NEAR EAST UNIEVERSITY

DEPERTMENT OF
COMPUTER ENGINEERING

GRADUATION PROJECT

" ERROR CONTROL CODING"

PROF.DR.FAHRETTIN

ZELİHA ÇAY
93651

INDEX

ERROR CONTROL CODING

Transmission errors in digital communication depend on the signal-to-noise ratio. If a particular system has a fixed value of S/N and the error rate is unacceptably high, then some other means of improving reliability must be sought Error-control coding often provides the best solution.

Error-control coding involves systematic addition of extra digits to the transmitted message. These extra check digits convey no information by themselves, but they make it possible to detect or correct errors in the regenerated message digits. In principle, information theory holds out the promise of nearly errorless transmission, as well be discussed in Chap.15. In practice, we seek some compromise between conflicting considerations of reliability, efficiency and equipment complexity. A multitude of error-control codes have therefore been devised to suit various applications.

This chapter starts with an overview of error-control coding, emphasizing the distinction between error detection and error correction and systems that employ these strategies. Subsequent sections describe the two major types of code implementations, block codes and convolutinal codes. We will stick entirely to binary coding, and we will omit formal mathematical analysis. Detailed treatments of error-control coding are provided by the references cited in the supplementary reading list.

ERROR DETECTION AND CORRECTION

Coding for error detection, without correction, is simpler than error-correction coding. When a two-way channel exists between source and destination, the receiver can request retransmission of information containing detected errors. This error-control strategy, called automatic repeat request (ARQ), particularly suits data communication systems such as computer networks. However, when retransmission is impossible or impractical, error control must take the form of forward error correction (FEC) using an error-correcting code. Both strategies will be examined here, after an introduction to simple but illustrative coding techniques.

Repetition and Parity-Check Codes

When you try to talk to someone across a noisy room, you may need to repeat yourself to be understood. A brute-force approach to binary communication over a noisy channel likewise employs repetition, so each message bit is represented by a codeword consisting of n identical bits. Any transmission error is a received codeword alters the repetition pattern by changing a 1 to a 0 or vice versa.

If transmission errors occur randomly and independently with probability P=x, then the binomial frequency function from Eq.(1), Sect.4.4,gives the probability of i errors in an n-bit codeword as

$$P(i,n) = \binom{n}{i} \alpha^i (1-\alpha)^{n-i} \quad (1a)$$

$$\approx \binom{n}{i} \alpha^i \qquad \alpha \ll 1$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)} = \frac{n(n-1).....(n-i+1)}{i!} \quad (1b)$$

We will proced on the assumption that $\alpha \ll 0.1$ –which does not necessary imply reliable transmission since $\alpha=0.1$ satisfies our condition but would be an unacceptable error probability for digital communication. Repetition codes improve reliability when α is sufficiently small that $P(I+1,n) \ll P(I,n)$ and, consequently, several errors per word are much less likely than a few errors per word.

Consider, for instance, a triple-repetition code with codeword 000 and 111. All the other received words, such as 001 or 101, clearly indicate the presence of errors. Depending on the decoding scheme, this code can detect or correct erroneous words. For error detection without correction, we say that any word other than 000 or 111 is a detected error. Single and double errors in a word are thereby detected, but triple errors result in an undetected word error with probability.

$$P_{we}=P(3,3)=\alpha^3$$

For error correction, we use majority-rule decoding based on the assumption that at least two of the three bits are correct. Thus, 001 and 101 are decoded as 000 and 111, respectively. This rule corrects words with single errors, but double or triple errors result in a decoding with probability.

$$P_{we}= P(2,3)+P(3,3)=3\alpha^2 -2\alpha^3$$

Since $P_e = \alpha$ would be the error probability without coding, we see that either decoding scheme for the triple-repetition code greatly improves reliability if, say, $\alpha \leq 0.01$. However implementation is gained at the cost of reducing the message bit rate by a factor of 1/3.

More efficient codes are based on the notion of parity. The parity of a binary word is said to be even when the word contains an even number of 1s, while odd parity means an odd number of 1s. the codewords for an error-detecting parity check code are constructed with n-1 message bits and one check bit chosen such that all codewords have the same parity. With n=3 and even parity, the valid codewords are 000,011,101, and 110, the last bit in each word being the parity, check. When a received word has odd parity, 001 for instance, we immediately know that it contains a transmission error-or three errors or, in general, an odd number of errors. Error correction is not possible because we don't know where the errors fall within the word. Furthermore, an even number of errors preserves valid parity and goes unnoticed.

Under the condition $\alpha \ll 1$, double errors occur far more often than four or more errors per word. Hence , the probability of an undetected error in an n-bit parity-check codeword is

$$P_{we} \approx P(2,n) \approx \frac{n(n-1)}{2}\alpha^2 \qquad (2)$$

For comparison purposes, uncoded transmission of words containing n-1 message bits would have

$$P_{uwe} = 1 - P(0,n-1) \approx (n-1)\alpha$$

Thus if $n=10$ and $\alpha=10^{-3}$ then $P_{uwe} \approx 10^{-2}$ whereas coding yields $P_{we} \approx 5\times10^{-5}$ with a rate reduction of just 9/10. These numbers help explain the popularity of parity checking for error detection in computer systems.

As an example of parity checking for error correction, Fig. 13.1-1 illustrates an error-correcting scheme in which the codeword is formed by arranging k message bits in a square array whose rows and columns are checked by 2squere k parity bits. A transmission error in one message bit causes a row and column

m_1	m_2	m_3
m_4	m_5	m_6
m_7	m_8	m_9
c_4	c_5	c_6

Figure 13.1-2 Interleaved check bits for error control with burst errors.

parity failure with the error at the intersection, so single errors can be corrected. This code also detects double errors.

Throughout the foregoing discussion we have assumed that transmission errors appear randomly and independently in a codeword. This assumption holds for errors caused by white noise or filtered white noise. But impulse noise produced by lightning and switching transients causes errors to occur in bursts that span several successive bits. Burst errors also appear when radio-transmission systems suffer from rapid fading. Such multiple errors wreak have on the performance of conventional codes and must be combated by special techniques. Parity checking controls burst errors if the check bits are interleaved so that the checked bits are widely spaced, as represented in where a curved line connects the message bits and check bit in one parity word.

Code Vectors and Hamming Distance

Rather than continuing a piecemeal survey of particular codes, we now introduce a more general approach in terms of code vectors. An arbitrary n-bit codeword can be visualized in an n-dimensional space as a vector whose elements or coordinates equal the bits in the codeword. We thus write the codeword 101 in row vector notation as X= (1 0 1). Figure 13.1-3 portrays all possible 3-bit codeword as dots corresponding to the vector tips in a three-dimension space. The solid dots in part (a)represent the triple- repetition code, while those in part (b) represent a parity-check code.

Notice that the triple-repetition code vectors have greater separation than the parity-check code vectors. This separation, measured in terms of the

Hamming distance, has direct bearing on the error-control power of a code. The Hamming distance d(X,Y) between two vectors X and Y is defined to equal the number of different elements. For instance, if X=(1 0 1) and Y=(1 1 0) then d(X,Y)=2 because the second and third elements are different.

The minimum distance d_{min} of a particular code is the smallest Hamming distance between valid code vectors. Consequently, error detection is always possible when the number of transmission errors in a codeword is less then d_{min} so the erroneous word is not a valid vector. Conversely, when the number of errors equals or exceeds d_{min}, the erroneous word may correspond to another valid vector and the errors be detected.

Further reasoning along this line leads to the following distance requirements for various degrees of error control capability.

Detect up to l errors per word	$d_{min} \geq l+1$	(3a)
Correct up to t errors per word	$d_{min} \geq 2\ l+1$	(4a)
Correct up to t errors and detect l>t	$d_{min} \geq 3l+1$	(3c)

By way of example, we see from **Fig.13.1-3** that the triple-repetition code has $d_{min}=3$. Hence, this code could be used to detect $t \leq 3-1=2$ errors per word or to correct $t \leq (3-1)/2=1$ error per word-in agreement with our previous observations. A more powerful code with $d_{min}=7$ could correct triple errors or it could correct double errors and detect quadruple errors.

The power of a code obviously depends on the number of bits added to each codeword for error-control purposes. In particular, suppose that the codewords consist of k<n message bits and n-k parity bits checking the message bits. This structure is known as an (n,k) block code. The minimum distance of an (n,k) block code is upper-bounded by

$$d_{min} \leq n-k+1$$

and the code's efficiency is measured by the code rate

$$R_c{}^{\Delta}=k/n$$

Regrettably, the upper bound in Eq.(4) is realized only by repetition codes, which have k=1 and very inefficient code rate $R_c=1/n$. Considerable effort has thus been devoted to the search for powerful and reasonably efficient codes, a topic we will return to in the next section.

FEC Systems

Now we are prepared to examine the forward error correction system diagrammed in Fig 13.1-4. Message bits come from an information source at the rate r_b. The encoder takes blocks of k message bits and constructed an (n,k) block code with

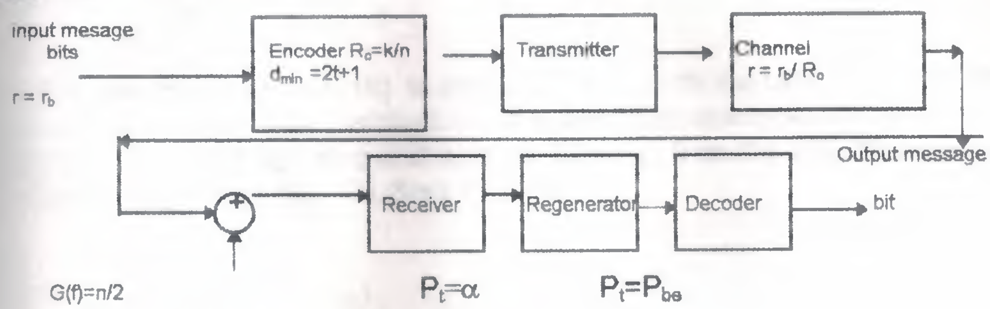


Figure 13.1-4 FEC System

code rate R_c=k/n<1 . The bit rate on the channel therefore must be greater than r_b , namely

$$r=(n/k)r_b = r_b/R_c \qquad (6)$$

The code has d_{min} =2t+1≤ n-k+1, and the decoder operates strictly in an error-correction mode. We will investigate the performance of this FEC system when additive white noise causes random errors with probability α<<1. The value of α depends, of course, on the signal energy and noise density at the receiver. If E_b represent the average energy per message bit, then the average energy per code bit is $R_c E_b$ and the ratio of bit energy to noise energy to noise density is

$$y_c = R_c E_b / n = R_c y_c \qquad (7)$$

where $y_b=E_b/n$. Our performance criterion will be the probability of output message-bit errors, denoted by P_{be} to distinguish it from the word error probability P_{we}.

The code always corrects up to t errors per word and some patterns of more than t errors may also be correctable, depending upon the specific code vectors. Thus, the probability of a decoding word error is upper-bounded by

$$P_{we} \leq \sum_{i=t+1}^{n} P(i,n)$$

For a rough but reasonable performance estimate, we will take the approximation

$$P_{we} \approx P(t+1,n) \approx \binom{n}{t+1} \alpha^{t+1} \qquad (8)$$

which means that an uncorrected word typically has t+1 bit errors. On the average, there will be (k/n)(t+1) message-bit errors per uncorrected word, the remaining errors being in check bits. When Nk bits are transmitted in $N>>1$ words, the expected total number of erroneous message bits at the output is $(k/n)(t+1)NP_{we}$. Hence,

$$P_{be} = \frac{t+1}{n} P_{we} \approx \binom{n-1}{t} \alpha^{t+1} \qquad (9)$$

in which we have used Eq.(1b) to combine $(t+1)/n$ with the binomial coefficient. If the noise has a gaussian distribution and the transmission system has been optimized (i.e., polar signaling and matched filtering), then the transmission error probability is given by Eq.(16), Sect.11.2, as

$$\alpha = Q(\sqrt{2y_c})Q(\sqrt{R_c y_b}) \qquad (10)$$
$$\approx (4pi\ R_c y_b)^{-1/2} e^{-R_c Y_b} \qquad R_c y_b \geq 5$$

The gaussian tail approximation invoked here follows, from Eq. (10), Sect.4.4,and is consistent with the assumption that $\alpha <<1$. Thus, our final result for the output error probability of the FEC system becomes

$$P_{be} = \binom{n-1}{t} [Q(\sqrt{2R_c y_b})]^{t+1} \qquad (11)$$

$$\approx \binom{n-1}{t} (4pi\ R_c y_b)^{-(t+1)/2} e^{-(t+1) R_c Y_b}$$

Uncoded transmission on the same channel would have

$$P_{ube} = Q(\sqrt{2y_b}) \approx (4pi\ y_b)^{-1/2} e^{-Y_b} \qquad (12)$$

since the signaling rate can be decreased from r_b/R_c to r_b

A comparison of Eqs(11) and (12) brings out the importance of the code parameters $t=(d_{min}-1)/2$ and $R_c=k/n$. The added complexity of an FEC system is justified provided that t and R_c yield a value of significantly less than P_{ube}. The exponential approximation show that this essentially requires $(t+1)R_c>1$. Hence, a code that only corrects single or double errors should have a relatively high code rate, while more powerful codes may succeed despite lower code rates. The channel parameter y_b also enters into the comparison, as demonstrated by the following example.

Example13.1-1 Suppose we have a (15,11) block code with $d_{min}=3$, so $t=1$ and $R_c=11/15$. An FEC system using this code would have $\alpha=Q[\sqrt{(22/15)y_b}]$ and $P_{be}=4\alpha^2$, whereas uncoded transmission on the same channel would yield $P_{ube}=Q(\sqrt{2y_b})$. These three probabilities are plotted versus y_b in dB Fig.13.1-5. If $y_b>8$ dB, we see that coding decreases the error probability by at least an order of magnitude compared to uncoded transmission. At $y_b=10$ dB, for instance, uncoded transmission yields $P_{ube}\approx 4\times10^{-6}$ whereas the FEC system has $P_{be}\approx 10^{-7}$ even through the higher channel bit rate increase the transmission error probability to $\alpha\approx 6\times10^{-5}$.

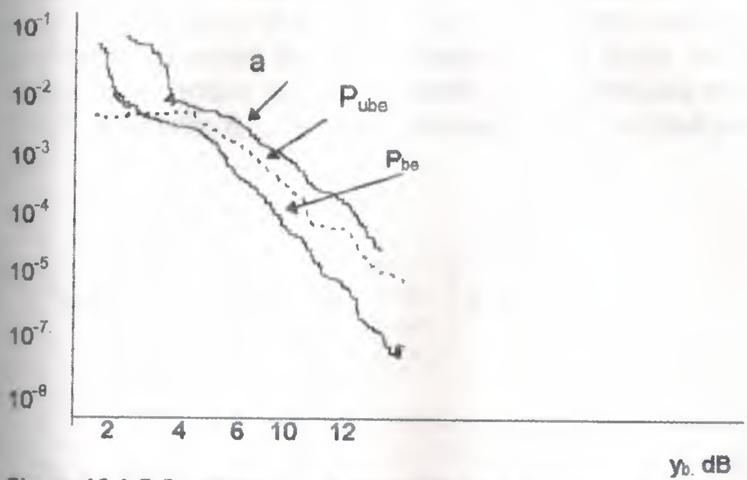


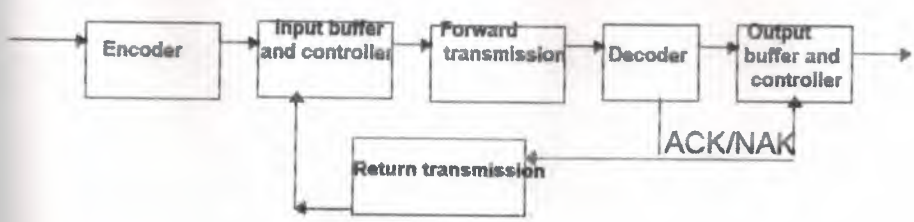
Figure 13.1-5 Curves of error probabilities in Example 13.1-1.

If y_b DB, however, coding does not significantly improve and actually makes matters worse when y_b <4 dB. Furthermore, an uncoded system could achieve better reliability that the FEC system simply by increasing the signal-to-noise ratio about 1.5 dB. Hence, this particular code doesn't save much signal much signal power, but it would be effective if y_b has a fixed value in the vicinity of 8-10 dB.

ARQ Systems

The automatic-repeat-request strategy for error control is based on error detection and retransmission rather than forward error correction. Consequently, ARQ systems differ from FEC systems in three important respects. First, an (n, k) block code designed for error detection generally requires fewer check bits and has a higher k/n ratio than code designed for error correction. Second, an ARQ system needs a return transmission path and additional hardware in order to implement repeat transmission of codewords with detected errors. Third, the forward transmission bit rate must make allowance for repeated word transmissions. The net impact of these differences becomes clearer after we describe the operation of the ARQ system represented by fig. 13.1-6.

Each codeword constructed by the encoder is stored temporarily and transmitted to the destination where the decoder looks for errors. the decoder issues positive acknowledgment (ACK) if no errors are detected, or a negative acknowledgment (NAK) if errors are detected. A negative acknowledgment causes the input controller to retransmit the appropriate word from those stored by the input buffer. A particular word may be transmitted just once or it may be transmitted two or more times, depending on the occurrence of transmission errors. The function of the output controller and buffer is to assemble the output bit stream from the codewords that have been accepted by the decoder.



