transmitted sequences. Such an optimal decoder, which minimizes the error probability (for the case where all transmitted sequences are equally likely), is known as a *maximum likelihood decoder*. The likelihood functions are given or computed from the specifications of the channel.

We will assume that the noise is additive white Gaussian with zero mean and the channel is *memoryless*, which means that the noise affects each code symbol *independently* of all the other symbols. For a convolutional code of rate $1/n$, we can therefore express the likelihood, $P(\mathbf{Z}|\mathbf{U}^{(m)})$ as follows:

$$P(\mathbf{Z}|\mathbf{U}^{(m)}) = \prod_{i=1}^{\infty} P(Z_i|U_i^{(m)}) = \prod_{i=1}^{\infty} \prod_{j=1}^{n} P(z_{ji}|u_{ji}^{(m)}) \tag{6.2}$$

where Z_i is the ith branch of the received sequence $\mathbf{Z}$, $U_i^{(m)}$ the ith branch of a particular codeword sequence $\mathbf{U}^{(m)}$, z_{ji} the jth code symbol of Z_i, and $u_{ji}^{(m)}$ the jth code symbol of $U_i^{(m)}$, each branch comprising n code symbols. The decoder problem consists of choosing a path through the trellis of Figure 6.7 (each possible path defines a codeword) such that

$$\prod_{i=1}^{\infty} \prod_{j=1}^{n} P(z_{ji}|u_{ji}^{(m)}) \text{ is maximized} \tag{6.3}$$

Generally, it is computationally more convenient to use the logarithm of the likelihood function since this permits the summation, instead of the multiplication, of terms. We are able to use this transformation because the logarithm is a monotonically increasing function and thus will not alter the final result in our codeword selection. We can define the log-likelihood function $\gamma_U(m)$ as

$$\gamma_U(m) = \log P(\mathbf{Z}|\mathbf{U}^{(m)}) = \sum_{i=1}^{\infty} \log P(Z_i|U_i^{(m)}) = \sum_{i=1}^{\infty} \sum_{j=1}^{n} \log P(z_{ji}|u_{ji}^{(m)}) \tag{6.4}$$

The decoder problem now consists of choosing a path through the tree of Figure 6.6 or the trellis of Figure 6.7 such that $\gamma_U(m)$ is maximized. For the decoding of convolutional codes, either the tree or the trellis structure can be used. In the tree representation of the code, the fact that the paths remerge is ignored. Since the number of possible sequences for an L-symbol-long sequence is 2^L, maximum likelihood decoding of an L-bit-long received sequence, using a tree diagram, requires the "brute force" or exhaustive comparison of 2^L accumulated log-likelihood metrics, representing all the possible different codewords that could have been transmitted. Hence it is not practical to consider maximum likelihood decoding with a tree structure. It is shown in a later section that with the use of the trellis representation of the code, it is possible to configure a decoder which can discard the paths that could not possibly be candidates for the maximum likelihood sequence. The decoded path is chosen from some reduced set of *surviving paths*. Such a decoder is still optimum in the sense that the decoded path is the same

as the decoded path obtained from a "brute force" maximum likelihood decoder, but the early rejection of unlikely paths reduces the decoding complexity.

For an excellent tutorial on the structure of convolutional codes, maximum likelihood decoding, and code performance, see Reference [8]. There are several algorithms that yield *approximate* solutions to the maximum likelihood decoding problem, including sequential [9, 10] and threshold [11]. Each of these algorithms is suited to certain special applications, but are all suboptimal. In contrast, the *Viterbi decoding algorithm* performs maximum likelihood decoding and is therefore optimal. This does not imply that the Viterbi algorithm is best for every application; there are severe constraints imposed by hardware complexity. The Viterbi algorithm is considered in Sections 6.3.3 and 6.3.4.

6.3.2 Channel Models: Hard versus Soft Decisions

Before specifying an algorithm that will determine the maximum likelihood decision, let us describe the channel. The codeword sequence $\mathbf{U}^{(m)}$, made up of branch words, with each branch word comprised of n code symbols, can be considered to be an endless stream, as opposed to a block code, in which the source data and their codewords are partitioned into precise block sizes. The codeword sequence shown in Figure 6.1 emanates from the convolutional encoder and enters the modulator, where the code symbols are transformed into signal waveforms. The modulation may be baseband (e.g., pulse waveforms) or bandpass (e.g., PSK or FSK). In general, ℓ symbols at a time, where ℓ is an integer, are mapped into signal waveforms $s_i(t)$, where $i = 1, 2, \ldots, M = 2^{\ell}$. When $\ell = 1$, the modulator maps each code symbol into a binary waveform. The channel over which the waveform is transmitted is assumed to corrupt the signal with Gaussian noise. When the corrupted signal is received, it is first processed by the demodulator and then by the decoder.

Consider that a binary signal, transmitted over a symbol interval $(0, T)$, is represented by $s_1(t)$ for a binary one and $s_2(t)$ for a binary zero. The received signal is $r(t) = s_i(t) + n(t)$, where $n(t)$ is a zero-mean Gaussian noise process. In Sections 2.9 and 3.4 we described the detection of $r(t)$ in terms of two basic steps. In the first step, the received waveform is reduced to a single number, $z(T) = a_i + n_0$, where a_i is the signal component of $z(T)$ and n_0 is the noise component. The noise component, n_0, is a zero-mean *Gaussian random variable*, and thus $z(T)$ is a *Gaussian random variable* with a mean of either a_1 or a_2 depending on whether a binary one or binary zero was sent. In the second step of the detection process a decision was made as to which signal was transmitted, on the basis of comparing $z(T)$ to a threshold. The conditional probabilities of $z(T)$, $p(z|s_1)$, and $p(z|s_2)$ are shown in Figure 6.8, labeled likelihood of s_1 and likelihood of s_2. The demodulator in Figure 6.1, converts the set of time-ordered random variables, $\{z(T)\}$, into a code sequence, $\mathbf{Z}$, and passes it on to the decoder. The demodulator output can be configured in a variety of ways. It can be implemented to make a *firm or hard decision* as to whether $z(T)$ represents a zero or a one. In this case, the output of the demodulator is quantized to two levels, zero and one, and fed

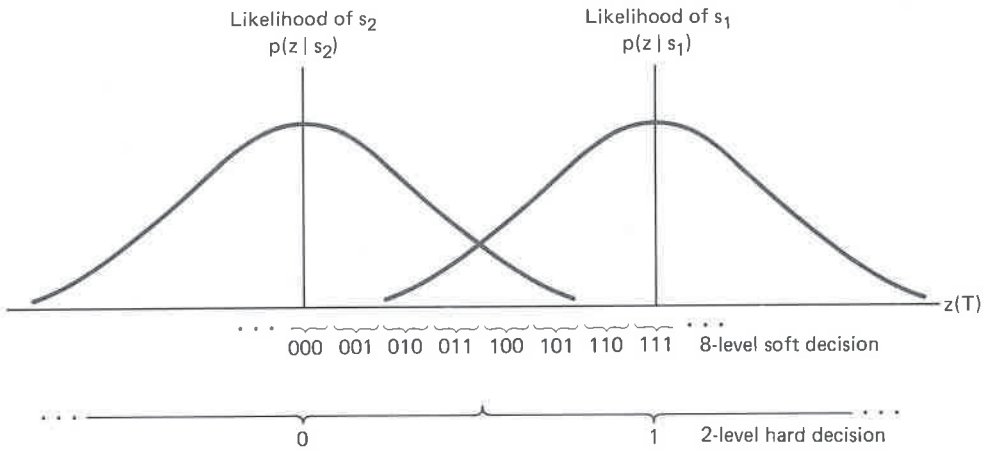


Figure 6.8 Hard and soft decoding decisions.

into the decoder (this is exactly the same threshold decision that was made in Chapters 2 and 3). Since the decoder operates on the hard decisions made by the demodulator, the decoding is called *hard-decision decoding*.

The demodulator can also be configured to feed the decoder with a *quantized value* of $z(T)$ *greater than two levels*, or with an unquantized or analog value of $z(T)$. Such an implementation furnishes the decoder with more information than is provided in the hard-decision case. When the quantization level of the demodulator output is greater than two, the decoding is called *soft-decision decoding*. Eight levels (3-bits) of quantization are illustrated on the abscissa of Figure 6.8. When the demodulator sends a hard binary decision to the decoder, it sends it a single binary symbol. When the demodulator sends a soft binary decision, quantized to eight levels, it sends the decoder a 3-bit word describing an interval along $z(T)$. In effect, sending such a 3-bit word in place of a single binary symbol is equivalent to sending the decoder a *measure of confidence* along with the code symbol. Referring to Figure 6.8, if the demodulator sends 1 1 1 to the decoder, this is tantamount to declaring the code symbol to be a one with very high confidence, while sending a 1 0 0 is tantamount to declaring the code symbol to be a one with very low confidence. It should be clear that ultimately, every message decision out of the decoder must be a hard decision; otherwise, one might see computer printouts that read: "think it's a 1," "think it's a 0," and so on. The idea behind the demodulator *not making hard decisions* and sending more data (soft decisions) to the decoder can be thought of as an interim step to provide the decoder with more information, which the decoder then uses for recovering the message sequence (with better error performance than it could in the case of hard-decision decoding).

For a Gaussian channel, eight-level quantization results in a performance improvement of approximately 2 dB in required signal-to-noise ratio compared to two-level quantization. This means that eight-level soft-decision decoding can provide the same probability of bit error as that of hard-decision decoding, but

requires 2 dB *less* E_b/N_0 for the same performance. Analog (or infinite-level quantization) results in a 2.2-dB performance improvement over two-level quantization; therefore, *eight-level quantization* results in a loss of approximately 0.2 dB compared to infinitely fine quantization. For this reason, quantization to more than eight levels can yield little performance improvement [12]. What price is paid for such improved soft-decision-decoder performance? In the case of hard-decision decoding, a single bit is used to describe each code symbol, while for eight-level quantized soft-decision decoding 3 bits are used to describe each code symbol; therefore, three times the amount of data must be handled during the decoding process. Hence the price paid for soft-decision decoding is an increase in required memory size at the decoder (and possibly a speed penalty).

Block decoding algorithms and convolutional decoding algorithms have been devised to operate with hard *or* soft decisions. However, soft-decision decoding is generally not used with block codes because it is considerably more difficult than hard-decision decoding to implement. The most prevalent use of soft-decision decoding is with the *Viterbi convolutional decoding algorithm*, since with Viterbi decoding, soft decisions represent only a trivial increase in computation.

6.3.2.1 Binary Symmetric Channel

A binary symmetric channel (BSC) is a discrete memoryless channel (see Section 5.3.1) that has binary input and output alphabets and symmetric transition probabilities. It can be described by the conditional probabilities

$$\begin{aligned} P(0|1) &= P(1|0) = p \\ P(1|1) &= P(0|0) = 1 - p \end{aligned} \tag{6.5}$$

as illustrated in Figure 6.9. The probability that an output symbol will differ from the input symbol is p, and the probability that the output symbol will be identical to the input symbol is $(1 - p)$. The BSC is an example of a *hard-decision channel*, which means that, even though continuous-valued signals may be received by the demodulator, a BSC allows only firm decisions such that each demodulator output symbol, z_{ji}, as shown in Figure 6.1, consists of one of two binary values. The indexing of z_{ji} pertains to the jth code symbol of the ith branch word, Z_i. The demodulator then feeds the sequence $\mathbf{Z} = \{Z_i\}$ to the decoder.

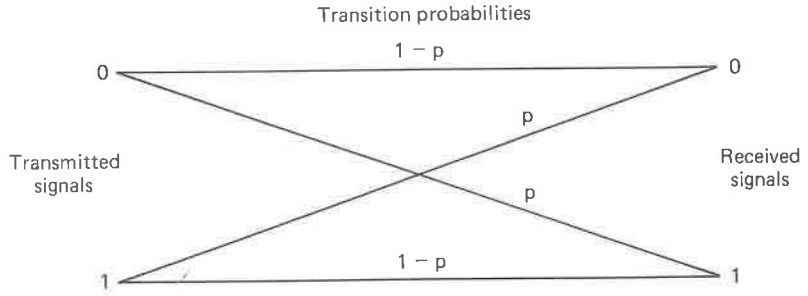


Figure 6.9 Binary symmetric channel (hard-decision channel).

Let $\mathbf{U}^{(m)}$ be a transmitted codeword over a BSC with symbol error probability p, and let $\mathbf{Z}$ be the corresponding received decoder sequence. As noted previously, a maximum likelihood decoder chooses the codeword $\mathbf{U}^{(m')}$ which maximizes the likelihood, $P(\mathbf{Z}|\mathbf{U}^{(m)})$, or its logarithm. For a BSC, this is equivalent to choosing the codeword, $\mathbf{U}^{(m')}$, that is closest in *Hamming distance* to $\mathbf{Z}$ [8]. Thus Hamming distance is an appropriate metric to describe the distance or closeness of fit between $\mathbf{U}^{(m)}$ and $\mathbf{Z}$. From all the possible transmitted sequences, $\mathbf{U}^{(m)}$, the decoder chooses the $\mathbf{U}^{(m')}$ sequence for which the distance to $\mathbf{Z}$ is minimum.

Suppose that $\mathbf{U}^{(m)}$ and $\mathbf{Z}$ are each L-bit-long sequences and that they differ in d_m positions [i.e., the Hamming distance between $\mathbf{U}^{(m)}$ and $\mathbf{Z}$ is d_m]. Then, since the channel is assumed memoryless, the probability that this $\mathbf{U}^{(m)}$ was transformed to the specific received $\mathbf{Z}$ at distance d_m from it can be written

$$P(\mathbf{Z}|\mathbf{U}^{(m)}) = p^{d_m}(1-p)^{L-d_m} \tag{6.6}$$

and the log-likelihood function is

$$\log P(\mathbf{Z}|\mathbf{U}^{(m)}) = -d_m \log\left(\frac{1-p}{p}\right) + L\log(1-p) \tag{6.7}$$

If we compute this quantity for each possible transmitted sequence, the second term will be constant in each case. Assuming that $p < 0.5$, we can express Equation (6.7) as

$$\log P(\mathbf{Z}|\mathbf{U}^{(m)}) = -Ad_m - B \tag{6.8}$$

where A and B are positive constants. Therefore, choosing the codeword $\mathbf{U}^{(m')}$ such that the Hamming distance, d_m, to the received sequence $\mathbf{Z}$ is minimized corresponds to *maximizing the likelihood or log-likelihood metric*. Consequently, over a BSC, the log-likelihood metric is conveniently replaced by the Hamming distance, and a maximum likelihood decoder will choose, in the tree or trellis diagram, the path whose corresponding sequence, $\mathbf{U}^{(m')}$, is at the *minimum Hamming distance* to the received sequence $\mathbf{Z}$.

6.3.2.2 Gaussian Channel

For a Gaussian channel, each demodulator output symbol, z_{ji}, as shown in Figure 6.1, is a value from a continuous alphabet. The symbol z_{ji} cannot be labeled as a correct or incorrect detection decision. Sending the decoder such soft decisions can be viewed as sending a family of conditional probabilities of the different symbols (see Section 5.3.1). It can be shown [8] that maximizing $P(\mathbf{Z}|\mathbf{U}^{(m)})$ is equivalent to maximizing the inner product between the codeword sequence, $\mathbf{U}^{(m)}$ (consisting of binary symbols), and the analog-valued received sequence, $\mathbf{Z}$. Thus the decoder chooses the codeword $\mathbf{U}^{(m')}$ if it maximizes

$$\sum_{i=1}^{\infty}\sum_{j=1}^{n} z_{ji}u_{ji}^{(m)} \tag{6.9}$$

This is equivalent to choosing the codeword $\mathbf{U}^{(m')}$ that is closest in *Euclidean distance* to $\mathbf{Z}$. Even though the hard- and soft-decision channels require different

metrics, the concept of choosing the codeword $\mathbf{U}^{(m')}$ that is closest to the received sequence, $\mathbf{Z}$, is the same in both cases. To implement the maximization of Equation (6.9) exactly, the decoder would have to be able to handle analog-valued arithmetic operations. This is impractical because the decoder is generally implemented digitally. Thus it is necessary to quantize the received symbols z_{ji}. Does Equation (6.9) remind you of the demodulation treatment in Chapter 3? Equation (6.9) is the discrete version of correlating an input received waveform, $r(t)$, with a reference waveform, $s_i(t)$, as expressed in Equation (3.34). The quantized Gaussian channel, typically referred to as a *soft-decision channel*, is the channel model assumed for the soft-decision decoding described earlier.

6.3.3 The Viterbi Convolutional Decoding Algorithm

The Viterbi decoding algorithm was discovered and analyzed by Viterbi [13] in 1967. The Viterbi algorithm essentially performs maximum likelihood decoding; however, it reduces the computational load by taking advantage of the special structure in the code trellis. The advantage of Viterbi decoding, compared with brute-force decoding, is that the complexity of a Viterbi decoder is not a function of the number of symbols in the codeword sequence. The algorithm involves calculating a *measure of similarity, or distance*, between the received signal, at time t_i, and all the trellis paths entering each state at time t_i. The Viterbi algorithm removes from consideration those trellis paths that could not possibly be candidates for the maximum likelihood choice. When two paths enter the same state, the one having the best metric is chosen; this path is called the *surviving path*. This selection of surviving paths is performed for all the states. The decoder continues in this way to advance deeper into the trellis, making decisions by eliminating the least likely paths. The early rejection of the unlikely paths reduces the decoding complexity. In 1969, Omura [14] demonstrated that the Viterbi algorithm is, in fact, maximum likelihood. Note that the goal of selecting the optimum path can be expressed, equivalently, as choosing the codeword with the *maximum likelihood metric*, or as choosing the codeword with the *minimum distance metric*.

