This chapter deals with convolutional coding. Chapter 6 presented the fundamentals of linear block codes, which are 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 capability and complexity of the code.

7.1 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 7.1. 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), and i is a time index. To be precise, one should denote the elements of $\mathbf{m}$ with an index for class membership (e.g., for binary codes, 1 or 0) and an index for time. However, in this chapter, for simplicity, indexing is only used to indicate time (or location within a sequence). 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 *code 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 7.1. 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 7.2, is mechanized with a kK-stage shift register and n modulo-2 adders, where K is the constraint length.

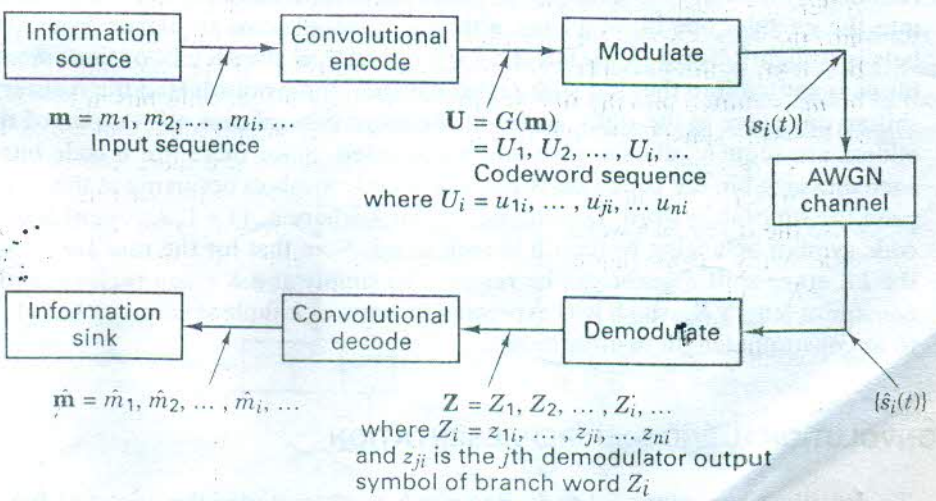

Figure 7.1 Encode/decode and modulate/demodulate portions of a communication link.

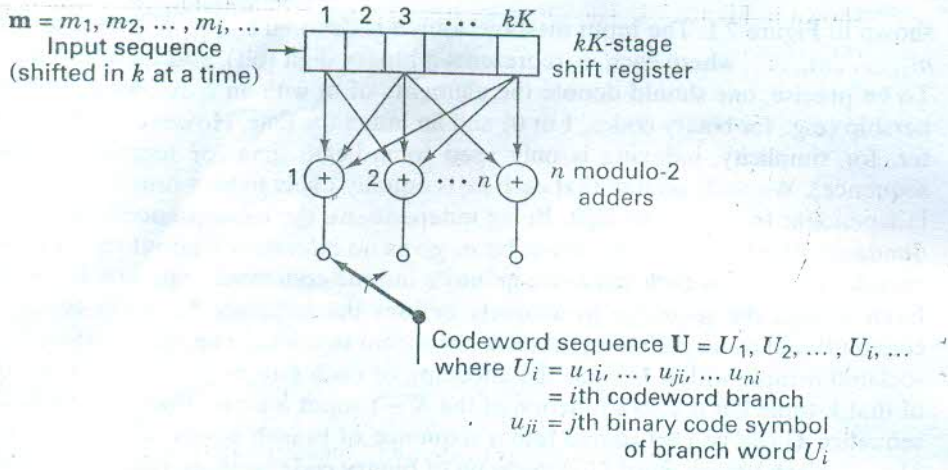

Figure 7.2 Convolutional encoder with constraint length K and rate k/n.

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

7.2 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.

7.2.1 Connection Representation

We shall use the convolutional encoder, shown in Figure 7.3, 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

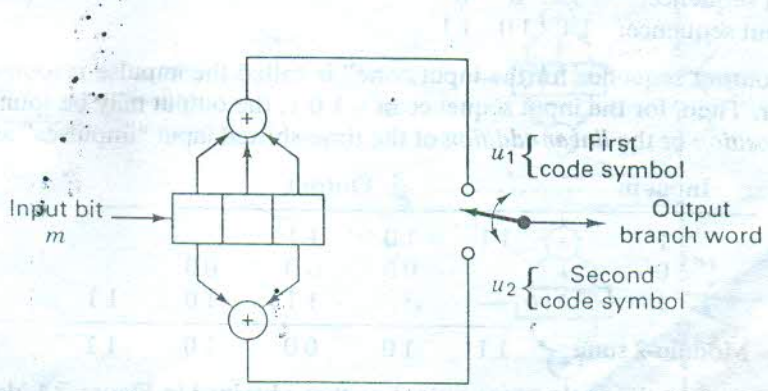

Figure 7.3 Convolutional encoder (rate $\frac{1}{2}$, $K = 3$).