Compared to forward transmission, return transmission of the ACK,NAK signal involves a low bit rate and we can reasonably assume a negligible error probability on the return path. Under this condition, all codewords with detected errors are transmitted as many times as necessary, so the only output errors appear in words with undetected errors. For an (n,k) block code with $d_{min}= i+1$, the corresponding output error probabilities are

$$P_{we}=\sum_{i=l+1}^{n} P(i,n)\approx P(l+1,n)\approx \binom{n}{l+1}\alpha^{l+1} \qquad (13)$$

$$P_{be}= \frac{l+1}{n} P_{we}\approx \binom{n-1}{l}\alpha^{l+1} \qquad (14)$$

which are identical to the FEC expressions, Eqs(8) and (9), with l in place of t. Since the decoder accepts words that have either no rrors or undetected errors. The words retransmission probability is given by

$$p\approx 1-[\, P(0,n)+P_{we}\,]$$

But a good error-detecting code should yield $P_{we} << P(0,n)$. Hence,

$$p\approx 1-P(0,n)=1-(1-\alpha)^{n}\approx n\alpha$$

where we have used the approximation $(1-\alpha)^{n}\approx 1- n\alpha$ based on $n\alpha<<1$.As for the retransmission process itself, there are three basic ARQ schemes illustrated by the timing diagrams in Fig. 13.1-7. The asterik marks words received with detected errors which must be retransmitted. The stop-and-wait scheme in part a requires the transmitter to stop after every word and wait for acknowledgment from the receiver. Just one word needs to be stored by the input buffer, but the transmission time delay t_d in which direction results in an idle time of duration $D \geq 2t_d$ between words. Idle time is eliminated by the go-back-N scheme in part b where codewords are transmitted continuously. When the receiver sends a NAK signal, the transmitter goes back N words in the buffer and

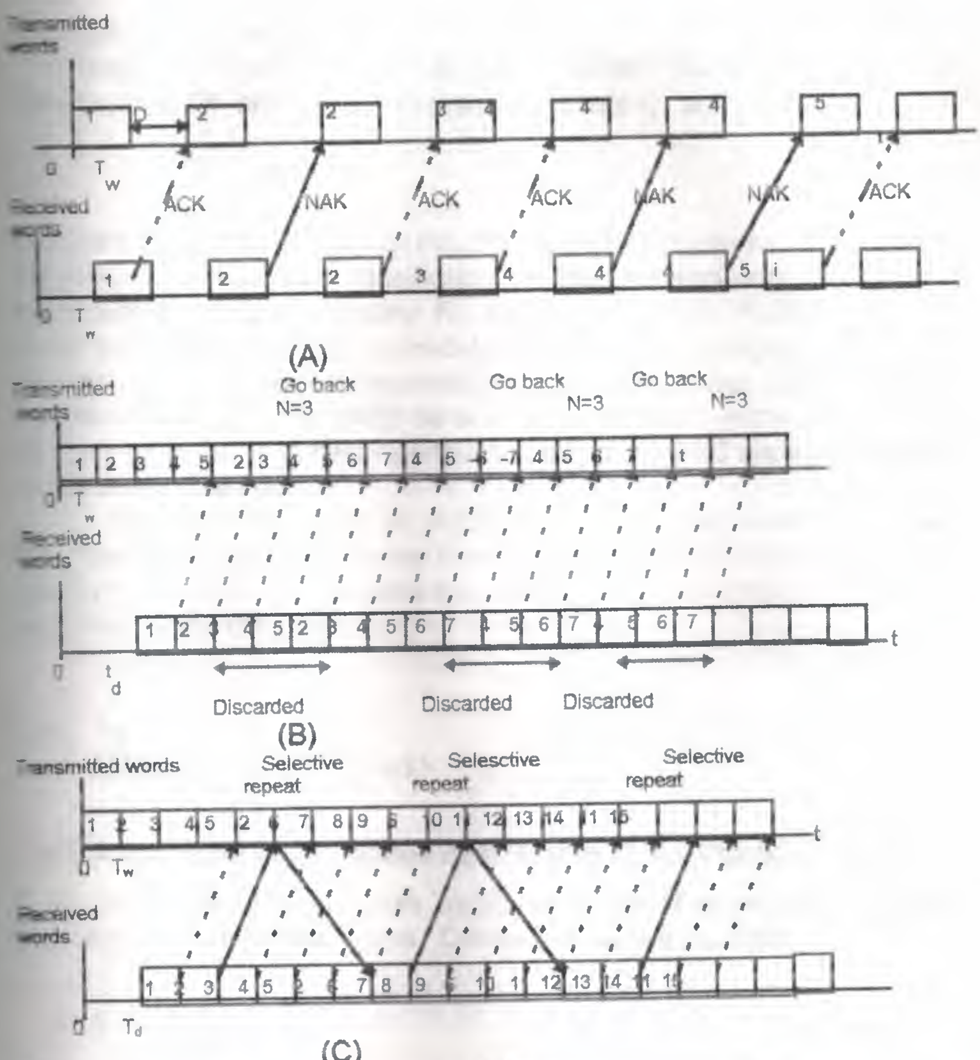


Figure 13.1-7 ARQ schemes. (A) Stop-and wait;(B)go-back-n; (C) selective-repeat

retransmits starting from that point. The receiver discards the N-1 intervening words, correct or not, in order to preserve proper sequence. The selective-repeat scheme in part c puts the burden of sequencing on the output controller and buffer, so that only words with detected errors need to be retransmitted.

Clearly, a selective.-repeat ARQ system has the highest throughput efficiency. To set this on a quantitative footing, we observe that the total number of transmission of a given word is discrete random variable m governed by the event probabilities $P(m=1)=1-p$, $P(m=2)=P(1-p)$ etc . The average number of transmitted words per accepted word is then

$$m=1(1-p)+2p(1-p)+3p^2(1-p)+\ldots\ldots$$

$$=(1-p)(1+2p+3p^2+\ldots\ldots)=\frac{1}{1-p} \qquad (16)$$

since $1+2p+3p^2+\ldots\ldots=(1-p)^{-2}$. On the average, the system must transmit nm bits for every k message bits, so the throughput efficiency is

$$R_c'=k/(nm)=(k(1-p))/n \qquad (17)$$

in which $p \approx n\alpha$, From Eq.(15).

We use the symbol R_c' here to reflect the fact that the forward-transmission bit rate r and the message bit rate r_b are related by.

$$R = r_b / R_c'$$

comparable to the relationship $r = r_b / R_c$ in an FEC system. Thus, when the noise has a gaussian distribution, the transmission error probability α is calculated from Eq.(10) using R_c' instead of $R_c = k/n$. Furthermore, if $p << 1$, then $R_c' \approx k/n$. But an error-detecting code has a larger k/n ratio than an error-correcting code of equivalent error-control power. Under these conditions, the more elaborate hardware needed for selective-repeat ARQ may pay off in terms of better performance than an FEC system would yield on the same channel.

The expression form m in Eq.(16) also applies to a stop-and wait ARQ system. However, the idle time reduces efficiency by the factor $T_w/(T_w+D)$ where is the round-trip delay and T_w is the word duration given by $T_w = n/r \leq k/r_b$. Hence,

$$R_c' \approx \frac{k}{n}\,\frac{1-p}{1+(D/T_w)} \leq \frac{k}{n}\,\frac{1-p}{1+(2t_d r_b/k)} \qquad (18)$$

in which the upper bound comes from writing $D/T_w \geq 2t_d r_b/k$.

A go back-N ARQ system has no idle time, but N words must be retransmitted for each word with detected errors. Consequently, we find that

$$m = 1 + \frac{Np}{1-p} \qquad (19)$$

and where the upper bound reflects the fact that $N \geq 2t_d/T_w$.

Unlike selective-repeat ARQ, the throughput efficiency of the stop-and-wait and go-back-N schemes depends on the round-trip delay. Equations (18) and (20) reveal that both of these schemes have reasonable efficiency if the delay and bit rate are such that $2t_d r_b << k$. However, stop-and-wait ARQ has very low efficiency when $2t_d r_b \geq k$, whereas the go-back-N scheme may still be satisfactory provided that the retransmission probability p is small enough.

Finally, we should at least describe the concept of hybrid ARQ systems. These systems consist of an FEC subsystem within the ARQ framework, thereby combining desirable properties of both error-control strategies. For instance, a hybrid ARC system might employ a block code with $d_{min} = t+l+1$, so the decoder can correct up to t errors per word and detect but not correct words with $l > t$ errors. Errors correction reduces the number of words that must be retransmitted, thereby increasing the throughput without sacrificing the higher reliability of ARQ.

13.2 Linear Block codes

This section describe the structure, probabilities, and implementation of block codes. We start with a matrix representation of the encoding process that generates the check bits for a given block of message bits. Then we use the matrix representation to investigate decoding methods for error detection and correction. The section closes with a brief introduction to the important class of cyclic block codes.

Matrix Representation of Block Codes

An (n,k) block code consist of n-bit vectors, each vector corresponding to a unique block of k<n message bit. Since there are different k-bit message blocks and 2^n possible n-bit vectors, the fundamental strategy of block coding is to choose the 2^k code vectors such that the minimum distance is as large as possible. But the code should also have some structure that facilities the encoding and decoding process. We will therefore focus on the class of systematic linear block codes.

Let an arbitrary code vector be represented by

$$X = (x_1\ x_2\ \ldots\ldots\ x_n)$$

where the elements $x_1\ x_2\ \ldots\ldots$ are, of course, binary digits. A code is linear if it includes the all-zero vector and if the sum of any code vectors produces another vector in the code. The sum of two vectors, say X and Z, is defined as

$$X+Z=(x_1 \oplus z_1\quad x_2 \oplus z_2 \ldots\ldots\ldots x_n z_n) \qquad (1)$$

in which the elements are combined according to the rules of mod-2 additional given in Eq..(2), Sect.11.4.

As a consequence of linearity, we can determine a code's minimum distance by the following argument. Let the number of nonzero elements of a vector X be symbolized by w(X), called the vector weight. The Hamming distance between any two code vectors X and Z is then

$$d(X,Z)=w(X+Z)$$

since $x_1 \oplus z_1=1$ if $x_1 \neq z_1$ etc. The distance between X and Z therefore equals the weight of another code vetor X+Z. But if Z=(0 0…..0) then X+Z=X; hence,

$$d_{min}=[w(X)]_{min} \qquad X\neq(0\ 0\ \ldots.0) \qquad (2)$$

In other words, the minimum distance of a linear block code equals the smallest nonzero vector weight.

A systematic block code consists of vectors whose first k elements(or last k elements) are identical to the message bits, the remaining n-k elements being check bits. A code vector then takes the form

$$X=(m_1\ m_2 \ldots\ldots\ m_k\ \ c_1\ c_2\ \ldots..c_q) \qquad (3a)$$

where

$q = n-k$

For convenience, we will also code vectors in the partitioned notation

$$X=(M \mid C)$$

in which M is a k-bit message vector and C is a q-bit check vector. Partitioned notations lends itself to the matrix representation of block codes.

Given a message vector M, the corresponding code vector X for a systematic linear (n,k) block code can be obtained by a matrix multiplication.

$$X=MG \qquad (4)$$

The matrix G is a k x n generator matrix having the general structure

$$G \triangleq [I_k \mid P]. \qquad (5a)$$

where I_k is the k x k identity matrix and P is a k x q submatrix od binary digits represented by

$$P=\begin{bmatrix} p_{11} & p_{12} \ldots\ldots & p_{1q} \\ p_{21} & p_{13} \ldots\ldots & p_{2q} \\ . & . & . \\ . & . & . \\ p_{k1} & p_{k2} \ldots\ldots & p_{kq} \end{bmatrix} \qquad (5b)$$

The identity matrix in G simply reproduces the message vector for the first k elements of X, while the submatrix P generates the check vector via

$$C=MP$$

This binary matrix multiplication follows the usual rules with mod-2 addition instead of conventional addition. Hence, the jth element of C is computed using the jth column of P , and

$$c_j=m_1p_{1j} + m_2p_{2j} + \ldots\ldots\ldots\ldots + m_kp_{kj} \qquad \text{(6b)}$$

for j=1,2,3,........q. All of these matrix operations are less formidable than they appear because every element equals either 0 or 1.

The matrix representation of a block code provides a compact analytical vehicle and, moreover, leads to hardware implementations of the encoder and decoder. But it does not tell us how to pick the elements of the P submatrix to achieve specified code parameters such as d_{min} and R_c . Consequently, good codes are discovered with the help of considerable inspiration and perspiration, guided by mathematical analysis. In fact, Hamming(1950) devised the first popular block codes several years before the underlying theory was formalized by Slepian(1956).

Example 13.2-1 Hamming code: A hamming code is an (n,k) linear block code with $q \geq 3$ check bits and

$$n=2^q-1 \qquad k=n+q \qquad (7a)$$

The code rate is

$$R_c=\frac{k}{n}=1-\frac{q}{2^q-1} \qquad (7b)$$

and thus $R_c \approx 1$ if $q>>1$. Independent of q, the minimum distance is fixed at

$$d_{min}=3 \qquad (7c)$$

so a Hamming code can be used for single-error correction or double detection. To construct a systematic Hamming code, you simply let the k rows of the P submatrix consist of q-bit words with two or more is, arranged in any order.

For example, consider systematic Hamming code with q=3, so $n=2^3-1=7$ and $k=7-3=4$. According to the previously stated rule, an appropriate generator matrix is

$$G=\left[\begin{array}{cccc|ccc} 1&0&0&0&1&0&1 \\ 0&1&0&0&1&1&1 \\ 0&0&1&0&1&1&0 \\ 0&0&0&1&0&1&1 \end{array}\right]$$

The last three columns constitute the P submatrix whose rows inculude all 3-bit words that have two or more Is. Given a block of message bits M=

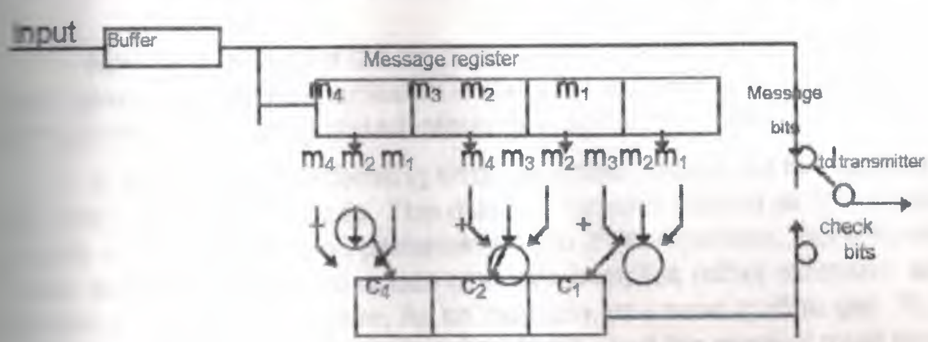


Figure 13.2-1 Encoder for (7,4) Hamming code.

$(m_1\, m_2\, m_3\ \ m_4)$, the check bits are determined from the set of equations

$$c_1= m_1 \oplus m_2 \oplus m_3 \oplus 0$$
$$c_2= 0 \oplus m_2 \oplus m_3 \oplus m_4$$
$$c_3= m_1 \oplus m_2 \oplus 0 \oplus m_4$$

These check-bit equations are obtained by substituting the elements of P into Eq.(6).

Figure 13.2-1 depicts an encoder that carries out the check-bit calculations for this(7,4) Hamming code. Each block of message bits going to the transmitter is also loaded into a message register. The cells of the message register are connected to exclusive-OR gates whose outputs equal the check bits. The check bits are stored in another register and shifted out to the transmitter after the message bits. An input buffer holds the next block of message bits while the check bits are shifted out. The cycle then repeats with the next blocks of message bits.

Table 13.2-1 lists the resulting $2^4 = 16$ codewords and their weights. The smallest nonzero weight equals 3, confirming that $d_{min}=3$.

Table 13.2-1 Codewords for the (7,4) Hamming code

M	C	w(X)	M	C	w(X)
0000	000	0	1000	101	3
0001	011	3	1001	110	4
0010	110	3	1010	011	4
0011	101	4	1011	000	3
0100	111	4	1100	010	3
0101	100	3	1101	001	4
0110	001	3	1110	100	4
0111	010	4	1111	111	7

write the check-bit equations and tabulate the codewords and their weights to show that $d_{min}=3$.

Syndrome Decoding

Now let Y stand for the received vector when a particular code vector X has been transmitted. Any transmission errors will result in $Y \neq X$. The decoder detects or corrects errors in Y using stored information about the code.

A direct way of performing error detection would be to compare Y with every vector in the code. This method requires storing all 2^k code vectors at the receiver and performing up to 2^k comparison. But efficient codes generally have large values of k, which implies rather extensive and expensive decoding hardware. As an example, you need $q \geq 5$ to get $R_c \geq 0.8$ with a Hamming code; then $n \geq 31$, $k \geq 26$, and the receiver must store a total of $n \times 2^k > 10^9$ bits1!.

