CHAPTER 10

Error control coding

10.1 Introduction

The fundamental resources at the disposal of a communications engineer are signal power, time and bandwidth. For a given communications environment (summarised in this context by an effective noise power spectral density) these three resources can be traded against each other. The basis on which the trade-offs are made will depend on the premium attached to each resource in a given situation. A general objective, however, is often to achieve maximum data transfer, in a minimum bandwidth *while maintaining an acceptable quality of transmission.* The quality of transmission, in the context of digital communications, is essentially concerned with the probability of bit error, P_b, at the receiver. (There are other factors which determine transmission quality, in its widest sense, of course, but focussing on P_b makes the discussion, and more especially the analysis, tractable.)

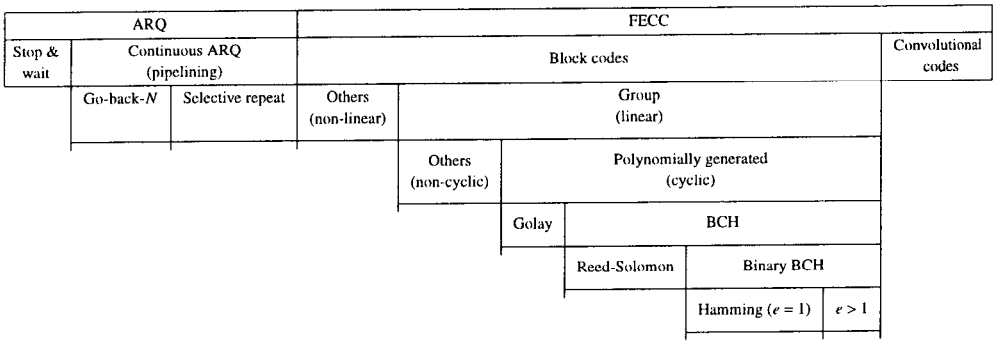
The Shannon-Hartley law (see section 11.4.1) for the capacity of a communications channel demonstrates two things. Firstly it shows (quantitatively) how bandwidth and signal power may be traded in an ideal system, and secondly it gives a theoretical limit for the transmission rate of (reliable, i.e. error free) data from a transmitter of given power, over a channel with a given bandwidth, operating in a given noise environment. In order to realise this theoretical limit, however, an appropriate coding scheme (which the Shannon-Hartley law assures us exists) must be found. (It should, perhaps, be noted at this point that there is one more quantity which must be traded in return for the advantage which such a coding scheme confers, i.e. time delay which results from the coding process.)

In practice, the objective of the design engineer is to realise the required data rate (often determined by the service being provided) within the bandwidth constraint of the available channel and the power constraint of the particular application. (In a mobile radio application, for example, bandwidth may be determined by channel allocations or frequency coordination considerations, and maximum radiated power may be determined by safety considerations or transceiver battery technology.) Furthermore this data rate

must be achieved with an *acceptable* BER and time delay. If an, essentially, uncoded PCM transmission cannot achieve the required BER within these constraints then the application of error control coding may be able to help, providing the constraints are not such as to violate the Shannon-Hartley law.

Error control coding (also referred to as channel coding) is used to detect, and often correct, symbols which are received in error. Error *detection* can be used as the initial step of an error *correction* technique by, for example, triggering a receiving terminal to generate an automatic repeat request (ARQ) signal which is carried, by the return path of a duplex link, to the originating terminal. A successful retransmission of the affected data results in the error being corrected. If ARQ techniques are inconvenient, as is the case, for example, when the propagation delay of the transmission medium is large, then forward error correction coding (FECC) may be appropriate [Blahut 1983, Clark and Cain, MacWilliams and Sloane]. FECC incorporates extra information (i.e. redundancy) into the transmitted data which can then be used not only to detect errors but also to correct them without the need for any retransmissions.

Table 10.1 *A taxonomy of error control codes.*



This chapter begins with a general discussion of error rate control, in its widest sense, as may be applied in digital communications systems. Five particular error control methods are identified and briefly described. It is one of these methods, namely FECC, which is then treated in detail during the remainder of the chapter. Some typical applications of FECC are outlined and the threshold phenomenon is highlighted. The Hamming distance between a pair of codewords and the weight of a codeword are defined. The discussion of FEC codes, that follows, is structured loosely around the taxonomy of codes shown in Table 10.1 starting with a description of block codes and including special mention of linear group codes, cyclic codes, the Golay code, BCH codes, Reed-Solomon codes and Hamming codes. The Hamming bound on the performance of a block code is derived and strategies for nearest neighbour, or maximum likelihood, decoding are discussed. Convolution coding is treated towards the end of the chapter. Tree, trellis and state transition diagrams are used to illustrate the encoding process. The significance of constraint length and decoding window length are discussed and Viterbi decoding is illustrated using a trellis diagram. Decoding is accomplished by finding the most likely path through the trellis and relating this back to the transmitted

data sequence.

Coding concepts are presented, here, with an essentially non-mathematical treatment, in the context of specific examples of group, cyclic and convolutional codes. The chapter concludes with a brief discussion of practical coders.

10.1.1 Error rate control concepts

The normal measure of error performance is bit-error rate (BER) or the probability of bit error (P_b). P_b is simply the probability of any given transmitted binary digit being in error. The bit error rate is, strictly, the average rate at which errors occur and is given by the product $P_b R_b$, where R_b is the bit transmission rate in the channel. Typical long term P_b for linear PCM systems is 10^{-7} while, for companded PCM, it is 10^{-5} and for ADPCM (Chapter 5) it is 10^{-4}. If the error rate of a particular system is too large then what can be done to make it smaller? The first and most obvious solution is to increase transmitter power, but this may not always be desirable, for example in man-portable systems where the required extra battery weight may be unacceptable.

A second possible solution, which is especially effective against burst errors caused by signal fading, is to use diversity. There are three main types of diversity: space diversity, frequency diversity, and time diversity. All these schemes incorporate redundancy in that data is, effectively, transmitted twice: i.e. via two paths, at two frequencies, or at two different times. In space diversity two or more antennas are used which are sited sufficiently far apart for fading at their outputs to be decorrelated. Frequency diversity employs two different frequencies to transmit the same information. (Frequency diversity can be in-band or out-band depending upon the frequency spacing between the carriers.) In time diversity systems the same message is transmitted more than once at different times.

A third possible solution to the problem of unacceptable BER is to introduce full duplex transmission, implying simultaneous 2-way transmission. Here when a transmitter sends information to a receiver, the information is 'echoed' back to the transmitter on a separate feedback channel. Information echoed back which contains errors can then be retransmitted. This technique requires twice the bandwidth of single direction (simplex) transmission, however, which may be unacceptable in terms of spectrum utilisation.

A fourth method for coping with poor BER is automatic repeat request (ARQ). Here a simple error *detecting* code is used and, if an error is detected in a given data block, then a request is sent via a feedback channel to retransmit that block. There are two major ARQ techniques. These are *stop and wait*, in which each block of data is positively, or negatively, acknowledged by the receiving terminal as being error free before the next data block is transmitted, and *continuous* ARQ, in which blocks of data continue to be transmitted without waiting for each previous block to be acknowledged. (In stop and wait ARQ data blocks are *timed out* if neither a positive nor negative acknowledgement is received within a predetermined time window. After timing out the appropriate data block is retransmitted in the same way as if it had been negatively acknowledged.) Continuous ARQ can, in turn, be divided into two variants. In the *go-*

back-n version, data blocks carry a sequence or reference number, n. (This is a *different n* to that used in the (n, k) block code notation later.) Each acknowledgement signal contains the reference number of a data block and effectively acknowledges all data blocks up to $n - 1$. When a negative acknowledgement is received (or a data block timed out) all data blocks starting from the reference number in the last acknowledgement signal are retransmitted. In the *selective repeat* version only those data blocks explicitly negatively acknowledged (or timed out) are retransmitted (necessitating data block reordering buffers at the receiver). Go-back-n ARQ has a well defined storage requirement whilst selective repeat ARQ, although very efficient, has a less well defined, and potentially much lager, storage requirement, especially when deployed on high speed links. ARQ is very effective, for example in facsimile transmission, Chapter 9. On long links with fast transmission rates, however, such as is typical in satellite communications, ARQ can be very difficult to implement.

The fifth technique for coping with high BER is to employ forward error correction coding (FECC). In common with three of the other four techniques FECC introduces redundancy, this time with data check bits interleaved with the information traffic bits. It relies on the number of errors in a long block of data being close to the statistical average and, being a forward technique, requires no return channel. The widespread adoption of FECC was delayed, historically, because of its complexity and high cost of implementation relative to the other possible solutions. Complexity is now less of a problem following the proliferation of VLSI custom coder/decoder chips.

FECC exploits the difference between the transmission rate or information bit rate R_b and the channel capacity $R_{\max}$ as given by the Shannon-Hartley law (see equation (11.38)). P_b can be reduced, at the expense of increasing the transmission delay [Schwartz, 1987], by using FECC with a sufficiently long block or constraint length. The increased transmission delay arises due to the need to assemble the data blocks to be transmitted and the time spent in examining received data blocks to correct errors. The benefits of error control, however, usually outweigh the inherent FECC processor delay disadvantages.

10.1.2 Threshold phenomenon

Figure 10.1 illustrates the error rate for an uncoded system in which P_b increases gradually as SNR decreases, as shown previously in Figure 6.3. Figure 10.1 is plotted as the ratio of bit energy to noise power spectral density (E_b/N_0), which is defined later in Chapter 11. With FECC the P_b versus E_b/N_0 curve is steeper. If the SNR is above a certain value, which here corresponds to an E_b/N_0 of around 6 dB, the error rate will be virtually zero. Below this value system performance degrades rapidly until the coded system is actually poorer than the corresponding uncoded system. (The reason for this is that there is a region of low E_b/N_0 where, in attempting to correct errors, the decoder approximately doubles the number of errors in a decoded codeword.) This behaviour is analogous to the threshold phenomenon in wide band frequency modulation. A coding gain can be defined, for a given P_b, by moving horizontally in Figure 10.1 from the uncoded to the coded curve. The value of the coding gain in dB is relatively constant for

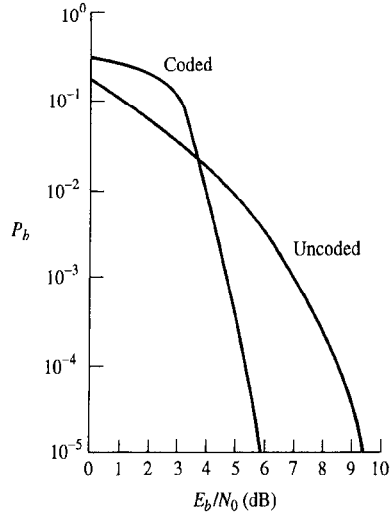


Figure 10.1 *The threshold phenomenon in FECC systems.*

$P_b \leq 10^{-5}$, and is dependant on the precise details of the FECC system deployed.