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:

$$\mathbf{g}_1 = 1\ 1\ 1$$
$$\mathbf{g}_2 = 1\ 0\ 1$$

Now consider that a message vector $\mathbf{m} = 1\ 0\ 1$ is convolutionally encoded with the encoder shown in Figure 7.3. The three message bits are inputted, one at a time, at times t_1, t_2, and t_3, as shown in Figure 7.4. 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 flushing is completed at time t_5. Thus, a new message can be entered at time t_6.

7.2.1.1 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 7.3 as a one moves through it:

Register contents	Branch word u_1	Branch word 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 7.4, demonstrating that *convolutional codes are linear*—just like the linear block codes of Chapter 6. It

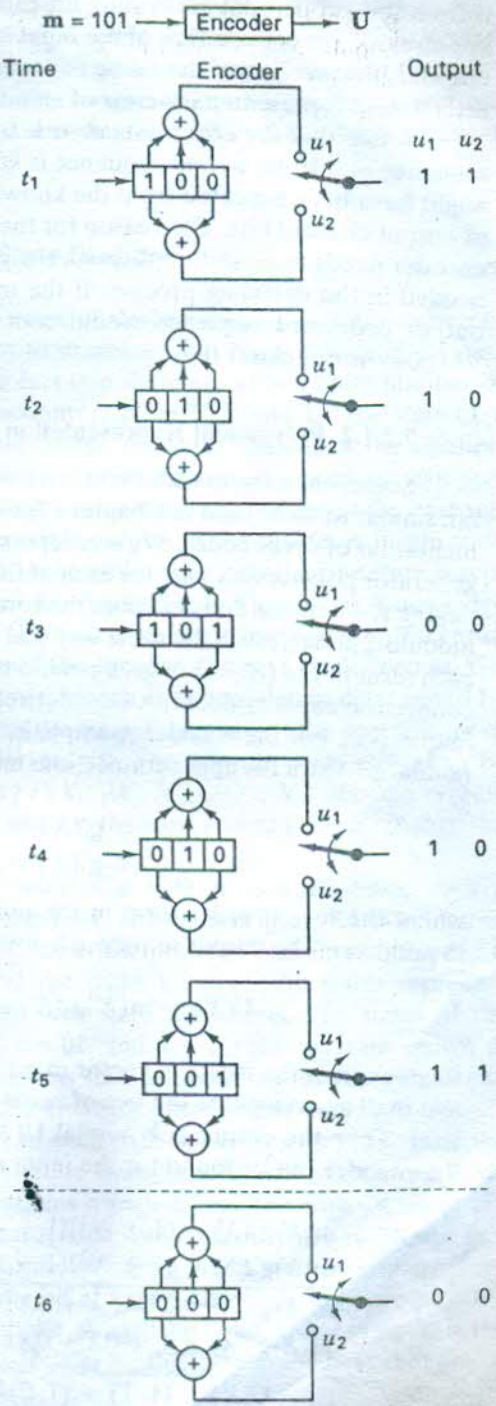

Figure 7.4 Convolutionally encoding a message sequence with a rate $\frac{1}{2}$, $K = 3$ encoder.

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, 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 $\frac{1}{2}$.

7.2.1.2 Polynomial Representation

Sometimes, the encoder connections are characterized by *generator polynomials,* similar to those used in Chapter 6 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 7.3, 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 7.3 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—namely, 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 7.4 and at 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].)

7.2.2 State Representation and the State Diagram

A convolutional encoder belongs to a class of devices known as *finite-state machines,* which is the general name given to machines that have a memory of past signals. The adjective *finite* refers to the fact that there are only a finite number of unique states that the machine can encounter. What is meant by the *state* of a finite-state machine? In the most general sense, the state consists of the smallest amount of information that, together with a current input to the machine, can predict the output of the machine. The state provides some knowledge of the past signaling events and the restricted set of possible outputs in the future. A future state is restricted by the past state. For a rate $1/n$ convolutional encoder, the state is represented by the contents of the rightmost $K - 1$ stages (see Figure 7.3). 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 7.3 is shown in Figure 7.5. 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 7.5 illustrates all the state transitions that are possible for the encoder in Figure 7.3. 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.