More practical decoding methods for codes with large k involve parity-check information derived from the code's P submatrix. Associated with any systematic linear (n,k) block code is a q x n matrix H called the parity-check matrix. This matrix is defined by

$$H^{T} \overset{\Delta}{=} \begin{bmatrix} P \\ I_q \end{bmatrix} \qquad (8)$$

Where H^T denotes the transpose of H and I_q is the q x q identity matrix. Relative to error detection, the parity-check matrix has the crucial propety.

$$X\,H^T=(0\ 0 .....0) \qquad (9)$$

provided that X belongs to the set of code vectors. However, when Y is not a code vector, the product YH^T contains at least one nonzero element.

Therefore, given H^T and a received vector Y, error detection can be based on

$$S= YH^T \qquad (10)$$

a q-bit vector called the syndrome. If all elements of S equal zero, then either equals the transmitted vector X and there are no transmission errors, or Y equal some other code vector and the transmission errors are undetectable. Otherwise errors are indicated by the presence of nonzero elements in S. Thus, a decoder for error detection simply takes the form of a syndrome calculator. A comparison of Eqs.(10) and (6) shows that the hardware needed is essentially the same as the encoding circuit.

Error correction necessarily entails more circuitry but it, too, can be based on the syndrome. We develop the decoding method by introducing an n-bit error vector E whose nonzero elements mark the positions of transmission errors in Y. For instance, if X=(1 0 1 1 0) and Y=(1 0 0 1 1) then E=(0 0 1 0 1). In general

$$Y=X+E \qquad (11a)$$

and conversely,

$$X=Y+E \qquad (11b)$$

since a second error in the same bit location would cancel the original error. Substituting Y=X+E into $S= YH^T$ and invoking Eq(9), we obtain

$$S=(X+E)H^T= XH^T+ EH^T= EH^T \qquad (12)$$

which reveals that the syndrome depends entirely on the pattern, not the specific transmitted vector.

However, there are only 2^q different syndromes generated by the 2^n possible n-bit error vectors, including the no-error case. Consequently, a given syndrome does not 2^q uniquely determine by the E . Or, putting this another way, we can correct just patterns with one or more errors, and the remaining patterns are uncorrecttable. We should therefore design the decoder to correct the most likely error patterns-namely those patterns with the fewest errors, since single errors are more probable than double errors, and so forth. This strategy, known as maximum-likelihood decoding, is optimum in the sense that it minimize the word error probability. Maximum-likelihood decoding corresponds to choosing the code vector that has the smallest Hamming distance from the received vector.

To carry out maximum-likelihood decoding, you must first compute the syndromes generated by the 2^q-1 most probable error vectors. The table-lookup decoder diagrammed in Fig 13.2-2 then operates as follows. The decoder calculates S from the received vector Y and looks up the assumed error vector E stored in table. The sum Y+E generated by exclusive-OR gates finally constitutes the code word. If there are no errors, or if the errors are uncorrectable, then S=(0 0 0) so Y+E=Y. The check bits in the last q elements of Y+E may be omitted if they are of no further interset.

The relationship between syndromes and error patterns also sheds some light on the design of error-correcting codes, since each of the 2^q-1 nonzero syndromes must represent a specific error pattern. Now there are single-error patterns for an n-bit word, double-error patterrns, and so forth. Hence, if a code is to correct up to t errors per word, q and must satsify

$$2^q-1 \geq n + \binom{n}{2} + \dots\dots\dots + \binom{n}{t} \qquad (13)$$

In the particular case of a single-error-correcting code, Eq(13) reduces to $2^q-1 \geq n$. Furthermore, when E corresponds to a single error in the jth bit of a codeword, we find from Eq(12) that S is identical to the jth row of H^t. Therefore, to provide a distinct syndromes for each single-error pattern and for the no error pattern, the rows of H^t (or columns of H) must all be different and each must contain at least one nonzero element. The generator matrix of a Hamming code is designed to satisfy these requirements on H, while q and n satisfy 2^q-1=n.

Example 13.2-2 Let's apply table-lookup decoding to a (7,4) Hamming code used for single-error correction. From Eq.(8) and the P submatrix given in Example 13.2-1, we obtain the 3 x 7 parity-check matrix.

$$H=[P^t | I_q]= \left[\begin{array}{cccc|ccc} 1&1&1&0&1&0&0 \\ 0&1&1&1&0&1&0 \\ 1&1&0&1&0&0&1 \end{array}\right]$$

There are 2^3-1=7 correctable single-error patterns, and the corresponding syndromes listed in Table 13.2-2 follow directly from the columns of H. To accommodate this table the decoder needs to store only (q+n)x 2^q=80 bits

Table 13.2-2 Syndromes for the (7,4) Hamming code

S	E
0 0 0	0 0 0 0 0 0 0
1 0 1	1 0 0 0 0 0 0
1 1 1	0 1 0 0 0 0 0
1 0 1	0 0 1 0 0 0 0
0 1 1	0 0 0 1 0 0 0
1 0 0	0 0 0 0 1 0 0
0 1 0	0 0 0 0 0 1 0
0 0 1	0 0 0 0 0 0 1

But suppose a received word happens to have two errors, such that E=(1 0 0 0 0 1 0). The decoder calculates S= YH^T=EH^T=(1 1 1) and the syndromes table gives the assumed single-error pattern

$E=(0\ 1\ 0\ 0\ 0\ 0\ 0)$. The decoded output word Y+E therefore contains three errors: the two transmission errors plus the erroneous correction added by the decoder.

If multiple transmission errors per word are sufficiently infrequent, we need not be concerned about the occasional extra errors committed by the decoder. If multiple errors are frequent, a more powerful code would be required. For instance, an extended Hamming code has an additional check bit that provides double-error detection along with single-error correction; see Prob13.2-12.

Exercises 13.2-2 Use Eqs. (8) and (10) to show that the jth bit of S given by

$$s_j = y_1p_{1j} \oplus y_2p_{2j} + \oplus \ldots\ldots \oplus y_kp_{kj} \oplus y_{k+j}$$

Then diagram the syndrome-calculation circuit for a (7,4) Hamming code, and compare it with Fig.13.2-1.

Cyclic Codes

The code for a forward-error-correction system must be capable of correcting $t \geq 1$ errors per word. It should also have a reasonably efficient code rate $R_c=k/n$. These two parameters are related by the inequality

$$1-R_c \geq \frac{1}{n}\log_2\left[\sum_{i=0}^{t}\binom{n}{i}\right]$$

which follows from Eq.(13) with $q = n-k = n(1-R_c)$. This inequality underscores the fact that if we want $R_c \approx 1$, we must use codewords with n>>1 and k>>1. However, the hardware requirements for encoding and decoding long codewords may be prohibitive unless we impose further structural conditions on the code. Cylic codes are a subclass of linear block codes with a cyclic structure that leads to more practical implementation. Thus, block codes used in FEC systems are almost always cyclic codes.

To describe a cyclic code, we will find it helpful to change our indexing scheme and express an arbitrary n-bit code vector in the form

$$X=(x_{n-1}\ \ x_{n-2} \ldots\ldots x_1\, x_0) \qquad (15)$$

Now suppose that X has been loaded into a shift register with feedback connection from the first to last stage. Shifting all bits one position to the left yields the cyclic shift of X, written as

$$X' \triangleq (x_{n-2}\ \ x_{n-3} \ldots\ldots x_1\, x_0\, x_{n-1}) \qquad (16)$$

A second shift produces $X'' = (x_{n-3} \ldots\ldots x_1\, x_0\, x_{n-1}\, x_{n-2})$ and so forth. A linear code is cyclic if every cyclic shift of a code vector X is another vector in the code. This cyclic property can be treated mathematically by associating a code vector X with the polynomial

$$X(p) = x_{n-1}p^{n-1} + x_{n-2}p^{n-2} + \ldots\ldots x_1p + x_0 \qquad (17)$$

and the syndromes table gives the assumed single-error pattern

E=(0 1 0 0 0 0 0). The decoded output word Y+E therefore contains three errors: the two transmission errors plus the erroneous correction added by the decoder.

If multiple transmission errors per word are sufficiently infrequent, we need not be concerned about the occasional extra errors committed by the decoder. If multiple errors are frequent, a more powerful code would be required. For instance, an extended Hamming code has an additional check bit that provides double-error detection along with single-error correction; see Prob13.2-12.

Exercises 13.2-2 Use Eqs. (8) and (10) to show that the jth bit of S given by

$$s_j = y_1p_{1j} \oplus y_2p_{2j} \oplus \dots \oplus y_kp_{kj} \oplus y_{k+j}$$

Then diagram the syndrome-calculation circuit for a (7,4) Hamming code, and compare it with Fig.13.2-1.

Cyclic Codes

The code for a forward-error-correction system must be capable of correcting $t \ge 1$ errors per word. It should also have a reasonably efficient code rate $R_c = k/n$. These two parameters are related by the inequality

$$1-R_c \ge \frac{1}{n}\log_2\left[\sum_{i=0}^{t}\binom{n}{i}\right]$$

which follows from Eq.(13) with $q = n-k = n(1-R_c)$. This inequality underscores the fact that if we want $R_c \approx 1$, we must use codewords with n>>1 and k>>1. However, the hardware requirements for encoding and decoding long codewords may be prohibitive unless we impose further structural conditions on the code. Cylic codes are a subclass of linear block codes with a cyclic structure that leads to more practical implementation. Thus, block codes used in FEC systems are almost always cyclic codes.

To describe a cyclic code, we will find it helpful to change our indexing scheme and express an arbitrary n-bit code vector in the form

$$X=(x_{n-1}\ x_{n-2} \dots\dots x_1\, x_0) \qquad (15)$$

Now suppose that X has been loaded into a shift register with feedback connection from the first to last stage. Shifting all bits one position to the left yields the cyclic shift of X , written as

$$X' \triangleq (x_{n-2}\ x_{n-3} \dots\dots x_1\, x_0\, x_{n-1}) \qquad (16)$$

A second shift produces $X'' = (x_{n-3} \dots\dots x_1\, x_0\, x_{n-1}\, x_{n-2})$ and so forth. A linear code is cyclic if every cyclic shift of a code vector X is another vector in the code. This cyclic property can be treated mathematically by associating a code vector X with the polynomial

$$X(p)= x_{n-1}p^{n-1}+ x_{n-2}p^{n-2}+\dots\dots x_1p+x_0 \qquad (17)$$

where p is an arbitrary real variable. The powers of p denote the positions of the codeword bits represented by the corresponding coefficients of p. Formally, binary code polynomials are defined in conjunction with Galois fields, a branch of modern algebra that provides the theory needed for a complete treatment of cyclic codes. For our informal overview of cyclic codes we will manipulate code polynomials using ordinary algebra modified in two respects. First, to be in agreement with our earlier definition for the sum of two code vectors, the sum of two polynomials is obtained by mod-2 addition of their respective coefficients. Second, since all coefficients are either 0 or 1, and since $1 \oplus 1=0$, the subtraction operation is the same as mod-2 addition. Consequently, if $X(p)+Z(p)=0$ then $X(p)=Z(p)$.

We develop the polynomial interpretation of cyclic shifting by comparing

$$pX(p)= x_{n-1}p^n+ x_{n-2}p^{n-1}+........x_1p^2+x_0p$$

with the shifted polynomial

$$X^{I}(p)= x_{n-2}p^{n-1}+ ....x_1p^2+x_0p+x_{n-1}$$

If we sum these polynomials, noting that $(x_1 + x_1)p^2=0$, etc., we get

$$pX(p)+ X^{I}(p)= x_{n-1}p^n +x_{n-1}$$

and hence

$$X^{I}(p)= pX(p)+ x_{n-1}(p^n+1)$$

Iteration yields similar expressions for multiple shifts.

The polynomial p^n+1 and its factors play major roles in cyclic codes. Specifically, an (n,k) cyclic code is defined by a generator polynomial of the form

$$G(p)=p^q+g_{q-1}p^{q-1}+..........+g_1p+1 \qquad (19)$$

where q = n-k and the coefficients are such that G(p) is a factor of p^n+1.Each codeword then corresponds to the polynomial product

$$X(p)=Q_M(p)G(p)$$

in which $Q_M(p)$ represent a block of k message bits. All such codeword satisfy the cyclic condition in Eq.(18) since G(p) is a factor of both X(p) and p^n+1. Any factor of p^n+1 that has degree q may serve as the generator polynomial for a cyclic code, but it does not necessarily generate a good code. Table 13.2-3 lists the generator polynomials of selected cyclic codes that have been demonstrated to posses desirable parameters for FEC systems. The table includes some cyclic Hamming codes, the famous Golay code, and a few members of the important family of BCH codes discovered by Bose, Chaudhuri, and Hocquenghem. The entries under G(p) denote the polynomial's coefficients; thus, for instance, 1 0 1 1 means that $G(p)=p^3+0+p+1$

Type	n	k	R_c	d_{min}	G(p)
Hamming code	7	4	0.57	3	1 011
	15	11	0.73	3	10 011
	31	26	0.84	3	100 101
BCH codes	15	7	0.46	5	111 010 001
	31	21	0.68	5	11 101 101 001
	63	45	0.71	7	1 111 000 001 011 001 111
Golay code	23	12	0.52	7	101 011 100 011

Cyclic codes may be systematic or nonsystematic, depending on the term $Q_M(p)$ in Eq.(20). For a systematic code, we define the message-bit and check-bit polynomials

$$M(p)=m_{k-1}p^{k-1}+\ldots\ldots+m_1p+m_0$$
$$C(p)=c_{q-1}p^{q-1}+\ldots\ldots+c_1p+c_0$$

and we want the codeword polynomials to be

$$X(p)=p^qM(p)+C(p) \qquad (21)$$

Equations (20) and (21) therefore require $p^qM(p)+C(p)=Q_M(p)G(p)$, or

$$\frac{p^qM(p)}{G(p)}=Q_M(p)+\frac{C(p)}{G(p)} \qquad (22a)$$

This expression says that C(p) equals the remainder left over after dividing $p^qM(p)$ by G(p), just as 14 divided by 3 leaves a remainder of 2 since 14/3=4+2/3. Symbolically, we write

$$C(p)=\text{rem}\left[\frac{p^qM(p)}{G(p)}\right]$$

where rem [ ]stands for the remainder of the division within the brackets.

The division operation needed to generate a systematic cyclic code is easily and efficiently performed by the shift-register encoder diagrammed in Fig.13.2-3

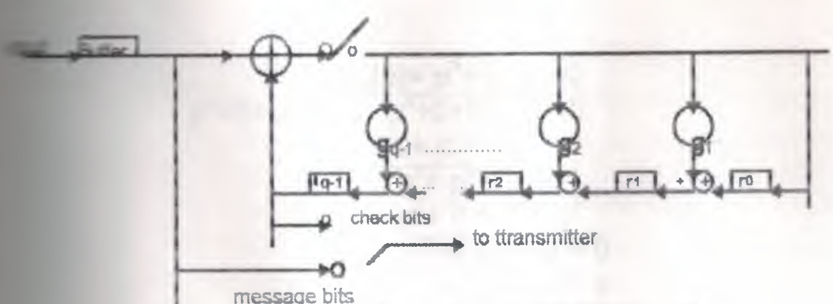


Figure 13.2-3 Shift-register encoder

Encoding starts with the feedback switch closed, the output switch in the message-bit position, and the register initialized to the all-zero state. The k message bits are shifted into the register and simultaneously delivered to the transmitter. After k shift cycles, the register contains the q check bits. The feedback switch is now opened and the output switch is moved to deliver the check bits to the transmitter 1.

Syndrome calculation at the receiver is equally simple. Given a received vector Y, the syndrome is determined from

$$S(p)=\text{rem}\left[\frac{Y(p)}{G(p)}\right] \qquad (23)$$

if Y(p) is a valid code polynomial, then G(p) will be factor of Y(p) and Y(p)/G(p) has zero remainder. Otherwise we get a nonzero syndrome polynomial indicating detected errors.

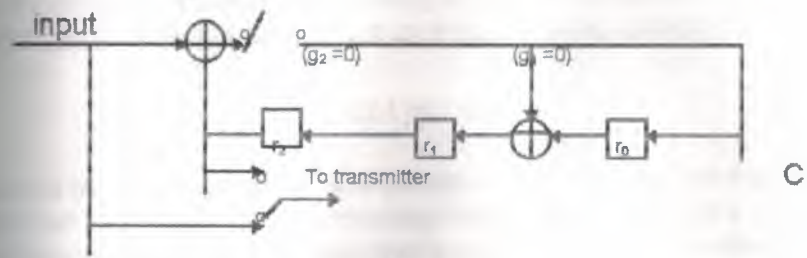
Besides simplified encoding and syndrome calculation, cyclic codes have other advantages over noncyclic block codes. The foremost advantage comes from the ingenious error-correcting decoding methods that have been devised for specific cyclic codes. These methods eliminate the storage needed for table lookup decoding and thus make it practical to use powerful and efficient codes with n>>1. Another advantage is the ability of cyclic codes to detect error bursts that span many successive bits. Detailed exposition of these properties are presented in texts such as Lin and Costello(1983).