10.1.3 Applications for error control

Compact disc players provide a growing application area for FECC. In CD applications the powerful Reed-Solomon code is used since it works at a symbol level, rather than at a bit level, and is very effective against burst errors, particularly when combined with interleaving to randomise the bursts. The Reed-Solomon code is also used in computers for data storage and retrieval. Cosmic particles create, on average, one error every two to three days in a 4 Mbyte memory, although small geometry devices are helping to reduce this probability. Digital audio and video systems are also areas in which FECC is applied. Error control coding, generally, is applied widely in control and communications systems for aerospace applications, in mobile (GSM) cellular telephony and for enhancing security in banking and barcode readers.

10.2 Hamming distance and codeword weight

Before embarking on a detailed discussion of code performance the following definitions are required. The Hamming distance between two codewords is defined as the number of places, bits or digits in which they differ. This distance is important since it determines how easy it is to change or alter one valid codeword into another. The weight of a binary codeword is defined as the number of ones which it contains.

EXAMPLE 10.1
Calculate the Hamming distance between the two codewords 11100 and 11011 and find the minimum codeword weight.

The two codewords 11100 and 11011 have a Hamming distance of 3 corresponding to the differences in the 3rd, 4th and 5th digit positions. Thus with three appropriately positioned errors in these locations the codeword 11100 could be altered to 11011.
In this example, 11011 has a weight of 4 due to the four ones and 11100 has a weight of 3. The minimum weight is thus 3. Hamming distance and weight will be used later to bound the error correcting performance of codewords.

10.3 (n, k) Block codes

Figure 10.2 illustrates a block coder with k information digits going into the coder and n digits coming out after the encoding operation. The n-digit codeword is thus made up of k information digits and $(n-k)$ redundant *parity check* digits. The rate, or efficiency, for this code (R) is k/n, representing the ratio of information digits to the total number of digits in the codeword. Rate is normally in the range ½ to unity. (Unlike source coding in which data is compressed, here redundancy is deliberately added, to achieve error detection.) This is an example of a *systematic* code in that the information digits are explicitly transmitted together with the parity check digits, Figure 10.2. In a non-systematic code the n digit codeword may not contain any of the information digits explicitly. There are two definitions of systematic codes in the literature. The stricter of the two definitions assumes that, for the code to be systematic, the k information digits must be transmitted contiguously as a block, with the parity check digits making up the codeword as another contiguous block. The less strict of the two definitions merely stipulates that the information digits must be included in the codeword but not necessarily in a contiguous block. The latter definition is the one which is adopted here.

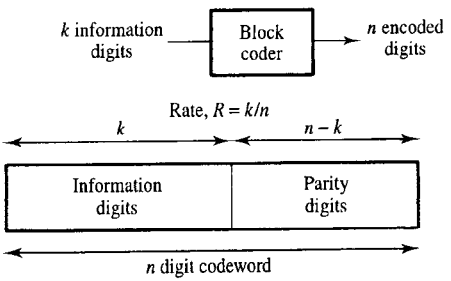


Figure 10.2 *(n, k) systematic block code.*

10.3.1 Single parity check code

The example in Figure 10.3 will be familiar to many as an option in ASCII coded data transmission (see Chapter 8). Consider the data sequence 1101000, to which a single parity check digit (P) is added. For even parity in this example sequence, P will be 1. For odd parity, P will be 0. Since the seven information digits contain three ones, another 1 must be added giving an even number of ones to achieve even parity. Alternatively if a zero is added, ensuring an odd number of ones, i.e. odd parity, the transmission of the all zero codeword is avoided. Even parity is, however, much more common.

Here the rate, $R = k/n$, is seven eighths which represents a very low level of redundancy. This scheme can only identify an odd number of errors because an even number of errors will not violate the chosen parity rule. Single error detection, as illustrated in Figure 10.4, is often used to extend a 7-bit word, with a checksum bit, into an 8-bit codeword. Another example, used in libraries, is the 10-digit ISBN codeword, Figure 10.5. This uses a modulo 11, weighted, checksum in which the weightings are 10 for the first digit, 9 for the second digit, etc. down to 2 for the ninth digit (and 1 for the checksum). The weighting can be applied either left to right or vice versa and a checksum digit of 10 is represented by the symbol C.

Single parity checks are also used on rows and columns of simple two-dimensional data arrays, Figure 10.6. Single errors in the array will be detected *and located* via the corresponding row and column parity bits. Such errors can therefore be corrected. Double errors can be detected but not necessarily corrected as several error patterns can produce the same parity violations. When the data is an array of ASCII characters the row and column check words can also be sent as ASCII characters.

In the English language there is a high level of redundancy. This is why spelling mistakes can be corrected and abbreviations expanded. There is, in fact, an approximate correspondence between the words of a language and code words as being discussed here, although in language contextual information goes beyond isolated words whilst in a block code each codeword is decoded in isolation.

Modulo $2^n - 1$ checksums are in widespread use for performing error detection on byte-serial network connections. They are usually computed by software during the data block (or packet) construction. One such error detection code is the Internet checksum for protocol messages [Comer], Chapter 18. If a message of length w (16-bit) bytes:

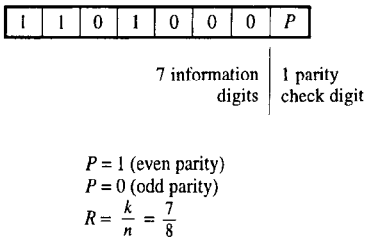


Figure 10.3 *Example of a single parity check digit codeword.*

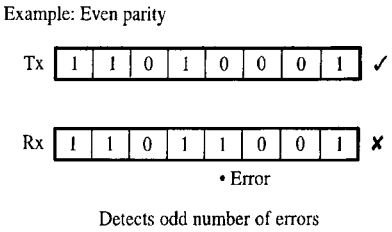


Figure 10.4 *Block code with single parity check error detecting capability.*

Checksum, $C = 11 - \sum_{i=1}^{9}(11 - i)k_i \pmod{11}$

i	1	2	3	4	5	6	7	8	9	C
ISBN	0	1	9	8	5	3	8	0	4	9

Figure 10.5 *ISBN codeword and checksum calculated to satisfy* $\sum_{i=1}^{10}(11 - i)k_i = 0$ *(mod-11).*

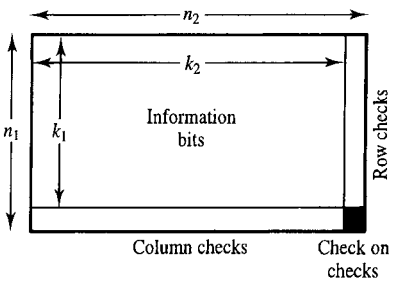


Figure 10.6 *Two-dimensional row–column array code.*

$m_{w-1}, \cdots, m_0$ is to be checksummed, the one-byte checksum is just the complement of:

$$\sum_{i=0}^{w-1} m_i \pmod{-65535}$$

The main operation required for this is summation modulo-65535 (i.e. summation using one's complement word addition). The checksum is included with the transmitted message allowing a recipient to check for transmission errors by performing a similar summation over the received data. If this sum is not zero then a channel error has occurred. Another error detection code is the ISO two-byte checksum [Fletcher]. The checksum is again included within the transmitted message and a recipient can perform a summation over the received data to confirm that both checksum bytes are zero; if not, a channel error has occurred.

EXAMPLE 10.2

Figure 10.7 illustrates a seven digit codeword with four information digits (I_1 to I_4) and three parity check digits (P_1 to P_3), commonly referred to as a (7,4) block code. The circles indicate how the information bits contribute to the calculation of each of the parity check bits. Assuming even parity, show the realisation of this encoder using 3-input modulo-2 adders. Calculate the individual parity check bits and encoding of P_1, P_2 and P_3 for the information digits 1011.

Figure 10.7 shows P_1 represented by the modulo-2 sum of I_1, I_3 and I_4. P_2 is the sum of I_1, I_2 and I_4, etc. (Modulo-2 arithmetic was used previously in Table 8.1.) The parity check digits are generated by the circuit in Figure 10.8. For the data sequence 1011, the 3-input modulo-2 adders count the total number of ones which are present at the inputs, and output the least significant bit as the binary coded sum. Thus $P_1 = 1$, $P_2 = 0$ and $P_3 = 0$, giving a coder output of 1011100.

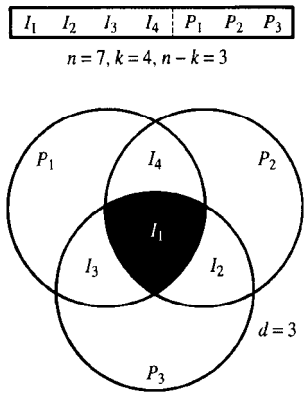


Figure 10.7 *Representation of relationship between parity check and data bits*

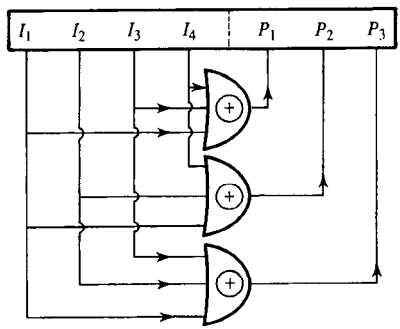


Figure 10.8 *(7,4) block code hardware generation of three parity check digits.*

Figure 10.9 shows how parity check equations for P_1, P_2 and P_3 in the above example may be written using $\oplus$ to represent the modulo-2 or exclusive-or arithmetic operation. Figure 10.9 also shows how these equations can be reduced to matrix form in a *parity check* matrix **H**. The coefficients of the information digits I_1, I_2, I_3 and I_4 are to the left of the dotted partition in the parity check matrix. The top row of the matrix contains the information about parity check P_1, the second row about parity check P_2 and the third row about parity check P_3.

Consider the top left hand part of the matrix (1011). The coefficients 1011 correspond to the information digits I_1, I_3 and I_4 in the equation for P_1. Similarly, the second row is 1101 to the left of the partition because I_3 is not involved in calculating parity check P_2 and the corresponding part of the bottom row is 1110 because I_4 is not involved in calculating the parity check P_3, Figure 10.8. To the right of the dotted partition there is a 3×3 diagonal matrix of ones. Each column in this diagonal matrix corresponds to a particular parity check digit. The first column (100) indicates parity check P_1. The second column indicates parity check P_2 and the third column parity check P_3. Later, in section 10.7, a *generator* matrix will be used to obtain the codeword directly from the information vector.