6.3.4 An Example of Viterbi Convolutional Decoding

For simplicity, a BSC is assumed; thus Hamming distance is a proper distance measure. The encoder for this example is shown in Figure 6.3, and the encoder trellis diagram is shown in Figure 6.7. A similar trellis can be used to represent the decoder, as shown in Figure 6.10. The basic idea behind the decoding procedure can best be understood by examining the Figure 6.7 encoder trellis in concert with the Figure 6.10 decoder trellis. For the decoder trellis it is convenient to label each trellis branch at time t_i with the *Hamming distance* between the received code symbols and the corresponding branch word from the encoder trellis. The example in Figure 6.10, shows a message sequence, $\mathbf{m}$, the corresponding codeword sequence, $\mathbf{U}$, and a noise corrupted received sequence, $\mathbf{Z}$ = 11 01 01 10 01 . . . . The branch words seen on the *encoder trellis* branches

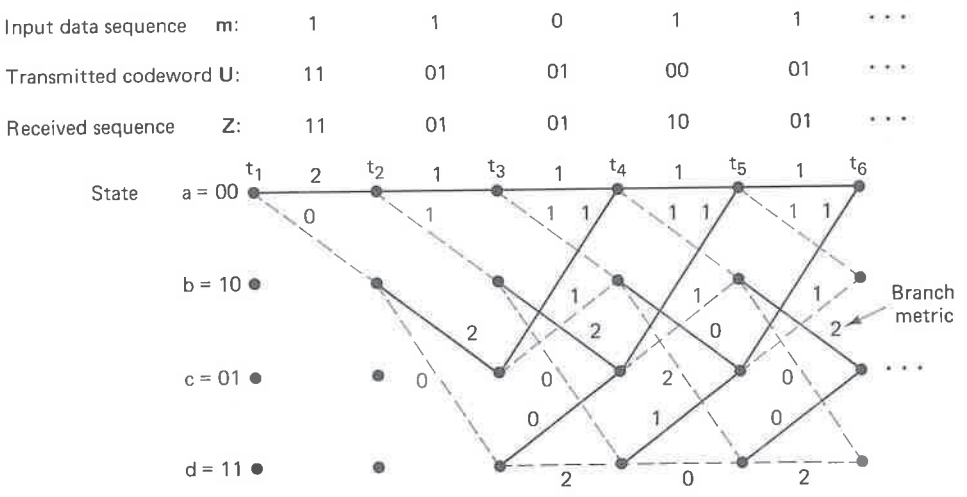


Figure 6.10 Decoder trellis diagram (rate $\frac{1}{2}$, $K = 3$).

characterize the encoder in Figure 6.3, and are known a priori to both the encoder and the decoder. These encoder branch words are the code symbols that would be expected to come from the encoder output as a result of each of the state transitions. The labels on the *decoder trellis* branches are accumulated by the decoder *on the fly*. That is, as the code symbols are received, each branch of the decoder trellis is labeled with a metric of similarity (Hamming distance) between the received code symbols and each of the branch words for that time interval. From the received sequence, **Z**, in Figure 6.10, we see that the code symbols received at time t_1 are 11. In order to label the decoder branches at time t_1 with the appropriate Hamming distance metric, we look at the Figure 6.7 encoder trellis. Here we see that a state 00 → 00 transition yields an output branch word of 00. But we received 11. Therefore, on the decoder trellis we label the state 00 → 00 transition with the Hamming distance between them, namely 2. Looking at the encoder trellis again, we see that a state 00 → 10 transition yields an output branch word of 11, which corresponds exactly with the code symbols we received at time t_1. Therefore, on the decoder trellis, we label the state 00 → 10 transition with a Hamming distance of 0. We continue labeling the decoder trellis branches in this way as the symbols are received at each time t_i. The decoding algorithm uses these Hamming distance metrics to find the *most likely* (minimum distance) path through the trellis.

The basis of *Viterbi decoding* is the following observation: If any two paths in the trellis merge to a single state, one of them can always be eliminated in the search for an optimum path. For example, Figure 6.11 shows two paths merging at time t_5 to state 00. Let us define the *cumulative Hamming path metric* of a given path at time t_i as the sum of the branch Hamming distance metrics along that path up to time t_i. In Figure 6.11 the upper path has metric 4; the lower has metric 1. The upper path cannot be a portion of the optimum path because the lower path, which enters the same state, has a lower metric. This observation

holds because of the Markov nature of the encoder state: The present state summarizes the encoder history in the sense that previous states cannot affect future states or future output branches.

At each time t_i there are 2^{K-1} states in the trellis, where K is the constraint length, and each state can be *entered by means of two paths*. Viterbi decoding consists of computing the metrics for the two paths entering each state and *eliminating one of them*. This computation is done for each of the 2^{K-1} nodes at time t_i; then the decoder moves to time t_{i+1} and repeats the process. The first few steps in our decoding example are as follows (see Figure 6.12). Assume that the input data sequence **m**, codeword **U**, and received sequence **Z** are as shown in Figure 6.10. Assume that the decoder knows the correct initial state of the trellis. (This assumption is not necessary in practice, but simplifies the explanation.) At time t_1 the received code symbols are 11. From state 00 the only possible transitions are to state 00 or state 10, as shown in Figure 6.12a. State 00 → 00 transition has branch metric 2; state 00 → 10 transition has branch metric 0. At time t_2 there are two possible branches leaving each state, as shown in Figure 6.12b. The cumulative path metrics of these branches are labeled λ_a, λ_b, λ_c, and λ_d, corresponding to the terminating state. At time t_3 in Figure 6.12c there are again two branches diverging from each state. As a result, there are two paths entering each state at time t_4. As noted previously, one path entering each state can be eliminated, namely, the one having the larger cumulative path metric. Should metrics of the two entering paths be of equal value, one path is chosen for elimination by using an arbitrary rule. The surviving path into each state is shown in Figure 6.12d. At this point in the decoding process, there is only a single surviving path between times t_1 and t_2. Therefore, the decoder can now decide that the state transition which occurred between t_1 and t_2 was 00 → 10. Since this transition is produced by an input bit one, the decoder outputs a one as the first decoded bit. Here we can see how the decoding of the surviving branch is facilitated by having drawn the lattice branches with solid lines for input zeros and dashed lines for input ones. Note that the first bit was not decoded until the path metric computation had proceeded to a much greater depth into the trellis. For a typical

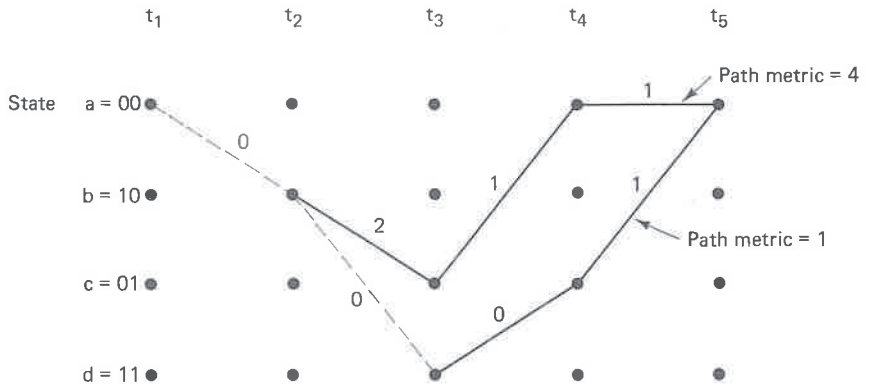


Figure 6.11 Path metrics for two merging paths.

decoder implementation, this represents a decoding delay which can be as much as five times the constraint length in bits.

At each succeeding step in the decoding process, there will always be two possible paths entering each state; one of the two will be eliminated by comparing the path metrics. Figure 6.12e shows the next step in the decoding process. Again, at time t_5 there are two paths entering each state, and one of each pair can be eliminated. Figure 6.12f shows the survivors at time t_5. Notice that in our example we cannot yet make a decision on the second input data bit because there still are two paths leaving the state 10 node at time t_2. At time t_6 in Figure 6.12g we again see the pattern of remerging paths, and in Figure 6.12h we see the survivors at time t_6. Also, in Figure 6.12h the decoder outputs one as the second decoded bit, corresponding to the single surviving path between t_2 and t_3. The decoder continues in this way to advance deeper into the trellis and to make decisions on the input data bits by eliminating all paths but one.

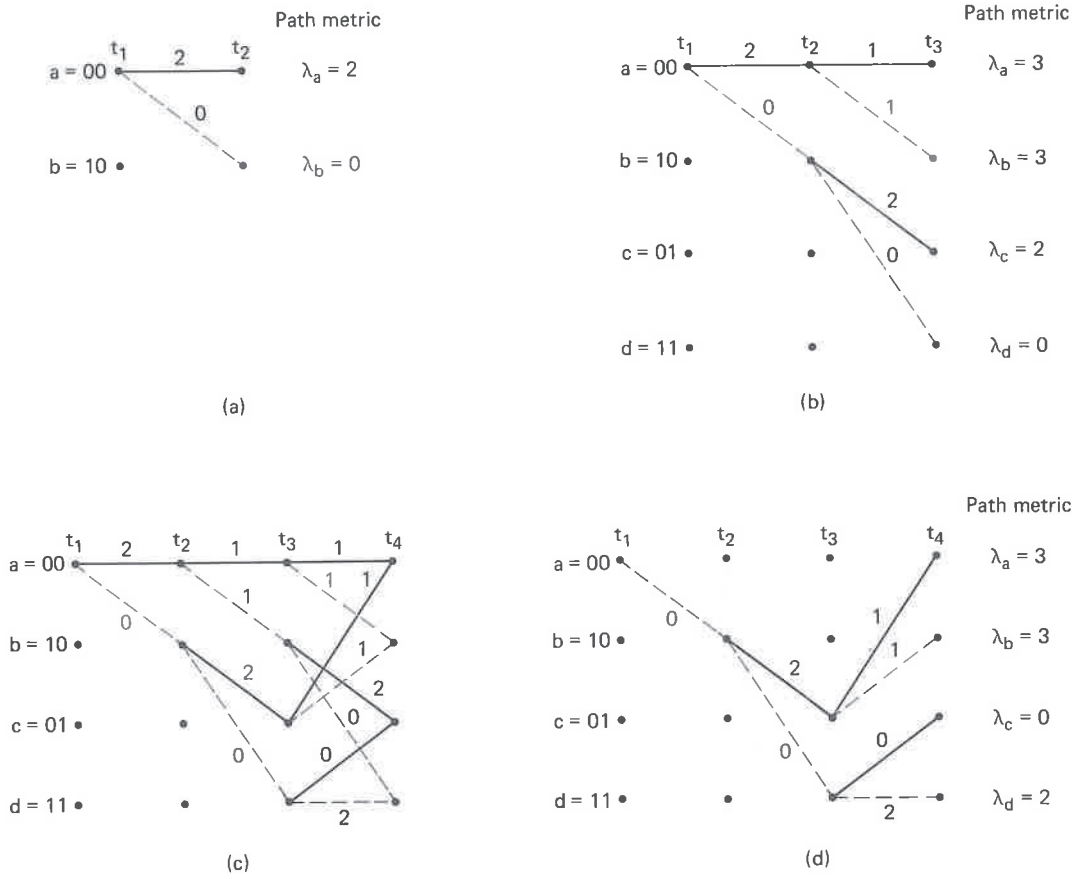


Figure 6.12 Selection of survivor paths. (a) Survivors at t_2. (b) Survivors at t_3. (c) Metric comparisons at t_4. (d) Survivors at t_4. (e) Metric comparisons at t_5. (f) Survivors at t_5. (g) Metric comparisons at t_6. (h) Survivors at t_6.

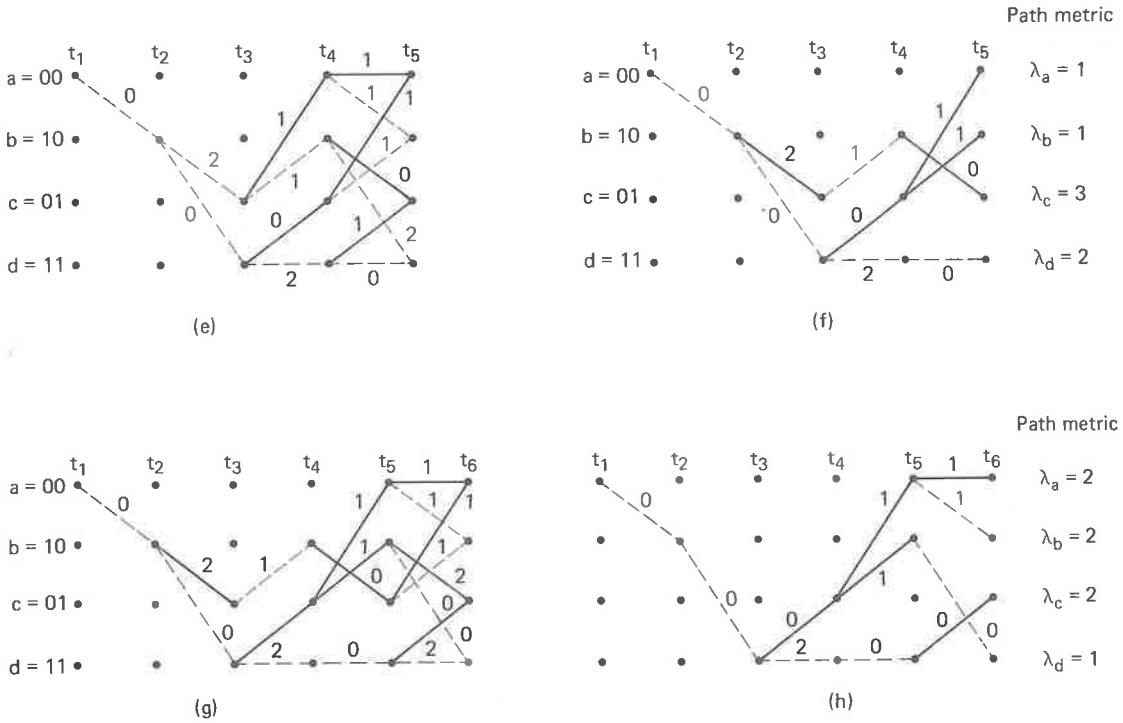


Figure 6.12 (*Continued*)

6.3.5 Path Memory and Synchronization

The storage requirements of the Viterbi decoder grow exponentially with constraint length K. For a code with rate $1/n$, the decoder retains a set of 2^{K-1} paths after each decoding step. With high probability, these paths will not be mutually disjoint very far back from the present decoding depth [12]. All of the 2^{K-1} paths tend to have a common stem which eventually branches to the various states. Thus if the decoder stores enough of the history of the 2^{K-1} paths, the oldest bits on all paths will be the same. A simple decoder implementation, then, contains a *fixed amount of path history* and outputs the oldest bit on an arbitrary path each time it steps one level deeper into the trellis. The amount of path storage required, u, is [12]

$$u = h2^{K-1} \tag{6.10}$$

where h is the length of the information bit path history per state. A refinement, which minimizes the value of h, uses the oldest bit on the most likely path as the decoder output, instead of the oldest bit on an arbitrary path. It has been demonstrated [12] that a value of h of 4 or 5 times the code constraint length is sufficient for near-optimum decoder performance. The storage requirement, u, is the basic limitation on the implementation of Viterbi decoders. The current state of the art

limits decoders to a constraint length of about $K = 10$. Efforts to increase coding gain by further increasing constraint length are met by the exponential increase in memory requirements (and complexity) that follows from Equation (6.10).

Branch word synchronization is the process of determining the beginning of a branch word in the received sequence. Such synchronization can take place without new information being added to the transmitted symbol stream because the received data appear to have an excessive error rate when not synchronized. Therefore, a simple way of accomplishing synchronization is to monitor some concomitant indication of this large error rate, that is, the rate at which the path metrics are increasing or the rate at which the surviving paths in the trellis merge. The monitored parameters are compared to a threshold, and synchronization is then adjusted accordingly.

6.4 PROPERTIES OF CONVOLUTIONAL CODES

6.4.1 Distance Properties of Convolutional Codes

Let us consider the distance properties of convolutional codes in the context of our simple encoder in Figure 6.3 and its trellis diagram in Figure 6.7. We want to evaluate the distance between all possible pairs of codeword sequences. As in the case of block codes (see Section 5.5.2), we are interested in the *minimum distance* between all pairs of such codeword sequences in the code, since the minimum distance is related to the error-correcting capability of the code. Because a convolutional code is a group or *linear code* [6], there is no loss in generality in simply finding the minimum distance between each of the codeword sequences and the all-zeros sequence. Assuming that the all-zeros input sequence was transmitted, the paths of interest are those that start and end in the 00 state and do not return to the 00 state anywhere in between. An error will occur whenever the distance of any other path that merges with the $a = 00$ state at time t_i is less than that of the all-zeros path up to time t_i, causing the all-zeros path to be discarded in the decoding process. In other words, given the all-zeros transmission, an error occurs whenever the *all-zeros path does not survive*. The minimum distance for making such an error can be found by exhaustively examining every path from the 00 state to the 00 state. First, let us redraw the trellis diagram, shown in Figure 6.13, labeling each branch with its Hamming distance from the all-zeros codeword instead of with its branch word symbols. The Hamming distance between two unequal-length sequences will be found by first appending the necessary number of zeros to the shorter sequence to make the two sequences equal in length. Consider all the paths that diverge from the all-zeros path and then remerge for the first time at some arbitrary node. From Figure 6.13 we can compute the distances of these paths from the all-zeros path. There is one path at distance 5 from the all-zeros path; this path departs from the all-zeros path at time t_1 and merges with it at time t_4. Similarly, there are two paths at distance 6, one which departs at time t_1 and merges at time t_5, and the other which departs at time t_1 and merges at time t_6, and so on. We can also see from the dashed and solid lines

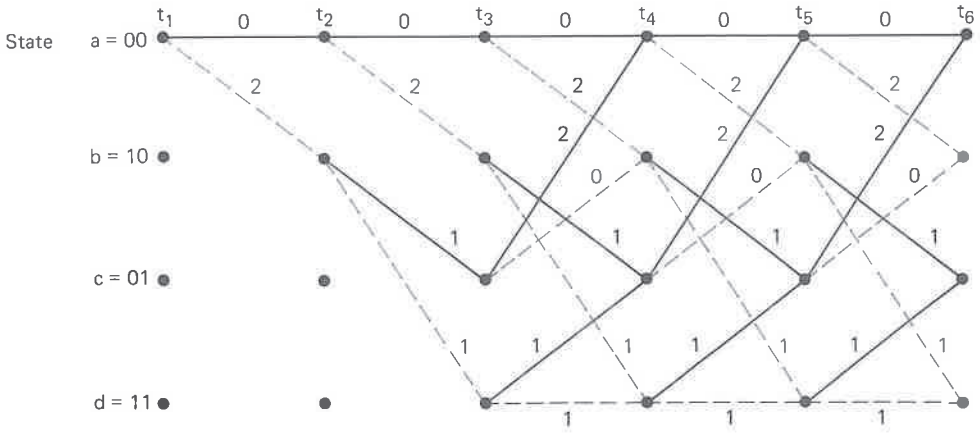


Figure 6.13 Trellis diagram, labeled with distances from the all-zeros path.

of the diagram that the input bits for the distance 5 path are 1 0 0; it differs in only one input bit from the all-zeros input sequence. Similarly, the input bits for the distance 6 paths are 1 1 0 0 and 1 0 1 0 0; each differs in two positions from the all-zeros path. The minimum distance in the set of all arbitrarily long paths that diverge and remerge, called the *minimum free distance* or simply the *free distance*, is seen to be 5 in this example. For calculating the error-correcting capability of the code, we repeat Equation (5.44) with the minimum distance, $d_{\min}$, replaced by the free distance, d_f.

$$t = \left\lfloor \frac{d_f - 1}{2} \right\rfloor \tag{6.11}$$

where $\lfloor x \rfloor$ means the largest integer no greater than x. Setting $d_f = 5$, we see that the code, characterized by the Figure 6.3 encoder, can correct any two channel errors.