Example 13.2-3 Consider the cyclic (7,4) Hamming code generated by $G(p)=p^3+0+p+1$. We will use long division to calculate the check-bit polynomial C(p) when M=(1 1 0 0). We first write the message-bit polynomial $M(p)=p^3+p^2+0+0$ so $p^qM(p)=p^6+p^5+0+0+0+0+0$. Next, we divide G(p) into $p^qM(p)$, keeping in mind that subtraction is the same as addition in mod-2 arithmetic . Thus,

$$
\begin{array}{r l}
 & \underline{Q_M(p)=p^3+p^2+p+0} \\
p^3+0+p+1\,| & p^6+p^5+0+0+0+0+0 \\
 & \underline{p^6+0+p^4+p^3} \\
 & \quad p^5+p^4+p^3+0 \\
 & \quad \underline{p^5+0+p^3+p^2} \\
 & \qquad p^4+0+p^2+0 \\
 & \qquad \underline{p^4+0+p^2+p} \\
 & \qquad\quad 0+0+p+0 \\
 & \qquad\quad \underline{0+0+0+0} \\
 & \qquad\quad C(p)=0+p+0
\end{array}
$$

so the complete code polynomial is

$$X(p)= p^3M(p)+C(p)= p^6+ p^5+0+0+0+p+0$$



(a)

Input bit m	Register bits before shift r_2	r_1	r_0	Register bits after shift $r_2' = r_1$	$r_1' = r_0 + r_2 + m$	$r_0' = r_2 + m$
1	0	0	0	0	1	1
1	0	1	1	1	0	1
0	1	0	1	0	0	1
0	0	0	1	0	1	0

(b)

Figure 13.2-4(a) Shift-register encoder for (7,4) Hamming code; (b) register bits when M=(1 1 0 0).

Which corresponds to the codeword

$$X=(1\ 1\ 0\ 0\ |0\ \ 1\ 0).$$

You will find this codeword back in Table 13.2-1, where you will also find the cyclic shift X'=(1 0 0 0 |1 0 1) and all multiples shifts.

Finally, Fig 13.2-4 shows the shift-register encoder and the register bits for each cycle of the encoding process when the input is M=(1 1 0 0). After four shift cycles, the register holds C=(0 1 0)--- in agreement with our manual division.

Exercises 13.2-3 Let Y(p)=X(p)+E(p) where E(p) is the error polynomial. Use Eqs.(20) and (23) to show that the syndrome polynomial S(p) depends on E(p) but not on X(p).

CONVOLUTIONAL CODES

Convolutional codes have a structure that efficiently extends over the entire transmitted bit stream, rather than being limited to codeword blocks. The convolutional structure is especially well suited to space and satellite communication systems that require simple encoders and achieve high performance by sophisticated decoding methods. Our treatment of this important family of codes consists of selected examples that introduce the salient features of convolutional encoding and decoding..

Convolutional Encoding

The fundamental hardware unit for convolutional encoding is a tapped shift register with L+1 stages, as diagrammed in Fig.13.3-1. Each tap gain g is binary digit epresenting a short-circuit connection or an open circuit. The message bits in the egister are combined by mod-2 additional to form the encoding bit.

$$X_j = X_{j-L}g_L \oplus \cdots\cdots \oplus X_{j-1}g_1 \oplus X_j g_0$$

$$= \sum_{i=0}^{L} m_{j-1} g_i \qquad \text{(mod-2)}$$

The name convolutional encoding comes from the fact that Eq(1) has the form a binary convolutional, analogous to the convolutional integral

$$x(t)=\int m(t-\lambda)g(\lambda)d\lambda$$

Notice that x_j depends on the current input m_j and on the state of the register defined by the previous L message bits. Also notice that a particular bit message influences a span of L+1 successive encoded bits as it shifts through the register.

To provide the extra bits needed for error control, a complete convolutional encoder must generate output bits at a rate greater than the message bit rate r_b. This is achieved by connecting two or more mod-2 summers to the register interleaving the encoded bits via a commutator switch. For example, the encoder in Fig.13.3-2 generates n=2 encoded bits

$$x'_j = m_{j-2} \oplus m_{j-1} \oplus m_j \qquad x''_j = m_{j-2} \oplus m_j$$

which are interleaved by the switch to produce the output steam

$$X = x'_1\ x''_1\ x'_2\ x''_1\ x'_2\ x'_3\ X''_3 \ldots\ldots$$

The output bit rate is therefore $2r_b$ and the code rate is $R_c=1/2$ - like an (n,k) block code with $R_c= k/n=1/2$.

However, unlike a block code, the input bits have not been grouped into words. Instead, each message bit influences a span of n(L+1)=6 successive output bits. The quantity n(L+1) is called the constraint length measured in terms

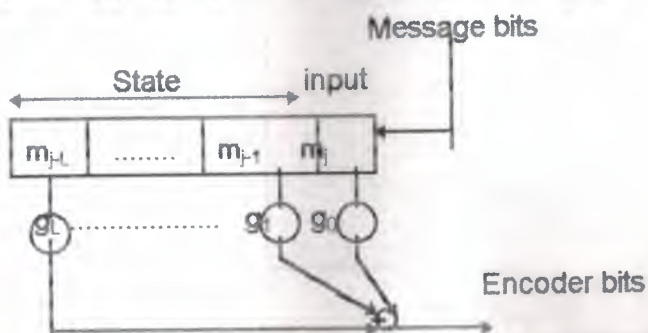


Figure 13.3-1 Tapped shift register for convolutional encoding

of encoded output bits, whereas L is the encode's memory measured in terms of input message bits. We say that this encoder produces an (n,k,L) convolutional code with n=2, k=1, and L=2.

Three different but related graphical representation have been devised for the study of convolutional encoding: the code tree, the code trellis, and the state diagram. We will present each of these for our (2,1,2) encoder in Fig 13.3-2, starting with the code tree. In accordance with normal operating procedure, we presume that the register has been cleared to contain all 0s when the first message bit m_1 arrives . Hence, the initial state is $m_{-1}m_0=00$ and Eq(2) gives the output $x'_1\ x''_1=00$ if $m_1=0$ or $x'_1\ x''_1=11$ if $m_1=1$.The code tree drawn in Fig.13.3-3 begins at a branch point the node labeled a representing the initial state. If $m_1=0$, you take the upper branch from node a to find the output 00 and the output 00 and the next state, which is also labeled a since $m_0m_1=00$ in this case. If $m_1=1$, you take the lower branch from a to find the output 11 and the next state $m_0m_1=01$ signified by the label b. The code tree progressively evolves in this fashion for each new input bit. Nodes are labeled with letters denoting the current state $m_{j-2}m_{j-1}$; you go up down from a node, depending on the value of m_j; each branch shows the resulting encoded output x'_j, x''_j calculated from Eq(2), and it terminates at another node labeled with the next state. There are 2^j possible branches for the jth message bit, but the branch pattern begins to repeat at j=3 since the register length is L+1=3. Having observed repetition in the code tree, we can construct a more compact picture called the code trellis and shown in Fig.13.3-4a. Here, the nodes on the left denote the four possible current states, while those on the right are the resulting next states. A solid line represent the state transition or branch for $m_j=0$, and a broken line represents the branch for $m_j=1$. Each branch is labeled with the resulting output bits x'_j, x''_j. Going one step further, we coalesce the left and right sides of the trellis to obtain the state diagram in Fig.13.3-4b. The self-loops at nodes a and d represent the state transitions a-a and d-d.

Given a sequence of message bits and the initial state, you can use either the code trellis or state diagram to find the resulting state sequence and output bits. The procedure is illustrated in Fig.13.3-4c, starting at initial state a.

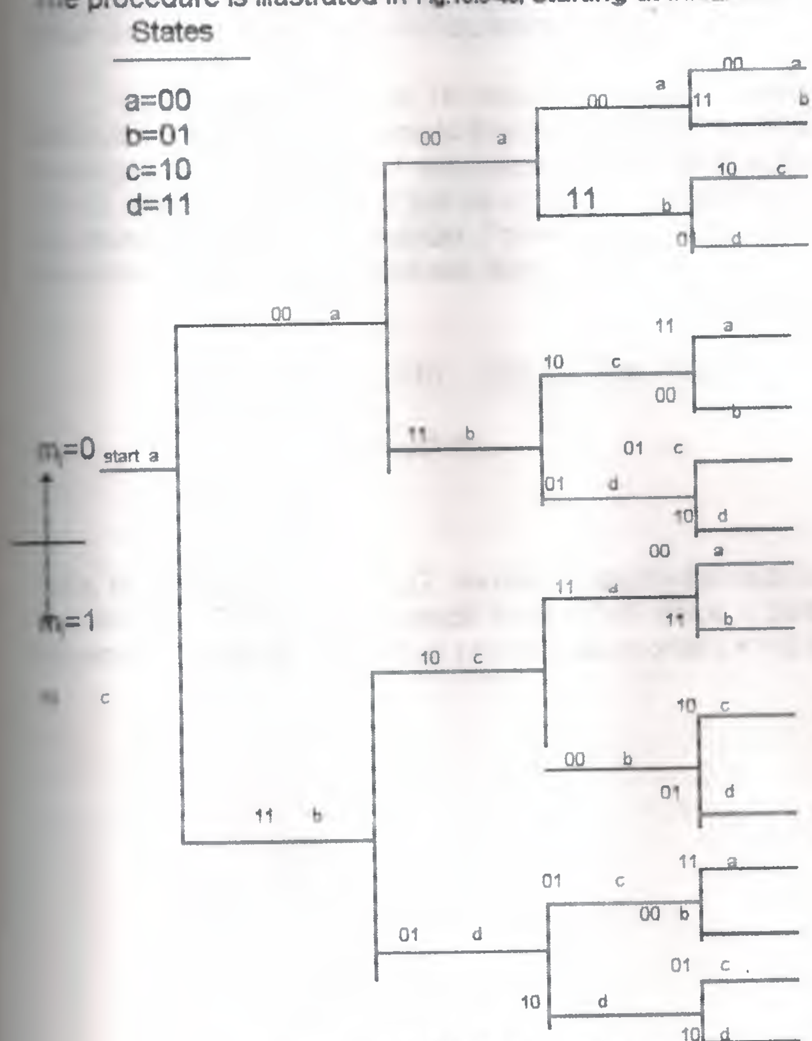
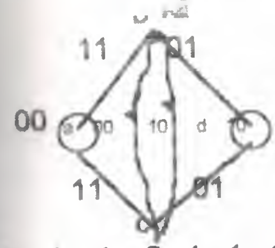


Figure 13.3-3 Code tree for(2,1,2) encoder.



(b)

input	1	1	0	1	1	1	0	0	1	0	0	0
state a	b	d	c	b	d	d	c	a	b	c	a	a
output	11	01	01	00	01	10	01	11	11	10	11	00

(c)

Figure 13.3-4-(a)Code trellis; (b) state diagram for (2,1,2)encoder

Numerous other convolutional codes are obtained by modifying the encoder in Fig.13.3-2 . If we just change the connections to the mod-2 summers, then the code tree, trellis, and state diagram retain the same structure since the state and branching pattern reflect only the register contents. The output bits would be different, of course, since they depend specifically on the summer connections.

If we extend the shift register to an arbitrary length L+1 and connect it to $n \geq 2$ mod-2 summers, we get an (n,k,L) convolutional code with k=1 and code rate $R_c=1/n \leq 1/2$. The state of the encoder is defined by L previous input bits, so the code trellis and state diagram have 2^L different states, and the code-tree pattern repeats j=L+1 branches. Connecting one comutator terminal directly to the first stage of the register yields the encoded bit stream

$$X=m_1x''_1\,x'''_1 \ldots\ldots\, m_2x''_2\,x'''_2 \ldots\ldots\, m_3x''_3\,x'''_3 \ldots\ldots$$

which defines a systematic convolutional code with $R_c=1/n$.

Code rates higher than 1/n require $k \geq 2$ shift registers and an input distributor switch. This scheme is illustrated by the (3,2,1) encoder in Fig.13.3-6. The message bits are distributed alternately between k=2 registers, each of length L+1=2. We regard the pair of bits $m_{j-1}m_j$ as the current input, while the pair $m_{j-3}m_{j-2}$ constitute the state of the encoder. For each input pair, the mod-2 summers generate n=3 encoded output bits given by

$$X'_j= m_j \oplus m_{j-2} \oplus m_j \qquad X''_j= m_{j-3} \oplus m_{j-1} \oplus m_j$$

$$X'''_j= m_{j-2} \oplus m_j \qquad (4)$$

Thus, the output bit rate is $3r_b/2$ corresponding to the code rate $R_c=k/n=2/3$. The constraint length is n(L+1)=6 since a particular input bit influences a span of n=3 output bits from each of its L+1=2 register positions.

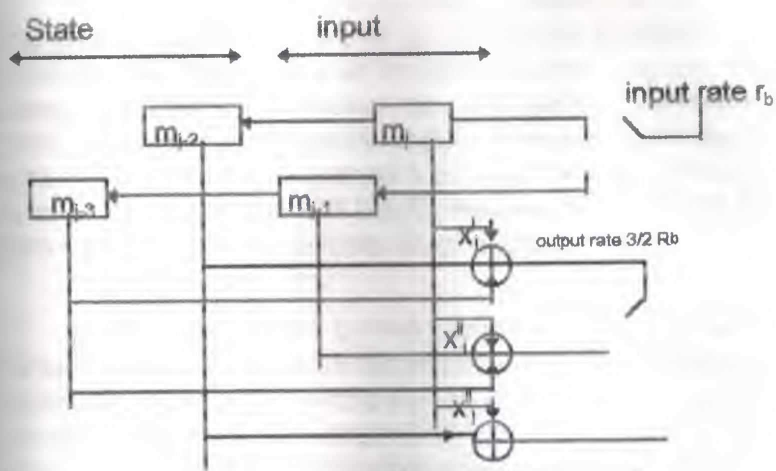


Figure 13.3-5(3,2,1) encoder

Graphical representation becomes more cumbersome for convolutional codes with k>1 because we must deal with input bits in groups of 2^k. Consequently, 2^k branches emanate and terminate at each node, and there are 2^{kL} different states. As an example, Fig 13.3-6 shows the state diagram for the (3,2,1) encoder in Fig.13.3-5. The branches are labeled with the k=2 input bits followed by the resulting n=3 output bits.

The convolutional codes employed for FEC systems usually have small values of n and k, while the constraint length typically falls in the range of 10 to 30. All convolutional encoders require a comutator switch at the output, as shown in Figs.13.3-1 and 13.3-5. For codes with k>1, the input distributor switch can be eliminated by using a single register of length kL and shifting the bits in groups of k. In any case, convolutional encoding hardware is simpler than the hardware for block encoding since message bits enter the register unit at a steady rate r_b and an input buffer is not needed.

Exercises 13.3-1 Conider a systematic (3,1,3) conolutional code. List the possible state and determine the state transition produced by m_j=0 and m_j=1. Then construct and label the state diagram taking the encoded output bits to be m_j, $m_{j-2} \oplus m_j$, and $m_{j-3} \oplus m_{j-1}$. (See Fig P13.3-4 for a convolutional eight-state pattern.)

Free Distance and Coding Gain

We previously found that the error-control power of a block code depends upon its minimum distance, determined from the weights of the codewords. A convolutional code does not subdivide into codewords, so we consider instead the weight w(X) of an entire transmitted sequence X generated by some message sequence.

The free distance of a convolutional code is then defined to be

$$d_f \triangleq [w(X)]_{min} \qquad X \neq 000\ldots$$

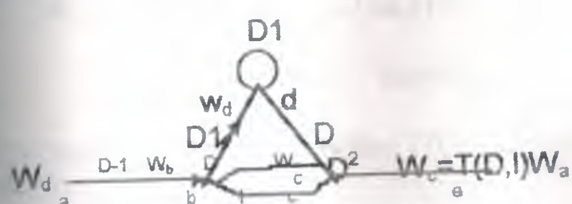
The value of d_f serves as a measure of error-control power. It would be an exceedingly dull and tiresome task to try to evaluate d_f by listing all possible transmitted sequences. Fortunately there's better way based on the normal operating produces of appending a"tailor" of 0s at the end of a message to clear the register unit and return the encoder to its initial state. This procedural eliminates certain branches from the code trellis for the last L transitions.

Take the code trellis in Fig.13.3-4a, for example. To end up at state a, the next-to-last state must be either a or c so the last few branches of any transmitted sequence X must follow one of the paths shown in Fig.13.3-7. Here the final state is denoted by e, and each branch has been labeled with the number of 1s in the encoded bits-- which equals the weight associated with that branch. The total weight of a transmitted sequence X equals the sum of the branch weights along the path of X. In accordance with Eq.(5), we seek the path that has the smallest branch-weight sum, other than the trivial all-zero path.

Looking backwords L+1=3 branches from e, we locate the last path that emanates from state a before terminating at e. Now suppose all earlier transitions followed the all-zero path along the top line, giving the state sequence aa....abce. Since an a-a branch hac weight 0, this state sequence corresponds to a minimum weight nontrivial path. We therefore conclude that d_f=0+0+.......0+2+1+2=5. There are other minimum-weights paths, such as aa....abcae and aa...abcbce, but not nontrivial path has less weight than d_f=5.

Another approach to the calculation of free distance involves the generating function of a convolutional code. The generating function may be viewed as the transfer function of the encoder with respect to state transitions. Thus, instead of relating the initial and final states by multiplication. Generating functions provide important information about code performance, including the free distance and decoding error probability.

We will develop the generating function for our (2,1,2) encoder using the modified state diagram in Fig.13.3-8a. This diagram has been derived from Fig13.3-4b. with four modifications.