10.4 Probability of error in n-digit codewords

What is the probability of having more than R' errors in an n-digit codeword? First consider the case of exactly j errors in n digits with a probability of error per digit of P_e. From Chapter 3, equation (3.8):

$$P(j \text{ errors}) = (P_e)^j \, (1 - P_e)^{n-j} \times {}^nC_j \tag{10.1}$$

The probability of having more than R' errors can be written as:

$$P(> R' \text{ errors}) = 1 - \sum_{j=0}^{R'} P(j) \tag{10.2}$$

Statistical stability controls the usefulness of this equation, statistical convergence occurring for long code words or blocks (see Figure 3.2). A long block effectively embodies a large number of trials, to determine whether or not an error will occur, and the number of errors in such a block will therefore be close to $P_e n$. Furthermore, the fraction of blocks containing a number of errors that deviates significantly from this value

$$\begin{aligned} P_1 &= 1 \times I_1 \oplus 0 \times I_2 \oplus 1 \times I_3 \oplus 1 \times I_4 \\ P_2 &= 1 \times I_1 \oplus 1 \times I_2 \oplus 0 \times I_3 \oplus 1 \times I_4 \\ P_3 &= 1 \times I_1 \oplus 1 \times I_2 \oplus 1 \times I_3 \oplus 0 \times I_4 \end{aligned}$$

$$\mathbf{H} = \left[\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Figure 10.9 *Representation of the code in Figure 10.7 by parity check equations and an* **H** *matrix.*

becomes smaller as the block length, n, becomes larger, Figure 3.2. Choosing a code that can correct $P_e n$ errors in a block will ensure that there are very few cases in which the coding system will fail. This is the rationale for long block codes. The attraction of block codes is that they are amenable to precise performance analysis. By far the most important, and most amenable, set of block codes are the linear group codes.

10.5 Linear group codes

The codewords in a linear group code have a one-to-one correspondence with the elements of a mathematical group. Group codes contain the all-zeros codeword and have the property referred to as closure. That is, taking any two codewords C_i and C_j, then $C_i \oplus C_j = C_k$. (For the all zeros codeword, when $i = 0$, then $k = j$.) Thus adding, modulo-2, corresponding pairs of digits in each of the codewords produces another codeword C_k. The presence of the all-zeros codeword and the closure property together make performance calculations with linear group codes particularly easy, as will be seen later. Figure 10.10 illustrates a simple group code which will be used as an example in this chapter. It is first used to illustrate the property of closure. Figure 10.10 depicts a source alphabet with 4 members: a, b, c and d (i.e. the number of information digits is $k = 2$). Each symbol is coded into an n-digit codeword (where $n = 5$) as shown. This is therefore a (5,2) code. (By the less strict definition, this is also a systematic code where the information digits are in columns 1/2 and 4/5.) Consider the codewords corresponding to c and b. Modulo-2 summing c and b gives the codeword d, illustrating the closure property.

10.5.1 Members of the group code family

Group codes can be divided into two types: those which are 'polynomial generated' in simple feedback shift registers; and others. The simplicity of the former have rendered the rest irrelevant. The polynomial generated codes can be further divided into subgroups, the main ones being the binary Bose–Chaudhuri–Hocquenghem (BCH) codes

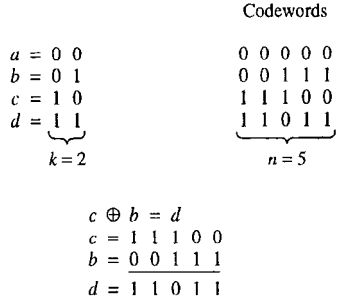


Figure 10.10 *Illustration of the closure property of a group code.*

and their important, non-binary counterpart, the Reed-Solomon codes. BCH codes are widely tabulated up to $n = 255$ with an error correcting capability of up to 30 digits [Blahut, 1983]. Generally speaking for the same error correcting capability a larger (e.g. $n = 255$) block size offers a higher rate than a shorter (e.g. $n = 63$) block size. Reed-Solomon, non-binary, byte organised codes are used extensively in compact disc players and computer memories.

10.5.2 Performance prediction

Normally all possible codeword pairs would have to be examined, and their Hamming distances measured, to determine the overall performance of a block code. For the case of group codes, however, consideration of each of the codewords with the all-zeros codeword is sufficient. This is a significant advantage of linear group codes and one reason why these codes are so important in relation to other block codes. (Analysis for large n becomes much simpler for group codes, since the number of combinations of codewords, which would otherwise have to be searched, is very large.)

The important quantity, as far as code performance prediction is concerned, is the minimum Hamming distance between any pair of codewords. For the four five-digit codewords in Figure 10.10 – 0 0 0 0 0, 0 0 1 1 1, 1 1 1 0 0, 1 1 0 1 1, inspection reveals a minimum Hamming distance of 3, i.e. $D_{\min} = 3$ for this (5,2) code. Other (n, k) block codes with this minimum distance of 3 are (3,1), (15,11), (31,26), etc.

The weight *structure* of a set of codewords is just a list of the weights of all the codewords. Consider the previous example with 4 codewords: the weights of these are 0, 3, 3 and 4. Ignoring the all-zeros word (as interest is concentrated in the distances from this codeword), the minimum weight in the weight structure (3) is equal to $D_{\min}$, the minimum Hamming distance for the code.

Consider the probability of the ith codeword (C_i) being misinterpreted as the jth codeword (C_j). This probability depends on the distance between these two codewords (D_{ij}). Since this is a linear group code, this distance D_{ij} is equal to the weight of a third codeword C_k which is actually the modulo-2 sum of C_i and C_j. The probability of C_i being mistaken for C_j is therefore equal to the probability of C_k being mistaken for C_0. Furthermore, the probability of C_k being mistaken for the all-zeros codeword (C_0) is equal to the probability of C_0 being misinterpreted as C_k (by symmetry). The probability of C_0 being misinterpreted as C_k, depends only on the weight of C_k.

This reasoning reveals the importance of a linear group code's weight structure since the performance of such a code can be determined completely by consideration of C_0 and the weight structure alone.

10.5.3 Error detection and correction capability

The maximum possible error correcting power, t, of a code is defined by its ability to correct *all* patterns of t or less errors. It is related to the code's minimum Hamming distance by:

$$t = int\left(\frac{D_{\min} - 1}{2}\right) \tag{10.3(a)}$$

where:

$$D_{\min} - 1 = e + t \tag{10.3(b)}$$

Here *int*( ) indicates 'the integer part of', e is the total number of *detectable* errors (including the correctable t errors) and $t \leq e$. Taking the case where $D_{\min}$ is 3, then there are at least two possible binary words which lie between each pair of valid codewords. In Example 10.1 these could be the binary words 11000 and 11001. If any single error occurs in one of the code words it can therefore be corrected. Alternatively, if there is no error correction $D_{\min} - 1$ errors can be detected (2 in this case as both 11000 and 11001 are detectable as errors). Note that the code *cannot* work in both these detection and correction modes simultaneously (i.e. detect two errors and correct one of them).

Longer codes with larger Hamming distances offer greater detection and correction capability by selecting different t and e values in equation (10.3). $D_{\min} = 7$ can offer $t =$ 1 bit correction combined with $e = 5$ bit error detection. If t is increased to 2 then e must decrease to 4. The UK Post Office Code Standards Advisory Group (POCSAG) code with $k = 21$ and $n = 32$ is an $R = 2/3$ code with $D_{\min} = 6$. This provides a 3-bit error *detection* or a 2-bit error *correction* capability, for a codeword which is widely used in pager systems, see Chapter 15. The $n = 63$, $k = 57$, BCH code gives $R = 0.9$ with $t = 1$ bit, while reducing k to 45 gives $R = 0.7$ with $t = 3$ bit. Further reducing k to 24 reduces R to below 0.4 but achieves a $t = 7$ bit correction capability. This illustrates the important trade-off between rate and error correction power. BCH codes can correct burst, as well as random, errors.

10.6 Nearest neighbour decoding of block codes

Encoding is achieved by use of a feedback shift register and is relatively simple as will be shown later. The two most important strategies for decoding are nearest neighbour and maximum likelihood decoding. These are equivalent if the probability of t errors is much greater than that of $t + 1$ errors, etc. as in Example 3.4. Using a decoding table based on nearest neighbours, therefore, implies the maximum likelihood decoding strategy, as discussed in the context of decision theory in Chapter 9. This is illustrated with a simple example.

Figure 10.11 is a nearest neighbour decoding table for the previous four-symbol example of Figure 10.10. The codewords are listed along the top of this table starting with the all-zeros codeword in the top left hand corner. Below each codeword all possible received sequences are listed which are at a Hamming distance of 1 from this codeword. (In the case of the all-zeros codeword these are the sequences 10000 to 00001.) If this were a t error correcting code this list would continue with all the patterns of 2 errors, 3 errors, etc. up to all patterns of t errors. Any detected bit pattern appearing in the table is interpreted as representing the codeword at the top of the relevant column,

thus allowing the bit errors to be corrected. Below the table in Figure 10.11 there are eight 5-bit words which lie outside the table. These received sequences are equidistant from two possible codewords, so these sequences lie on a decision boundary. It is not possible, therefore, to decide which of the two original codewords they came from, and consequently the errors cannot be corrected. These sequences were referred to previously as *detectable* error sequences.