Although Figure 6.13 presents the computation of free distance in a straightforward way, a more direct closed-form expression can be obtained by starting with the state diagram in Figure 6.5. First, we label the branches of the state diagram as either $D^0 = 1$, D^1, or D^2, shown in Figure 6.14, where the exponent of D denotes the Hamming distance from the branch word of that branch to the all-zeros branch. The self-loop at node a can be eliminated since it contributes nothing to the distance properties of a codeword sequence relative to the all-zeros sequence. Furthermore, node a can be split into two nodes (labeled a and e), one of which represents the input and the other the output of the state diagram. All paths originating at $a = 00$ and terminating at $e = 00$ can be traced on the modified state diagram of Figure 6.14. We can calculate the transfer function of path $a\ b\ c\ e$ (starting and ending at state 00) in terms of the indeterminate "placeholder" D, as $D^2\ D\ D^2 = D^5$. The exponent of D represents the cumulative tally of the number of ones in the path, and hence the Hamming distance from the all-zeros path. Similarly, the paths $a\ b\ d\ c\ e$ and $a\ b\ c\ b\ c\ e$ each have the transfer

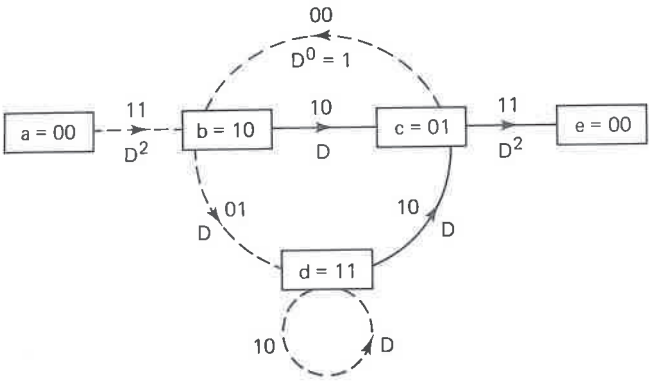


Figure 6.14 State diagram, labeled according to distance from the all-zeros path.

function D^6 and thus a Hamming distance of 6 from the all-zeros path. We now write the following state equations:

$$\begin{aligned} X_b &= D^2X_a + X_c \\ X_c &= DX_b + DX_d \\ X_d &= DX_b + DX_d \\ X_e &= D^2X_c \end{aligned} \tag{6.12}$$

where $X_a, \ldots, X_e$ are dummy variables for the partial paths to the intermediate nodes. The *transfer function*, $T(D)$, sometimes called the *generating function* of the code can be expressed as $T(D) = X_e/X_a$. By solving the state equations shown in Equation (6.12), we obtain [15, 16]

$$\begin{aligned} T(D) &= \frac{D^5}{1 - 2D} \\ &= D^5 + 2D^6 + 4D^7 + \cdots + 2^\ell D^{\ell+5} + \cdots \end{aligned} \tag{6.13}$$

The transfer function for this code indicates that there is a single path of distance 5 from the all-zeros path, two of distance 6, four of distance 7, and in general, there are 2^ℓ paths of distance $\ell + 5$ from the all-zeros path, where $\ell = 0, 1, 2, \ldots$. The free distance d_f of the code is the Hamming weight of the lowest-order term in the expansion of $T(D)$. In this example $d_f = 5$. In evaluating distance properties, the transfer function, $T(D)$, cannot be used for long constraint lengths since the complexity of $T(D)$ increases exponentially with constraint length.

The transfer function can be used to provide more detailed information than just the distance of the various paths. Let us introduce a factor L into each branch of the state diagram so that the exponent of L can serve as a counter to indicate the number of branches in any given path from state $a = 00$ to state $e = 00$. Furthermore, we can introduce a factor N into all branch transitions caused by the input bit one. Thus, as each branch is traversed, the cumulative exponent on N increases by one, only if that branch transition is due to an input bit one. For the convolutional code characterized in our Figure 6.3 example, the additional

factors L and N are shown on the modified state diagram of Figure 6.15. We can now modify Equations (6.12) as follows:

$$\begin{aligned} X_b &= D^2LNX_a + LNX_c \\ X_c &= DLX_b + DLX_d \\ X_d &= DLNX_b + DLNX_d \\ X_e &= D^2LX_c \end{aligned} \tag{6.14}$$

The transfer function of this augmented state diagram is

$$\begin{aligned} T(D, L, N) &= \frac{D^5L^3N}{1 - DL(1 + L)N} \\ &= D^5L^3N + D^6L^4(1 + L)N^2 + D^7L^5(1 + L)^2N^3 \\ &\quad + \cdots + D^{\ell+5}L^{\ell+3}N^{\ell+1} + \cdots \end{aligned} \tag{6.15}$$

Thus we can verify some of the path properties displayed in Figure 6.13. There is one path of distance 5, length 3, which differs in one input bit from the all-zeros path. There are two paths of distance 6, one of which is length 4, the other length 5, and both differ in two input bits from the all-zeros path. Also, of the distance 7 paths, one is of length 5, two are of length 6, and one is of length 7; all four paths correspond to input sequences that differ in three input bits from the all-zeros path. Thus if the all-zeros path is the correct path and the noise causes us to choose one of the incorrect paths of distance 7, three bit errors will be made.

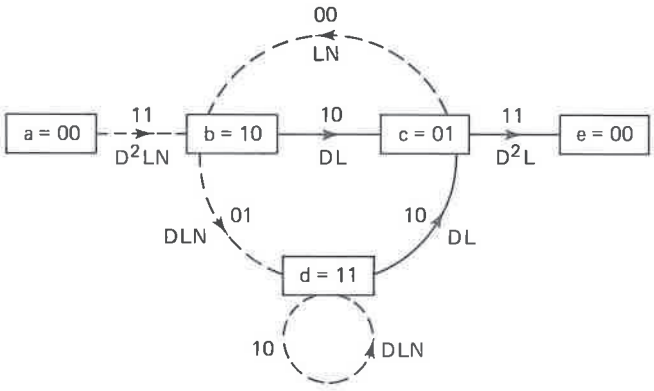


Figure 6.15 State diagram, labeled according to distance, length, and number of input ones.

6.4.1.1 Error-Correcting Capability of Convolutional Codes

In the study of block codes in Chapter 5, we saw that the error-correcting capability, t, represented the number of code symbol errors that could, with maximum likelihood decoding, be corrected in each block length of the code. However, when decoding convolutional codes, the error-correcting capability cannot

be stated so succinctly. With regard to Equation (6.11), we can say that the code can, with maximum likelihood decoding, correct t errors within a few constraint lengths, where "few" here means 3 to 5. The exact length depends on how the errors are distributed. For a particular code and error pattern, the length can be bounded using transfer function methods. A computer program for convolutional decoding with the Viterbi algorithm, called VITALG, is provided in Appendix E. The interested reader can use this tool for verifying the capability of Viterbi decoding of convolutional codes with various choices of code generators, code rates, constraint lengths, and path memory lengths.

6.4.2 Systematic and Nonsystematic Convolutional Codes

A *systematic* convolutional code is one in which the input k-tuple appears as part of the output branch word n-tuple associated with that k-tuple. Figure 6.16 shows a binary, rate $\frac{1}{2}$, $K = 3$ systematic encoder. For linear block codes, any nonsystematic code can be transformed into a systematic code with the same block distance properties. This is not the case for convolutional codes. The reason for this is that convolutional codes depend largely on *free distance*; making the convolutional code systematic, in general, *reduces* the maximum possible free distance for a given constraint length and rate.

Table 6.1 shows the maximum free distance for rate $\frac{1}{2}$ systematic and nonsystematic codes for $K = 2$ through 8. For large constraint lengths the results are even more widely separated [17].

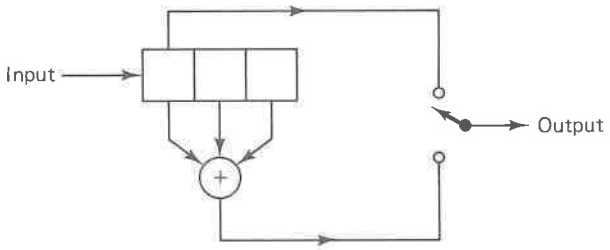


Figure 6.16 Systematic convolutional encoder, rate $\frac{1}{2}$, $K = 3$.

6.4.3 Catastrophic Error Propagation in Convolutional Codes

A *catastrophic error* is defined as an event whereby a finite number of code symbol errors cause an infinite number of decoded data bit errors. Massey and Sain [18] have derived a necessary and sufficient condition for convolutional codes to display catastrophic error propagation. For rate $1/n$ codes with register taps designated by polynomial generators, as described in Section 6.2.1, the condition for catastrophic error propagation is that the generators have a *common polynomial factor* (of degree at least one). For example, Figure 6.17a illustrates a rate $\frac{1}{2}$, $K = 3$ encoder with upper polynomial $\mathbf{g}_1(X)$ and lower polynomial $\mathbf{g}_2(X)$, as follows:

$$\begin{aligned} \mathbf{g}_1(X) &= 1 + X \\ \mathbf{g}_2(X) &= 1 + X^2 \end{aligned} \tag{6.16}$$

TABLE 6.1 Comparison of Systematic and Nonsystematic Free Distance, Rate $\frac{1}{2}$

Constraint length	Free distance systematic	Free distance nonsystematic
2	3	3
3	4	5
4	4	6
5	5	7
6	6	8
7	6	10
8	7	10

Source: A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill Book Company, New York, 1979, p. 251.

The generators $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$ have in common the polynomial factor, $1 + X$, since

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

Therefore, the encoder in Figure 6.17a can manifest *catastrophic error propagation*.

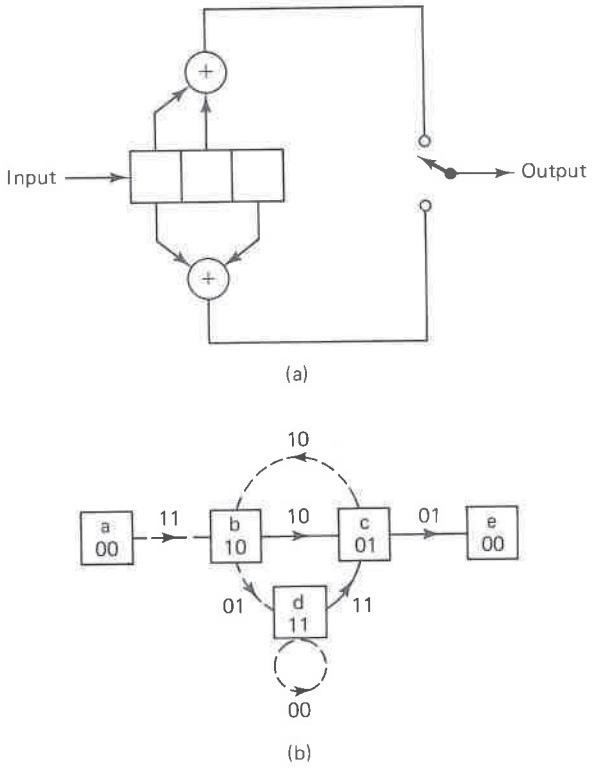


Figure 6.17 Encoder displaying catastrophic error propagation. (a) Encoder. (b) State diagram.

In terms of the state diagram for any-rate code, catastrophic errors can occur if, and only if, any closed-loop path in the diagram has zero weight (zero distance from the all-zeros path). To illustrate this, consider the example of Figure 6.17. The state diagram in Figure 6.17b is drawn with the state $a = 00$ node split into two nodes, a and e, as before. Assuming that the all-zeros path is the correct path, the incorrect path $a\ b\ d\ d\ \ldots\ d\ c\ e$ has exactly 6 ones, no matter how many times we go around the self-loop at node d. Thus for a BSC, for example, three channel errors may cause us to choose this incorrect path. An arbitrarily large number of errors (two plus the number of times the self-loop is traversed) can be made on such a path. We observe that for rate $1/n$ codes, if each adder in the encoder has an even number of connections, the self-loop corresponding to the all-ones data state will have zero weight, and consequently, *the code will be catastrophic.*

The only advantage of a systematic code, described earlier, is that it can never be catastrophic, since each closed loop must contain at least one branch generated by a nonzero input bit, and thus each closed loop must have a nonzero code symbol. However, it can be shown [19] that only a small fraction of non-systematic codes (excluding those where all adders have an even number of taps) are catastrophic.

6.4.4 Performance Bounds for Convolutional Codes

The probability of bit error, P_B, for a binary convolutional code using hard-decision decoding can be shown [8] to be upper bounded as follows:

$$P_B \leq \left.\frac{dT(D, N)}{dN}\right|_{N=1, D=2\sqrt{p(1-p)}} \tag{6.17}$$

where p is the probability of channel symbol error. For the example of Figure 6.3, $T(D, N)$ is obtained from $T(D, L, N)$ by setting $L = 1$ in Equation (6.15).

$$T(D, N) = \frac{D^5 N}{1 - 2DN} \tag{6.18}$$

and

$$\left.\frac{dT(D, N)}{dN}\right|_{N=1} = \frac{D^5}{(1 - 2D)^2} \tag{6.19}$$

Combining Equations (6.17) and (6.19), we can write

$$P_B \leq \frac{\{2[p(1 - p)]^{1/2}\}^5}{\{1 - 4[p(1 - p)]^{1/2}\}^2} \tag{6.20}$$

For coherent BPSK modulation over an additive white Gaussian noise (AWGN) channel, it can be shown [8] that the bit error probability is bounded by

$$P_B \leq Q\left(\sqrt{2d_f \frac{E_c}{N_0}}\right) \exp\left(d_f \frac{E_c}{N_0}\right) \frac{dT(D, N)}{dN}\bigg|_{N=1, D=\exp(-E_c/N_0)} \tag{6.21}$$

where

$E_c/N_0 = rE_b/N_0$

E_b/N_0 = ratio of information bit energy to noise power spectral density

E_c/N_0 = ratio of channel symbol energy to noise power spectral density

$r = k/n$ = rate of the code

and where $Q(x)$ is defined in Equations (2.42) and (2.43) and tabulated in Table B.1. Therefore, for the rate $\frac{1}{2}$ code with free distance $d_f = 5$, in conjunction with coherent BPSK and hard-decision decoding, we can write

$$\begin{aligned} P_B &\leq Q\left(\sqrt{\frac{5E_b}{N_0}}\right) \exp\left(\frac{5E_b}{2N_0}\right) \frac{\exp(-5E_b/2N_0)}{[1 - 2\exp(-E_b/2N_0)]^2} \\ &\leq \frac{Q(\sqrt{5E_b/N_0})}{[1 - 2\exp(-E_b/2N_0)]^2} \end{aligned} \tag{6.22}$$

6.4.5 Coding Gain

Coding gain is defined as the reduction, usually expressed in decibels, in the required E_b/N_0 to achieve a specified error probability of the coded system over an uncoded system with the same modulation and channel characteristics. Table 6.2 lists an upper bound on the coding gains, compared to uncoded coherent BPSK, for several maximum free distance convolutional codes with constraint lengths varying from 3 to 9 over a Gaussian channel with hard-decision decoding. The table illustrates that it is possible to achieve significant coding gain even with

TABLE 6.2 Coding Gain Upper Bounds for Some Convolutional Codes

Rate $\frac{1}{2}$ codes			Rate $\frac{1}{3}$ codes		
K	d_f	Upper bound (dB)	K	d_f	Upper bound (dB)
3	5	3.97	3	8	4.26
4	6	4.76	4	10	5.23
5	7	5.43	5	12	6.02
6	8	6.00	6	13	6.37
7	10	6.99	7	15	6.99
8	10	6.99	8	16	7.27
9	12	7.78	9	18	7.78

Source: V. K. Bhargava, D. Haccoun, R. Matyas, and P. Nuspl, *Digital Communications by Satellite*, John Wiley & Sons, Inc., New York, 1981.

a simple convolutional code. The actual coding gain will vary with the required bit error probability [20].

Table 6.3 lists the measured coding gains, compared to uncoded coherent BPSK, achieved with hardware implementation or computer simulation over a Gaussian channel with soft-decision decoding [21]. The uncoded E_b/N_0 is given in the leftmost column. From Table 6.3 we can see that coding gain increases as the bit error probability is decreased. However, the coding gain cannot increase indefinitely; it has an upper bound as shown in the table. This bound in decibels can be shown [21] to be

$$\text{coding gain} \leq 10 \log_{10} (rd_f) \tag{6.23}$$

where r is the code rate and d_f is the free distance. Examination of Table 6.3 also reveals that at $P_B = 10^{-7}$, for code rates of $\frac{1}{2}$ and $\frac{2}{3}$, the weaker codes tend to be closer to the upper bound than are the more powerful codes.

Typically, Viterbi decoding is used over binary input channels with either hard or 3-bit soft quantized outputs. The constraint lengths vary between 3 and 9, the code rate is rarely smaller than $\frac{1}{3}$, and the path memory is usually a few constraint lengths [12]. The path memory refers to the depth of the input bit history stored by the decoder. From the Viterbi decoding example in Section 6.3.4, one might question the notion of a fixed path memory. It seems from the example that the decoding of a branch word, at any arbitrary node, can take place as soon as there is only a single surviving branch at that node. That is true; however, to actually implement the decoder in this way would entail an extensive amount of processing to continually check when the branch word can be decoded. Instead, *a fixed delay is provided*, after which the branch word is decoded. It has been shown [12, 22] that a fixed amount of path history, namely 4 or 5 times the constraint length, is sufficient to limit the degradation from the optimum decoder performance to about 0.1 dB for the BSC and Gaussian channels. Typical error performance curves are shown in Figure 6.18 for rate $\frac{1}{2}$ codes using coherent BPSK over a soft (8-level) quantized channel, with Viterbi decoding, and a 32-bit path memory. Also plotted are the transfer function bounds for infinitely fine quantized received data [12]. Figure 6.19 gives the simulation results for Viterbi decoding with hard decision quantization [12]. Notice that each increment in constraint

TABLE 6.3 Basic Coding Gain (dB) for Soft Decision Viterbi Decoding

Uncoded E_b/N_0 (dB)	Code rate		$\frac{1}{3}$		$\frac{1}{2}$			$\frac{2}{3}$		$\frac{3}{4}$	
	P_B	K	7	8	5	6	7	6	8	6	9
6.8	10^{-3}		4.2	4.4	3.3	3.5	3.8	2.9	3.1	2.6	2.6
9.6	10^{-5}		5.7	5.9	4.3	4.6	5.1	4.2	4.6	3.6	4.2
11.3	10^{-7}		6.2	6.5	4.9	5.3	5.8	4.7	5.2	3.9	4.8
Upper bound			7.0	7.3	5.4	6.0	7.0	5.2	6.7	4.8	5.7

Source: I. M. Jacobs, "Practical Applications of Coding," *IEEE Trans. Inf. Theory*, vol. IT20, May 1974, pp. 305–310.