(a)

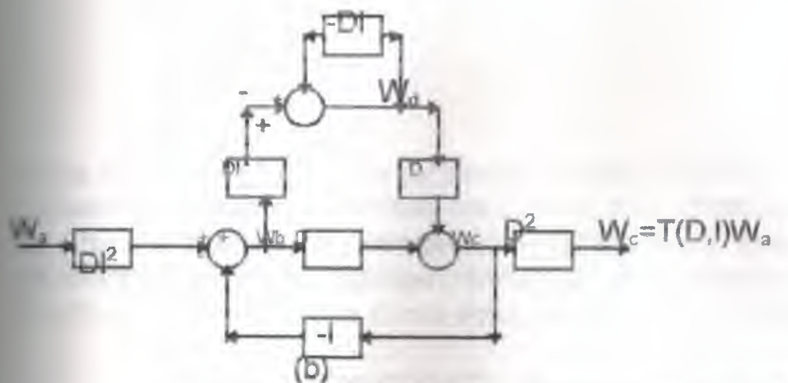


Figure 13.3-8 (a) Modified state diagram for (2,1,2) encoder;(b)equivalent block diagram

First, we have eliminated the a-a loop which contributes nothing to the weight of a sequence X. Second, we have drawn the c-a branch as the final c-e transition. Third, we have assigned a state variable W_a at node a, and likewise at all other nodes. Fourth we have labeled each branch with two 'gain' variables. D and I such that the exponent of D equals the branch weight (as in Fig 13.3-7), while the exponent of I equals the corresponding number of nonzero message bits (as signified by the solid or dashed branch line). For instance since the c-e branch represents $x'_j x''_j=11$ and $m_j=0$, it is labeled with $D^2I^0=D^2$. This exponential trick allows us to perform sums by multiplying the D and I terms, which will become the independent variables of the generating function.

Our modified state diagram now looks like a signal-flow graph of the type sometimes used to analyze feedback systems. Specifically, if we treat the nodes as summing junctions and the DI terms as branch gains, then Fig.13.3-8a represents the set of algebraic state equation

$$W_b= D^2IW_a+ IW_c \qquad W_c= DW_b+DW_d \qquad (6a)$$

$$W_d= DIW_b+ DIW_d \qquad W_e= D^2W_c$$

The encoder's generating function T(D,I) can now be defined by the input-output equation

$$T(D,I) \triangleq W_e/W_a$$

These equations are also equivalent to the block diagram in Fig.13.3-8b, which further emphasizes the relationship between the state variables, the branch gains, and the generating function. Note that minus signs have been introduces here so that the two feedback paths c-b and d-d corresponds to negative feedbackç

Next, the expression for T(D,I) is obtained by algebraic solution of Eq(6), or by block-diagram reduction of Fig.13.3-8b. using the transfer-function relations for parallel, cascade, and feedback connections in Fig.3.1-8.(If you know Mason's rule you could also apply it to Fig.13.3-8a). Any of these methods produces the final result

$$T(D,I)= \frac{D^5I}{1-2DI}$$

$$= D^5I + 2D^6I^2 + 4D^7I^3 + \ldots\ldots\ldots\ldots$$

$$=\sum_{d=5}^{\infty} 2^{d-5} D^{d} I^{d-4}$$

where we have $1/(1-2DI)^{-1}$ to get the series in Eq(7b). Keeping in mind that T(D,I) represent all possible transmitted sequences that terminate with a c-e transition Eq. (7b) has the following interpretation: for any $d \geq 5$, there are exactly 2^{d-5} valid paths are generated by message containing d-4 nonzero bits. The smallest value of w(X) is the free distance, so we again conclude that $d_f=5$.

As a generalization of Eq.(7), the generating function for an arbitrary convolutional code takes the form

$$T(d,I)=\sum_{d=d_f}^{\infty} \sum_{i=0}^{\infty} A(d,i) D^{d} I^{i} \tag{8}$$

Here, A(d,i) denotes the number of different input-output paths through the modified state diagram that have weight d and are weight d and are generated by messages containig I nonzero bits.

Now consider a received sequence Y=X+E, where E represents transmission errors. The path of Y then diverges from the path of X and may or may not be a valid path for the code in question. When Y does not correspond to a valid path, a maximum-likelihood decoder should seek out the valid path that has the smallest Hamming distance from Y. Before describing how such a decoder might be implemented, we will state the relationship between generating functions, free distance, and error probability in maximum-likelihood decoding of convolutional codes.

If transmission errors occur with equal and independent probability α per bit, then the probability of a decoded message-bit error is upper-bounded by

$$P_{be} \leq \frac{1}{k} \frac{\partial T(D,I)}{\partial I} \Bigg|_{D=2\sqrt{\alpha(1-\alpha)},\, I=1}$$

The derivation of this bound is given in Lin and Costello (1983, chap.11) or Viterbi and Omura (1979,chap.4). When α is sufficiently small, series expansion of T(D,I) yields the approximation

$$P_{be} \approx \frac{M(d_f)\, 2^{d_f} \alpha^{d_f/2}}{k} \qquad \sqrt{\alpha} \ll 1$$

where

$$M(d_f)=\sum_{i=1}^{\infty} i A(d_f,i)$$

The quantity $M(d_f)$ simply equals the total number of nonzero message bits over all minimum-weight input-output paths in the modified state diagram.

Equation (10) supports our earlier assertion that the error-control power of a convolutional code depends upon its free distance. For a performance comparison with uncoded transmission we will make the usual assumption of gaussian white noise and $(S/N)_R=2R_c y_b \geq 10$ so Eq(10), Sect.13.1, gives the transmission error probability

$$\alpha \approx (4\pi R_c y_b)^{-1/2} e^{-R_c Y_b}$$

The decoded error probability then becomes

$$P_{be} \approx \frac{M(d_f) 2^{d_f} e^{-(R_c d_f/2) y_b}}{k(4\pi R_c y_b)^{d_f/4}} \qquad (11)$$

whereas uncoded transmission would yield

$$P_{be} \approx \frac{1 \quad e^{-y_b}}{(4\pi y_b)^{1/2}} \qquad (11)$$

Since the exponential terms dominate in these expression, we see that convolutional coding improves reliability when $R_c d_f/2 > 1$. Accordingly , the quantity $R_c d_f/2$ is known as the coding gain, usually expressed in dB.

Explicit design formulas for d_f do not exists, unfortunately, so good convolutional codes must be discovered by computer search and simulation. Table 13.3-1 lists the maximum free distance and coding gain of convolutional codes for selected values of n,k, and L.. Observe that the free distance and coding gain increase with increasing memory L when the code rate R_c is held fixed. All listed codes are nonsystematic ;a systematic convolutional code has a smaller d_f than optimum nonsystematic code with the same rate and memory.

Table 13.3-1 Maximum free distance and coding gain of selected convolutional codes

n	k	R_c	l	d_f	$R_c d_f/2$
4	1	¼	3	13	1.63
3	1	1/3	3	10	1.68
2	1	½	3	6	1.50
			6	10	2.50
			9	12	3.00
3	2	2/3	3	7	2.33
4	3	¾	3	8	3.00

Example 13.3-1 The (2,1,2) encoder back in Fig 13.3-2 has $T(D,I)=D^5 I/(1-2DI)$, so $\partial T(D,I)/\partial I= D^5/(1-2DI)^2$.Equation (9) therefore gives

$$P_{be} \leq \frac{2^5[\alpha(1-\alpha)]^{5/2}}{[1-4\sqrt{\alpha(1-\alpha)}]^2} \approx 2^5 \alpha^{5/2}$$

and the small-αapproximation agrees with Eq.(10). Specifically, in Fig 13.3-8a we find just one minimum-weight nontrivial path abce, which has $w(X)=5=d_f$ and is generated by a message containing one nonzero bit, so $M(d_f)=1$. If $y_b=10$, then R_c

$_b$=5, $\alpha \approx 8.5\text{x}10^{-4}$, and maximum-likelihood decoding yields P_{be}=6.7x10^{-7}, as compared with P_{ube}=4.1x10^{-6}. This rather small reliability improvement agrees with the small coding gain $R_c\, d_f/2$=5/4.

Exercises13.3-2 Let the connections to the mod-2 summers in Fig13.3-2 be changed such that $x'_i = m_i$ and $x''_i = m_{i-2} \oplus m_{i-1} \oplus m_i$.

(a) Construct the code trellis and modified state diagram for this systematic code. Show that there are two minimum-weight paths in the state diagram, and that d_f=4 and $M(d_f)$=3. It is not necessary to find T(D,I).

(b) Now assume γ_b=10. Calculate α, P_{be}, and P_{ube}. What do you conclude about the performance of a conolutional code when $R_c d_f/2$=1?

Decoding Methods

There are three generic methods for decoding convolutional codes. At one extreme, the Veterbi algorithm executes maximum-likelihood decoding and achieves optimum performance but requires extensive hardware for computation and storage. At the other extreme, feedback decoding sacrifices performance in exchange for simplified hardware. Between these extremes, sequential decoding approaches optimum performance to a degree that depends upon the decoder's complexity. We will describe how these methods work with a (2,1,L) code. The extension to other codes is conceptually straight forward, but becomes messy to portray for k>1.

Recall that a maximum-likelihood decoder must examine an entire received sequence Y and find a valid path that has the smallest Hamming distance from Y. However, there are 2^N possible paths for an arbitrary message sequence of N bits (or Nn/k bits in Y), so an exhaustive comparison to 2^{kL} surviving paths, independent of N, thereby bringing maximum-likelihood decoding into the realm of feasibility.

A Viterbi decoder assigns to each branch of each surviving path a metric that equals its Hamming distance from the corresponding branch of Y. (we assume here that 0s and 1s have the same transmission-error probability; if not, the branch metric must be redefined to account for the differing probabilities). Summing the branch metrics yields the path metric, and Y is finally decoded as the surviving path with smallest metric. To illustrate the metric calculations and explain how surviving paths are selected, we will walk through an example of Viterbi decoding

Suppose that our (2,1,2) encoder is used at the transmitter, and the transmitter, and the received sequence starts with Y=11 01 11. Figura 13.3-9 shows the first three branches of the valid paths emanating from the initial node a_0 in the code trellis. The number in parentheses beneath each branch is the branch metric, obtained by counting the differences between the encoded bits and the corresponding bits in Y. The circled number at the right-hand end of each branch is the running path metric, obtained by summing branch metrics from a_0. For instance, the metric of the path $a_0\, b_1\, c_2\, b_3$ is 0+2+2=4.

Now observe that another path $a_0 a_1 a_2\, a_3$ also arrives at node b_3 and has a smaller metric 2+1+0=3. Regardless of what happens subsequently, this path will have a smaller Hamming distance from Y than the other path arriving at b_3 and is therefore more likely to represent a the actual transmitted sequence. Hence, we discard the larger-metric path, marked by an X, and we declare the path with the smaller metric to be the survivor at this node. Likewise, we discard the larger metric paths arriving at nodes a_3,c_3 and d_3, leaving to total of $2^{kL}=4$ surviving paths. The fact that none of the surviving path metrics equals zero indicated the presence of detectable errors in Y.Fig13.3-10 depicts the continuation of Fig.13.3-9 for a complete message of N=12 bits, including tail 0s. All discarded branches and all labels expects the running path metrics have been omitted for the sake of clarity. The latter T under a node indicates that the two arriving paths had equal running matrix, in which case we just flip a coin to choose the survivor (why?). The maximum-likelihood path follows the heavy line from a_0 to a_{12} and the final value of the path metric signifies at least two transmission sequence Y+E and message sequence M written below the trellis.

A Viterbi decoder must calculate two metrics for each node and store 2^{kL} surviving paths, each consisting of N branches. Hence, decoding complexity increases exponentially with L and linearly with N. The exponentially factor limits practical applications of the Viterbi algorithm to codes with small values of L.

When N>>1, storage requirements can be reduced by a truncation process based on the following metric-divergence effect: if two surviving paths emanated from the same node at some point, then the running metric of the less likely path tends to increase more rapidly than the metric of the other survivor within about 5L branches from the common node. This effect appears several times in Fig.13.3-10; consider, for instance, the two paths emanating from node b_1. Hence, decoding need not be delayed until the end of the transmitted sequence. Instead, the first k message bits can be decoded and the first set of branches can be deleted from memory after the first 5Ln received bits have been processed. Successive groups of k message bits are then decoded for each additional n bits received thereafter.

Sequential decoding, which was invented before the Viterbi algorithm, also relies on the metric-divergence effect. A simplified version of the sequential algorithm is illustrated in Fig.13.3-11a, using the same trellis, received sequence, and

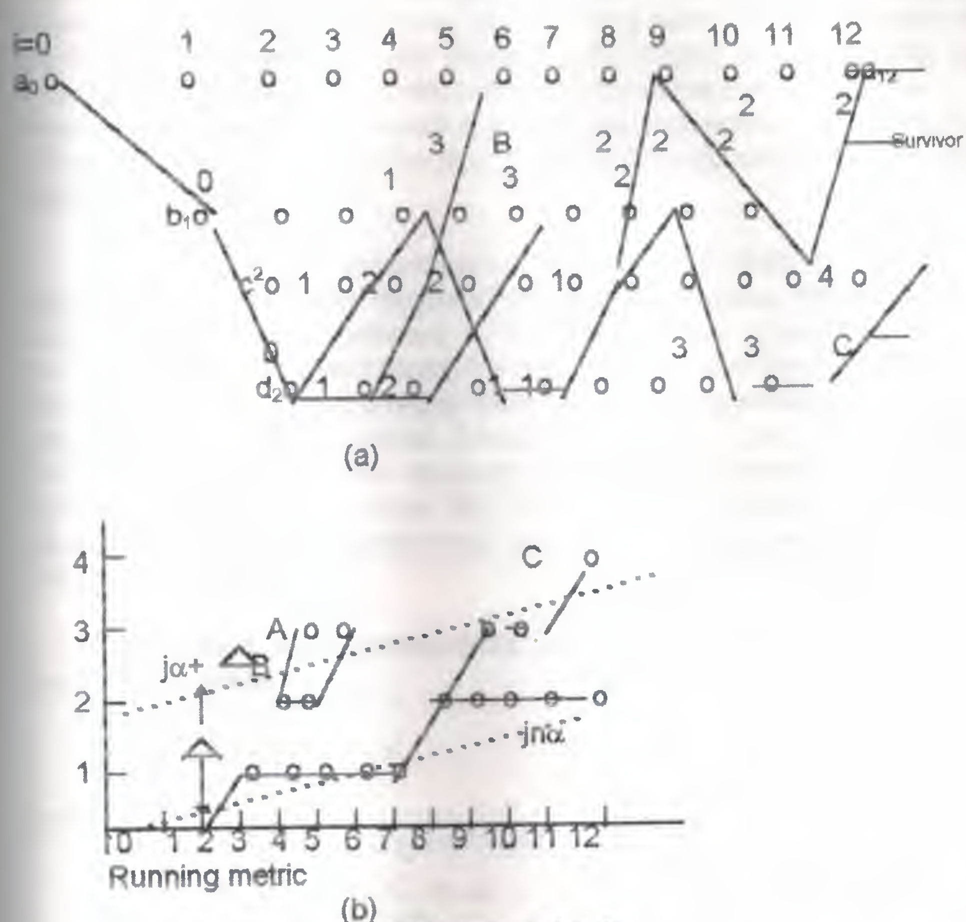


Figure 13.3-11 Illustration of sequential decoding

metrics as in Fig 13.3-10 .. Starting at a_0 the sequential decoder purpose a single path by taking the branch with the smallest branch metric at each successive node. If two or more branches form one node have the same metric, such as at node b_2,the decoder selects one at random and continues on. Whenever the current path happens to be unlikely, the running metric rapidly increases and the decoder eventually decides to go back to a lower-metric node and try +another path. There are three of these abandoned paths in our example. Even so, a comparison with Fig.13.3-10 shows that sequential decoding involves less computation than Viterbi decoding.

The decision to backtrack and try again is based on the expected value of the running metric at a given node. Specifically, if α is the transmission error probabilities per bit, then the expected running metric at the jth node of the correct path equals $jn\alpha$, the expected number of bits errors in Y at that point. The sequential decoder abandons a path when its metric exceeds some specified threshold Δ above $jn\alpha$. If no path survives the threshold test, the value of Δ is increased and the decoder backtracks again. Figure 13.3-11b plots the running metrics versus j, along with $jn\alpha$ and the threshold line $jn\alpha,+\Delta$ for α=1/16 and Δ=2.

Sequential decoding approaches the performance of maximum-likelihood decoding when the threshold is loose enough to permit exploration of all probable paths. However, the frequent backtracking requires more computations and results in a decoding delay significantly greater than Viterbi decoding. A tighter thresholds reduces computations and decoding delay but may actually eliminate the most probable path, thereby increasing the output error probability compared to that of maximum-likelihood decoding with the same coding gain. As compensation, sequential decoding permits practical application of convolutional codes with large L and large coding gain since the decoder's complexity is essentially independent of L.

We have described sequential decoding and Vitebi decoding in terms of algorithm rather than block diagrams of hardware. Indeed, these methods are usually implemented as software for a computer or microprocessor that performs the metric calculations and stores the path data. When circumstances preclude algorithmic decoding, and a higher error probability is tolerable, feedback decoding may be the appropriate method. A feedback decoder actsin general like a "sliding block decoder" that decodes message bits one by one based on a block of L or more successive tree branches. We will focus on the special class of feedback decoding that employs majority logic to achieve the simplest hardware realization of a convolutional decoder.