10.6.1 Hamming bound

Consider the possibility of a code with codewords of length n, comprising k information digits and having error correcting power t. There is an upper bound on the performance of block codes which is given by:

$$2^k \leq \frac{2^n}{1 + n + {}^nC_2 + {}^nC_3 + \cdots + {}^nC_t} \tag{10.4}$$

The simplest way to derive equation (10.4) is to inspect the nearest neighbour decoding table for the (n, k), t-error correcting code. Figure 10.12 develops Figure 10.11 into the general case of a t-error correcting code with 2^k codewords. There are, thus, 2^k columns

Codewords	00000	11100	00111	11011
Single-bit error correctable patterns	10000	01100	10111	01011
	01000	10100	01111	10011
	00100	11000	00011	11111
	00010	11110	00101	11001
	00001	11101	00110	11010
Double-bit error detectable patterns	10001	01101	10110	01010
	10010	01110	10101	01001

Figure 10.11 *Nearest neighbour decoding table for the group code of Figure 10.9.*

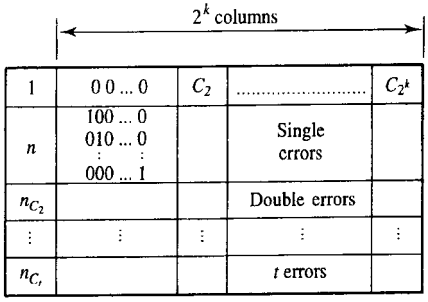


Figure 10.12 *Decoding table for a t-error correcting (n, k) block code.*

in the decoding table. Consider the left hand column. The all-zeros codeword itself is, obviously, one possible correctly received sequence or valid codeword. Also there are n single error patterns associated with that all-zeros codeword. Further, there are ${}^{n}C_2$ patterns of 2 errors, etc. down to ${}^{n}C_t$ patterns of t errors. Totalling the number of entries in this column reveals the total number of rows in the table and the value of the denominator in equation (10.4). Taking this number of rows and dividing into 2^n (which is the total number of possible received sequences), as in equation (10.4), gives the maximum possible number of columns and hence the maximum number of codewords in the given code. If the left hand side of equation (10.4) is greater than the right hand side then no such code exists and n must be increased, k decreased, or t decreased, until equation (10.4) is satisfied. For a *perfect* code, equation (10.4) is an equality. This implies that there are no bit patterns which lie outside the decoding table, avoiding the problem of equidistant errors which occurred in the code of Figure 10.11.

10.7 Syndrome decoding

The difficulty with decoding of block codes using the nearest neighbour decoding table of Figure 10.12 is the physical size of the table for large n. The syndrome decoding technique described here provides a solution to this problem.

10.7.1 The generator matrix

The generator matrix is a matrix of basis vectors. The rows of the generator matrix **G** are used to derive the actual transmitted codewords. This is in contrast with the **H** (or parity check) matrix, Figure 10.9, which does not contain any codewords. The generator matrix **G** for an (n, k) block code can be used to generate the appropriate n-digit codeword from any given k-digit data sequence. The **H** and corresponding **G** matrices for the example (7,4) block code of Figure 10.8 are shown below:

$$\mathbf{H} = \begin{bmatrix} 1\,0\,1\,1 : 1\,0\,0 \\ 1\,1\,0\,1 : 0\,1\,0 \\ 1\,1\,1\,0 : 0\,0\,1 \end{bmatrix} \tag{10.5}$$

$$\mathbf{G} = \begin{bmatrix} 1\,0\,0\,0 : 1\,1\,1 \\ 0\,1\,0\,0 : 0\,1\,1 \\ 0\,0\,1\,0 : 1\,0\,1 \\ 0\,0\,0\,1 : 1\,1\,0 \end{bmatrix} \tag{10.6}$$

Study of **G** shows that on the left of the dotted partition there is a 4×4 unit diagonal matrix and on the right of the partition there is a parity check section. This part of **G** is the transpose of the left hand portion of **H**. As this code has a single error correcting capability then $D_{\min}$, and the weight of the codeword, must be 3. As the identity matrix has a single one in each row then the parity check section must contain at least two ones. In addition to this constraint, rows cannot be identical.

Continuing the (7,4) example, we now show how **G** can be used to construct a codeword, using the matrix equation (4.13) [Spiegel]. Assume the data sequence is 1001. To generate the codeword associated with this data sequence the data vector 1001 is multiplied by **G** using modulo-2 arithmetic:

$$[1\ 0\ 0\ 1]\begin{bmatrix} 1\ 0\ 0\ 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 0\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 1\ 1\ 1\ 0 \end{bmatrix} = [1\ 0\ 0\ 1\ 0\ 0\ 1] \tag{10.7}$$

The 4×4 unit diagonal matrix in the lefthand portion of **G** results in the data sequence 1001 being repeated as the first four digits of the codeword and the right hand (parity check) portion results in the three parity check digits P_1, P_2 and P_3 (in this case 001) being calculated. (This generator matrix could, therefore, be applied to solve the second part of Example 10.2.)

It is now possible to see why the columns to the right of the partition in **G** are the rows of **H** to the left of its partition. From another standpoint the construction of a codeword is viewed as a weighted sum of the rows of **G**. The digits of the data sequence perform the weighting. With digits 1001 in this example, the top row of **G** is weighted by 1, the second row by 0, the third row by 0 and the fourth row by 1. After weighting the corresponding digits from each row are added modulo-2 to obtain the required codeword.

10.7.2 Syndrome table for error correction

Recall the strong inequality that the probability of t errors is much greater than the probability of $t+1$ errors. This situation always holds in the P_e regime where FECC systems normally operate. Thus nearest neighbour decoding is equivalent to maximum likelihood decoding. Unfortunately, the nearest neighbour decoding table is normally too large for practical implementation which requires a different technique, involving a smaller table, to be used instead. This table, referred to as the syndrome decoding table, is smaller than the nearest neighbour table by a factor equal to the number of codewords in the code set (2^k). This is because the syndrome is independent of the transmitted codeword and only depends on the error sequence as is demonstrated below.

When **d** is a message vector of k digits, **G** is the $k \times n$ generator matrix and **c** is the n digit codeword corresponding to the message **d**, equation (10.7) can be written as:

$$\mathbf{d}\,\mathbf{G} = \mathbf{c} \tag{10.8}$$

Furthermore:

$$\mathbf{H}\,\mathbf{c} = \mathbf{0} \tag{10.9}$$

where **H** is the (even) parity check matrix corresponding to **G** in equation (10.8). Also:

$$\mathbf{r} = \mathbf{c} \oplus \mathbf{e} \tag{10.10}$$

where **r** is the sequence received after transmitting **c**, and **e** is an error vector representing the location of the errors which occur in the received sequence **r**. Consider the product **H**

$\mathbf{H}\,\mathbf{e} = \mathbf{s}$

Error pattern	Syndrome
0 0 0 0 0 0 0	0 0 0
1 0 0 0 0 0 0	1 1 1
0 1 0 0 0 0 0	0 1 1
0 0 1 0 0 0 0	1 0 1
0 0 0 1 0 0 0	1 1 0
0 0 0 0 1 0 0	1 0 0
0 0 0 0 0 1 0	0 1 0
0 0 0 0 0 0 1	0 0 1

Figure 10.13 *The complete syndrome table for all possible single error patterns.*

r which is referred to as the syndrome vector **s**:

$$\begin{aligned}\mathbf{s} &= \mathbf{H}\,\mathbf{r} = \mathbf{H}\,(\mathbf{c} \oplus \mathbf{e}) \\ &= \mathbf{H}\,\mathbf{c} \oplus \mathbf{H}\,\mathbf{e} = \mathbf{0} \oplus \mathbf{H}\,\mathbf{e}\end{aligned} \tag{10.11}$$

Thus **s** is easily calculated and, if there are no received errors, the syndrome will be the all-zero vector **0**. Calculating the vector **s** provides immediate access to the vector **e** and hence the position of the errors. A syndrome table is constructed by assuming transmission of all the zeros codeword and calculating the syndrome vector associated with each correctable error pattern:

$$\begin{bmatrix} 1&0&1&1&1&0&0 \\ 1&1&0&1&0&1&0 \\ 1&1&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{10.12}$$

Equation 10.12 illustrates the case of no errors in the received sequence leading to the all zeros syndrome for the earlier (7,4) code example. Figure 10.13 shows the full syndrome table for this (7,4) code. In this case only single errors are correctable and the syndrome table closely resembles the transposed matrix $\mathbf{H}^T$. If a double error occurs then it will normally give the same syndrome as some single error and, since single errors are much more likely than double errors, a single error will be assumed and the wrong codeword will be output from the decoder resulting in a 'sequence' error. This syndrome decoding technique is still a nearest neighbour (maximum likelihood) decoding strategy.

EXAMPLE 10.3

As an example of the syndrome decoding technique, assume that the received vector for the (7,4) code is **r** = 1001101 and find the correct transmitted codeword.

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

The above matrix equation illustrates calculation of the corresponding syndrome (100). Reference to the syndrome table (Figure 10.13) reveals the corresponding error pattern as (0000100). Finally $\mathbf{c} = \mathbf{r} \oplus \mathbf{e}$:

$\mathbf{r} = 1\,0\,0\,1\,1\,0\,1$
$\mathbf{e} = 0\,0\,0\,0\,1\,0\,0$
$\mathbf{c} = 1\,0\,0\,1\,0\,0\,1$

to give the corrected transmitted codeword **c** as 1001001.

EXAMPLE 10.4

For a (6, 3) systematic linear block code, the codeword comprises $I_1\ I_2\ I_3\ P_1\ P_2\ P_3$ where the three parity-check bits $P_1\ P_2$ and P_3 are formed from the information bits as follows:

$P_1 = I_1 \oplus I_2$
$P_2 = I_1 \oplus I_3$
$P_3 = I_2 \oplus I_3$

Find: (a) the parity check matrix; (b) the generator matrix; (c) all possible codewords. Determine (d) the minimum weight; (e) the minimum distance; and (f) the error detecting and correcting capability of this code. (g) If the received sequence is 101000, calculate the syndrome and decode the received sequence.

(a)

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

(b)

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

(c) and (d)

Message $\times$ **G**	=	Codeword	Weight
[0 0 0] $\times$ **G**	=	0 0 0 0 0 0	0
[0 0 1] $\times$ **G**	=	0 0 1 0 1 1	3
[0 1 0] $\times$ **G**	=	0 1 0 1 0 1	3
[1 0 0] $\times$ **G**	=	1 0 0 1 1 0	3
[0 1 1] $\times$ **G**	=	0 1 1 1 1 0	4

$$\begin{array}{lll} [1\,0\,1]\times\mathbf{G} & = 1\,0\,1\,1\,0\,1 & 4 \\ [1\,1\,0]\times\mathbf{G} & = 1\,1\,0\,0\,1\,1 & 4 \\ [1\,1\,1]\times\mathbf{G} & = 1\,1\,1\,0\,0\,0 & 3 \end{array}$$

Minimum weight = 3.

(e) $D_{\min}$ = minimum weight = 3.

(f) The code is thus single error correcting (or double error detecting if no correction is required).

(g) Now using $\mathbf{H}\,\mathbf{r} = \mathbf{s}$, equation (10.11) becomes:

$$\mathbf{s} = \begin{bmatrix} 1\,1\,0\,1\,0\,0 \\ 1\,0\,1\,0\,1\,0 \\ 0\,1\,1\,0\,0\,1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

The decoded codeword could be found by constructing the syndromes for all possible error patterns and modulo-2 adding the appropriate error pattern to the received bit pattern. It is clear, however, that the decoded codeword is [1 1 1 0 0 0] as this is the closest valid codeword to the received bit pattern.