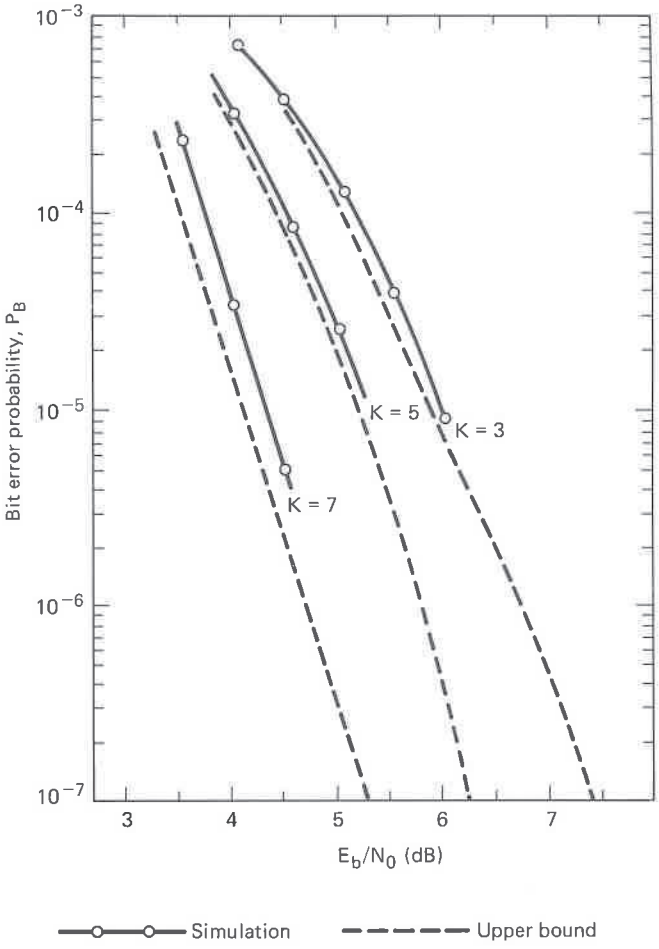


Figure 6.18 Bit error probability versus E_b/N_0 for rate $\frac{1}{2}$ codes using coherent BPSK over a soft quantized channel, Viterbi decoding, and a 32-bit path memory. (Reprinted with permission from J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.*, vol. COM19, no. 5, October 1971, Fig. 5, p. 84. © 1971 IEEE.)

length improves the required E_b/N_0 by a factor of approximately 0.5 dB at $P_B = 10^{-5}$. Also, as expected, the 3-bit soft decisions of the channel output result in approximately a 2-dB gain over the hard quantized BSC.

6.4.6 Best Known Convolutional Codes

The connection vectors or polynomial generators of a convolutional code are usually selected based on the code's free distance properties. The first criterion is to select a code that does not have catastrophic error propagation and that has the maximum free distance for the given rate and constraint length. Then the number of paths at the free distance d_f, or the number of data bit errors the paths represent, should be minimized. The selection procedure can be further refined by considering the number of paths or bit errors at $d_f + 1$, at $d_f + 2$, and so on, until only one code or class of codes remains. A list of the best known codes of rate $\frac{1}{2}$, $K = 3$ to 9, and rate $\frac{1}{3}$, $K = 3$ to 8, based on this criterion was compiled by Odenwalder [3, 23] and is given in Table 6.4. The connection vectors in this

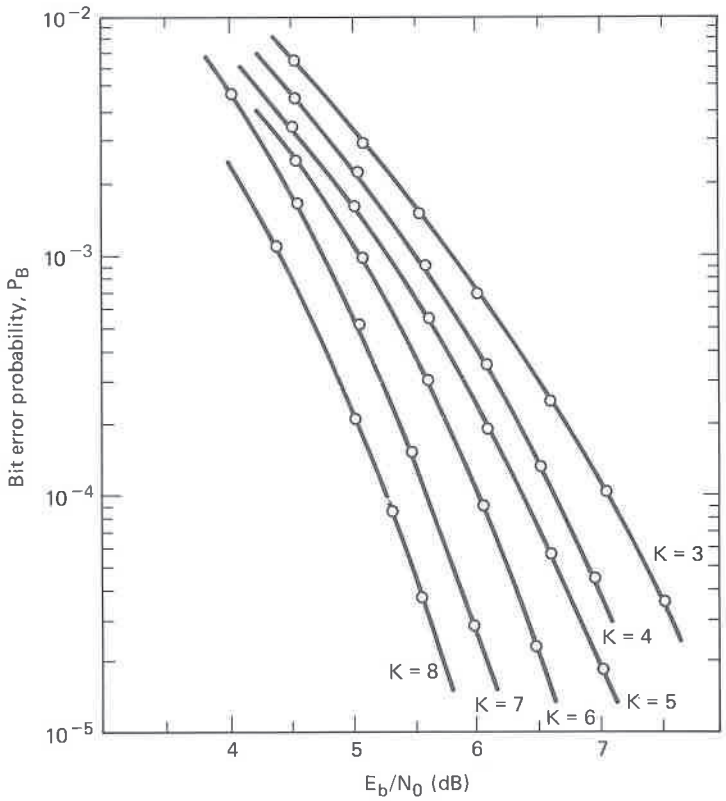


Figure 6.19 Bit error probability versus E_b/N_0 for rate $\frac{1}{2}$ codes using coherent BPSK over a BSC, Viterbi decoding, and a 32-bit path memory. (Reprinted with permission from J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.*, vol. COM19, no. 5, October 1971, Fig. 7, p. 84. © 1971 IEEE.)

table represent the presence or absence (1 or 0) of a tap connection on the corresponding stage of the convolutional encoder. The leftmost term corresponds to the leftmost stage of the encoder register, and the rightmost term corresponds to the rightmost stage, following the notation established in Figure 6.3. It is interesting to note that these connections can be inverted (leftmost and rightmost can be interchanged in the above description). Under the condition of Viterbi decoding, the inverted connections give rise to codes with identical distance properties, and hence identical performance, as those in Table 6.4.

6.4.7 Convolutional Code Rate Trade-Off

6.4.7.1 Performance with Coherent PSK Signaling

The error-correcting capability of a coding scheme increases as the number of channel symbols n per information bit k increases or the rate, k/n, decreases. However, the channel bandwidth and the decoder complexity both increase with n. The advantage of lower code rates when using convolutional codes with co-

TABLE 6.4 Optimum Short Constraint Length Convolutional Codes (Rate $\frac{1}{2}$ and Rate $\frac{1}{3}$)

Rate	Constraint length	Free distance	Code vector
$\frac{1}{2}$	3	5	111 101
$\frac{1}{2}$	4	6	1111 1011
$\frac{1}{2}$	5	7	10111 11001
$\frac{1}{2}$	6	8	101111 110101
$\frac{1}{2}$	7	10	1001111 1101101
$\frac{1}{2}$	8	10	10011111 11100101
$\frac{1}{2}$	9	12	110101111 100011101
$\frac{1}{3}$	3	8	111 111 101
$\frac{1}{3}$	4	10	1111 1011 1101
$\frac{1}{3}$	5	12	11111 11011 10101
$\frac{1}{3}$	6	13	101111 110101 111001
$\frac{1}{3}$	7	15	1001111 1010111 1101101
$\frac{1}{3}$	8	16	11101111 10011011 10101001

Source: J. P. Odenwalder, *Error Control Coding Handbook*, Linkabit Corp., San Diego, Calif., July 15, 1976.

herent PSK, is that the required E_b/N_0 is decreased (for a large range of code rates), permitting the transmission of higher data rates for a given amount of power, or permitting reduced power for a given data rate. Simulation studies have shown [16, 22] that for a fixed constraint length, a decrease in the code rate from $\frac{1}{2}$ to $\frac{1}{3}$ results in a reduction of the required E_b/N_0 of roughly 0.4 dB. However, the corresponding increase in decoder complexity is about 17%. For smaller values of code rate, the improvement in performance relative to the increased decoding complexity diminishes rapidly [22]. Eventually, a point is reached where further decrease in code rate is characterized by a reduction in coding gain.

6.4.7.2 Performance with Noncoherent Orthogonal Signaling

In contrast to PSK, there is an optimum code rate of about $\frac{1}{2}$ for noncoherent orthogonal signaling. Error performance at rates of $\frac{1}{3}$, $\frac{2}{3}$, and $\frac{3}{4}$ are each worse than those for rate $\frac{1}{2}$. For a fixed constraint length, the rate $\frac{1}{3}$, $\frac{2}{3}$, and $\frac{3}{4}$ codes typically degrade by about 0.25, 0.5, and 0.3 dB, respectively, relative to the rate $\frac{1}{2}$ performance [16].

6.5 OTHER CONVOLUTIONAL DECODING ALGORITHMS

6.5.1 Sequential Decoding

Prior to the discovery of an optimum algorithm by Viterbi, other algorithms had been proposed for decoding convolutional codes. The earliest was the *sequential decoding algorithm*, originally proposed by Wozencraft [24, 25] and subsequently modified by Fano [2]. A sequential decoder works by generating hypotheses about the transmitted codeword sequence; it computes a metric between these hypotheses and the received signal. It goes forward as long as the metric indicates that its choices are likely; otherwise, it goes backward, changing hypotheses until, through a systematic trial-and-error search, it finds a likely hypothesis. Sequential decoders can be implemented to work with hard or soft decisions, but soft decisions are usually avoided because they greatly increase the amount of the required storage and the complexity of the computations.

Consider that using the encoder shown in Figure 6.3, a sequence $\mathbf{m} = 1\ 1\ 0\ 1\ 1$ is encoded into the codeword sequence $\mathbf{U} = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1$, as shown in Example 6.1. Assume that the received sequence $\mathbf{Z}$ is, in fact, a *correct* rendition of $\mathbf{U}$. The decoder has available a replica of the encoder code tree, shown in Figure 6.6, and can use the received sequence $\mathbf{Z}$ to penetrate the tree. The decoder starts at the time t_1 node of the tree and generates both paths leaving that node. The decoder follows that path which agrees with the received n code symbols. At the next level in the tree, the decoder again generates both paths leaving that node, and follows the path agreeing with the second group of n code symbols. Proceeding in this manner, the decoder quickly penetrates the tree.

Suppose, however, that the received sequence $\mathbf{Z}$ is a *corrupted* version of $\mathbf{U}$. The decoder starts at the time t_1 node of the code tree and generates both paths leading from that node. If the received n code symbols coincide with one of the generated paths, the decoder follows that path. If there is not agreement, the decoder follows the *most likely path* but keeps a cumulative count on the number of disagreements between the received symbols and the branch words on the path being followed. If two branches appear equally likely, the receiver uses an arbitrary rule, such as following the zero input path. At each new level in the tree, the decoder generates new branches and compares them with the next set of n received code symbols. The search continues to penetrate the tree along the most likely path and maintains the cumulative disagreement count.

If the disagreement count exceeds a certain number (which may increase as

we penetrate the tree), the decoder decides that it is on an incorrect path, backs out of the path, and tries another. The decoder keeps track of the discarded pathways to avoid repeating any path excursions. For example, assume that the encoder in Figure 6.3 is used to encode the message sequence $\mathbf{m} = 1\ 1\ 0\ 1\ 1$ into the codeword sequence $\mathbf{U}$ as shown in Example 6.1. Suppose that the fourth and seventh bits of the transmitted sequence $\mathbf{U}$ are received in error, such that:

Time:		t_1	t_2	t_3	t_4	t_5
Message sequence:	$\mathbf{m} =$	1	1	0	1	1
Transmitted sequence:	$\mathbf{U} =$	1 1	0 1	0 1	0 0	0 1
Received sequence:	$\mathbf{Z} =$	1 1	0 0	0 1	1 0	0 1

Let us follow the decoder path trajectory with the aid of Figure 6.20. Assume that a cumulative path disagreement count of 3 is the criterion for backing up and trying an alternative path. On Figure 6.20 the numbers along the path trajectory represent the current disagreement count.

1. At time t_1 we receive symbols 11 and compare them with the branch words leaving the first node.
2. The most likely branch is the one with branch word 11 (corresponding to an input bit one or downward branching), so the decoder decides that input bit one is the correct decoding, and moves to the next level.
3. At time t_2, the decoder receives symbols 00 and compares them with the available branch words 10 and 01 at this second level.
4. There is no "best" path, so the decoder arbitrarily takes the input bit zero (or branch word 10) path, and the disagreement count registers a disagreement of 1.
5. At time t_3, the decoder receives symbols 01 and compares them with the available branch words 11 and 00 at this third level.
6. Again, there is no best path, so the decoder arbitrarily takes the input zero (or branch word 11) path, and the disagreement count is increased to 2.
7. At time t_4, the decoder receives symbols 10 and compares them with the available branch words 00 and 11 at this fourth level.
8. Again, there is no best path, so the decoder takes the input bit zero (or branch word 00) path, and the disagreement count is increased to 3.
9. But a disagreement count of 3 is the turnaround criterion, so the decoder "backs out" and tries the alternative path. The disagreement counter is reset to 2.
10. The alternative path is the input bit one (or branch word 11) path at the t_4

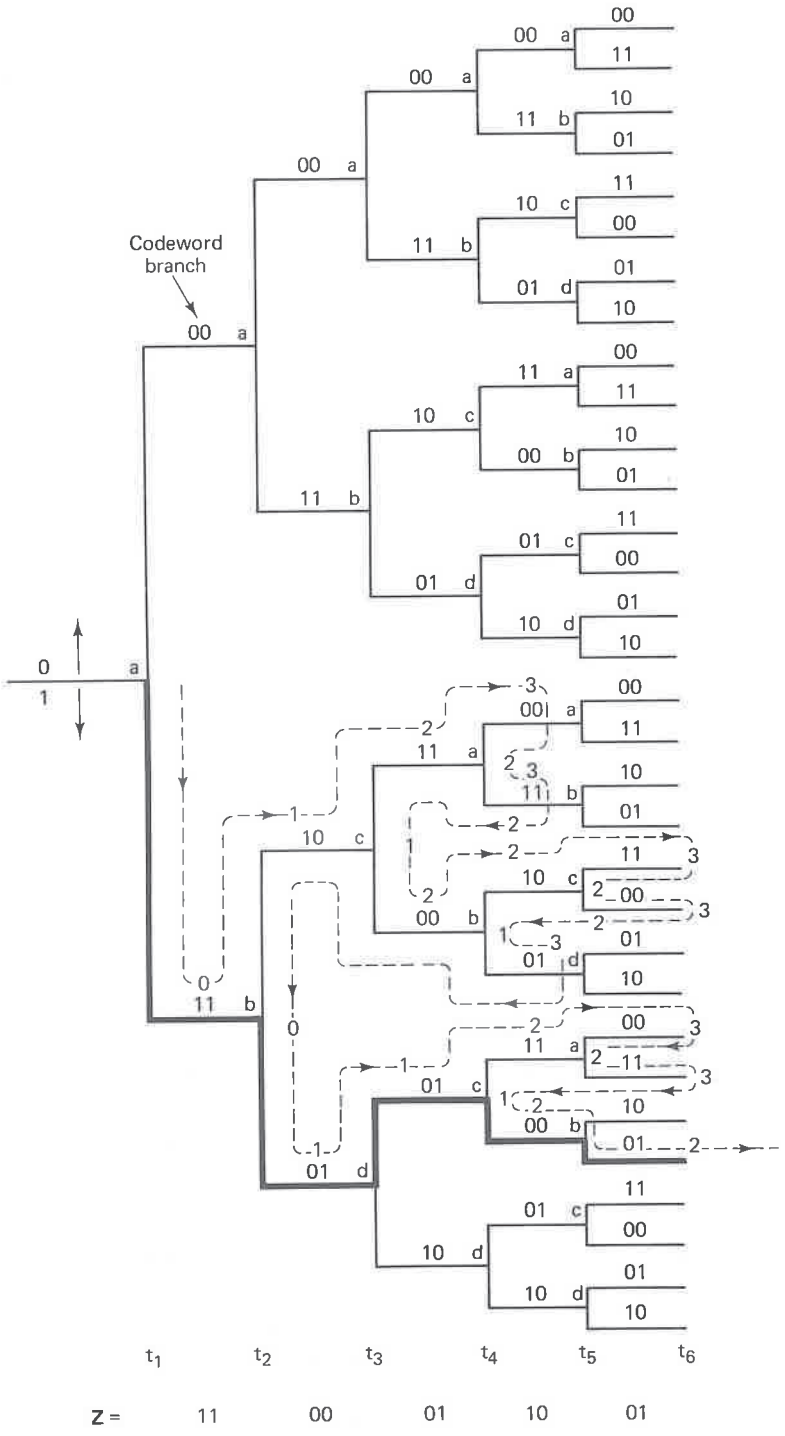


Figure 6.20 Sequential decoding example.

level. The decoder tries this, but compared to the received symbols 10, there is still a disagreement of 1, and the counter is reset to 3.

11. But, 3 being the turnaround criterion, the decoder backs out of this path, and the counter is reset to 2. All of the alternatives have now been traversed at this t_4 level, so the decoder returns to the node at t_3, and resets the counter to 1.
12. At the t_3 node, the decoder compares the symbols received at time t_3, namely 01, with the untried 00 path. There is a disagreement of 1, and the counter is increased to 2.
13. At the t_4 node, the decoder follows the branch word 10 that matches its t_4 code symbols of 10. The counter remains unchanged at 2.
14. At the t_5 node, there is no best path, so the decoder follows the upper branch, as is the rule, and the counter is increased to 3.
15. At this count, the decoder backs up, resets the counter to 2, and tries the alternative path at node t_5. Since the alternate branch word is 00, there is a disagreement of 1 with the received code symbols 01 at time t_5, and the counter is again increased to 3.
16. The decoder backs out of this path, and the counter is reset to 2. All of the alternatives have now been traversed at this t_5 level, so the decoder returns to the node at t_4 and resets the counter to 1.
17. The decoder tries the alternative path at t_4, which raises the metric to 3 since there is a disagreement in two positions of the branch word. This time the decoder must back up all the way to the time t_2 node because all of the other paths at higher levels have been tried. The counter is now decremented to zero.
18. At the t_2 node, the decoder now follows the branch word 01, and because there is a disagreement of 1 with the received code symbols 00 at time t_2, the counter is increased to 1.