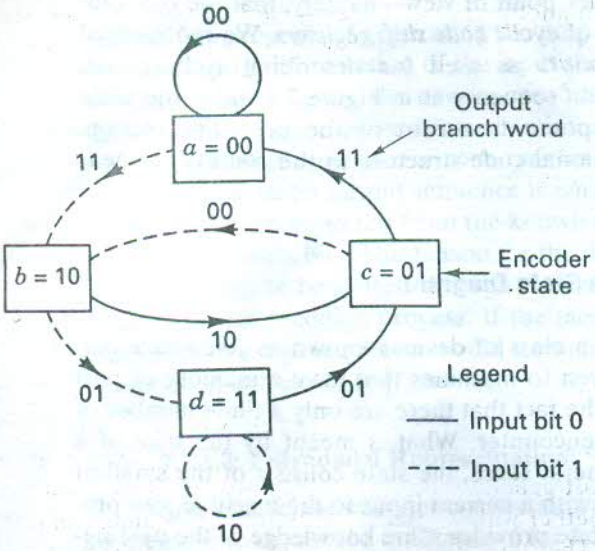

Figure 7.5 Encoder state diagram (rate $\frac{1}{2}$, $K = 3$).

Example 7.1 Convolutional Encoding

For the encoder shown in Figure 7.3, 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
—	000	00	00	—	
1	100	00	10	1	1
1	110	10	11	0	1
0	011	11	01	0	1
1	101	01	10	0	1
1	110	10	11	0	1
0	011	11	01	0	1
0	001	01	00	1	1

state t_i state t_{i+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 7.2 Convolutional Encoding

In Example 7.1 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 7.1 with the assumption that the given input sequence is preceded by two one bits, and verify that now the codeword sequence $\mathbf{U}$ for input sequence $\mathbf{m} = 1\ 1\ 0\ 1\ 1$ is different than the codeword found in Example 7.1.

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

By comparing this result with that of Example 7.1, 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.

7.2.3 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 convolutional encoder shown in Figure 7.3 is illustrated in Figure 7.6. 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