10.8 Cyclic codes

Cyclic codes are a subclass of group codes which do not possess the all-zeros codeword. The Hamming code is an example of a cyclic code. Their properties and advantages are as follows:

- Their mathematical structure permits higher order correcting codes.
- Their code structures can be easily implemented in hardware by using simple shift registers and exclusive-or gates.
- Cyclic code members are all lateral, or cyclical, shifts of one another.
- Cyclic codes can be represented as, and derived using, polynomials.

The third property, listed above, can be expressed as follows. If:

$$C = (I_1, I_2, I_3, \cdots, I_n) \tag{10.13}$$

then:

$$C_i = (I_{i+1}, \cdots, I_n, I_1, I_2, \cdots, I_i) \tag{10.14}$$

(C_i is an i bit cyclic shift of C, and $I_1, \cdots, I_n$ now represents both parity and information bits.) The following three codewords provide an example:

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

Full description of these codes requires a detailed discussion of group and field theory and takes us into a discussion of prime numbers and Galois fields. Non-systematic cyclic codes are obtained by multiplying the data vector by a generator polynomial with modulo arithmetic. In general cyclical codes are generated from parity-check matrices which have cyclically related rows.

Cyclic codes encompass BCH codes and the Reed-Solomon non-binary codes. The Reed-Solomon (RS) code is made up of n symbols where each symbol is m bits long. m can be any length depending on the application, for example if $m = 8$ bits then each symbol would represent a byte. Thus RS codes operate on a multiple bit symbol principle and not a bit principle as other cyclic codes do. An important property of the RS codes is their burst error correction property. Their error correcting power is:

$$t = \frac{n-k}{2} \tag{10.15}$$

where n and k here relate to encoded symbols, not bits. For example the (31,15) Reed-Solomon code has 31 5-bit encoded symbols which represent 15 symbols of input information or 75 input information bits. This can correct 8 independent bit errors or 4 bursts of length equal to or less than the 5-bit symbol duration.

10.8.1 Polynomial codeword generation

Systematic cyclic redundancy checks (CRC) are in widespread use for performing error detection, but not correction, on bit-serial channels. The operation of CRCs can be considered as an algebraic system in which 0 and 1 are the only values and where addition and subtraction involve no carry operations (i.e. arithmetic is modulo-2). (Addition and subtraction are both, therefore, the same as the logical exclusive-or operation.) If a message of length k bits: $m_{k-1}, \cdots, m_1, m_0$ (from most to least significant bit) is to be transmitted over a channel then, for coding purposes, it may be considered to represent a polynomial of order $k-1$:

$$M(x) = m_{k-1}x^{k-1} + \cdots + m_1 x + m_0 \tag{10.16}$$

The message $M(x)$ is modified by the generator polynomial $P(x)$ to form the channel coded version of $M(x)$. This is accomplished by multiplying, or bit shifting, $M(x)$ by the order of $P(x)$. $P(x)$ is then divided into the bit shifted or extended version of $M(x)$ and the remainder is then appended to $M(x)$ replacing the zeros which were previously added by the bit shifting operation. Note that the quotient is discarded.

EXAMPLE 10.5

Generate a polynomial codeword from the data sequence 1001 and the generator polynomial $1 + x + x^3$.

```
          1111
1101 ⌈1001000
    -1101
     100000
    -1101
      10100
     -1101
       1110
      -1101
        011
```

Figure 10.14 *Long division calculation of remainder for appending to data to obtain codeword.*

For $M(x) = 1001$ (i.e. $1 + x^3$) and the polynomial $P(x) = 1101$ (i.e. $1 + x + x^3$) the bit shifted sequence $kM(x)$ is 1001000. This is now divided by $P(x)$, Figure 10.14, to obtain the remainder 011. The appended zeros in $kM(x)$ are then replaced by the remainder to realise the transmitted codeword as 1001011.

The required division process can be conveniently implemented in hardware. An encoding circuit, which is equivalent to the long division operation of Figure 10.14, is shown in Figure 10.15. The encoding circuit works as follows. The information bits are transmitted as part of the cyclic code but are also fed back via the feedback loop. During this step, the feedback switch is kept closed. As we clock the shift register and input the message codeword the states $A, \cdots, E$ in the shift register follow this bit pattern:

Input A = 1001
B = 1010
previous B, i.e. C = 0101
D = 1111
previous D, i.e. E = 0111
previous E, i.e. F = 0011

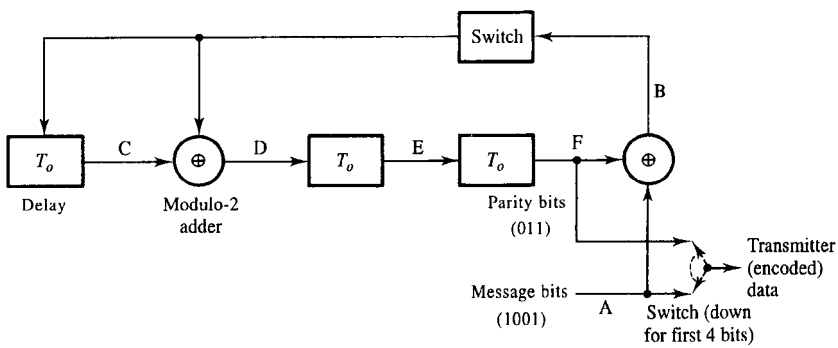


Figure 10.15 *Encoder for a (7,4) cyclic code generated by* $P(x) = 1 + x + x^3$.

When the final bit of information is transmitted, the gate in the feedback loop is opened and the information remaining at the shift register locations E, D, B is then appended to the data for transmission using the output switch. These remaining parity check bits, in this example, would be 110, resulting in the (7,4) Hamming code.

On reception the received data is again divided by $P(x)$ and if the remainder is zero then no errors have been introduced. Further examination of a non-zero remainder in a syndrome table can allow the bit error positions to be determined and the errors corrected by adding in the error pattern in the decoder [Blahut 1983], as previously shown in Example 10.3. The syndrome table can be found either by mathematical manipulation or by successive division for each error location.

The hardware decoding scheme, see Figure 10.16, is basically an inverse version of the encoding scheme of Figure 10.15. If the decoder receives an error it is capable of identifying the position of the error digit via the remainder and the syndrome table.

Thus, the polynomial coded message consists of the original k message bits, followed by $n-k$ additional bits. The generator polynomial is carefully chosen so that almost all errors will be detected. Using a generator of degree k allows the detection of all burst errors affecting up to k consecutive bits. The generator chosen by ITU-T for the V.41 standard, which is the same as that used extensively on wide area networks, is:

$$M(x) = x^{16} + x^{12} + x^5 + 1 \tag{10.17}$$

The generator chosen by IEEE, used extensively on local area and FDDI networks (see Chapter 18), is:

$$M(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \tag{10.18}$$

CRC six-bit codewords are transmitted within the plesiochronous multiplex, Chapter 19, to improve the robustness of frame alignment words. The error correcting power of the CRC code is low and it is mainly used when ARQ retransmission is deployed, rather than

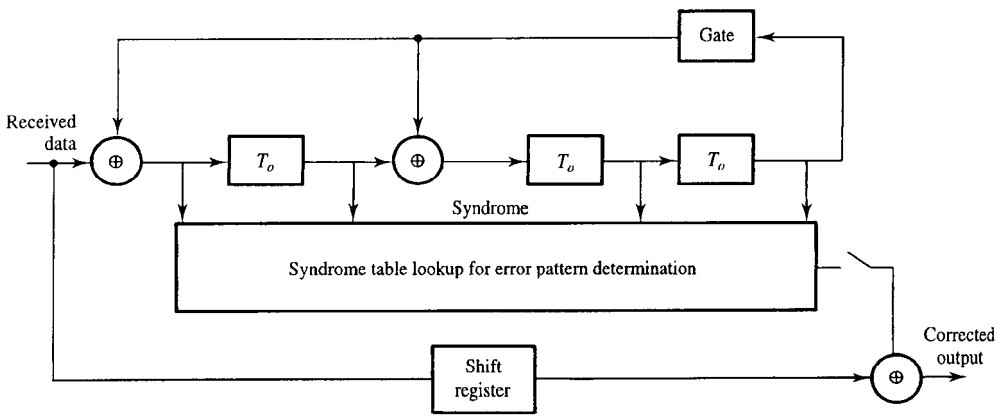


Figure 10.16 *Syndrome calculator and decoder for a (7,4) cyclic code.*

for error correction itself.

10.8.2 Interleaving

The largest application area of block codes is in compact disc players which employ a powerful concatenated and cross-interleaved Reed-Solomon coding scheme to handle random and burst errors. Partitioning data into blocks and then splitting the blocks and interleaving them means that a burst transmission error usually degrades only part of each original block. Thus, using FECC, it is possible to correct for a long error burst, which might have destroyed all the information in the original block, at the expense of the delay required for the interleaver encoder/decoder function. In other applications bit-by-bit interleaving is employed to spread burst errors across a data block prior to decoding. Figure 10.17 shows how an input data stream is read, column by column, into a temporary array and then read out, row by row, to achieve the bit interleaving operation. Now, for example, a burst of three errors in the consecutive transmitted data bits, I_1, I_5, I_9, is converted into isolated single errors in the de-interleaved data.

10.9 Encoding of convolutional codes

Convolutional codes are generated by passing a data sequence through a shift register which has two or more sets of register taps (effectively representing two or more different filters) each set terminating in a modulo-2 adder. The code output is then produced by sampling the output of all the modulo-2 adders once per shift register clock period. The coder output is obtained by the convolution of the input sequence with the impulse response of the coder, hence the name convolutional code. Convolution applies even though there are exclusive or and switch operations rather than multiplies. Figure 10.18

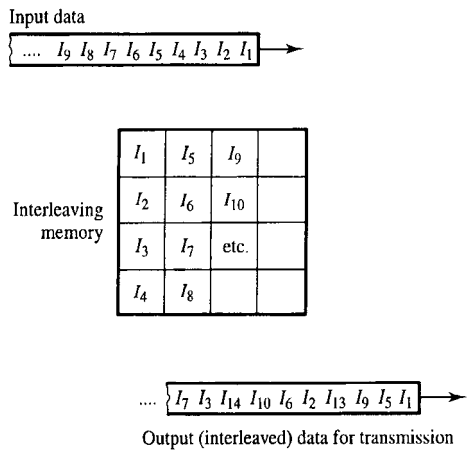


Figure 10.17 *Data block interleaving to overcome burst errors.*

illustrates this with a simple example where the output encoder operation can be defined by two generator polynomials. The first and second encoded outputs, $P_1(x)$, $P_2(x)$, can be defined by $P_1(x) = 1 + x^2$ and $P_2(x) = 1 + x$, as in Example 10.5. This shows a 3-stage encoder giving a constraint length $n = 3$. The error correcting power is related to the constraint length, increasing with longer lengths of shift register.

