Corso di Information Theory and Coding

Prof. Francesco A. N. Palmieri

Dipartimento di Ingegneria, Università della Campania Luigi Vanvitelli

Corso di Laurea Magistrale in Ingegneria Informatica

AA 2020-21

CODICI A BLOCCO

Estratto dal libro:

S. Benedetto, E Biglieri, *Priciples of Digital Transmission with* Wireless Applications, Kluwer Academic Press, 1999

10

Improving the transmission reliability: Block codes

Designers of primitive digital communication systems sought to obtain low bit error probabilities by transmitting at high power or by using larger bandwidth than strictly necessary. This approach is adequate if the required error probability is not too low and/or the data rate is not too high: it buys performance with the most precious of resources: spectral bandwidth and power.

The lesson taught by Shannon (see Section 3.3) was that high performance is indeed obtainable by calling a third resource into play, the system complexity. The concurrence of two basic facts, i.e., the sky-rocketing requirements of transmission speed and the affordability, thanks to the modern electronic technology, of extremely sophisticated receivers has made the Shannon dream an every-day reality, so that highly complex co-decoding techniques are now widely used in digital transmission systems to protect the information.

Techniques to control the error probability are based on the addition of redundancy to the information sequence. Traditionally, codes aimed at improving the transmission reliability are called *error correcting codes*. This concept is bound to a particular operating mode of the demodulator and decoder, in which the received signal sequence is *hard-detected* by the demodulator, before being transferred to the decoder. As a consequence, the binary sequence entering the decoder contains errors that the decoder may or may not be able to correct. This mode of operation, however, entails some degree of suboptimality, and is replaced, whenever feasible, by *soft-decoding*, in which the demodulator derives the sufficient statistics in analog or quantized form, and supplies it to the decoder, which, in turn, performs the final task of estimating the transmitted information sequence. When this is the operation mode, talking of "error correcting" codes

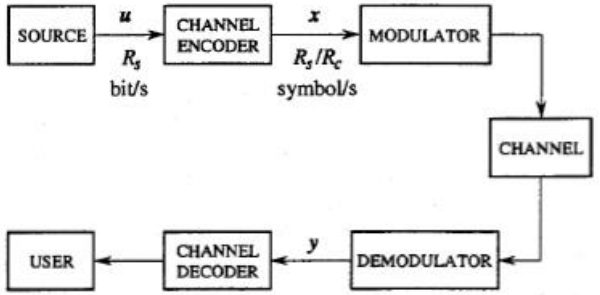

Figure 10.1: *Block diagram of a transmission system employing channel coding.*

does not make sense, since no true correction of error takes place, but, rather, the pair encoder-decoder prevents errors from occurring. In this situation, it would better to talk of *error control* codes (see Blahut, 1983). Most of the algorithms for decoding *tree codes* make use of the soft information in a straightforward manner. The use of soft-decision in block codes is somewhat more involved and generally requires significant changes in the decoding algorithms.

For the reasons previously explained, we will generally speak of codes that improve the transmission reliability, or of *channel codes*, in the sense that these codes aim at protecting the information from impairments occurring during its transmission over the channel.

In this chapter, we will first propose a taxonomy of the codes employed to protect the transmitted information, and then define and analyze *linear* block codes. In the next chapter, we will consider convolutional and concatenated codes.

10.1. A taxonomy of channel codes

Consider the simple block diagram of Fig. 10.1. Using the terminology of Blahut (1983), we distinguish a source producing a binary sequence, the *data stream*: it is the binary sequence emitted directly by the source, or by the source encoder. We assume that it is formed by independent identically distributed (iid) binary random variables (RVs). The data stream enters the channel encoder which maps it into a *code stream*. For constructing the code, additional structure may be defined on the data stream by segmenting it into pieces called *data words*. Likewise, the code stream is segmented into pieces called *code words*. For an (n, k) block code, the data words consist of k bits and the code words of n bits. A