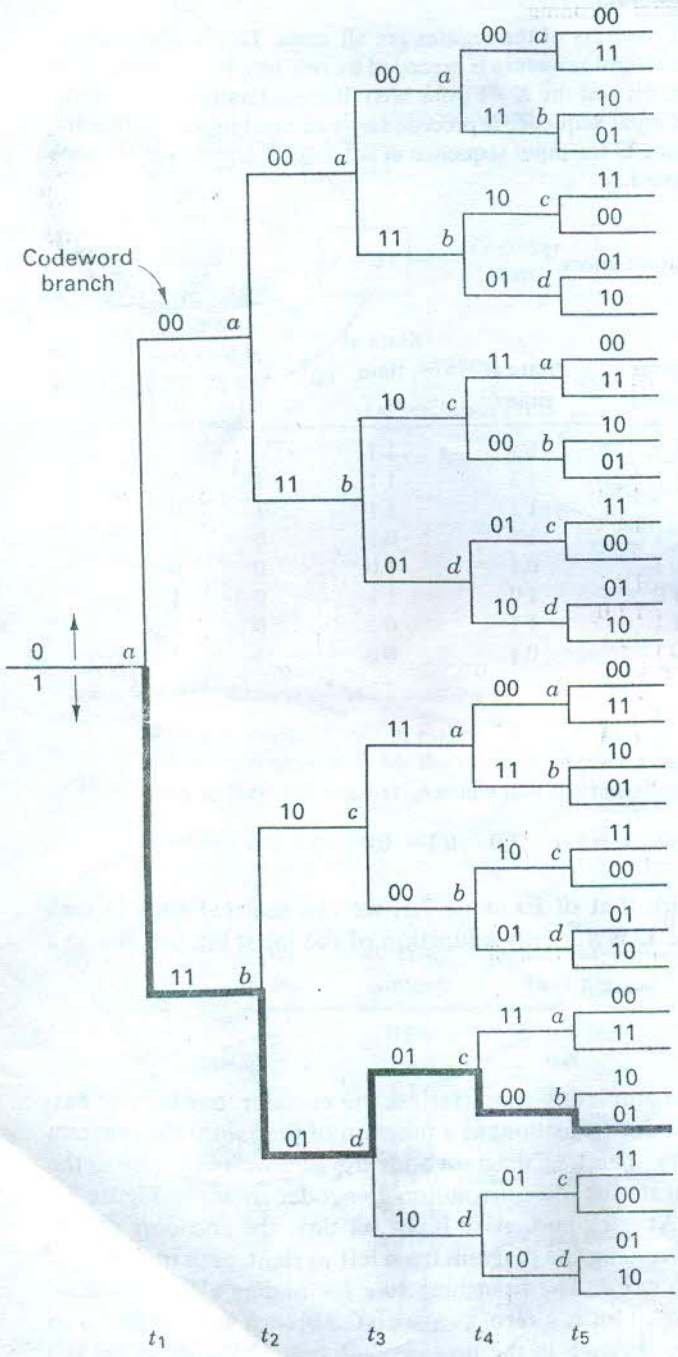

Figure 7.6 Tree representation of encoder (rate $\frac{1}{2}$, $K = 3$).

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 diagram in Figure 7.6. This path corresponds to the 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 increases as a function of 2^L, where L is the number of branch words in the sequence. You would quickly run out of paper, and patience.

7.2.4 The Trellis Diagram

Observation of the Figure 7.6 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 7.6 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 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 7.3. 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 7.6, we obtain another diagram, called the trellis diagram. The *trellis diagram,* 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 7.3 is shown in Figure 7.7.

In drawing the trellis diagram, we use the same convention that we introduced with the state diagram—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

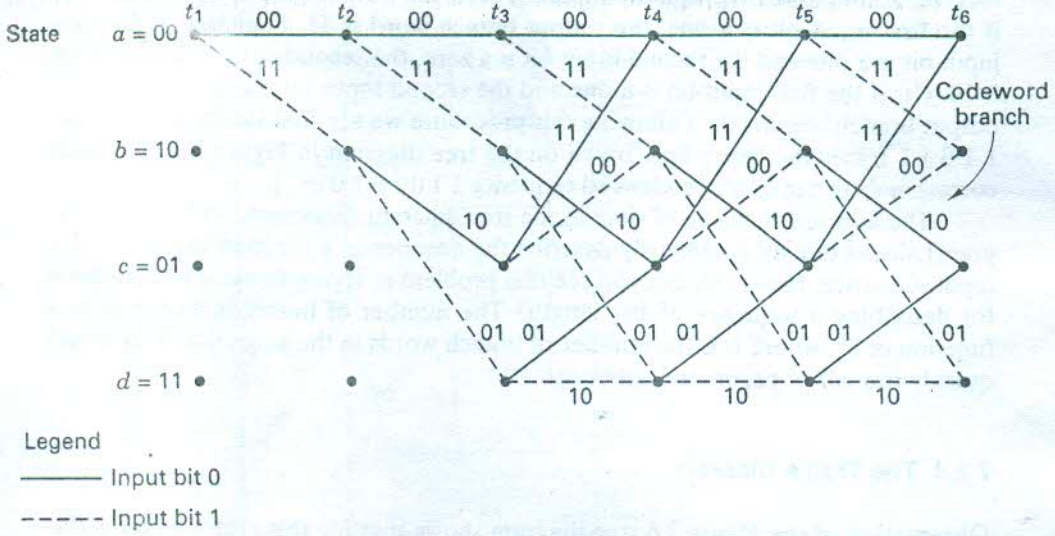

Figure 7.7 Encoder trellis diagram (rate $\frac{1}{2}$, $K = 3$).

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. At this point and thereafter, 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 7.7 the output branch words corresponding to the state transitions appear as labels on the trellis branches.

One time-interval section of a fully-formed encoding trellis structure completely defines the code. The only reason for showing several sections is for viewing a code-symbol sequence as a function of time. The state of the convolutional encoder is represented by the contents of the rightmost $K - 1$ stages in the encoder register. Some authors describe the state as the contents of the leftmost $K - 1$ stages. Which description is correct? They are both correct in the following sense. Every transition has a starting state and a terminating state. The rightmost $K - 1$ stages describe the starting state for the current input, which is in the leftmost stage (assuming a rate $1/n$ encoder). The leftmost $K - 1$ stages represent the terminating state for that transition. A code-symbol sequence is characterized by N branches (representing N data bits) occupying N intervals of time and associated with a particular state at each of $N + 1$ times (from start to finish). Thus, we launch bits at times $t_1, t_2, \ldots, t_N$, and are interested in state metrics at times $t_1, t_2, \ldots, t_{N+1}$. The convention used here is that the current bit is located in the leftmost stage (not on a wire leading to that stage), and the rightmost $K - 1$ stages start in the all-zeros state. We refer to this time as the *start time* and label it t_1. We refer to the concluding time of the last transition as the *terminating time* and label it t_{N+1}.