Assume the data sequence 1101 is input to the three-stage shift register which is initiated or flushed with zeros prior to clocking the sequence through. This example depicts a rate ½ coder (R = ½) since there are two output digits for every input information digit. The coder is non-systematic since the data digits are not explicitly present in the transmitted data stream. The first output following a given input is obtained with the switch in its first position and the second output is obtained with the switch in its second position, etc.

This encoder may be regarded as a finite state machine. The first stage of the shift register holds the next input sample and its contents determine the transition to the next state. The final two stages of the shift register hold past inputs and may be regarded as determining the 'memory' of the machine. In this example there are 2 'memory stages'

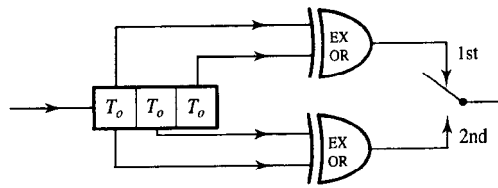


Figure 10.18 *A simple example of a rate ½ convolutional encoder.*

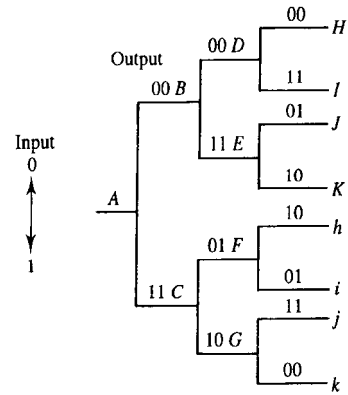


Figure 10.19 *Tree diagram representation of the coder in Figure 10.18.*

and hence four possible states. In general an n-stage register would have $2^{(n-1)}$ states. For the $n = 3$ stage coder the four states correspond to the data bit pairs 00, 10, 01, 11 (from prior input data). The convolutional encoder operation may be represented by a tree diagram.

10.9.1 Tree diagram representation

Figure 10.19 depicts the tree diagram corresponding to the example of Figure 10.18. Assume that the encoder is 'flushed' with zeros prior to the first input of data and that it is in an initial state which is labelled A. Conventionally the tree diagram is drawn so that inputting a zero results in exiting the present state by the upper path, while inputting a one causes it to exit by the lower path. Assuming a zero is input, the machine will move to state B and output 00. Outputs are shown on the corresponding branches of the diagram. Alternatively if the machine is in state A and a 1 is input then it proceeds to state C via the lower branch and 11 is output. Figure 10.19 depicts the first three stages of the tree diagram, after which there appear to be eight possible states. This is at variance with the previous statement, that the state machine in this example has only four states.

There are, in fact, only four distinct states here, (00, 10, 01, 11), but each state appears twice. Thus H is equivalent to h, for example. This duplication of states results from identical prior data bits being stored in the shift register of the encoder, The path B, D to H represents the input of two zeros, as does the path C, F to h, resulting in identical data being stored in the shift register. After the fourth stage each state would appear four times, etc. Two states are identical if, on receiving the same input, they respond with the same output. Following through input data in Figure 10.19 by the path which this data generates allows the states (00, 10, 01, 11) to be identified and the figure annotated accordingly to identify the redundancy. The apparent exponential growth rate in the number of states can be contained by identifying the identical states and overlaying them. This leads to a trellis diagram.

10.9.2 Trellis diagram

Figure 10.20 shows the trellis diagram corresponding to the tree diagram of Figure 10.19. The horizontal axis represents time while the states are arranged vertically. On the arrival of each new bit the tree diagram is extended to the right. Here five stages are shown with the folding of corresponding tree diagram states being evident at the fourth and fifth stages (states *HIJK*, *LMNO*) by the presence of two entry paths to each state. There are still too many states here and inspection will show that H and L, for example, are equivalent. Thus four unique states may be identified. These are labelled a, b, c and d on this diagram again corresponding to the binary data 00, 10, 01, and 11 being stored in the final two stages of the shift register of Figure 10.18.

The performance of the convolutional coder is basically dependent on the Hamming distance between the valid paths through the trellis, corresponding to all the possible, valid, data bit patterns which can occur. The final step in compacting the graphical

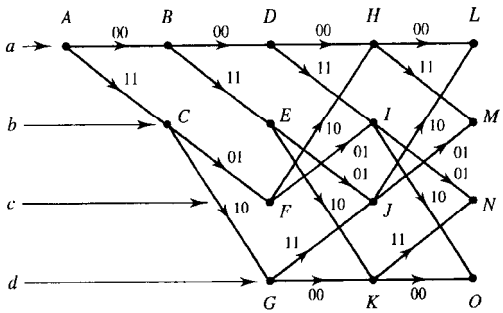


Figure 10.20 *Trellis diagram representation of the coder in Figure 10.18.*

representation of the convolutional encoder is to reduce this trellis diagram to a state transition diagram.

10.9.3 State transition diagram

Here the input to the encoder is shown on the appropriate branch and the corresponding outputs are shown in brackets beside the input, Figure 10.21. For example if the encoder is in state a (the starting state) and a zero is input then the transition is along the self-loop returning to state a. The corresponding output is 00, as shown inside the brackets (and along the top line of Figure 10.20). If, on the other hand, a 1 is input while in state a, then 11 is output and the state transition is along the branch from a to b, etc.

10.10 Viterbi decoding of convolutional codes

There are three main types of decoder. These are based on sequential, threshold (majority logic) and Viterbi decoding techniques. The Viterbi technique is by far the most popular.

Data encoded by modern convolutional coders are usually divided into message blocks for decoding, but unlike the block coded messages, where $n < 255$, the convolutional coded message typically ranges from 500 to >10,000 bits, depending on the application. This makes decoding of convolutional codes potentially onerous. (The decoder memory requirements grow with message length.) The coder operation is

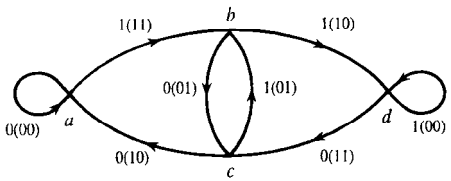


Figure 10.21 *State transition diagram representation of the coder in Figure 10.18.*

illustrated here by the processing of short fixed length blocks, which are fed through the encoder after it has been 'flushed' with zeros to bring it into state a. The block of data is followed with trailing zeros to return the encoder back to state a at the end of the coding cycle. This simplifies decoding and 'flushes' the encoder ready for the next block. The zeros do not, however, carry any information and the efficiency, or rate, of the code is consequently reduced.

Secondly, the Viterbi decoding algorithm is used at each stage of progression through the decoding trellis, retaining only the most likely path to a given node and rejecting all other possible paths on the grounds that their Hamming distance is larger and that they are, thus, less likely events than that represented by the shorter distance path. This leads to a linear increase in storage requirement with block length as opposed to an exponential increase.

The Viterbi decoding algorithm implements a nearest neighbour decoding strategy. It picks the path through the decoding trellis, which assumes the minimum number of errors (the probability of t errors being much greater than that of $t+1$ errors, etc.). Conceptually a decoding trellis, similar to the corresponding encoding trellis, is used for decoding.

EXAMPLE 10.6 – Decoding trellis construction
Assume a received data sequence 1010001010 for the encoder operation of Figure 10.18. Identify the errors and derive the corresponding transmitted data sequence.

The ten transmitted binary digits correspond to five information digits. We assume that the first three of these digits are unknown data and the last two are flushing zeros.

Decoding begins by building a decoding trellis corresponding to the encoding trellis starting at state A as shown in Figure 10.22. We assume that the first input to the encoder is a zero. Reference to the encoding trellis indicates that on entering a zero with the encoder completely flushed 00 would be output, but 10 has been received. This means that the received sequence is a Hamming distance of 1 from the possible transmitted sequence (with an error in the first output bit). This distance metric is noted along the upper branch from A to B.

The possibility that the input data may have been a 1 is now investigated. Again, reference to the encoding trellis indicates that if a 1 is input to the encoder in state A, the encoder will output 11 and follow the lower path to state C. In fact, 10 was received, so again, the actual received sequence is a Hamming distance of 1 from this possible transmitted sequence. (Here the error

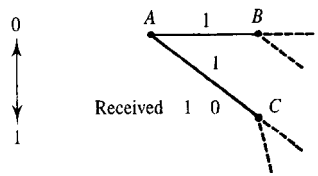


Figure 10.22 *First stage in constructing the decoding trellis for a received sequence from the encoder of Figure 10.18 after receiving two encoded data bits.*

would be in the second bit.) The distance metric is thus noted as 1 along the branch from A to C.

Now we return to state B and assume that the input was zero followed by another zero. If this were the case, the encoder would have gone from state B to state D and output 00 (Figure 10.23). The third and fourth digits received, however, were 10 and again there is a Hamming distance of 1 between the received sequence and this possible transmitted sequence. This distance metric is noted on the branch B to D and a similar operation is performed on branch B to E where the distance is also 1. Next we consider inputting zero while in state C. This would create 01 whilst, in fact, 10 was received. The Hamming distance here is 2. This metric is noted on branch C to F and attention is turned finally to branch C to G. Starting in state C and inputting a 1 would have output 10 and, in fact, 10 was received, so at last there is a received pair of digits which does not imply any errors. The *cumulative* distances along the various paths, i.e. the path metrics, are now entered in square brackets above the final states in Figure 10.23.

Figure 10.24 illustrates a further problem. Decoding is now at stage 3 in the decoding trellis and the possibility of being in state J is being considered. From this stage on, each of the four states in this example has two entry, and two exit, paths. Conventionally on reaching a state like J with two input paths, the cumulative Hamming distance or path metric of the upper route ($ABEJ$) is shown first and the cumulative Hamming distance for the lower path ($ACGJ$) is shown second in the square brackets adjacent to state J. The real power of the Viterbi algorithm lies in its rejection of one of those two paths, retaining one path which is referred to as the 'survivor'. If the two paths have different Hamming distances then, since this is a nearest neighbour (maximum likelihood) decoding strategy, the path with the larger Hamming distance is rejected and the path with the smaller Hamming distance, or path metric, is carried forward as the survivor.