Consider a message sequence $M= m_1 m_2$ and the systematic (2,1,L) encoded sequence

$$X= x'_1\, x''_1\, x'_2\, x''_1 \qquad (13a)$$

where

$$x'_j = m_j \qquad x''_j = \sum_{i=0}^{L} m_{j-1}\, g_i \quad \text{(mod-2)} \qquad (13b)$$

We will view the entire sequence X as codeword of indefinite length. Then, borrowing from the matrix representation used for block codes, we will define a generator matrix G and a parity-check matrix H such that

$$X=MG \qquad XH^T=0\ 0\ \ldots..$$

To represent Eq.(13), must be a semi-infinite matrix with a diagonal structure given by

$$G=\begin{bmatrix} 1 & g_0 & 0 & g_1 & 0 & \ldots\ldots & 0 & g_L & & & \\ & & 1 & g_0 & 0 & g_1 & 0 & \ldots\ldots & 0 & g_L & \\ & & & \cdot & \cdot & \cdot & & & & \cdot & \\ & & & & & \cdot & \cdot & \cdot & & & \\ & & & & & & \cdot & \cdot & & \cdot & \end{bmatrix} \qquad (14a)$$

This matrix extends indefinitely to the right and down, and the triangular blank spaces denote elements that equal zero. The parity-check matrix is

$$H=\begin{bmatrix} g_0 & 1 & & & & & & \\ g_1 & 0 & g_0 & 1 & & & & \\ . & . & g_1 & 0 & g_0 & 1 & & \\ . & . & . & . & & & . & . \\ . & . & . & . & & & . & . \\ g_L & 0 & . & . & & & . & \\ & & g_L & 0 & & & & \\ & & & & . & . & & \\ & & & & & . & . & \\ & & & & & & . & \end{bmatrix}$$

which also extended indefinitely to the right and down.

Next, let E be the transmission error pattern in a received sequence Y=X+E. We will write these sequences as

$$Y=y'_1\, y''_1\, y'_2\, y''_2 \ldots\ldots \quad E=e'_1\, e''_1\, e'_2\, e''_2 \ldots\ldots$$

so that $y'_j=m_j \oplus e'_j$. Hence, given the error bit e'_j, the jth message bit is

$$m_j=y'_j \oplus e'_j$$

A feedback decoder estimates errors from the syndrome sequence

$$S=YH^T=(X+E)\,H^T=E\,H^T$$

Using Eq.(14b) for H, the jth bit of S is

$$s_j=\sum_{i=0}^{L} y'_{j-i}g_i \oplus y''_j=\sum_{i=0}^{L} e'_{j-i}g_i \oplus e''_j \qquad (16)$$

where the sums are mod-2 and it is understand that $y'_{j-i}=e'_{j-i}=0$ for $j-i\leq 0$. As a specific example, take a (2,1,6) encoder with $g_0=g_2=g_5=g_6=1$ and $g_1=g_3=g_4=0$, so

$$s_j=y'_{j-6}+y'_{j-5}+y'_{j-2}+y'_j+y''_j \qquad (17a)$$

$$=e'_{j-6}+e'_{j-5}+e'_{j-2}+e'_j+e''_j \qquad (17b)$$

Equation (17a) leads directly to the shift-register circuit for syndrome calculation diagrammed an Fig. 13.3-12. Equation (17b) is called a parity-check sum and will leads us eventually to the remaining portion of the feedback decoder.

To that end, consider the parity-check table Fig.13.3-13a. where checks indicated which error bits appear in the sums $s_{j-6}, s_{j-4}, s_{j-1}$, and s_j. This table brings out the fact that e'_{j-6} is checked by all four of the listed sums, while no other bit is checked by more than one. Accordingly, this set of check sums is said to be orthogonal on e'_{j-6}. The tap gains of the encoder were carefully chosen to obtain orthogonal check sums.

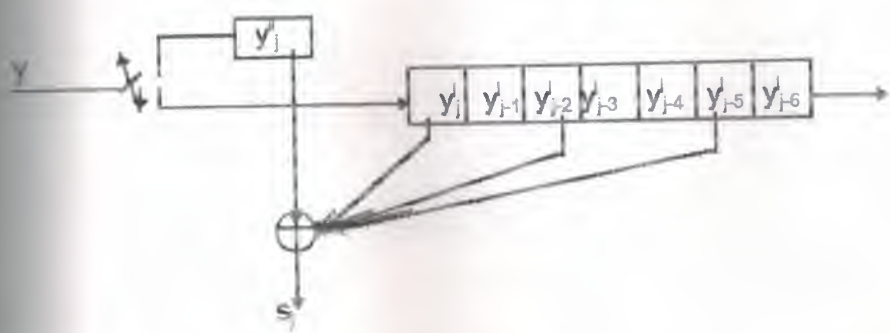


Figure 13.3-13 Parity-check table for a systematic (2,1,6) code.

When the transmission error probability is reasonably small, we expect to find at most one or two errors in the 17 transmitted bits represented by the parity-check table. If one of the errors corresponds to $e'_{j-6}=1$, then the four check sums will contain three 1s. Otherwise, the check sums contain less than three 1s. Hence, we can apply these four check sums to a majority-logic gate to generate the most likely estimate of e'_{j-6}.

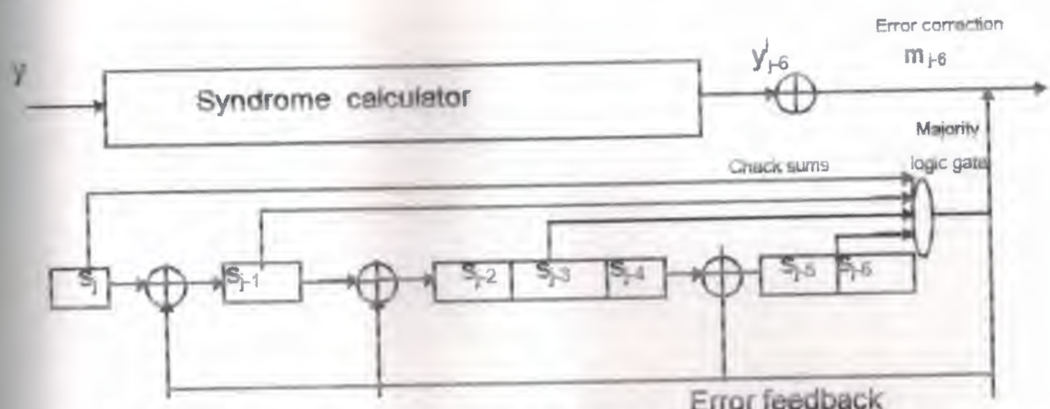


Figure 13.3-14 Majority-logic feedback decoder for a systematic (2,1,6) code.

Figure 13.3-14 diagrams a complete majority-logic feedback decoder for our systematic (2,1,6) code.The syndrome calculator from Fig 13.3-12 has two outputs y'_{j-6} and s_j. The syndrome bit goes into another shift register with taps that connect the check sums to the majority-logic gate, whose output equals the estimated error e'_{j-6}.The mod-2 addition $y'_{j-6} \oplus e'_{j-6}$ carries out error correction based on Eq(15). The error is also feedback to the syndrome register to improve the reliability of subsequent check sums. This feedback path accounts for the name feedback decoding.

Our example decoder can correct any single-error or double-error pattern in six consecutive message bits. However, more than two transmission errors produces erroneous corrections and error propagation via the feedback path. These effects result in a higher output error than that of maximum-likelihood decoding. See Lin and Costello (1983, chap.13) for the error analysis and further treatment of majority-logic-decoding.

ERROR DETECTION, CORRECTION AND CONTROL

A major design criterion for all telecommunication systems is to achieve error free transmission. Errors, unfortunately, do occur. There are many types and causes originating from various sources ranging from lightning strikes to dirty switch contacts at the central office. A method of detecting and in some cases correcting for their occurrence, is a necessity. To achieve this, two basic techniques are employed. One is to detect the error and request a retransmission of the corrupted message. The second technique is to correct the error at the error at the receiving end without having to retransmit the message. The trade-off for either technique is the redundancy that must be built into the transmitted bit stream. This redundancy decreases system throughput

Many of today's communication systems employ elaborate error-control protocols. Some of these protocols are software packages designed to facilitate file transfers between personal computers and mainframes. More recent error controllers are completely self-contained within a hardware module, thus relieving the CPU of the burden of error control. The entire process is transparent to the user.

In this chapter we consider some of the most common methods used for error detection and correction, including error-controlling protocols specifically designed for data-communications equipment.

12.1- Parity

Parity is the most simplest and oldest method of error detection. Although it is not very effective in data transmission, it is still widely used due to its simplicity. A single bit called the parity bit is added to a group of bits representing a letter, number, or symbol. ASCII characters on a keyboard, for example, are typically encoded into seven bits with an eight bit acting as parity. The parity bit is computed by the transmitting device based on the number of 1-bits set in the character. Parity can be either **odd** or **even**. If odd parity is selected, the parity bit is set to a 1 or 0 to make the total number of 1-bits in the character, including the parity bit itself, equal to an odd value. If even parity is select, the opposite is true; the parity bit is set to a 1 or 0 to make the total number of 1-bits, including the parity bit itself, equal to an even number. The receiving device performs the same computation on the received number of 1-bits for each character and checks the computed parity against what was received. If they do not match, an error has been detected. Table1 lists examples of even and odd parity.

The selection of even or odd parity is generally arbitrary. In most cases it is a matter of custom or preference. The transmitting and receiving stations, however, must be set to same mode. Some system designers prefer odd parity over even. The advantages is that when a string of several data characters are anticipated to be all zeros the parity bit would be set to 1 for each character, thus allowing for ease of character identification and synchronization.

Table 1 Even and Odd Parity for a Seven-Bit Data Character

Data character	Odd parity bit	Data character	Even parity bit
1101000	0	1011101	1
0010111	1	1110111	0
1010110	1	0011010	1
1010001	0	1010111	1

12. 2-Parity Generating and Checking

Parity generating circuits can easily be implemented with a combination of exclusive-bit data word. Odd parity can be obtained by simple adding an inverter at the output of the given circuit. Additional gates can be included in the circuit for extended word lengths. The same circuits can be used for parity checking by adding another exclusive-OR gate to accommodate the received parity bit. The received data word and parity bit are applied at the circuits input. For even parity, the output should always be low unless an error occurs. Conversely, for odd parity checking, the output should always be high unless an error occurs.

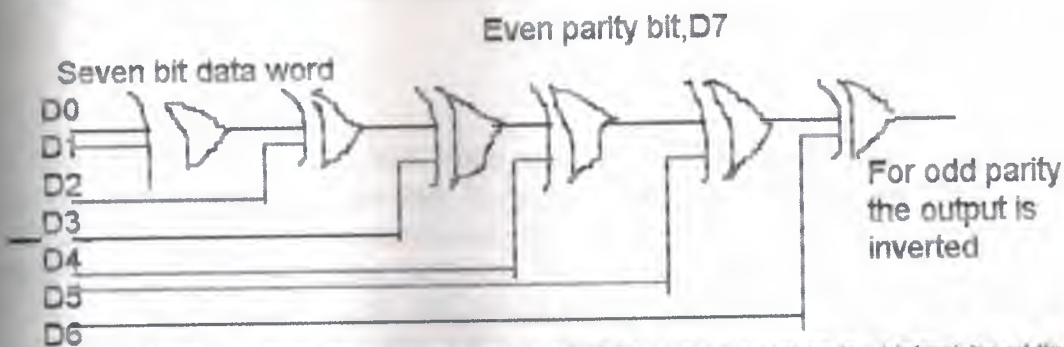


Figure 12-1 Even parity generating circuit. Odd parity generation is obtained by adding an inverted at the output.

Figure 12-2 depicts another design that can be used for parity generation and checking.

12.3 THE DISADVANTAGE WITH PARITY

A major shortcoming with parity is that it is only applicable for detecting when one bit or an odd number of bits have been changed in a character. Parity checking does not detected when an even number of bits have changed. For example, suppose that bit D2 in Example 1 were to change during the course of a transmission for an odd parity system. Example 1 shows how the bit errors is detected.

Example1	Parity (odd)	D7	D6	D5	D4	D3	D2	D1	D0
Transmitted	0	0	1	1	0	1	1	0	1
Received	0	0	1	1	0	1	0	0	1

single bit error

The receive parity bit, a zero is in conflict with the computed number of 1-bits that was received; in this case four, an even number of 1-bits. The parity

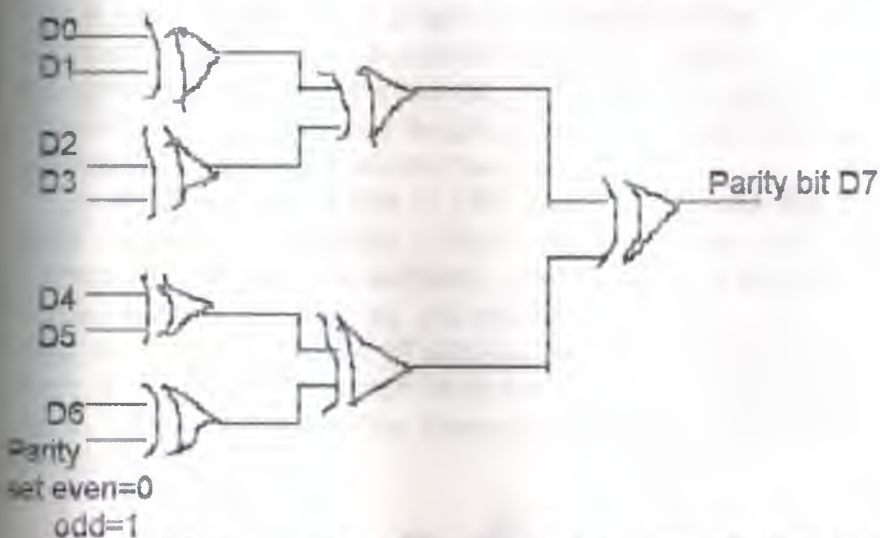


Figure 12-2 Even or odd parity generation is achieved in this circuit by setting the appropriate level at the parity set input.

bit should have been equal to a value making the total number of 1-bits odd. An error has been properly detected. If, on the other hand, bit D2 and bit D1 were both altered during the transmission, the computed parity bit would still be in agreement with the received parity bit. This err would go undetected, as shown in Example2. A little through will reveal that an even number of errors in a character, for odd or even parity, will go undetected.

Example

	Parity (odd)	D7	D6	D5	D4	D3	D2	D1	D0
Transmittted	0	0	1	1	0	1	1	0	1
Received	0	0	1	1	0	1	0	1	1

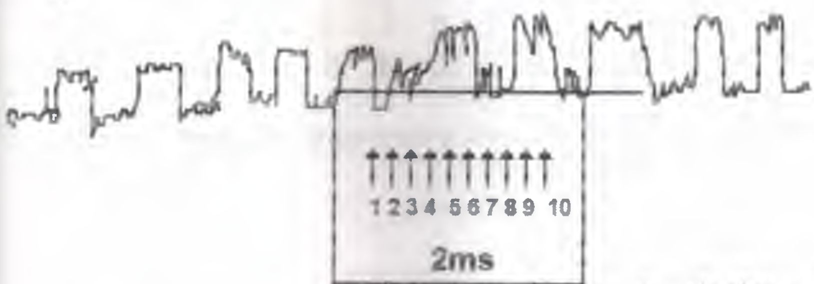
Errors: two bits or an even number of bits go undetected

Parity, being a single-bit error-detection scheme, presents another problem in accommodating today's high-speed transmission rates. Many errors are a result of impulse noise, which tends to be bursty in nature. Noise impulses may last several milliseconds, consequently destroying several bits. The higher the transmission rate, teh greater teh effect. Figure 12-3 depicts a 2-ms noise burst imposed on a 4800-bps signal is 208 ms (1/4800). As many as 10 bits are affected. At least two characters are destroyed here, with the possibility of both character errors going undetected.

12.4 VERTICAL AND LONGITUDINAL REDUNDANCY CHECK (VRC AND LRC)

Thus far, the discussion of parity has been on a per character basis. This is often referred to as a vertical redundancy check (vrc). Parity can also be computed based on an accumulation of the value of each character's LSB through MSB, including the vrc bit, as shown in figure 12-4. This method of parity checking is referred to as a longitudinal redundancy check(LRC). The resulting word is called the block check character (BCC).

Additional parity bits in LRC used to produce the BCC provide extra error detection capabilities . Single-bit errors can now be detected and corrected. For example, suppose that the LSB of the letter y in the message in Figure 12-4 was received as a 0 instead of a 1. The computed parity bit for the LRC would indicate that a bit was received in error. By itself, the detected LRC error does not specify which bit in the row of LSB bit received is in error . The same is true for the vrc. The computed parity bit in the column of the character in errory.



Noise burs; ten bits are destroyed

Figure 12-3 Effected of a 2-ms noise burst on a 4800-bps signal.

Would be a 1 instead of a 0. By itself, the detected vrc error does not specify which bit in the y column has been received in error . A cross-check, however will reveal that the intersection of the detected parity error, for the vrc and lrc check identifies the exact bit was received in error. By inverting this bit, the error can be corrected.

Unfortunately, an even number of bit errors is not detected by either the lrc or vrc check. Cross-checks cannot be performed; consequently bit errors cannot be corrected.

12.1CYCLIC REDUNDANCY CHECKING (CRC)

Parity checking has major shortcomings. It is much efficient to eliminate the parity bit of each character in the block entirely and utilize the redundant bits at the end of block.