The decoder continues in this way. As shown in Figure 6.20, the final path, which has not increased the counter to its turnaround criterion, yields the correctly decoded message sequence, 1 1 0 1 1. Sequential decoding can be viewed as a trial-and-error technique for searching out the correct path in the code tree. It performs the search in a sequential manner, always operating on just a single path at a time. If an incorrect decision is made, subsequent extensions of the path will be wrong. The decoder can eventually recognize its error by monitoring the path metric. The algorithm is similar to the case of an automobile traveler following a road map. As long as the traveler recognizes that the passing landmarks correspond to those on the map, he continues on the path. When he notices strange landmarks (an increase in his dissimilarity metric) the traveler eventually assumes that he is on an incorrect road, and he backs up to a point where he can now recognize the landmarks (his metric returns to an acceptable range). He then tries an alternative road.

6.5.2 Comparisons and Limitations of Viterbi and Sequential Decoding

The major drawback of the Viterbi algorithm is that while error probability decreases exponentially with constraint length, the number of code states, and consequently decoder complexity, *grows exponentially with constraint length*. On the other hand, the computational complexity of the Viterbi algorithm is independent of channel characteristics (compared to hard-decision decoding, soft-decision decoding requires only a trivial increase in the number of computations). Sequential decoding achieves asymptotically the same error probability as maximum likelihood decoding but without searching all possible states. In fact, with sequential decoding the number of states searched is essentially *independent of constraint length*, thus making it possible to use very large ($K = 41$) constraint

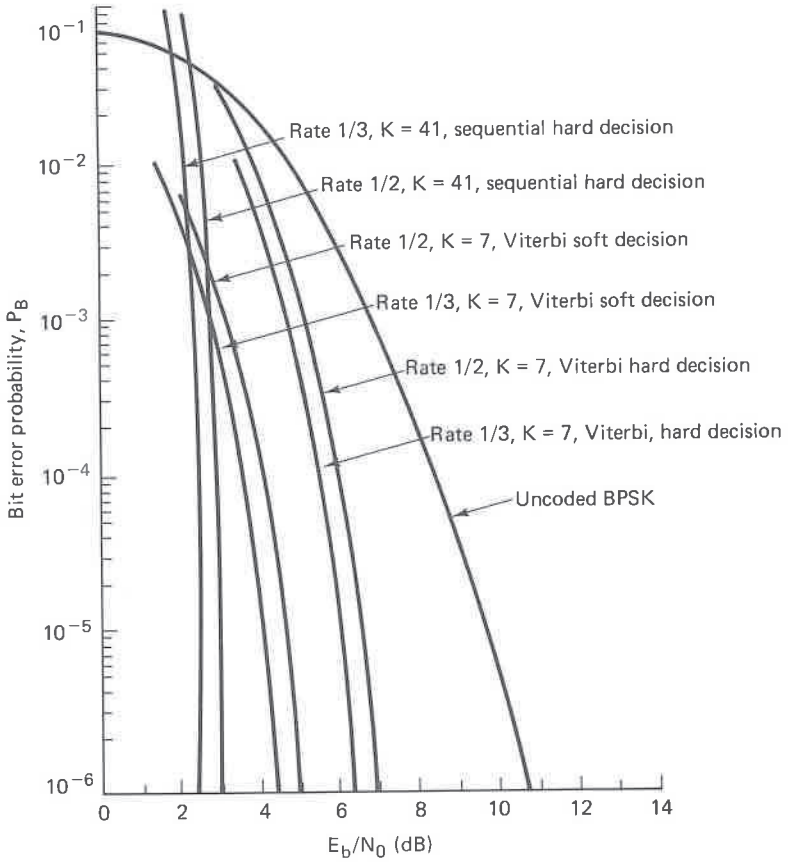


Figure 6.21 Bit error performance for various Viterbi and sequential decoding schemes using coherent BPSK over an AWGN channel. (Reprinted with permission from J. K. Omura and B. K. Levitt, "Coded Error Probability Evaluation for Antijam Communication Systems," *IEEE Trans. Commun.*, vol. COM30, no. 5, May 1982, Fig. 4, p. 900. © 1982 IEEE.)

lengths. This is an important factor in providing such low error probabilities. The major drawback of sequential decoding is that the number of state metrics searched is a random variable. For sequential decoding, the expected number of poor hypotheses and backward searches is a function of the channel SNR. With a low SNR, more hypotheses must be tried than with a high SNR. Because of this variability in computational load, buffers must be provided to store the arriving sequences. Under low SNR, the received sequences must be buffered while the decoder is laboring to find a likely hypothesis. If the average symbol arrival rate exceeds the average symbol decode rate, the buffer will overflow, no matter how large it is, causing a loss of data. The sequential decoder typically puts out error-free data until the buffer overflows, at which time the decoder has to go through a recovery procedure. The buffer overflow threshold is a very sensitive function of SNR. Therefore, an important part of a sequential decoder specification is the *probability of buffer overflow*.

In Figure 6.21, some typical P_B versus E_b/N_0 curves for these two popular solutions to the convolutional decoding problem, Viterbi decoding and sequential decoding, illustrate their comparative performance using coherent BPSK over an AWGN channel. The curves compare Viterbi decoding (rates $\frac{1}{2}$ and $\frac{1}{3}$ hard decision, $K = 7$) versus Viterbi decoding (rates $\frac{1}{2}$ and $\frac{1}{3}$ soft decision, $K = 7$) versus sequential decoding (rates $\frac{1}{2}$ and $\frac{1}{3}$ hard decision, $K = 41$). One can see from Figure 6.21 that coding gains of approximately 8 dB at $P_B = 10^{-6}$ can be achieved with sequential decoders. Since the work of Shannon [26] foretold the potential of approximately 11 dB of coding gain compared to uncoded BPSK, it appears that the major portion of what is theoretically possible can already be accomplished.

6.5.3 Feedback Decoding

A *feedback decoder* makes a hard decision on the data bit at stage j based on metrics computed from stages $j, j + 1, \ldots, j + m$, where m is a preselected positive integer. *Look-ahead length*, L, is defined as $L = m + 1$, the number of received code symbols, expressed in terms of the corresponding number of encoder input bits that are used to decode an information bit. The decision of whether the data bit is zero or one depends on which branch the minimum Hamming distance path traverses in the *look-ahead window* from stage j to stage $j + m$. The detailed operation is best understood in terms of a specific example. Let us consider the use of a feedback decoder for the rate $\frac{1}{2}$ convolutional code shown in Figure 6.3. Figure 6.22 illustrates the tree diagram and the operation of the feedback decoder for $L = 3$. That is, in decoding the bit at branch j, the decoder considers the paths at branches $j, j + 1$, and $j + 2$.

Beginning with the first branch, the decoder computes 2^L or eight cumulative Hamming path metrics and decides that the bit for the first branch is zero if the minimum distance path is contained in the upper part of the tree, and decides one if the minimum distance path is in the lower part of the tree. Assume that the received sequence is $\mathbf{Z} = 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$. We now examine the eight paths from time t_1 through time t_3 in the block marked A in Figure 6.22, and compute

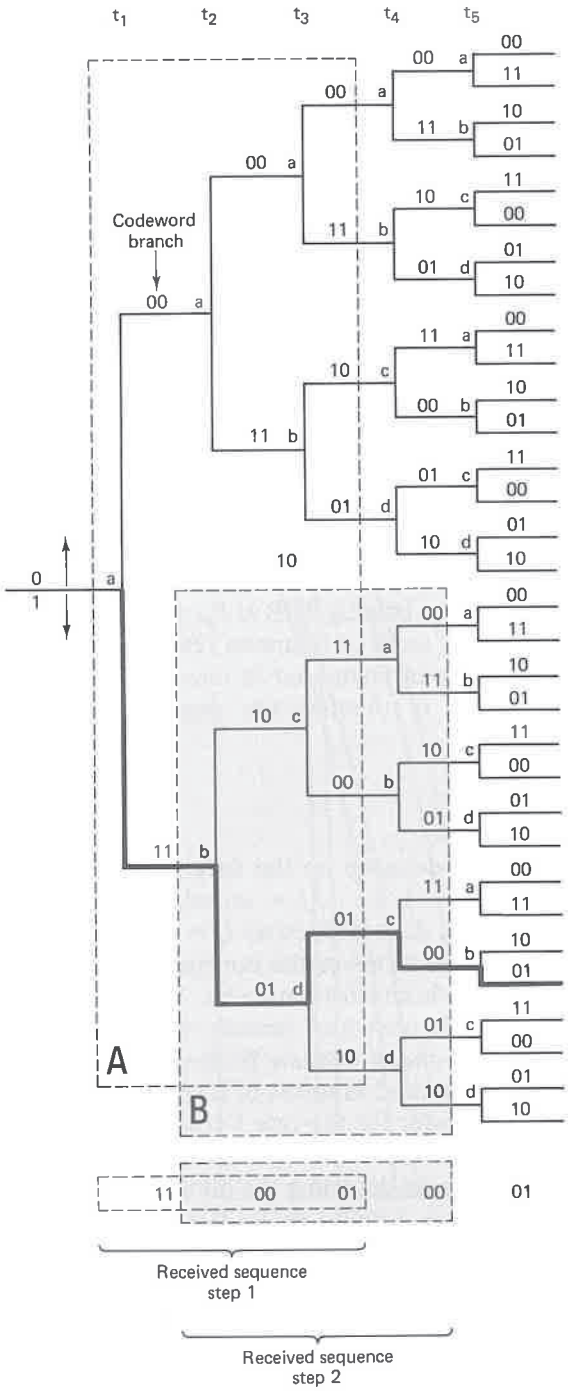


Figure 6.22 Feedback decoding example.

metrics comparing these eight paths with the first six received code symbols (three branches deep times two symbols per branch). Listing the Hamming cumulative path metrics (starting from the top path), they are:

Upper-half metrics: 3, 3, 6, 4

Lower-half metrics: 2, 2, 1, 3

We see that the minimum metric is contained in the lower part of the tree. Therefore, the first decoded bit is one (characterized by a downward movement on the tree). The next step is to extend the lower part of the tree (the part that survived) one stage deeper, and again compute eight metrics, this time from t_2 through t_4. Having decoded the first two code symbols, we now slide over two code symbols to the right and again compute the path metrics for six code symbols. This takes place in the block marked B in Figure 6.22. Again, listing the metrics from top path to bottom path, they are:

Upper-half metrics: 2, 4, 3, 3

Lower-half metrics: 3, 1, 4, 4

For the assumed received sequence, the minimum metric is found in the lower half of block B. Therefore, the second decoded bit is one.

The same procedure continues until the entire message is decoded. The decoder is called a *feedback decoder* because the detection decisions are *fed back* to the decoder in determining the subset of code paths that are to be considered next. On the BSC, the feedback decoder can perform nearly as well as the Viterbi decoder [17] in that it can correct all the more probable error patterns, namely all those of weight $(d_f - 1)/2$ or less, where d_f is the free distance of the code. An important design parameter for feedback convolutional decoders is L, the look-ahead length. Increasing L increases the coding gain but also increases the decoder implementation complexity.

6.6 INTERLEAVING AND CONCATENATED CODES

Throughout this chapter and Chapter 5 we have assumed that the channel is *memoryless,* since we have considered codes that are designed to combat random independent errors. A channel that has *memory* is one that exhibits mutually dependent signal transmission impairments. An example of such a channel is a fading channel, particularly when the fading varies slowly compared to one symbol time. Another type of impairment, called *multipath,* involves signal arrivals at the receiver over two or more paths of different lengths. The effect is that the signals *arrive out of phase* with each other, and the cumulative received signal is distorted. High-frequency (HF) and tropospheric propagation radio channels suffer from such phenomena. Also, some channels suffer from switching noise and other burst noise (e.g., telephone channels or channels disturbed by pulse jamming). All of these time-correlated impairments result in statistical dependence

among successive symbol transmissions. That is, the disturbances tend to cause errors that occur in bursts, instead of as isolated events.

Under the assumption that the channel has memory, the errors no longer can be characterized as single randomly distributed bit errors whose occurrence is independent from bit to bit. Most block or convolutional codes are designed to combat random independent errors. The result of a channel having memory on such coded signals is to cause degradation in error performance. Coding techniques for channels with memory have been proposed [27, 28], but the greatest problem with such coding is the difficulty in obtaining accurate models of the often time-varying statistics of such channels. One technique, which only requires a knowledge of the *duration or span* of the channel memory, *not* its exact statistical characterization, is the use of time diversity or *interleaving*.

Interleaving the coded message before transmission and deinterleaving after reception causes bursts of channel errors to be spread out in time and thus to be handled by the decoder as if they were random errors. Since, in all practical cases, the channel memory decreases with time separation, the idea behind interleaving is to separate the codeword symbols in time. The intervening times are similarly filled by the symbols of other codewords. Separating the symbols in time effectively transforms a channel with memory to a *memoryless* one, and thereby enables the random-error-correcting codes to be useful in a burst-noise channel.

The interleaver shuffles the code symbols over a span of several block lengths (for block codes) or several constraint lengths (for convolutional codes). The span required is determined by the burst duration. The details of the bit redistribution pattern must be known to the receiver in order for the symbol stream to be deinterleaved before being decoded. Figure 6.23 illustrates a simple interleaving example. In Figure 6.23a we see seven uninterleaved codewords, A through G. Each codeword is comprised of seven code symbols. Let us assume that the code has a single-error-correcting capability within each seven-symbol sequence. If the memory span of the channel is one codeword in duration, such a seven-symbol-time noise burst could destroy the information contained in one or two codewords. However, suppose that, after having encoded the data, the code symbols were then *interleaved* or shuffled, as shown in Figure 6.23b. That is, each code symbol of each codeword is separated from its preinterleaved neighbors by a span of seven symbol times. The interleaved stream is then used to modulate a waveform that is transmitted over the channel. A contiguous channel noise burst occupying seven symbol times is seen in Figure 6.23b, to affect one code symbol from each of the original seven codewords. Upon reception, the stream is first deinterleaved so that it resembles the original coded sequence in Figure 6.23a. Then the stream is decoded. Since each codeword possesses a single-error-correcting capability, the burst noise has no degrading effect on the final sequence.

Interleaving techniques have proven useful for all the convolutional and block codes described here and in Chapter 5. Two types of interleavers are commonly used, *block interleavers* and *convolutional interleavers*. They are each described below.

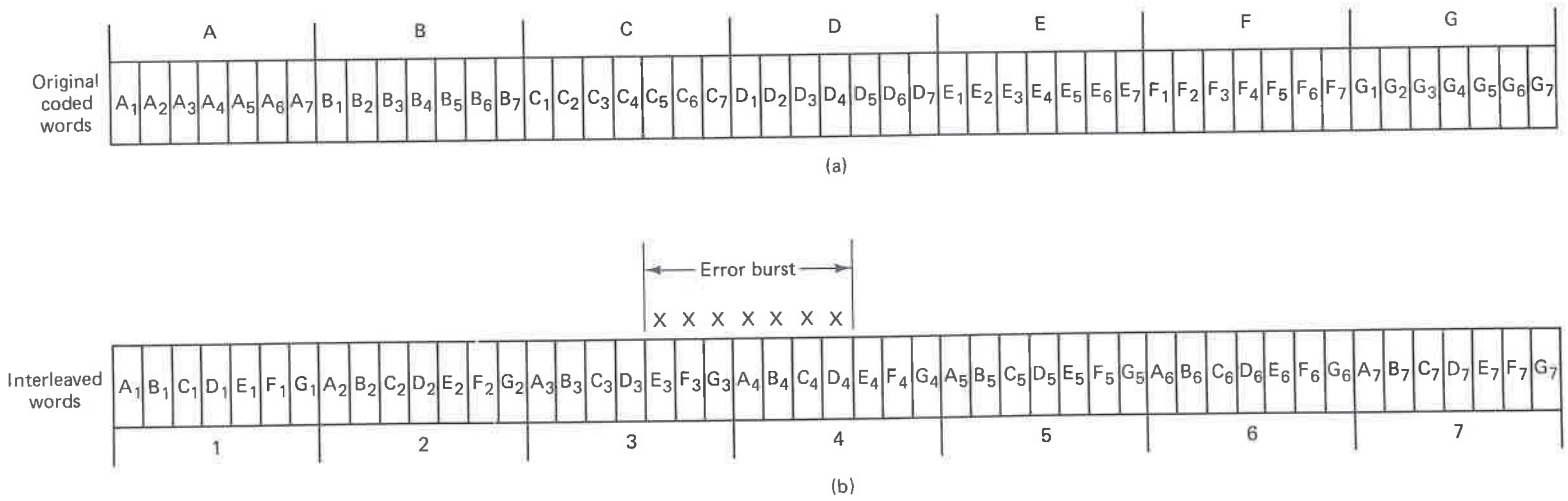


Figure 6.23 Interleaving example. (a) Original uninterleaved codewords, each comprised of seven code symbols. (b) Interleaved code symbols.