In the case illustrated in Figure 10.24 the distances are identical and the decoder must flag an uncorrectable error sequence, if using incomplete decoding. If using complete decoding a random choice is made between the two paths (bearing in mind there is a 50% probability of being wrong). Fortunately, this situation is rare in practice. (The probability of error has been deliberately increased here for illustrative purposes.)

There are two paths to state H with path metrics [2,4], Figure 10.25. The path of distance 4 may thus be rejected as being less likely than the path of distance 2, etc. To state I there are also two paths of equal length [4,4]. In the final stage state P has been labelled as being the finishing point of the decoding process since in this example only three unknown data digits are being transmitted followed by 00 to flush the encoder and bring the decoder back to state a. Only the

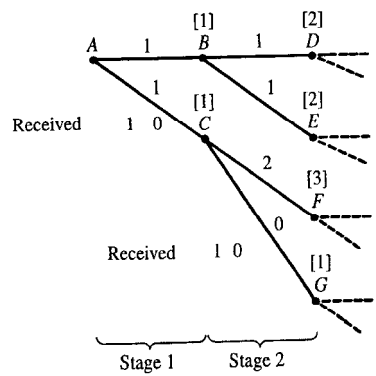


Figure 10.23 *Second stage of trellis of Figure 10.18 after decoding four data bits.*

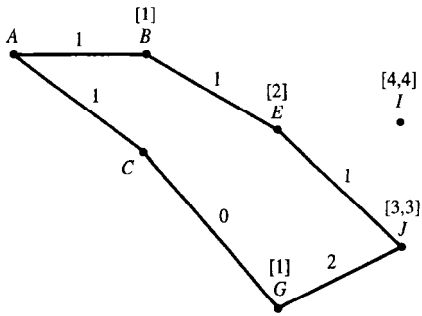


Figure 10.24 *Illustration of a sequence containing a detectable but uncorrectable error pattern.*

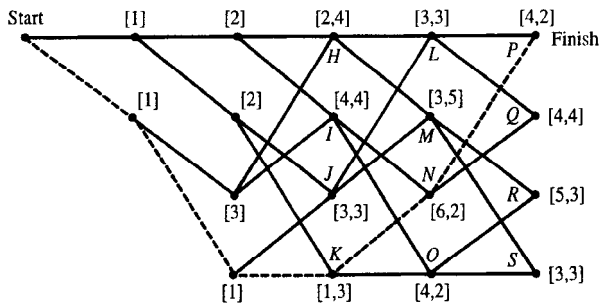


Figure 10.25 *Complete decoding trellis for Example 10.6 with the dashed preferred path and resulting decoded sequence 11100.*

more likely of the two paths to state P is retained. This is the lower path with a Hamming distance of 2. Note that although state O also has distances of 2 we cannot progress from O to P to complete the decoding operation. Figure 10.25 shows the complete decoding trellis. Tracing back along the most likely (dashed) path provides the corresponding decoded sequence as (11100) and the implied correct received data as 1110001110. Although state Q also has a cumulative distance of 2 this cannot terminate the correct path as this decoder must be flushed with zeros ready for the next block of data.

10.10.1 Decoding window

In a practical convolutional decoder the block length would usually be very much larger than in the simple example above. The data from a complete frame of a video coded image, Chapter 16, may be sent as a single message block, for example. In such a case the overhead requirement, for accommodating the flushing zeros, becomes negligible. There is a constraint on the length of data which can be retained in the decoder memory, however, when performing the Viterbi decoding operation. The practical limitation is

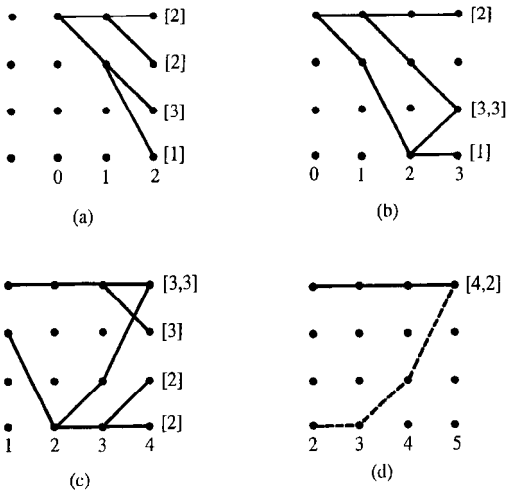


Figure 10.26 *Viterbi decoding within a finite length decoding window.*

known as the decoding window. In a practical decoder the new distance metrics are added to the previous path metrics to obtain updated path metrics. Details of the paths, which correspond to these various distances, are carried forward in the decoding process. In Figure 10.25 the decoding was performed over a window length of 5 input data bits. (With long block lengths the same procedure would be followed.) Figure 10.26 shows an example of decoding with a window length of only 4. This demonstrates how the decoding of Figures 10.22 to 10.25 moves through the window (sequence (a) to (d)) with only the most likely paths being retained.

When the window length is restricted to 4 then, on the arrival of the final received bit pair, it is no longer possible to continue to examine the start bit as this would have propagated out of the restricted length of the decoding window (through which the trellis is viewed). The window length should be long enough to cover all bursts of decoding errors but, since longer lengths involve more computation, a compromise must be made and the decoder's performance verified by computer simulation. (For practical coders a good rule of thumb is for the decoding window to be set at five times the constraint length of the encoder.)

10.10.2 Sequential decoding

Viterbi's algorithm requires all the surviving sequences to be followed throughout the decoding process and leads to excessive memory requirements for long constraint lengths. Complexity can be reduced by sequential decoding, which directly constructs the sequence of states by performing a distance measure at each step. Sequential decoding proceeds forwards until complete decoding is accomplished or the cumulative distance exceeds a preset threshold. When this occurs the algorithm backtracks and

selects an alternative path until a satisfactory overall distance is maintained. This works well at low error rates but, when the error rate is high, the number of backward steps can become very large.

The Viterbi decoder has three main components, Figure 10.27. The first is the branch metric value (BMV) calculation unit which finds the Hamming distances for each new branch in the trellis. The add-compare-select unit is the second which calculates and updates the overall path history or path metric values (PMVs) for each path arriving at each node in the trellis. In the third component, the output determination unit, only the surviving paths are retained (i.e. selected) as these have the smallest distances, the paths with higher distances being discarded. When working through the complete trellis, only the most likely overall path is finally retained as the ultimate survivor. Trellis decoding is not restricted to convolutional codes as it is also used for soft decision decoding of block codes [Honary and Markarian]. (A similar decoding trellis is also used later for trellis coded modulation, section 11.4.8 [Biglieri *et al.*, Ungerboeck].)

10.11 Practical coders

Examples of practical block codes are the BCH (127, 64) which has an error correction capability of $t = 10$ bit; the Reed-Solomon (16,8) or (64,32) codes achieve $t = 4$ symbol capability while the shorter block length of the Golay (23,12) code has a $t = 3$ bit capability. The error rate performance of these, and some other, codes is compared in Figure 10.28 [Farrell], all for DPSK modulation, in which the horizontal axis is energy per input *information* data bit divided by one sided noise power spectral density E_b/N_0. Figure 10.28 echoes Figure 10.1, showing clearly the point at which the FECC systems outperform uncoded differential phase shift keyed (DPSK) transmission (see Chapter 11).

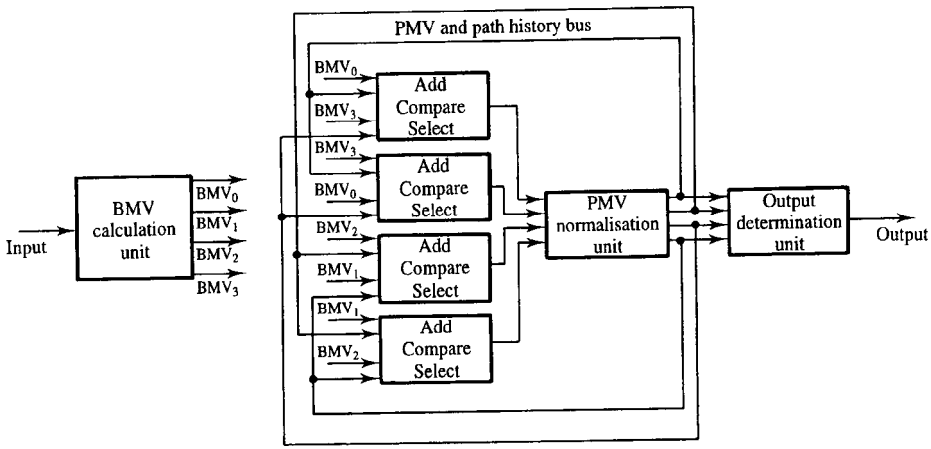


Figure 10.27 *Viterbi decoder circuit for decoding the trellis of Figure 10.20.*

Generally for a $t = 1$ BCH block code, which has a larger block length than the simple Hamming (7,4) example, i.e. a (31,26) or a (63,57) code, then the coding gain over the uncoded system, at a P_b of 10^{-5}, is >2 dB. For the longer $t = 3$ bit BCH (127,106) code the coding gain approaches 4 dB. Linear block codes are usually restricted to $n < 255$ by the decoder complexity. In Figure 10.28 the $t = 4$ symbol Reed-Solomon code is inferior to $t = 3$ bit Golay code at high P_b, but this performance is reversed at lower P_b. The (23,12) Golay code, which corrects triple errors, is a perfect code as equation (10.4) becomes:

$$2^{12} \leq \frac{2^{23}}{{}^{23}C_0 + {}^{23}C_1 + {}^{23}C_2 + {}^{23}C_3} \tag{10.19(a)}$$

i.e.:

$$2^{12} \leq \frac{2^{23}}{1 + 23 + 253 + 1771} \tag{10.19(b)}$$

and hence is satisfied as the equality:

$$2^{12} = \frac{2^{23}}{2048} = \frac{2^{23}}{2^{11}} \tag{10.20}$$

In general Reed-Solomon codes are attractive for bursty error channels where extremely low P_b values (e.g. 10^{-10}) are required. Convolutional codes are favoured for Gaussian noise channels where more moderate P_b values (e.g. 10^{-6}) are required.