A more powerful method than the combination of LRC and VRC for error detection in blocks is cyclic redundancy checking (CRC). CRC is the most commonly used method

	H	a	e	sp	a	sp	n	i	c	e	sp	d	a	y		BBC(LRC)
LSB	0	1	0	0	1	0	0	1	1	1	0	0	1	1	1	0
	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
	0	0	1	0	0	0	1	0	0	1	0	1	0	0	0	0
	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1
	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
MSB	1	1	1	0	1	0	1	1	1	1	0	1	1	1	0	1
VRC	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1

Figure 12.4Computing the block check character (BCC) for a message block with vrc and lrc odd parity checking.

for error detection in block transmission. A minimal amount of hardware is required (slightly more than lrc/vrc systems), and its effectiveness in cting errors is greater than 99.9%.

CRC involves a division of the transmitted message block , by a constant called the generator polynomial. The quotient is discarded and the remainder is transmitted as the block check character (BCC). This is shown in figure 12-5. Some protocols refer to the BCC as the frame check sequence (FCS). The receiving station performs the same computation on the received from the transmitter. If the two match, then no errors have been detected in the message block. If the two match, either a request for retransmission is made by the receiver or the errors are corrected through the use of special coding techniques.

Cyclic codes contain a specific number of bits, governed by the size of the character within the message block. There of the most commonly used cyclic codes are CRC-12, CRC-16, and CRC-CCITT. Blocks containing characters that are six bits in length typically use CRC-12, a 12-bit CRC. Blocks formatted with eight-bit characters typically use CRC-16 or CRC-CCITT, both of which are 16-bit codes. The BCC for these three cyclic codes is deried from the following generator polynomials, G(x):

CRC-12 generator polynomial:	$G(x)=x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16 generator polynomial:	$G(x)=x^{16}+x^{15}+x^2+1$
CRC-CCITT generator polynomial:	$G(x)=x^{16}+x^{12}+x^5+1$

A combination of multistage shift registers employing feedback through exclusive-OR gates is used to implement the mathematical function performed on the message block to obtain the BCC, Figure 12-6 depicts three CRC generating circuits for CRC-12, CRC-16, and CRC-CCITT. The BCC is accumulated by shifting the data stream into the data input of the register. When the final bit of the message block is shifted in , the register contains the BCC. The BCC is transmitted at the end of the message block, LSB first.

Generator polynomial, G(x)

↓ Quotient discarded

Constant | Mesage block

Constant

Next character

Constant

Next character

Constant

Next character

Constant

Nextcharacter

Constant

Remainder ← BCC

Fiaure12-5 compubne the block check character (BCC) of a messaae block usine CRC.

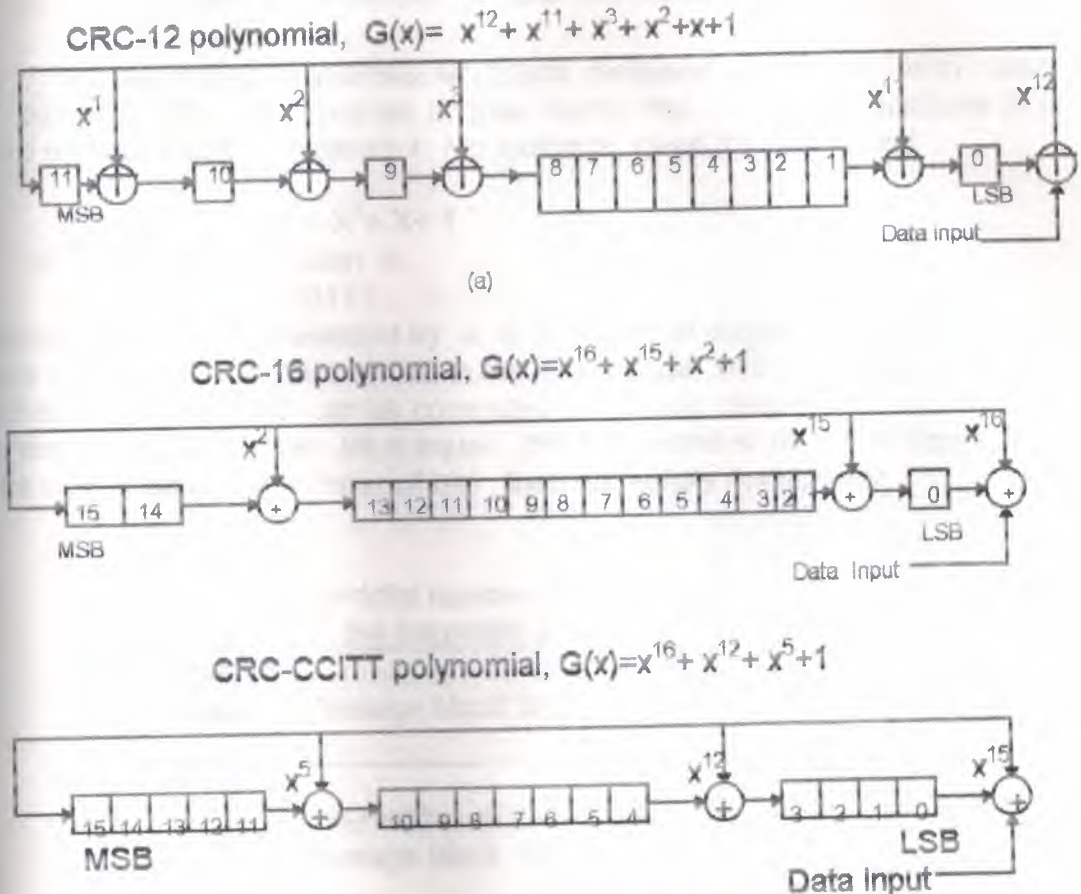


Figure 12-6 (a)CRC generating circuit for CRC-12; (b) CRC generating circuit forCRC-16;(c)CRC-CCITT

12.5.1 Computing the Block Check Character

The generating polynomial, $G(x)$, and message polynomial , $M(x)$, used for computing the BCC include degree terms that represent positions is a group of bits that are a binary 1: For example, given the polynomial

$$X^5+X^2+X+1$$

its binary representation is

100111

Missing terms are represented by a 0. The highest degree in the polynomial is one less than the number of bits in the binary code. The following discussion illustrates how the BCC can be computed using long division.
Referring to Figure 12-7, if we let n equal the total number of bits in transmitted block and k equal the number of bits , then n-k equals the number

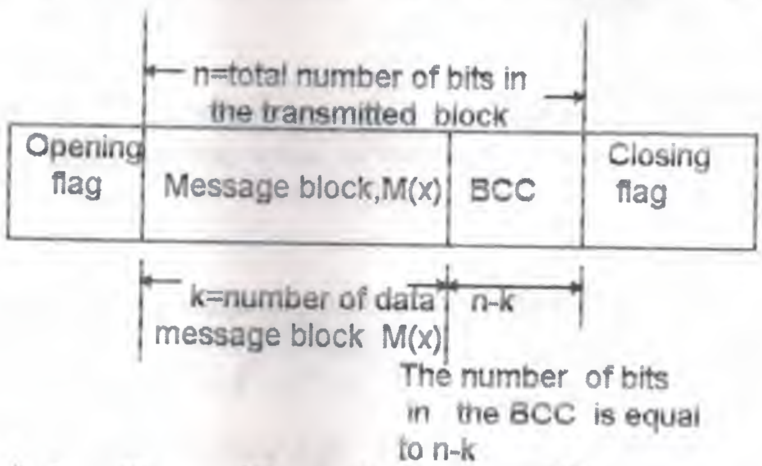


Figure 12-7 Format of a message block for computing the BCC in CRC

of bits in the BCC . The message polynomial $M(x)$, is multiplied by X^{n-k} to achieved the correct number of bits for the BCC. The resulting product is then divided by the generator polynomial, $G(x)$. The quotient is discarded and the end of the message block. The long-division process. Rather, an exclusive-OR operation is performed. As we will see in the following examples , this will yield a BCC having a total number of bits one less equal to the highest degree of the generator polynomial. The entire transmitted

$$T(x)=X^{n-k}[M(x)+B(x)]$$

where $T(x)$= total transmitted message block

X^{n-k}= multiplication factor
$B(x)$= BCC

Consider the following examples.

Example3 In this example, the transmitted message block will include a total number of bits, n, equal to 14. Nine of the 14 bits are data k, Therefore the BCC consist of five bits(n-k=5).

Given:

Generator polynomial, $G(x)=x^5+ x^2+ x+1$

$=100111$

Message polynomial, $M(x)= x^8+ x^6+ x^3+ x^2+ 1$

$=101001101$

The number of bits in the BCC, n-k=5(the highest degree of the generator polynomial). Compute the value of the BCC.

Solution:

1.Multiply the message polynomial, M(x), by x^{n-k};

$x^{n-k} M(x)= x^5(x^8+ x^6+ x^3+ x^2+ 1)$

$= x^{13}+ x^{11}+ x^8+ x^7+ x^5$

$=1010110100000$

2.Divided $x^{n-k} M(x)$ by the generator polynomial and discard the quotient. The remainder is the BBC, B(x).

```
             101111110  ←— discard quotient
      100111)10100110100000
             100111
              111010
              100111
              ------
              111011
              100111
              ------
               111000
               100111
               ------
                111110
                100111
                ------
                 110010
                 100111
                 ------
                  101010
                  100111
                  ------
                   11010  ←— BBC
```

3. To determine the total transmitted message block, T(x), add the BBC, B(x), to x^{n-k} M(x)

```
T(x)=x^(n-k)M(x)+B(x)
        =10100111000000
      +          11010    BBC,B(x)
        10100110111010    transmitted block T(x)
```

At the receiving end, the transmitted message block, T(x), is divided by the same generating polynomial, G(x) . If the remainder is zero, the block was received without errors.

```
              101111110          discard quotient
       100111/10110111010
              100111
               111010
               100111
               ------
                111011
                100111
                 111001
                 100111
                  111101
                  100111
                   110100
                   100111
                    100111
                    100111       Remainder equals
                         0  <----       zero (no errors)
```

Example : For simplicity, a 16-bit message (k=16) using CRC=16 will be used. The total number of bits in the transmitted message block, n, is therefore 32.

Given:

Generator polynomial for CRC-16, $G(x)=x^{16}+x^{15}+x^{2}+1$
Message Polynomial, $M(x)=x^{15}+x^{13}+x^{11}+x^{10}+x^{7}+x^{5}+x^{4}+1$

The number of bits in the BCC, n-k=16 (the highest degree of the generator polynomial).

Solution:

1. Multiply the message polynomial, M(x), by x^{n-k};

$$x^{n-k} M(x)=x^{16}(x^{15}+x^{13}+x^{11}+x^{10}+x^{7}+x^{5}+x^{4}+1)$$

$$=x^{31}+x^{29}+x^{27}+x^{26}+x^{23}+x^{21}+x^{20}+x^{16}$$

$$=10101100101100010000000000000000$$

Transmitted Data   Transmitted Block   Received Data

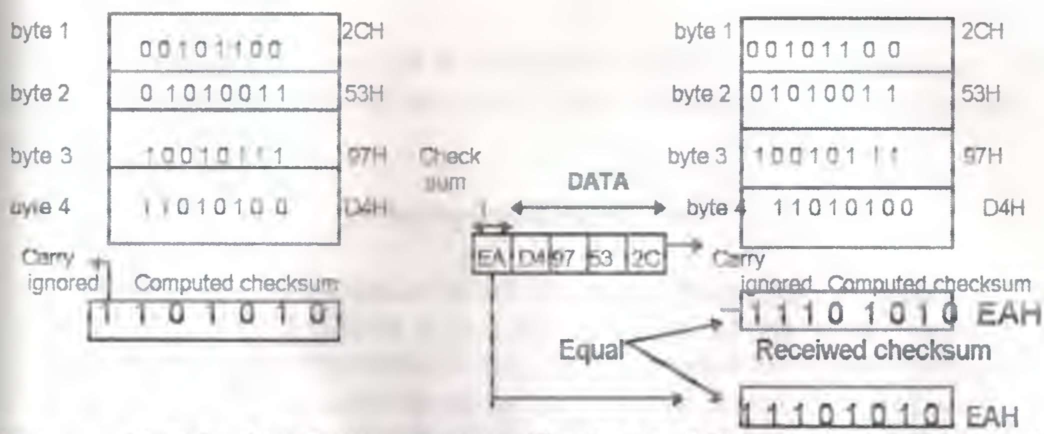


Figure12-8 a single-precision checksum is generated and transmitted as a BCC at the end of four-byte block. The receiver verifies the block by regenerating the checksum and comparing it a against the original.

1. Single-precision
2. Double-precision
3. Honeywell
4. Residue

1- Single-Precision Checksum

The most fundamental checksum computation is the single-precision checksum. Here, the checksum is derived simply the performing a binary addition of each n-bit data word in the message block. Any carry or overflow during the addition process is ignore thus the resultant checksum is also bit in length. Figure 12-8 illustrates how the single -precision checksum is derived transmitted as the BCC, and used to verify the integred of the received data for simplicity a four data block is used. Note that the sum of the data exceeds 2^n-1 and there for all carry occurs out of the MSB. This carry is ignore and only the eight-bit (n-bit) checksum is send as the BCC.

An inhernet problem with the single-precision checksum is if the MSB of the n-bit data word becomes logically stuck at (SAI), the checksum becomes SAI as well. A little through will reveal that the regenerated checksum on the received data will equal the original checksum and the SAI fault will go undetected. A more elaborate scheme may be necessary.

2-Double- Precision Checksum

As its name implies, the double-precision checksum extends the computed checksum to 2n bits in length, where n is the size of the data word in the message block. For example, the eight-bit data words used in the single-precision checksum example above would have a 16-bit checksum. Message blocks with 16-bit data words would have a 32-bit checksum, and so forth. Summation of data words in the message block can now extend up to modulo 2^{2n}, there by decreasing the probability of an erroneous checksum. In addition, the SAI (stuck at 1) error discussed earlier would be detected as a checksum error at the received, Figure 12-9 depicts how the double-precision checksum is derived, transmitted as the BCC, and used to verify the integrity of the received data. For simplicity, a four byte data block is used again. Hexadecimal notation bis also used. Note that the carryout of the MSB position of the low-order checksum byte is not ignored .Instead, it becomes part of the 16-bit checksum result. Any carryout of the MSB of the 16-bit checksum is ignored.

3- Honeywell Checksum

The Honeywell checksum is an alternative from of the double-length. Its length is also 2n bits, where n is again the size of the data word in the message block. The difference is that the Honeywell checksum is based on interleaving consecutive data words to from double- length words. The double-length words are then summed together two from a double-precision checksum. This is shown in Figure 12-10 .The advantage of the Honeywell checksum is that stuck at 1 (SAI) and stuck at o (SA=) bit errors occurring in the same bit positions of all words can be detected during the error in the upper and lower words of the checksum. At least two bit positions in the checksum are affected.

4- Residue Checksum

The last from of checksum in our discussion is the residue checksum. The residue checksum is identical to the single-precision checksum, except that any carryout of the MSB position of the checksum word is 'wrapped around' and added to the LSB position. This added complexity permits the detection of SA1 errors that go undetected. This is illustrated in Figure 12-11

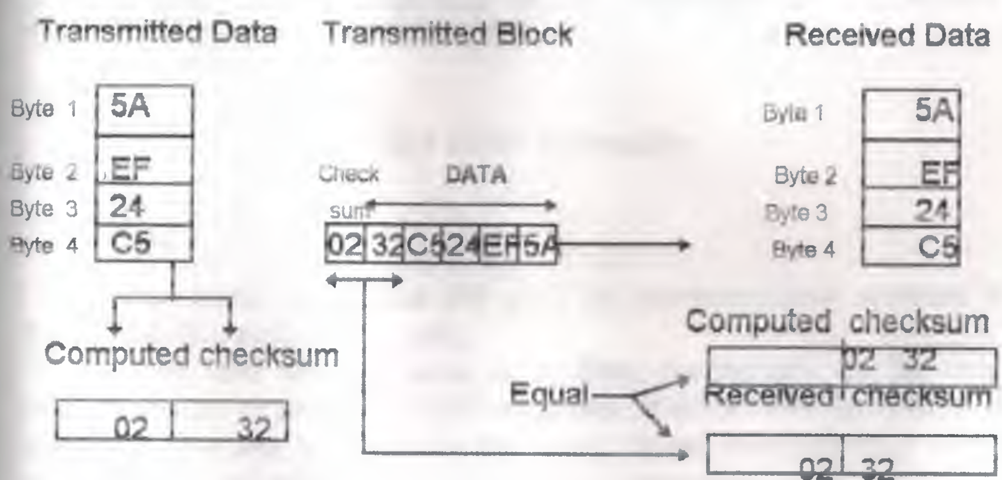


Figure12-9 A double-precision checksum is generated and transmitted as a BCC at the end of a four-byte block. The receiver verifies the block by regenerating the checksum and comparing it against the original

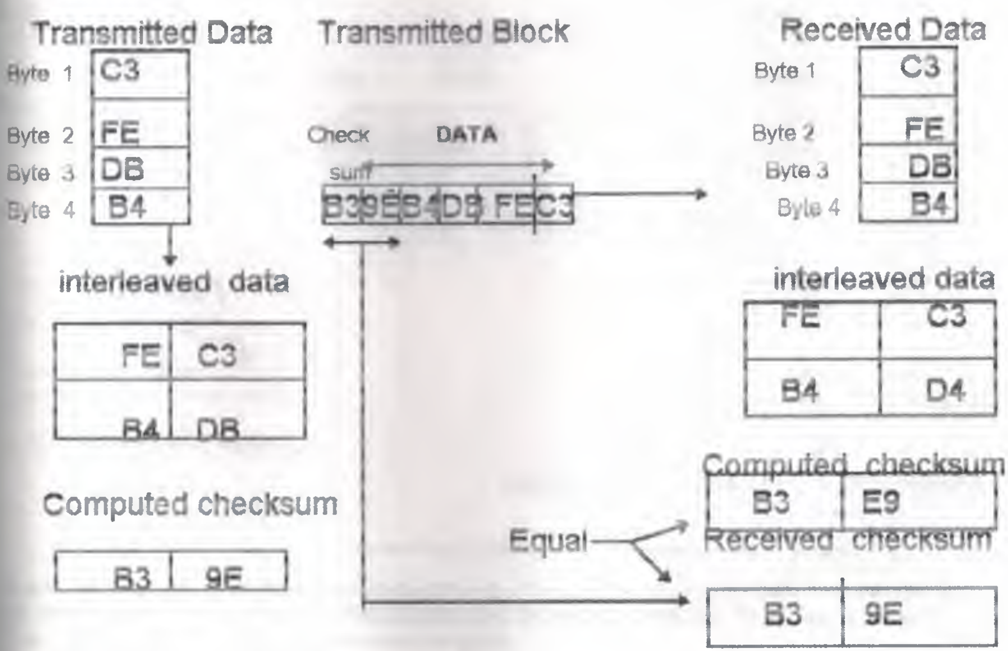


Figure12-10 Structure of the Honeywell checksum. The checksum is generated and transmitted as the BCC at the end of a four-byte block. The receiver verifies the block by regenerating the checksum and comparing it against the original.