6.6.1 Block Interleaving

A block interleaver accepts the coded symbols in blocks from the encoder, permutes the symbols, and then feeds the rearranged symbols to the modulator. The usual permutation of the block is accomplished by *filling the columns* of an M-row-by N-column ($M \times N$) array with the encoded sequence. After the array is completely filled, the symbols are then fed to the modulator *one row at a time* and transmitted over the channel. At the receiver, the deinterleaver performs the inverse operation; it accepts the symbols from the demodulator, deinterleaves them, and feeds them to the decoder. Symbols are entered into the deinterleaver array by rows, and removed by columns. Figure 6.24a illustrates an example of an interleaver with $M = 4$ rows and $N = 6$ columns. The entries in the array illustrate the order in which the 24 code symbols are placed into the interleaver. The output sequence to the transmitter consists of code symbols removed from the array by rows, as shown in the figure. The most important characteristics of such a block interleaver are as follows:

1. Any burst of less than N contiguous channel symbol errors results in isolated errors at the deinterleaver output that are separated from each other by at least M symbols.
2. Any bN burst of errors, where $b > 1$, results in output bursts from the deinterleaver of no more than $\lceil b \rceil$ symbol errors. Each output burst is separated from the other bursts by no less than $M - \lfloor b \rfloor$ symbols. The notation $\lceil x \rceil$ means the smallest integer no less than x, and $\lfloor x \rfloor$ means the largest integer no greater than x.
3. A periodic sequence of single errors spaced N symbols apart results in a single burst of errors of length M at the deinterleaver output.
4. The interleaver/deinterleaver end-to-end delay is approximately $2MN$ symbol times. To be precise, only $M(N - 1) + 1$ memory cells need to be filled before transmission can begin (as soon as the first symbol of the last column of the $M \times N$ array is filled). A corresponding number needs to be filled at the receiver before decoding begins. Thus the minimum end-to-end delay is $(2MN - 2M + 2)$ symbol times, not including any channel propagation delay.
5. The memory requirement is MN symbols for each location (interleaver and deinterleaver). However, since the $M \times N$ array needs to be (mostly) filled before it can be read out, a memory of $2MN$ symbols is generally implemented at each location to allow the emptying of one $M \times N$ array while the other is being filled, and vice versa.

Example 6.3 Interleaver Characteristics

Using the $M = 4$, $N = 6$ interleaver structure of Figure 6.24a, verify each of the block interleaver characteristics described above.

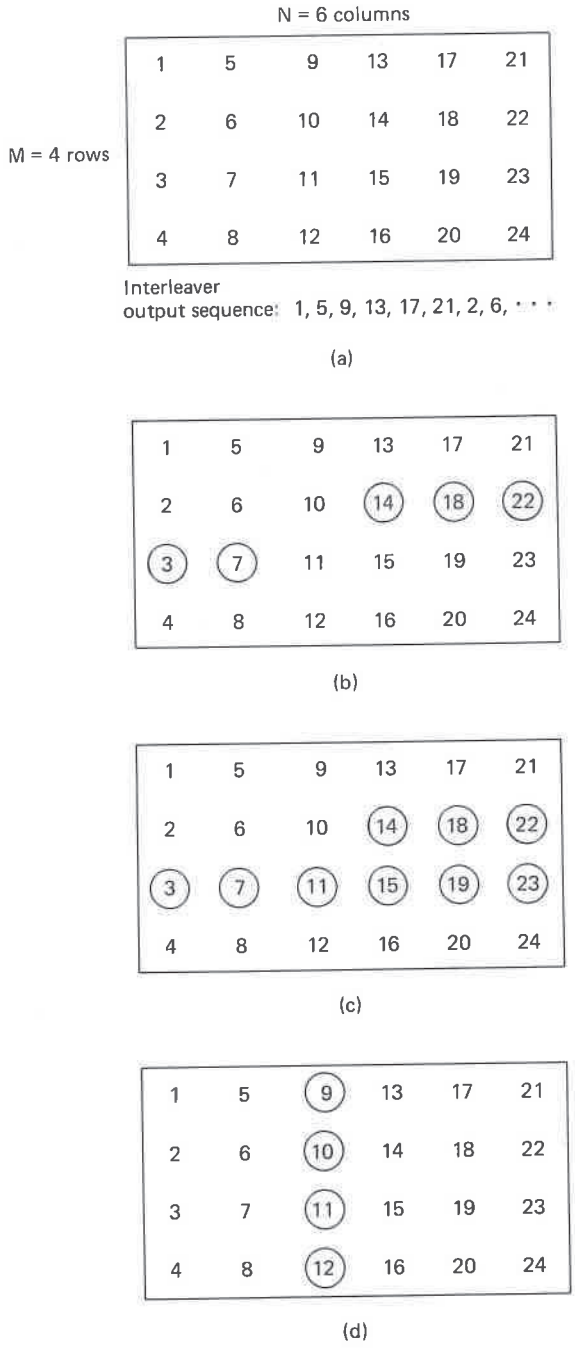


Figure 6.24 Block interleaver example. (a) $M \times N$ block interleaver. (b) Five-symbol error burst. (c) Nine-symbol error burst. (d) Periodic single-error sequence spaced $N = 6$ symbols apart.

Solution

1. Let there be a noise burst of five symbol times, such that the symbols shown encircled in Figure 6.24b experience errors in transmission. After deinterleaving at the receiver, the sequence is

   1 2 ③ 4 5 6 ⑦ 8 9 10 11 12
   13 ⑭ 15 16 17 ⑱ 19 20 21 ㉒ 23 24

   where the encircled symbols are in error. It is seen that the smallest separation between symbols in error is $M = 4$.

2. Let $b = 1.5$ so that $bN = 9$. Figure 6.24c illustrates an example of a nine-symbol error burst. After deinterleaving at the receiver, the sequence is

   1 2 ③ 4 5 6 ⑦ 8 9 10 ⑪ 12
   13 ⑭ ⑮ 16 17 ⑱ ⑲ 20 21 ㉒ ㉓ 24

   Again, the encircled symbols are in error. It is seen that the bursts consist of no more than $\lceil 1.5 \rceil = 2$ contiguous symbols and that they are separated by at least $M - \lfloor 1.5 \rfloor = 4 - 1 = 3$ symbols.

3. Figure 6.24d illustrates a sequence of single errors spaced by $N = 6$ symbols apart. After deinterleaving at the receiver, the sequence is

   1 2 3 4 5 6 7 8 ⑨ ⑩ ⑪ ⑫
   13 14 15 16 17 18 19 20 21 22 23 24

   It is seen that the deinterleaved sequence has a single error burst of length $M = 4$ symbols.

4. End-to-end delay: The minimum end-to-end delay due to the interleaver and deinterleaver is $(2MN - 2M + 2) = 42$ symbol times.

5. Memory requirement: The interleaver and the deinterleaver arrays are each of size $M \times N$. Therefore, storage for $MN = 24$ symbols is required at each end of the channel. As mentioned earlier, storage for $2MN = 48$ symbols would generally be implemented.

Typically, for use with a single-error-correcting code the interleaver parameters are selected such that the number of columns N overbounds the *expected burst length*. The choice of the number of rows M is dependent on the coding scheme used. For block codes, M should be larger than the code block length, while for convolutional codes, M should be larger than the constraint length. Thus a burst of length N can cause at most a single error in any block codeword; similarly, with convolutional codes, there will be at most a single error in any decoding constraint length. For t-error-correcting codes, the choice of N need only overbound the expected burst length divided by t.

6.6.2 Convolutional Interleaving

Convolutional interleavers have been proposed by Ramsey [29] and Forney [30]. The structure proposed by Forney appears in Figure 6.25. The code symbols are

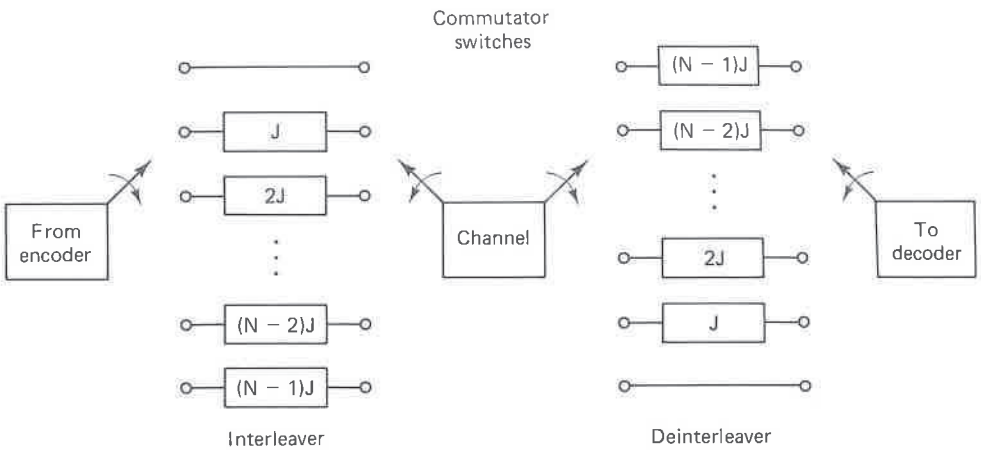


Figure 6.25 Shift register implementation of a convolutional interleaver/deinterleaver.

sequentially shifted into the bank of N registers; each successive register provides J symbols more storage than did the preceding one. The zeroth register provides no storage (the symbol is transmitted immediately). With each new code symbol the commutator switches to a new register, and the new code symbol is shifted in while the oldest code symbol in that register is shifted out to the modulator/transmitter. After the $(N - 1)$th register, the commutator returns to the zeroth register and starts again. The deinterleaver performs the inverse operation, and the input and output commutators for both interleaving and deinterleaving must be synchronized.

Figure 6.26 illustrates an example of a simple convolutional four-register ($J = 1$) interleaver being loaded by a sequence of code symbols. The synchronized deinterleaver is shown simultaneously feeding the deinterleaved symbols to the decoder. Figure 6.26a shows symbols 1 to 4 being loaded; the $\times$s represent unknown states. Figure 6.26b shows the first four symbols shifted within the registers and the entry of symbols 5 to 8 to the interleaver input. Figure 6.6c shows symbols 9 to 12 entering the interleaver. The deinterleaver is now filled with message symbols, but nothing useful is being fed to the decoder yet. Finally, Figure 6.6d shows symbols 13 to 16 entering the interleaver, and at the output of the deinterleaver, symbols 1 to 4 are being passed to the decoder. The process continues in this way until the entire codeword sequence, in its original preinterleaved form, is presented to the decoder.

The performance of a convolutional interleaver is very similar to that of a block interleaver. The important advantage of convolutional over block interleaving is that with convolutional interleaving the end-to-end delay is $M(N - 1)$ symbols, where $M = NJ$, and the memory required is $M(N - 1)/2$ at both ends of the channel. Therefore, there is a reduction of one-half in delay and memory over the block interleaving requirements [16].

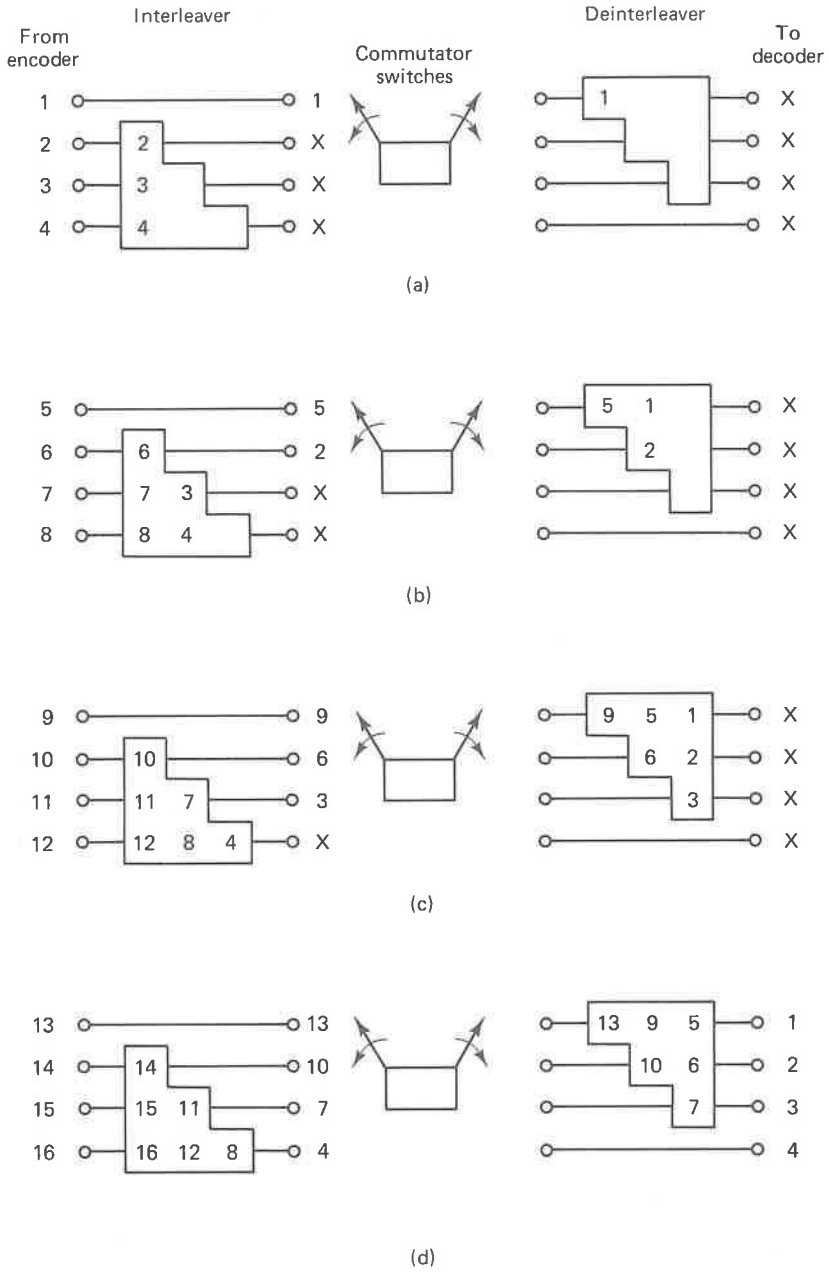


Figure 6.26 Convolutional interleaver/deinterleaver example.

6.6.3 Concatenated Codes

A concatenated code is one that uses two levels of coding, an inner code and an outer code, to achieve the desired error performance. Figure 6.27 illustrates the order of encoding and decoding. The inner code, the one that interfaces with the modulator/demodulator and channel, is usually configured to correct most of the channel errors. The outer code, usually a higher-rate (lower-redundancy) code, then reduces the probability of error to the specified level. The primary reason for using a concatenated code is to achieve a low error rate with an overall implementation complexity which is less than that which would be required by a single coding operation. In Figure 6.27 an interleaver is shown between the two coding steps. This is usually required to spread any error bursts that may appear at the output of the inner coding operation.

One of the most popular concatenated coding systems uses a Viterbi-decoded convolutional inner code and a Reed–Solomon (R–S) outer code, with interleaving between the two coding steps [23]. Operation of such systems with E_b/N_0 in the range 2.0 to 2.5 dB to achieve $P_B = 10^{-5}$ (only about 4 dB away from the Shannon limit) is now feasible with practical hardware [16]. In this system, the demodulator outputs soft quantized code symbols to the inner convolutional decoder, which in turn outputs hard quantized code symbols with bursty errors to the R–S decoder. (In a Viterbi-decoded system, the output errors tend to occur in bursts.) The outer R–S code is formed from m-bit segments of the binary data stream (see Section 5.7.4). The performance of such a (nonbinary) R–S code depends only on the number of *symbol errors* in the block. The code is undisturbed by burst errors within an m-bit symbol. That is, for a given symbol error, the R–S code performance is the same whether the symbol error is due to one bit being in error or m bits being in error. However, the concatenated system performance is severely degraded by correlated errors among successive symbols. Hence the interleaving between codes needs to take place at the symbol level (not at the bit level). In the next section we consider a popular consumer application of such symbol interleaving in a concatenated system.

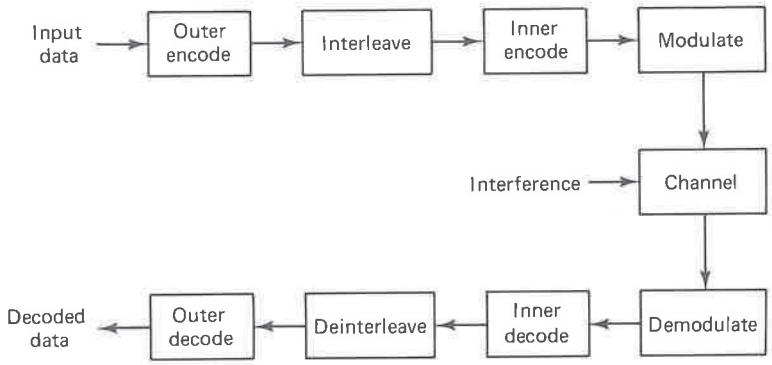


Figure 6.27 Block diagram of a concatenated coding system.

6.7 CODING AND INTERLEAVING APPLIED TO THE COMPACT DISC DIGITAL AUDIO SYSTEM

In 1979, Philips Corp. of the Netherlands and Sony Corp. of Japan defined a standard for the digital storage and reproduction of audio signals, known as the *compact disc (CD) digital audio system*. This CD system has become the world standard for achieving fidelity of sound reproduction that far surpasses any other available technique. A plastic disc 120 mm in diameter is used to store the digitized audio waveform. The waveform is sampled at 44.1 kilosamples/s to provide a recorded bandwidth of 20 kHz; each audio sample is uniformly quantized to one of 2^{16} levels (16 bits/sample), resulting in a dynamic range of 96 dB and a total harmonic distortion of 0.005%. A single disc (playing time approximately 70 minutes) stores about 10^{10} bits in the form of minute *pits* that are optically scanned by a laser.

There are several sources of channel errors: (1) small unwanted particles or air bubbles in the plastic material or pit inaccuracies arising in manufacturing, and (2) fingerprints or scratches during handling. It is difficult to predict how, on the average, a CD will get damaged; but in the absence of an accurate channel model, it is safe to assume that the channel mainly has a *burstlike* error behavior, since a scratch or fingerprint will cause *several* consecutive data samples to be in error. An important aspect of the system design contributing to the high-fidelity performance is a concatenated control scheme called the *cross-interleave Reed–Solomon* code (CIRC). The data are rearranged in time so that digits stemming from contiguous samples of the waveform are *spread out in time*. In this way, error bursts are made to appear as single random events (see the earlier sections on interleaving). The digital information is protected by adding parity bytes derived in two Reed–Solomon (R–S) encoders (see Section 5.7.4). Error control applied to the compact disc depends mostly on R–S coding and multiple layers of interleaving. Material on the CD is treated in this chapter rather than in Chapter 5 with R–S coding because it follows naturally after the subject of interleaving and concatenated codes in the previous sections.

In digital audio applications, an undetected decoding error is very serious since it results in clicks, while occasional *detected* failures are not so serious because they can be concealed. The CIRC error-control scheme in the CD system involves both *correction* and *concealment* of errors. The performance specifications for the CIRC are given in Table 6.5. From the specifications in the table it would appear that the CD can endure much damage (e.g., 8-mm holes punched in the disc) without any noticeable effect on the sound quality.