Practical convolution codes often have constraint lengths of $n = 7$ with a rate ½ coder which employs 7 delay stages. This requires a decoding window in the trellis of 35 to

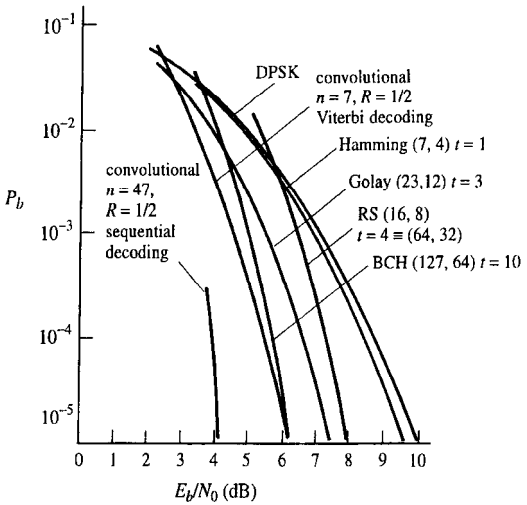


Figure 10.28 *Performance of rate ½ codes for DPSK signals in AWGN with hard decision decoding (source: Farrell, 1989, reproduced with permission of the IEE).*

achieve the theoretical coding gain. With such a window, the encoder message block size can be set appropriate to the specific application. Such a coder then has $D_{min} = 10$ and its performance is equivalent to the BCH (127,64) block code. The convolutional coder is preferred, however, because the encoder is simpler and the decoder can more easily incorporate soft decision techniques (in which a confidence level is retained for the received data). Such soft decisions, which implement maximum likelihood decoding, give approximately 2 dB improvement in coding gain compared to hard decisions. (The Voyager space probes employed soft decision decoders adding 5 to 8 dB coding gain to their link budget, see Chapter 12.)

In general reducing the efficiency, or rate, R, of the coder increases the Hamming distance D_{min} between permissible paths and improves the error correcting power (i.e. the coding gain in dB compared to the uncoded DPSK system). If the performance is normalised by the rate R then there may be little benefit in going to rates of less than ½ as the improvement in error rate is exactly balanced by the reduction in data rate. Whilst additional coding gain improvement is typically 1 dB for an $n = 9$ constraint length, compared to $n = 7$, the improvements beyond this are very small and the decoder complexity is excessively high.

The largest application area for convolutional coders is in rate ¾ compact disc codes which employ a concatenated and cross-interleaved coding scheme. Sequential decoding is a powerful search path technique for use in the decoding trellis, especially with longer constraint length, Figure 10.28. Sophisticated VLSI Viterbi decoders are now available for speeds of 250 kbit/s to 25 Mbit/s with constraint lengths of $n = 7$ at (1996) costs of £10 and above, per FECC decoder. Table 10.2, lists some Qualcom (Q) and Stanford Telecom (ST) convolutional coder chipsets.

Table 10.2 *Examples of commercially available convolutional coder chipsets.*

Data rate (Mbit/s)	25	12	2.5	1	¼
Coder rate, R	$\frac{7}{8}, \frac{3}{4}, \frac{1}{2}, \frac{1}{3}$	$\frac{3}{4}, \frac{2}{3}, \frac{1}{2}$	$\frac{7}{8}, \frac{3}{4}, \frac{1}{2}, \frac{1}{3}$	$\frac{1}{2}, \frac{1}{3}$	$\frac{7}{8}, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}$
Constraint length	7	7	7	6	6–7
Soft decision capability (bits)	3	3	3	–	3–4
Supplier	Q	Q/ST	Q	ST	Q/ST

10.12 Summary

Error rate control is necessary for many systems to ensure that the probability of bit error is acceptably low. Bit error rates may be reduced by increasing transmitted power, applying various forms of diversity, using echo back and retransmission, employing ARQ, or incorporating FECC. In this chapter the focus has been on channel coding for error detection and correction which are prerequisites, respectively, for ARQ and FECC error control systems.

Channel codes may be systematic or unsystematic. Systematic codes use codewords which contain the information digits, from which the codeword is derived, explicitly. The rate, R, of a code is the ratio of information bits transmitted to total bits transmitted. The Hamming distance between a pair of binary codewords, which is given by their modulo-2 sum, is a measure of how easily one codeword can be transformed into the other. The weight of a binary code word is equal to the number of binary ones which it contains.

Block codes divide the precoded data into k bit lengths and add $(n - k)$ parity check bits to create a post-coded block, with length n bits. An (n, k) block code, therefore, has an efficiency, or rate, of $R = k/n$. Single parity check codes are block codes which append a single digital one, or zero, to each codeword in order to ensure that all codewords have either an even (for even parity) or an odd (for odd parity) weight. This allows single error detection. Data can also be arranged in two dimensional, rectangular, arrays allowing parity check digits to be added to the ends of rows, and the bottoms of columns. This achieves single error correction.

Group codes (also called linear block codes) are block codes which contain the all-zeros codeword and have the closed set, or linear group, property, i.e. the modulo-2 sum of any pair of valid codewords in the set is another valid codeword. The error correcting power, t, of a linear group code is given by $int((D_{min} - 1)/2)$ where D_{min} is the minimum Hamming distance between any pair of codewords. The error detecting power, e, of a linear group code is given by $D_{min} - t - 1$. Group codes are the most important block codes due to the ease with which their performance can be predicted. (D_{min} is given by the weight of the minimum weight codeword, excluding the all-zeros codeword, in the group). Block codes can be generated using a generator matrix. Complete codewords are generated by the product of the precoded data vector with the generator matrix.

Block codes are most easily decoded using a nearest neighbour strategy. This is equivalent to maximum likelihood decoding providing that $P(t \text{ errors}) \gg P(t + 1 \text{ errors})$ which is always the case in practice. Nearest neighbour decoding can be implemented using a nearest neighbour table or a syndrome table. (The syndrome vector for a given received codeword is the product of the parity check matrix and the received codeword vector. It is also the product of the parity check matrix and the error pattern vector of the received codeword allowing error patterns to be determined from a table of syndromes.) Syndrome decoding is advantageous when block size is large since the syndrome table for all (single) error patterns is much smaller than the nearest neighbour decoding table.

Cyclic codes are linear group codes in which the codeword set consists of only, and all, cyclical shifts of any one member of the codeword set. They are particularly easily generated using shift registers with appropriate feedback connections. Their syndromes are also easily calculated using shift register hardware. CRC codes are systematic cyclic codes which are potentially capable of error correction but are often used for error detection only. A polynomial representation of a precoded block of information bits is multiplied (i.e. bit shifted) by the order of a generator polynomial and then divided by the generator polynomial. The remainder of the division process is appended to the block of information bits to form the complete codeword. Division of received codeword polynomials by the generator polynomial leaves zero remainder in the absence of errors.

Convolution codes are unsystematic and operate on long data blocks. The encoding operation can be described (in increasing order of economy) using tree, trellis or state diagrams. The Viterbi algorithm, which implements a nearest neighbour decoding strategy, is usually used to decode convolution codes. The error correction capability of convolutional coders is not inherently in excess of that for block coders but their decoder and, particularly, encoder designs are simpler.

FECC combinations of block and convolutional codes are widely applied to accommodate random and burst errors which may both arise in communication channels.

10.13 Problems

10.1. Assume a binary channel with independent errors and $P_e = 0.05$. Assume k digit symbols from the source alphabet are encoded using an (n, k) block code which can correct all patterns of three or fewer errors. Assume $n = 20$. (a) What is the average number of errors in a block? [1] (b) Assuming binary transmission at 20,000 binary digits per second, derive the symbol error rate at the decoder output. [15.8 symbol/s]

10.2. A binary signal is transmitted through a channel which adds zero mean, white, Gaussian noise. The probability of bit error is 0.001. What is the probability of error in a block of 4 data bits? If the bandwidth is expanded to accommodate a (7,4) block code, what would be the probability of an error in a block of 4 data bits? [0.0040, 0.0019]

10.3. Assume a systematic (n, k) block code where $n = 4$, $k = 2$ and the four codewords are 0000, 0101, 1011, 1110. (a) Construct a maximum likelihood decoding table for this code. (b) How many errors will the code correct? Are there any errors which are detectable but not correctable? [8 corrected and 8 detected but not corrected] (c) Assume this code is used on a channel with a $P_e = 0.01$. What is the probability of having a detectable error sequence? What is the probability of having an undetectable error sequence? [0.0388, 0.0006]

10.4. For a (6,3) systematic linear block code, the three parity check digits are:

$$P_1 = 1 \times I_1 \oplus 1 \times I_2 \oplus 1 \times I_3$$

$$P_2 = 1 \times I_1 \oplus 1 \times I_2 \oplus 0 \times I_3$$

$$P_3 = 0 \times I_1 \oplus 1 \times I_2 \oplus 1 \times I_3$$

(a) Construct the generator matrix $\mathbf{G}$ for this code. (b) Construct all the possible codewords generated by this matrix. (c) Determine the error-correcting capabilities for this code. [single] (d) Prepare a suitable decoding table. (e) Decode the received words 101100, 000110 and 101010. [111100, 100110, 101011]

10.5. Given a code with the parity check matrix:

$$\mathbf{H} = \begin{bmatrix} 1110\ 100 \\ 1101\ 010 \\ 1011\ 001 \end{bmatrix}$$

(a) Write down the generator matrix showing clearly how you derive it from $\mathbf{H}$. (b) Derive the complete weight structure for the above code and find its minimum Hamming distance. How many errors can this code correct? How many errors can this code detect? Can it be used in correction and detection modes simultaneously? [3, 1, 2, No] (c) Write down the syndrome table for this code showing how the table may be derived by consideration of the all-zeros codeword. Also comment

on the absence of an all-zeros column from the **H** matrix. (d) Decode the received sequence 1001110, indicate the most likely error pattern associated with this sequence and give the correct codeword. Explain the statement 'most likely error pattern'. [0000010, 1001100]

10.6. When generating a (7,4) cyclic block code using the polynomial $1 + x^2 + x^3$: (a) What would the generated codewords be for the data sequences 1000 and 1010? [1000101, 1010011] (b) Check that these codewords would produce a zero syndrome if received without error. (c) Draw a circuit to generate this code and show how it generates the parity bits for the two data sequences in part (a). (d) If the codeword 1000101 is corrupted to 1001101, i.e. an error occurs in the fourth bit, what is the syndrome at the receiver? Check this is the same as for the codeword 1010011 being corrupted to 1011011. [011]

10.7. Given the ½ rate convolutional encoder defined by $P_1(x) = 1 + x + x^2$ and $P_2(x) = 1 + x^2$, and assuming data is fed into the shift register one bit at a time, draw the encoder: (a) tree diagram; (b) trellis diagram; (c) state transition diagram. (d) State what the rate of the encoder is. (e) Use the Viterbi decoding algorithm to decode the received block of data, 10001000.
Note: there may be errors in this received vector. Assume that the encoder starts in state a of the decoding trellis in Figure 10.20 and, after the unknown data digits have been input, the encoder is driven back to state a with two 'flushing' zeros. [0000]