12-7 Error Correction

Two basic techniques are used by communication systems to ensure the reliable transmission of data.

They are shown Figure 12-12. One technique is to request the retransmission of the data block received in error. This technique, the more popular of the two, is known as automatic repeat request(ARQ). When a data block is received without error, a positive acknowledgment is sent back to the transmitter via the reverse channel. ACK alternating in BISYNC is an example of a protocol that uses ARQ for error correction. A second technique is called

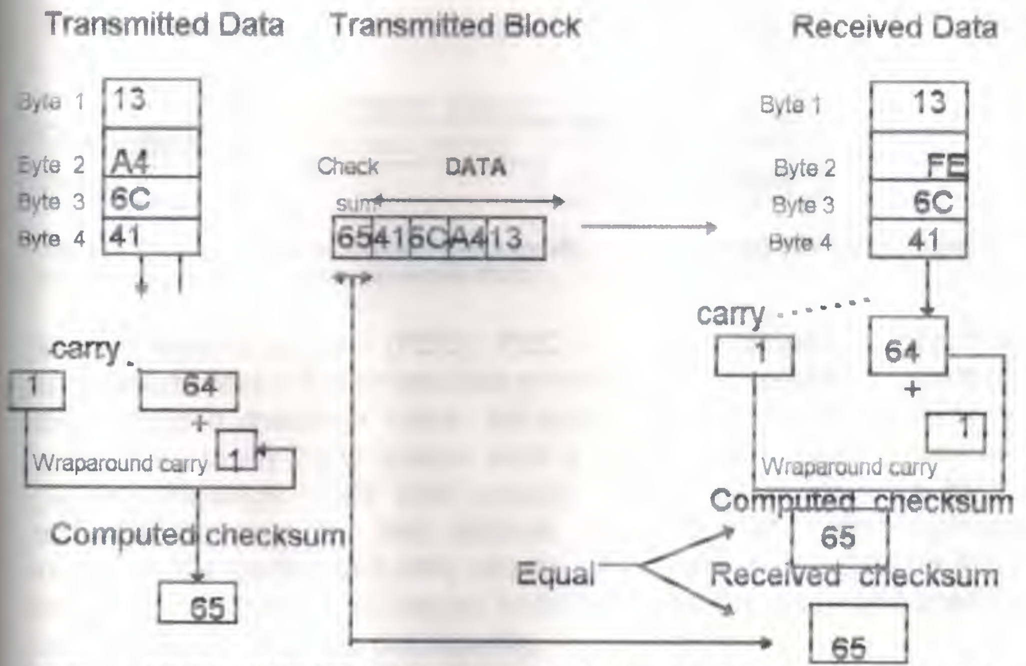
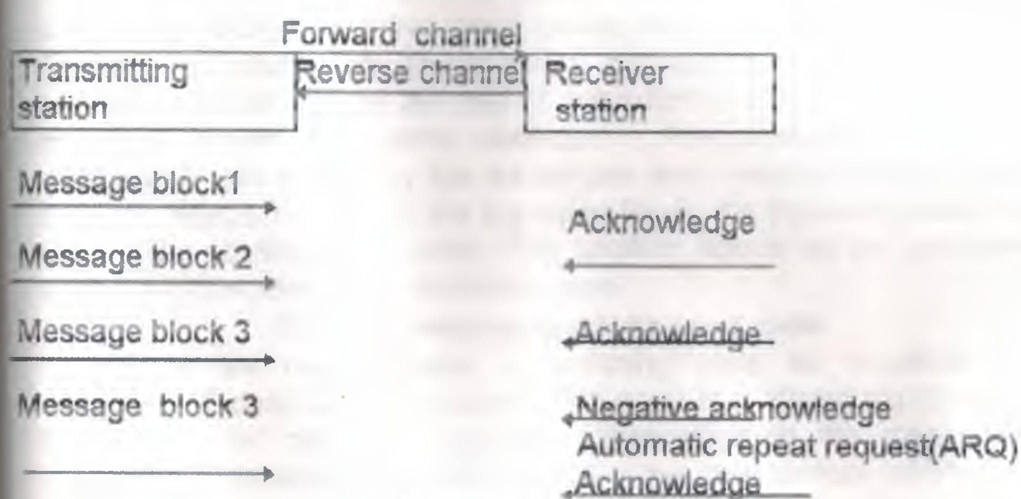


Figure12-10 Structure of the Residue checksum. The checksum is generated and transmitted as the BCC at the end of a four-byte block. The receiver verifies the block by regenerating the checksum and comparing it against the original.



(a)

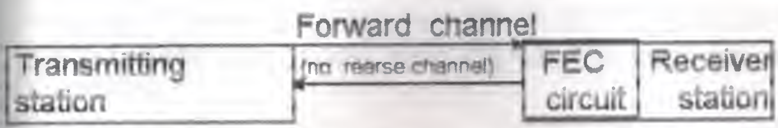


Figure12-12 (a)Error correction using the automatic the repeat request(ARQ) technique; error correction using forward error correction (FEC).

forward error correction (FEC), FEC is used in simplex communications or applications where it is impractical or impossible to request a retransmission of the corrupted message block. An example might be the telemetry signals transmitted to an Earth station from a satellite on a deep space mission. A garbled message could take several minutes or even hours to travel the distance between the two stations. Redundant error-correction coding is include in the transmitted data stream .If an error is detected by the receiver, the redundant code is extracted from the message block and used to predict and possibly correct the discrepancy.

12.7.1 Hamming code

In FEC a return path is not used for requesting the retransmission of a message block in error, hence the name forward error correction. Several codes have been developed to suit applications requiring FEC. Those most commonly recognized have been based on the research of mathematician Trichard W. Hamming. These codes are referred to as ,Hamming codes. Hamming codes employ the use of redundant bits that are inserted into the message stream for error correction . The positions of these bits are established and known by the transmitter and received before hand. If the receiver detects an error in the message block, the Hamming bits are used to identify the position of the error. This position, known as the syndrome ,is the underlying principle of the Hamming code.

12.7.1.1 Developing a Hamming code

We will now develop a Hamming code for single-bit FEC. For simplicity, 10 data bits will be used . The number of Hamming bits depends on the number of data bits m_0, m_1......transmitted in the message stream, including the Hamming bits. If n is equal to the real number of bits transmitted in a message stream and m is equal to the number of Hamming bits, then m is the smallest number governed by the equation.

$$2^m > n+1$$

For a message of 10 data bits, m is equal to 4 and n is equal 14 bits (10+4)

$$2^4>(10+4)+1$$

If the syndrome is to indicate the position of the bit error, check bits, or Hamming bits c_0,c_1........ serving as parity can be inserted into the message stream to perform a parity check based on the binary representation of each bit position. How is this possible? Note in Table 12-2 that the binary representation of each bit position forms an alternating bit pattern in the vertical direction.. Each column proceeding from the LSB to the MSB alternates at one-half the rate of the

TABLE 12-2 Check Bits Can Be Used As Parity on Binary Weighted Positions in a Message Stream

Bit position in message	Binary representation	Check bit	Position set
1	0001	c_0	1,3,5,7,9,11,13
2	0010	c_1	2,3,6,7,10,11,14
3	0011	c_2	4,5,6,7,12,13,14
4	0100	c_3	8,9,10,11,12,13,14,
5	0101		
6	0110		
7	0111		
8	1000		
9	1001		
10	1010		
11	1011		
12	1100		
13	1101		
14	1110		
.	.		
.	.		
.	.		

previous column. The LSB alternates with every positions. The next bit alternates every two bit positions, and so forth.

To illustrate how the check bits are encoded, the 10-bit message 1101001110 is labeled m_9 through m_0, as illustrates in Figure12-13 By inserting the check bits into the message length n is extended to 14 bits. For simplicity, bit positions1,2,4,8 will be used for the .check bits. Even or odd parity generation can be performed own the bit positions associated with each can be performed by exclusive -ORing individual bits an a group of the bit. For even parity, PEO through PE3 can serve as weight parity checks over the bit positions listed in Table 12-2 Exclusive-ORing these bit positions together with the data corresponding to the 14 bit

message stream shown in Figure12-13, we the following:

$$\begin{array}{l} \qquad\quad 13 \quad\; 11 \quad 9 \quad\; 7 \quad\; 6 \quad\; 5 \quad\; 3 \quad 1 \leftarrow \text{bit position} \\ PE0 = 0 = m_8 \oplus m_6 \oplus m_4 \oplus m_3 \oplus m_2 \oplus m_1 \oplus m_0 \oplus c_0 \end{array}$$

$$\begin{array}{l} \qquad\quad 14 \quad\; 11 \quad 10 \quad\; 7 \quad\; 6 \quad\; 3 \quad\; 2 \leftarrow \text{bit position} \\ PE1 = 0 = m_9 \oplus m_6 \oplus m_5 \oplus m_3 \oplus m_2 \oplus m_0 \oplus c_1 \end{array}$$

$$\begin{array}{l} \qquad\quad 14 \quad 13 \quad 12 \quad\; 7 \quad\; 6 \quad\; 5 \quad\; 4 \leftarrow \text{bit position} \\ PE2 = 0 = m_9 \oplus m_8 \oplus m_7 \oplus m_3 \oplus m_2 \oplus m_1 \oplus c_2 \end{array}$$

$$\begin{array}{l} \qquad\quad 14 \quad 13 \quad 12 \quad 11 \quad 10 \quad\; 9 \quad\; 8 \leftarrow \text{bit position} \\ PE3 = 0 = m_9 \oplus m_8 \oplus m_7 \oplus m_6 \oplus m_5 \oplus m_4 \oplus c_3 \end{array}$$

To determine the value of the check bits c_0 through c_3 the equations above can be rearranged as follows:

m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
1	1	0	1	0	0	1	1	1	0

Original bit stream, 10 bits

check bits

m_9	m_8	m_7	m_6	m_5	m_4	c_3	m_3	m_2	m_1	c_2	m_0	c_1	c_0	
1	1	0	1	0	0	1	1	1	1	1	0	0	0	
14	13	12	11	10	9	8	7	6	5	4	3	2	1	← Bit position

Transmitted bit stream, 14 bits

Figure 12-13 Check bits are inserted into a message stream for FEC.

$$c_0 = m_8 \oplus m_6 \oplus m_4 \oplus m_3 \oplus m_1 \oplus m_0$$
$$= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$c_1 = m_9 \oplus m_6 \oplus m_5 \oplus m_3 \oplus m_2 \oplus m_0$$
$$= 1 + 1 + 0 + 1 + 1 + 0 = 0$$

even parity

$$c_2 = m_9 \oplus m_8 \oplus m_7 \oplus m_3 \oplus m_2 \oplus m_1$$
$$= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$c_3 = m_9 \oplus m_8 \oplus m_7 \oplus m_6 \oplus m_5 \oplus m_4$$
$$= 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

Thus the check bits inserted into the message stream in positions 8,4,2,and1 are

$c_3=1$
$c_2=1$
$c_1=0$
$c_0=0$

Let us now look at how a bit error can be identified and corrected by the weighted parity checks. Suppose that an error has been detected in the transmitted message stream. Bit positions 7 has been lost in the transmission

1 1 0 1 0 0 1 1 1 1 1 0 0 0 ← transmitted bit stream

↓ lost in transmission

1 1 0 1 0 0 1 0 1 1 1 0 0 0 ← received bit stream

↑ error in bit position 7

The receiver performs an even parity check over the same bit positions discussed above. Even parity should result for each parity check if there are no errors. Since a bit error has occurred, however, the syndrome(location of the error) will be identified by the binary number produced by the parity checks PEO through PE3, as follows:

14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	0	0	1	0	1	1	1	0	0	0

Check 0 : 13 11 9 7 5 3 1 bit position
PE0 = 1+ 1 + 0 + 0 + 1 + 0 + 0 = 1 (even parity failure)1
Check 1 : 14 11 10 7 6 3 2 bit position
PE1 = 1 +1 +0 + 0 + 1 + 0 + 0 =1 (even parity failure)1
Check 2 : 14 13 12 7 6 5 4 bit position
PE1 = 1 +1 +0 + 0 + 1 + 1 + 1 =1 (even parity failure)1
Check 3 : 14 13 12 11 10 9 8 bit position
PE3 = 1 +1 +0 +1 + 0 + 0 + 1 =0 (correct)0

syndrome
=0111
=7

The resulting syndrome is 0111, or bit position 7. This bit is simply inverted and the parity checks will result in 000(corrected) . The check bits are removed from positions 1, 2 , 4 and 8, there by resulting in the original message. One nice feature of this Hamming code is that once the message is encoded there is no differences between the check bits and the original message bits; that is. The syndrome can just as well identify a check bit in error.

17.2.1.2An alternative method.

Now we have established the principle behind a Hamming code, an alternative method for correcting a single-bit error will be given here. The disadvantage with this method, however, is that it 10-bit message stream, 1101001110,will be used. Therefore, the number of Hamming bits, four, remains the same. The Hamming bits can actually be placed anywhere in the transmitted message stream as their positions are known by the transmitter and received. The procedure is outlined as follows:

1- Compute the number of Hamming bits m required for a message of n bits.

Original message stream:: 1 1 0 1 0 0 1 1 1 0 (10bits)

$$2^m > n+1$$
$$2^4 > (10+4)+1 \quad m=4, \quad n=14$$

2 -Insert the Hamming bits H into the original message stream.

Transmitted message stream:

14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	H	1	0	H	1	0	0	1	H	1	H	1	0

3 - Express each bit positions containing as 1 as a four-bit binary number and exclusive-Or each of these numbers together. Starting from the left, bit positions 14, 12, 9, 6, 4, and 2 are exclusive-ORed together. This will b result in the value of the Hamming bits.

$$\begin{array}{rl}
 & 1110=14 \\
+ & \underline{1100=12} \\
 & 0010 \\
+ & \underline{1001=9} \\
 & 1011 \\
+ & \underline{0110=6} \\
 & 1101 \\
+ & \underline{0100=4} \\
 & 1001 \\
+ & \underline{0010=2} \\
 & 1011 \longleftarrow \text{Hamming bits}
\end{array}$$

4-Places the value of the Hamming bits into the H transmitted message stream shown in step 2.

14	13	12	11	10	9	8	7	6	5	4	3	2	1	bit position
1	1	1	0	0	1	0	0	1	1	1	1	1	0	transmitted bit stream
1	1	1	0	0	1	0	0	0	1	1	1	1	0	received bit stream

error in bit position 6

Let us now assume that bit position 6 was received in error.

5- The Hamming bits are extracted from the received message stream and exclusive-Ored with the binary representation of the bit positions containing a 1.This will detect the bit positions in error, or the syndrome,

14 13 12 11 10 9 8 7 6 5 4 3 2 1

1 1 0 1 0 0 0 1 1 0

1 0 1 1 extracted Hamming bits

1011 Hamming bits
\+ 1110=14
0101
\+ 1100=12
1001
\+ 1001=9
0000
\+ 0100=4
0100
\+ 0010=2
0110 syndrome equals bit position 6

To detect multiple bit errors, more elaborate FEC techniques are necessary. Additional redundancy must be built into the message stream. This further reduces the efficiency of the channel and lowers the transmission system's throughput. Unlike ARQ, which is extremely reliable, the best FEC techniques are not particularly in cases where multiple bits are destroyed due to noise bursts. Generally, FEC is employed only in applications where ARQ is not feasible. The detection of multiple- bit errors is beyond the scope of this book.

CONCLUSION

This graduation project training was the point where theoretic came together. The training gave a lot me. I discovered the computer world. There were learn the error correction, detection and control. Also I having great knowledge in computers. It helped me. You allows to project . Thank you for Prof.Dr. Fahrettin