The CIRC system achieves its error control by a hierarchy of the following techniques:

1. The decoder provides a level of error correction.
2. If the error correction capability is exceeded, the decoder provides a level of erasure correction (see Section 5.5.5).
3. If the erasure correction capability is exceeded, the decoder attempts to

TABLE 6.5 Specifications for the CD Cross-Interleave Reed–Solomon Code

Maximum correctable burst length	$\simeq$ 4000 bits (2.5-mm track length on the disc)
Maximum interpolatable burst length	$\simeq$ 12,000 bits (8 mm)
Sample interpolation rate	One sample every 10 hours at $P_B = 10^{-4}$ 1000 samples/min at $P_B = 10^{-3}$
Undetected error samples (clicks)	Less than one every 750 hours at $P_B = 10^{-3}$ Negligible at $P_B \leq 10^{-4}$
New discs are characterized by	$P_B \simeq 10^{-4}$

conceal unreliable data samples by *interpolating* between reliable neighboring samples.

4. If the interpolation capability is exceeded, the decoder blanks out or *mutes* the system for the duration of the unreliable samples.

6.7.1 CIRC Encoding

Figure 6.28 illustrates the basic CIRC encoder block diagram (within the CD recording equipment) and the decoder block diagram (within the CD player equipment). Encoding consists of the encoding and interleaving steps designated as: Δ interleave, C_2 encode, D* interleave, C_1 encode, and D interleave. The decoder steps, consisting of deinterleaving and decoding, are performed in the *reverse* order of the encoding steps and are designated as: D deinterleave, C_1 decode, D* deinterleave, C_2 decode, and Δ deinterleave.

Figure 6.29 illustrates the basic system frame time, comprised of six sampling periods, each made up of a stereo sample pair (16-bit left sample and 16-bit right sample). The bits are organized into symbols or bytes of 8 bits each. Therefore, each sample pair contains 4 bytes, and the uncoded frame contains $k = 24$ bytes. Figure 6.29a–e summarizes the *five encoding steps* that characterize the CIRC system. The function of each of these steps will best be understood when we consider the decoding operation.

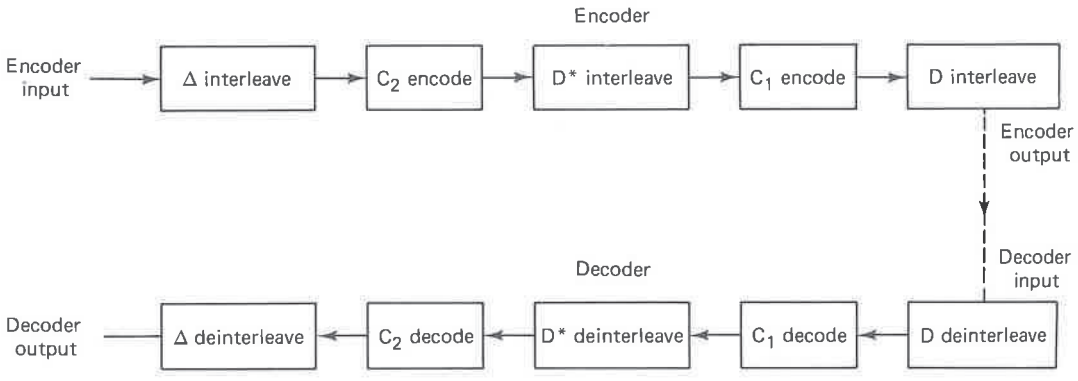


Figure 6.28 CIRC encoder and decoder.

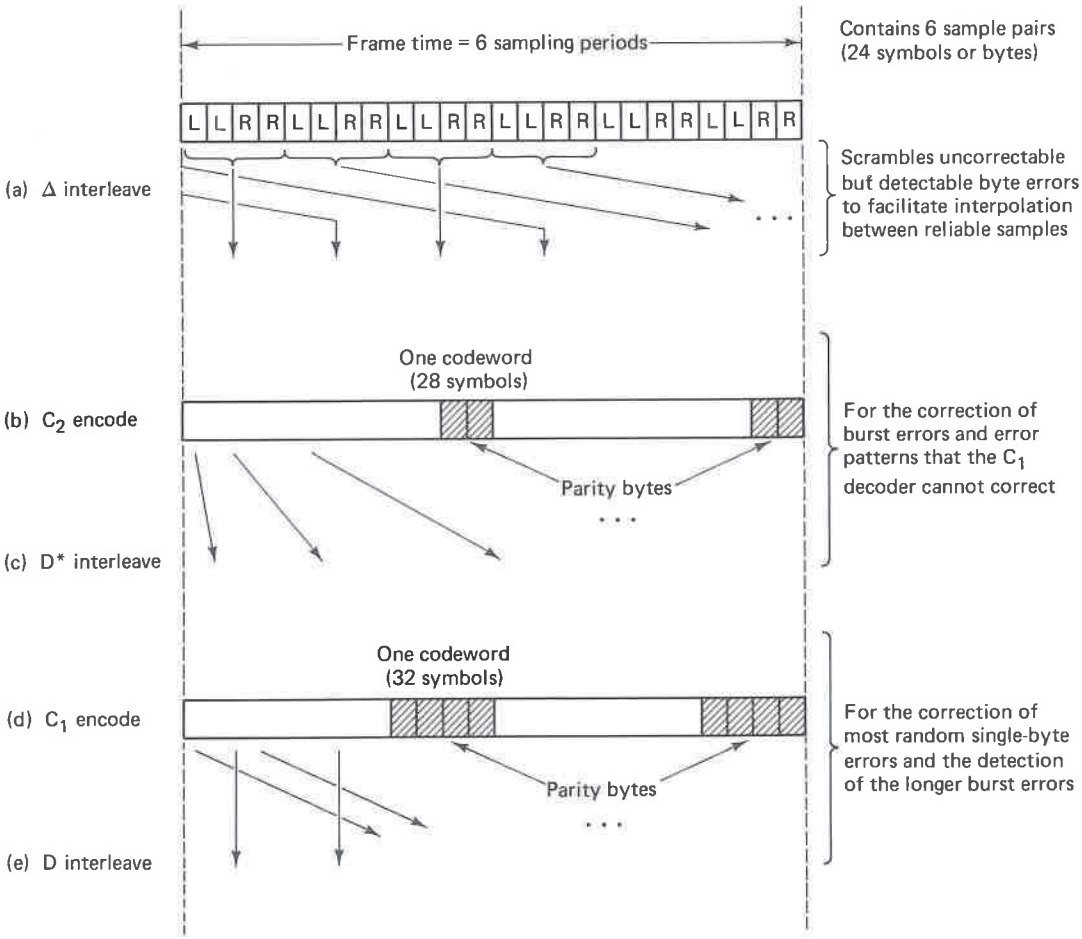


Figure 6.29 Compact disc encoder. (a) Δ interleave. (b) C_2 encode. (c) D^* interleave. (d) C_1 encode. (e) D interleave.

(a) Δ *interleave*. Even-numbered samples are separated from odd-numbered samples by two frame times in order to scramble uncorrectable but detectable byte errors. This facilitates the interpolation process.

(b) C_2 *encode*. Four Reed–Solomon (R–S) parity bytes are added to the Δ-interleaved 24-byte frame, resulting in a total of $n = 28$ bytes. This (28, 24) code is called the *inner code*.

(c) D^* *interleave*. Here each byte is delayed a different length, thereby spreading errors over several codewords. C_2 encoding together with D* interleaving have the function of providing for the correction of burst errors and error patterns that the C_1 decoder cannot correct.

(d) C_1 *encode*. Four R–S parity bytes are added to the $k = 28$ bytes of the D*-interleaved frame, resulting in a total of $n = 32$ bytes. This (32, 28) code is called the *outer code*.

(e) *D interleave*. The purpose is to *cross-interleave* the *even bytes* of a frame with the *odd bytes* of the next frame. By this procedure, two consecutive bytes on the disc will always end up in two different codewords. Upon decoding, this interleaving, together with the C_1 decoding, results in the correction of most random single errors and the detection of longer burst errors.

6.7.1.1 Shortening the R–S Code

In Section 5.7.4 an (n, k) R–S code is expressed in terms of $n = 2^m - 1$ total symbols and $k = 2^m - 1 - 2t$ data symbols, where m is the number of bits per symbol and t is the error-correcting capability of the code in symbols. For the CD system, where a symbol is made up of 8 bits, a 2-symbol error-correcting code can be configured as a (255, 251) code. However, the CD system uses a considerably shorter block length. Any block code (in systematic form) can be shortened without affecting the number of errors that can be corrected within a block length. In terms of the (255, 251) R–S code, imagine that 227 of the 251 data symbols are a set of all-zero symbols (which are not actually transmitted and hence are not subject to any errors). Then the code is really a (28, 24) code with the same 2-symbol error-correcting capability. This is what is done in the C_1 encoder of the CD system.

We can think of the 28 total symbols out of the C_1 encoder as the data symbols into the C_2 encoder. Again, we can configure a shortened 2-symbol error-correcting (255, 251) code by throwing away 223 data symbols—the result being a (32, 28) code.

6.7.2 CIRC Decoding

The inner and outer R–S codes with (n, k) values (32, 28) and (28, 24) each use four parity bytes. The code rate of the CIRC is $(k_1/n_1)(k_2/n_2) = 24/32 = 3/4$. From Equation (5.78) the minimum distance of the C_1 and C_2 R–S codes is $d_{\min} = n - k + 1 = 5$. From Equations (5.79) and (5.50),

$$t \leq \frac{d_{\min} - 1}{2} \tag{6.24}$$

$$\rho \leq d_{\min} - 1 \tag{6.25}$$

where t is the error-correcting capability and ρ is the erasure-correcting capability, it is seen that the C_1 or C_2 decoder can correct a maximum of 2 symbol errors or 4 symbol erasures per codeword. Or, as described by Equation (5.51), it is possible to correct any pattern of α errors and γ erasures simultaneously provided that

$$d_{\min} \geq 2\alpha + \gamma + 1 \tag{6.26}$$

There is a trade-off between error correction and erasure correction; the larger the error correcting capability used, the smaller will be the erasure correcting capability.

The benefits of CIRC are best seen at the *decoder*, where the processing steps, shown in Figure 6.30, are in the reverse order of the encoder steps. The decoder steps are as follows:

1. *D deinterleave*. This function is performed by the alternating delay lines marked D. The 32 bytes ($B_{i1}, \ldots, B_{i32}$) of an encoded frame are applied in parallel to the 32 inputs of the D deinterleaver. Each delay is equal to the duration of 1 byte, so that the information of the *even bytes* of a frame is cross-deinterleaved with that of the *odd bytes* of the next frame.
2. *C_1 decode*. The D deinterleaver and the C_1 decoder are designed to correct a single byte error in the block of 32 bytes and to detect larger burst errors. If multiple errors occur, the C_1 decoder passes them on unchanged, attaching to all 28 remaining bytes an erasure flag, sent via the dashed lines (the four parity bytes used in the C_1 decoder are no longer retained).
3. *D* deinterleave*. Due to the different lengths of the deinterleaving delay lines D*(1, . . . , 27), errors that occur in one word at the output of the C_1 decoder are *spread over a number of words* at the input of the C_2 decoder. This results in reducing the number of errors per input word of the C_2 decoder, enabling the C_2 decoder to correct these errors.
4. *C_2 decode*. The C_2 decoder is intended for the correction of burst errors that the C_1 decoder could not correct. If the C_2 decoder cannot correct these errors, the 24-byte codeword is passed on unchanged to the Δ deinterleaver

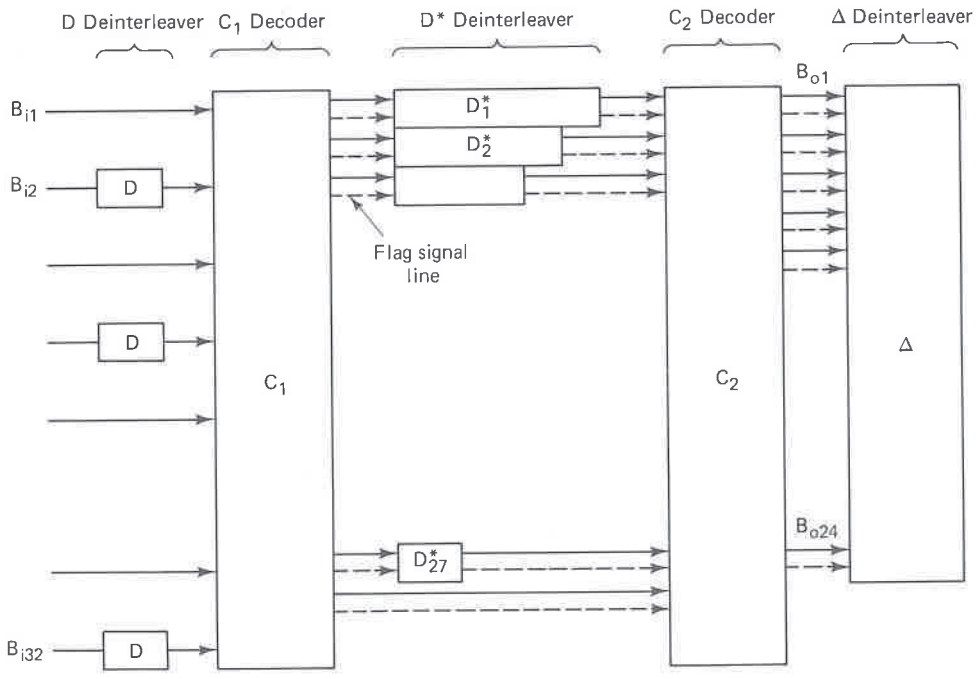


Figure 6.30 Compact disc decoder.

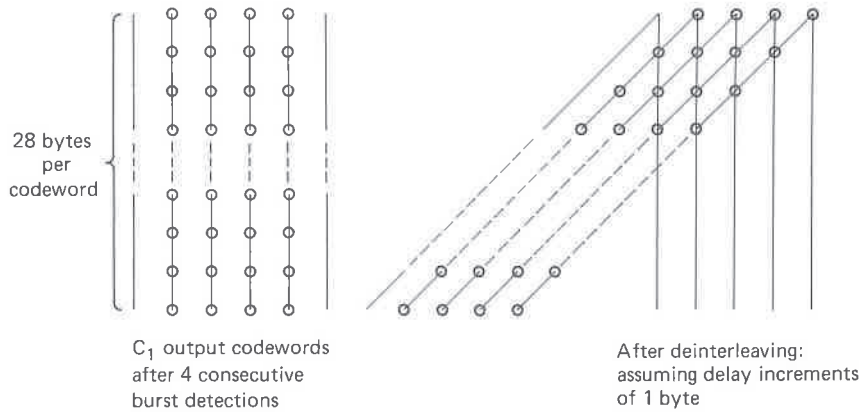


Figure 6.31 Example of 4-byte erasure capability. (Rightmost event is at the earliest time.)

and the associated positions are given an *erasure flag* via the dashed output lines, $B_{o1}, \ldots, B_{o24}$.

5. Δ *deinterleave*. The final operation deinterleaves uncorrectable but detected byte errors in such a way that *interpolation* can be used between reliable neighboring samples.

Figure 6.31 highlights the decoder steps 2, 3, and 4. At the output of the C_1 decoder is seen a sequence of four 28-byte codewords that have exceeded the 1 byte per codeword error correction design. Therefore, each of the symbols in these codewords is tagged with an erasure flag (shown with circles). The D* deinterleaver provides a staggered delay for each byte of a codeword, so that the bytes of a given codeword arrive in different codewords at the input to the C_2 decoder. If we assume that the delay increments of the D* deinterleaver in Figure 6.31 are 1 byte, it would be possible to correct error bursts of as many as four consecutive C_1 codewords (since the C_2 decoder is capable of four erasure corrections per codeword). In the actual CD system, the delay increments are 4 bytes; therefore, the maximum burst error correction capability consists of 16 consecutive uncorrectable C_1 words.

6.7.3 Interpolation and Muting

Samples that cannot be corrected by the C_2 decoder could cause audible disturbances. The function of the *interpolation* process is to insert new samples, estimated from reliable neighbors, in place of the unreliable ones. If an entire C_2 word is detected as unreliable, this would make it impossible to apply interpolation without additional interleaving, since both even- and odd-numbered samples are unreliable. This can happen if the C_1 decoder fails to detect an error but the C_2 decoder detects it. It is the purpose of Δ deinterleaving (over a span of two frame

Figure 6.32 Effect of interleaving. (Rightmost event is at the earliest time.)

times) to obtain a pattern where even-numbered samples can be interpolated from reliable odd-numbered samples, or vice versa.

Two successive unreliable words consisting of 12 sample pairs are shown in Figure 6.32. A sample pair consists of a sample (2 bytes) from the right audio channel and a sample from the left audio channel. The numbers indicate the ordering of the sets of samples. An encircled sample set denotes an *erasure flag*. After Δ deinterleaving, the unreliable samples shown in the figure are estimated by a first-order linear interpolation between neighboring samples that stem from a different location on the disc.

In CD players, another level of error control is provided in case a burst length of 48 frames is exceeded and 2 or more consecutive unreliable samples result. In this case the system is *muted* (audio is softly blanked out), which is not discernible to the human ear if the muting time does not exceed a few milliseconds. For a more detailed treatment of the CIRC coding scheme in the CD system, see References [31–34].

6.8 CONCLUSION

In the last decade, coding emphasis has been in the area of convolutional codes since in almost every application, convolutional codes outperform block codes for the same implementation complexity of the encoder–decoder. For satellite communication channels, forward error correction techniques can easily reduce the required SNR for a specified error performance by 5 to 6 dB. This coding gain can translate directly into an equivalent reduction in required satellite effective radiated power (EIRP), with consequently reduced satellite weight and cost.

In this chapter we have outlined the essential structural difference between block codes and convolutional codes—the fact that rate $1/n$ convolutional codes have a memory of the prior $K - 1$ bits, where K is the encoder constraint length. With such memory, the encoding of each input data bit not only depends on the value of that bit but on the values of the $K - 1$ input bits that precede it. We presented the decoding problem in the context of the maximum likelihood algorithm, examining all the candidate codeword sequences which could possibly be created by the encoder, and selecting the one that appears statistically most likely; the decision is based on a distance metric for the received code symbols. The error performance analysis of convolutional codes is more complicated than the simple binomial expansion describing the error performance of many block codes. We laid out the concept of free distance, and we presented the relationship between free distance and error performance in terms of bounds. We also described the basic idea behind sequential decoding and feedback decoding and showed some comparative performance curves and tables for various coding schemes.

Finally, we described a technique, interleaving, that allows the popular block and convolutional coding schemes to be used over channels that exhibit bursty noise or periodic fading, without suffering degradation. We used the CD digital

audio system as an example of how interleaving plays an important role in ameliorating the effects of burst noise.

Appendix E consists of a FORTRAN program called VITALG for the convolutional encoding and Viterbi decoding of messages. The messages can be in the form of binary sequences or ASCII characters. The user has a choice of code rate, constraint length, connection vectors, and path memory length. The program can be used to insert errors into a bit stream after it has been encoded. From the program output, the user sees the error correcting that results from the use of hard-decision Viterbi decoding of his chosen message. It should prove interesting to use the VITALG program for verifying the performance of the optimum Odenwalder codes shown in Table 6.4.

REFERENCES

1. Gallager, R. G., *Information Theory and Reliable Communication,* John Wiley & Sons, Inc., New York, 1968.
2. Fano, R. M., "A Heuristic Discussion of Probabilistic Decoding," IRE *Trans. Inf. Theory,* vol. IT9, no. 2, 1963, pp. 64–74.
3. Odenwalder, J. P., *Optimal Decoding of Convolutional Codes,* Ph.D. dissertation, University of California, Los Angeles, 1970.
4. Curry, S. J., *Selection of Convolutional Codes Having Large Free Distance,* Ph.D. dissertation, University of California, Los Angeles, 1971.
5. Larsen, K. J., "Short Convolutional Codes with Maximal Free Distance for Rates $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{4}$," *IEEE Trans. Inf. Theory,* vol. IT19, no. 3, 1973, pp. 371–372.
6. Lin, S., and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983.
7. Forney, G. D., Jr., "Convolutional Codes: I. Algebraic Structure," *IEEE Trans. Inf. Theory,* vol. IT16, no. 6, Nov. 1970, pp. 720–738.
8. Viterbi, A., "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun. Technol.,* vol. COM19, no. 5, Oct. 1971, pp. 751–772.
9. Forney, G. D., Jr., and Bower, E. K., "A High Speed Sequential Decoder: Prototype Design and Test," *IEEE Trans. Commun. Technol.,* vol. COM19, no. 5, Oct. 1971, pp. 821–835.
10. Jelinek, F., "Fast Sequential Decoding Algorithm Using a Stack," *IBM J. Res. Dev.,* vol. 13, Nov. 1969, pp. 675–685.
11. Massey, J. L., *Threshold Decoding,* The MIT Press, Cambridge, Mass., 1963.
12. Heller, J. A., and Jacobs, I. W., "Viterbi Decoding for Satellite and Space Communication," *IEEE Trans. Commun. Technol.,* vol. COM19, no. 5, October 1971, pp. 835–848.
13. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inf. Theory,* vol. IT13, April 1967, pp. 260–269.

14. Omura, J. K., "On the Viterbi Decoding Algorithm" (correspondence), *IEEE Trans. Inf. Theory,* vol. IT15, Jan. 1969, pp. 177–179.
15. Mason, S. J., and Zimmerman, H. J., *Electronic Circuits, Signals, and Systems,* John Wiley & Sons, Inc., New York, 1960.
16. Clark, G. C., Jr., and Cain, J. B., *Error-Correction Coding for Digital Communications,* Plenum Press, New York, 1981.
17. Viterbi, A. J., and Omura, J. K., *Principles of Digital Communication and Coding,* McGraw-Hill Book Company, New York, 1979.
18. Massey, J. L., and Sain, M. K., "Inverse of Linear Sequential Circuits," *IEEE Trans. Comput.,* vol. C17, Apr. 1968, pp. 330–337.
19. Rosenberg, W. J., *Structural Properties of Convolutional Codes,* Ph.D. dissertation, University of California, Los Angeles, 1971.
20. Bhargava, V. K., Haccoun, D., Matyas, R., and Nuspl, P., *Digital Communications by Satellite,* John Wiley & Sons, Inc., New York, 1981.
21. Jacobs, I. M., "Practical Applications of Coding," *IEEE Trans. Inf. Theory,* vol. IT20, May 1974, pp. 305–310.
22. Linkabit Corporation, "Coding Systems Study for High Data Rate Telemetry Links," *NASA Ames Res. Center, Final Rep. CR-114278,* Contract NAS-2-6-24, Moffett Field, Calif., 1970.
23. Odenwalder, J. P., *Error Control Coding Handbook,* Linkabit Corporation, San Diego, Calif., July 15, 1976.
24. Wozencraft, J. M., "Sequential Decoding for Reliable Communication," *IRE Natl. Conv. Rec.,* vol. 5, pt. 2, 1957, pp. 11–25.
25. Wozencraft, J. M., and Reiffen, B., *Sequential Decoding,* The MIT Press, Cambridge, Mass., 1961.
26. Shannon, C. E., "A Mathematical Theory of Communication," *Bell Syst. Tech. J.,* vol. 27, 1948, pp. 379–423, 623–656.
27. Brayer, K., "Error Correcting Code Performance on HF, Troposcatter, and Satellite Channels," *IEEE Trans. Commun. Technol.,* vol. COM19, 1971, pp. 835–848.
28. Kohlenberg, A., and Forney, G. D., "Convolutional Coding for Channels with Memory," *IEEE Trans. Inf. Theory,* vol. IT2, 1968, pp. 618–626.
29. Ramsey, J. L., "Realization of Optimum Interleavers, *IEEE Trans. Inf. Theory,* vol. IT16, no. 3, May 1970, pp. 338–345.
30. Forney, G. D., "Burst-Correcting Codes for the Classic Bursty Channel," *IEEE Trans. Commun. Technol.,* vol. COM19, Oct. 1971, pp. 772–781.
31. Peek, J. B. H., "Communications Aspects of the Compact Disc Digital Audio System," *IEEE Commun. Mag.,* vol. 23, no. 2, Feb. 1985, pp. 7–20.
32. Berkhout, P. J., and Eggermont, L. D. J., "Digital Audio Systems," *IEEE ASSP Mag.,* Oct. 1985, pp. 45–67.
33. Driessen, L. M. H. E., and Vries, L. B., "Performance Calculations of the Compact Disc Error Correcting Code on a Memoryless Channel," *Fourth Int. Conf. Video and Data Record.,* Southampton, England, Apr. 20–23, 1982, *IERE Conf. Proc.,* vol. 54, pp. 385–395.
34. Hoeve, H., Timmermans, J., and Vries, L. B., "Error Correction in the Compact Disc System," *Philips Tech. Rev.,* vol. 40, no. 6, 1982, pp. 166–172.

PROBLEMS

6.1. Draw the state diagram, tree diagram, and trellis diagram for the $K = 3$, rate $\frac{1}{3}$ code generated by

$$g_1(X) = X + X^2$$

$$g_2(X) = 1 + X$$

$$g_3(X) = 1 + X + X^2$$

6.2. Given a $K = 3$, rate $\frac{1}{2}$, binary convolutional code with the partially completed state diagram shown in Figure P6.1, find the complete state diagram and sketch a diagram for the encoder.

6.3. Draw the state diagram, tree diagram, and trellis diagram for the convolutional encoder characterized by the block diagram in Figure P6.2.

6.4. Suppose that you were trying to find the quickest way to get from London to Vienna by boat or train. The diagram in Figure P6.3 was constructed from various schedules. The labels on each path are travel times. Using the Viterbi algorithm, find the fastest route from London to Vienna. In a general sense, explain how the algorithm works, what calculations must be made, and what information must be retained in the memory used by the algorithm.

6.5. Consider the convolutional encoder shown in Figure P6.4.
(a) Write the connection vectors and polynomials for this encoder.
(b) Draw the state diagram, tree diagram, and trellis diagram.

6.6. What is the impulse response of the encoder of Problem 6.5? Using the impulse response, determine the output sequence when the input is 1 0 1. Verify by using the generator polynomials.

6.7. Does the encoder of Problem 6.5 allow catastrophic error propagation? Justify your answer with an example.

6.8. Find the free distance of the encoder of Problem 6.3 by the transfer function method.

6.9. Let the codewords of a coding scheme be

$$a = 0\ 0\ 0\ 0\ 0\ 0$$

$$b = 1\ 0\ 1\ 0\ 1\ 0$$

$$c = 0\ 1\ 0\ 1\ 0\ 1$$

$$d = 1\ 1\ 1\ 1\ 1\ 1$$

If the received sequence over a binary symmetric channel is 1 1 1 0 1 0 and a maximum likelihood decoder is used, what will be the decoded symbol?

6.10. Consider that the $K = 3$, rate $\frac{1}{2}$ encoder of Figure 6.3 is used over a binary symmetric channel (BSC). Assume that the initial encoder state is the 00 state. At the output of the BSC, the sequence $\mathbf{Z}$ = (1 1 0 0 0 0 1 0 1 1 rest all "0") is received.
(a) Find the maximum likelihood path through the trellis diagram, and determine the first 5 decoded information bits. If a tie occurs between any two merged paths, choose the upper branch entering the particular state.
(b) Identify any channel bits in $\mathbf{Z}$ that were inverted by the channel during transmission.

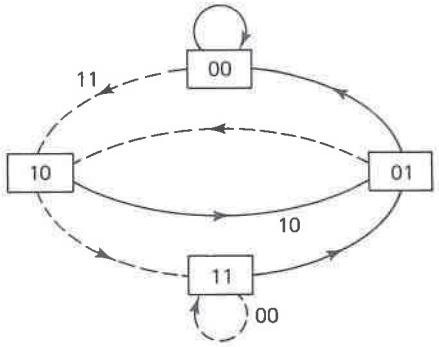


Figure P6.1

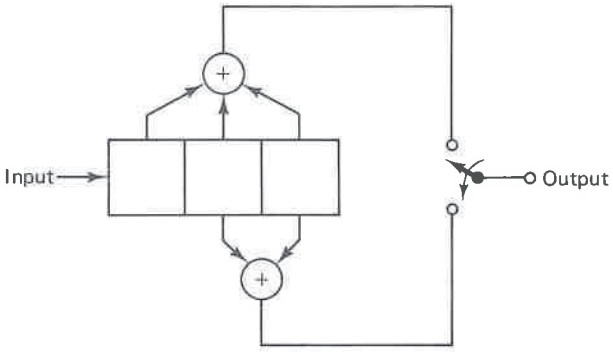


Figure P6.2

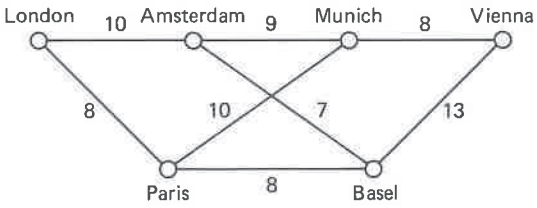


Figure P6.3

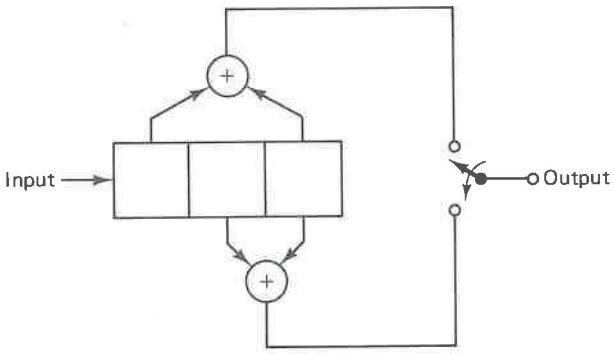


Figure P6.4

6.11. Determine which of the following rate $\frac{1}{2}$ codes are catastrophic.

(a) $g_1(X) = X^2, \quad g_2(X) = 1 + X + X^3$

(b) $g_1(X) = 1 + X^2, \quad g_2(X) = 1 + X^3$

(c) $g_1(X) = 1 + X + X^2, \quad g_2(X) = 1 + X + X^3 + X^4$

(d) $g_1(X) = 1 + X + X^3 + X^4, \quad g_2(X) = 1 + X^2 + X^4$

(e) $g_1(X) = 1 + X^4 + X^6 + X^7, \quad g_2(X) = 1 + X^3 + X^4$

(f) $g_1(X) = 1 + X^3 + X^4, \quad g_2(X) = 1 + X + X^2 + X^4$

6.12. (a) Consider a coherently detected BPSK signal encoded with the encoder shown in Figure 6.3. Find an upper bound on the bit error probability, P_B, if the available E_b/N_0 is 6 dB. Assume hard decision decoding.

(b) Compare P_B with the uncoded case and calculate the improvement factor.

6.13. Using sequential decoding, illustrate the path along the tree diagram shown in Figure 6.20 when the received sequence is 0 1 1 1 0 0 0 1 1 1. The backup criterion is three disagreements.

6.14. Repeat the decoding example of Problem 6.13 using feedback decoding, with a look-ahead length of 3. In the event of a tie, select the upper half of the tree.

6.15. Figure P6.5 depicts a constraint length 2 convolutional encoder.

(a) Draw the state diagram, tree diagram, and trellis diagram.

(b) Assume that a received message from this encoder is 1 1 0 0 1 0. Use a feedback decoding algorithm with a look-ahead length of 2 to decode the coded message sequence.

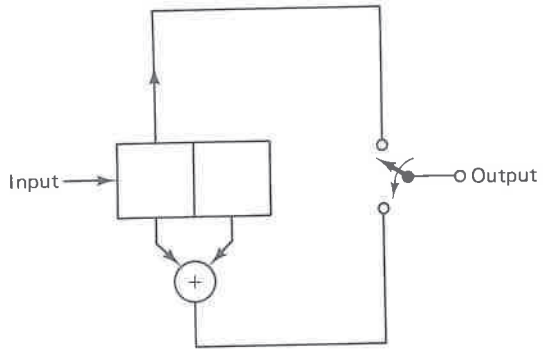


Figure P6.5

6.16. Using the branch word information on the encoder trellis of Figure 6.7, decode the sequence $\mathbf{Z}$ = (01 11 00 01 11 rest all "0"), using hard-decision Viterbi decoding.

6.17. Consider the rate $\frac{2}{3}$ convolutional encoder shown in Figure P6.6. In this encoder, $k = 2$ bits at a time are shifted into the encoder and $n = 3$ bits are generated at the encoder output. There are $kK = 4$ stages in the register, and the constraint length is $K = 2$ in units of 2-bit bytes. The state of the encoder is defined as the contents of the rightmost $K - 1$ k-tuple stages. Draw the state diagram, the tree diagram, and the trellis diagram.

6.18. Find the ratio of the predetection signal-to-noise spectral density, P_r/N_0, in decibels, required to yield a decoded data rate of 1 Mbit/s with a bit error probability of 10^{-5}. Assume binary noncoherent FSK modulation. Also, assume convolutional encoding with the following decoder relationship:

$$P_b = 2000\, P_B^4$$

where P_B and P_b are bit error probabilities into and out of the decoder, respectively.

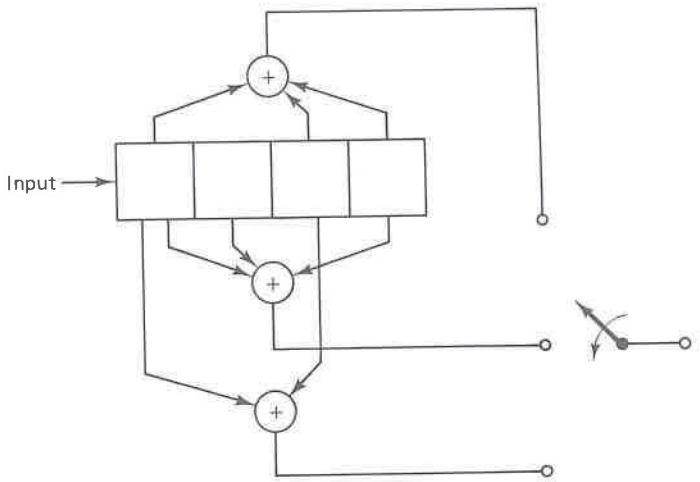


Figure P6.6

6.19. The sequence

1 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0

is the input to a 4 × 4 block interleaver. What is the output sequence? The same input sequence is applied to the convolutional interleaver of Figure 6.26. What is the output sequence?

6.20. Using the computer program VITALG listed in Appendix E, perform the following calculations. Let the uncoded message consist of a sequence of binary zeros, where the number of zeros is 10 times the constraint length of the code being used. Convolutionally encode the message and emulate a memoryless AWGN channel by inserting random transmission errors into the coded sequence (space the errors at approximately a uniform distance from one another). Decode the corrupted coded message using the Viterbi algorithm with the path memory chosen to be five times the constraint length. Use the code generators described as optimum by Odenwalder in Table 6.4. Record the errors corrected by the code, and tabulate the maximum number of errors that can be corrected using each of the following codes:

(a) Rate $\frac{1}{2}$, constraint length 3
(b) Rate $\frac{1}{3}$, constraint length 3
(c) Rate $\frac{1}{2}$, constraint length 5
(d) Rate $\frac{1}{3}$, constraint length 5
(e) Rate $\frac{1}{2}$, constraint length 7
(f) Rate $\frac{1}{3}$, constraint length 7

Explain the results.

6.21. Repeat Problem 6.20. However, instead of a memoryless AWGN channel, emulate a channel that has memory by inserting error bursts into the coded message. Let a burst consist of an uninterrupted sequence of errors placed approximately in the middle of the message. Tabulate the maximum number of errors that can be corrected using each of the code types (a) through (f) listed in Problem 6.20, and compare the error-correcting capabilities of the codes with these two different channel environments. Explain the results.

6.22. Repeat Problem 6.20, parts (e) and (f), for both the uniformly spaced error pattern and the burst error pattern (described in Problems 6.20 and 6.21). In each case compare the performance of the Odenwalder generators in Table 6.4 to other generators of your own choosing. Tabulate the results. Do your findings support the premise that the Table 6.4 generators are optimum?

6.23. For each of the following conditions, design an interleaver for a communication system operating over a bursty noise channel at a transmission rate of 19,200 coded symbols/s.

(a) A contiguous noise burst typically lasts for 250 ms. The system code consists of a (127, 36) BCH code with $d_{\min} = 31$. The end-to-end delay is not to exceed 5 s.

(b) A contiguous noise burst typically lasts for 20 ms. The system code consists of a rate $\frac{1}{2}$ convolutional code with a feedback decoding algorithm that corrects an average of 3 symbols in a sequence of 21 symbols. The end-to-end delay is not to exceed 160 ms.

6.24. **(a)** Calculate the probability of a byte (symbol) error after decoding the data stored on a compact disc (CD) as described in Section 6.7. Assume that the probability of a channel symbol error for the disc is 10^{-3}. Also assume that the inner and outer R–S decoders are each configured to correct all 2-symbol errors, and that the interleaving process results in channel symbol errors being uncorrelated from one another.

(b) Repeat part (a) for a disc that has a probability of channel symbol error equal to 10^{-2}.