channel code C is the set of 2^k n-tuples of bits, the code words $\mathbf{x}$. An *encoder* E is the set of the 2^k pairs $(\mathbf{u}, \mathbf{x})$, where $\mathbf{u}$ is a data word, i.e., a k-tuple of bits, and $\mathbf{x}$ the corresponding code word. These definitions should clarify the fundamental difference between the notion of a code and the notion of an encoder. The code is a collection of code words and is independent of the way they are obtained. The term encoder refers to the one-to-one correspondence between data words and code words, and also applies to the device that implements this assignment. With respect to how the encoder assigns code words to data words, we say that the (n, k) code is a *block* code when the encoder is memoryless, i.e., when to the same k bits in the data word there correspond the same n code word bits. The block code is an (n, k) code, and the ratio $R_c \triangleq k/n$ is the *rate* of the code. Each data word (block) is encoded independently without interaction with earlier or later data words. When the correspondence between data words and code words has memory, i.e., the n bits of the code word do not depend only on the k bits of the data word, but also on some previous data words, we say that the code is a *tree* code. In this case, it is often convenient to think of infinitely long data streams and code streams, or sequences, which start at time zero and continue indefinitely in the future. A tree code breaks the data stream into segments called *data frames*, each consisting of k_0 data bits, k_0 normally a small integer. The encoder is a finite-state machine that retains some memory of earlier data frames; in the simplest case, it simply stores the m most recent data frames unchanged. A single code frame consists of n_0 bits that are computed from the mk_0 data bits of the m data frames stored in the encoder memory, and the k_0 bits of the incoming data frame; these n bits are shifted out to the channel as the new k_0 data bits enter the encoder. The ratio $R_c \triangleq k_0/n_0$ is still called the code rate. Tree codes with a special memory and linearity structure, to be defined in the next chapter, are called *convolutional* codes.

With respect to the properties of the set of code words, we distinguish between *linear* and *nonlinear* codes. For a linear code, the set of code words (or code streams, for tree codes) is closed under component-wise modulo-2 addition, an operation denoted simply by "+" in this chapter.[1] This property has several important implications that will be made clear in the next sections. According to how the system makes use of the code capabilities, we distinguish between *error detecting* and *error correcting* codes. This does not represent a distinction between the codes themselves, but, rather, between the strategies followed by the system.

Two different strategies can be used in the channel decoder. Conceptually,

[1]Modulo-2 addition can also be defined as the addition operation in the Galois field GF(2). Since it is beyond the scope of this book to introduce Galois fields, we will always speak of modulo-2 addition.

these strategies can be related to Fano's inequality (see Chapter 3, (3.67)). In the first strategy, the decoder observes the hardly-demodulated received sequence and detects whether or not errors have occurred. A certain measure of uncertainty is eliminated, which corresponds to the term $H(e)$ in (3.67). Error detection is used to implement one of two possible schemes: error monitoring or automatic repeat request (ARQ). In the case of error monitoring, the decoder supplies to the user a continuous indication regarding the quality of the received sequence, so that, when the reliability becomes too low, the sequence can be discarded. In the case of ARQ, the transmitter is asked to repeat unsuccessful transmissions. To this end, a feedback channel from the receiver to the transmitter must be available.

The second strategy is called *forward error correction* (FEC). The decoder attempts to restore the correct transmitted sequence whenever errors are detected in the received sequence. In this case, an additional quantity of uncertainty must be removed corresponding to the term $P(e)\log(2^k - 1)$ of (3.67). It is intuitive that this strategy requires, for the same code, more complex decoding algorithms. The choice between the two strategies depends on the particular application and on the complexity of the transmission system considered. For example, the ARQ scheme is usually applied in the communication between computers, since a two-way transmission channel is available together with large memory devices that store the incoming information while performing, upon request, the retransmission procedure. On the other hand, FEC is adopted when the information must be protected on a one-way channel, or when real-time, or strictly-controlled delays are required. Examples pertain to deep-space communication and digitized interactive voice transmission.

With respect to the encoder operations, we say that the encoder E is *systematic* when the first k bits of each code word $\mathbf{x}$ coincide with the k bits of the data word $\mathbf{u}$. It is common in textbooks to say that a code, rather that its encoder, is systematic. In the following, we too will sometimes indulge in this imprecision. The reader is warned, though, that the concept of systematicity entails the mapping of data words into code words, and, thus, only pertains to the encoder.

To analyze the benefits due to channel encoding in comparison with the uncoded schemes of Chapter 5, let us consider again the model of Fig. 10.1. The source emits binary digits[2] at a rate of R_s bit/s and the encoder represents each data word of k source bits using $n = k/R_c$ bits. R_c is the code rate. To keep the pace of the source, the transmission speed on the channel must be increased to the value R_s/R_c binary symbols per second, and thus the required bandwidth must also be increased by the same factor $1/R_c$. As a result, the use of chan-

[2]Since we make the assumption that the data stream is made of iid binary RVs (0 and 1), we will use indifferently the words "bits" and "binary digits."

nel encoding decreases the bandwidth efficiency with respect to the uncoded transmission by a factor $1/R_c$. The binary symbols produced by the channel encoder are presented to the modulator, and converted into a sequence of waveforms using one of the modulation schemes described in Chapter 5 or 6. For the purposes of this preliminary discussion, we assume that the modulator uses an antipodal binary modulation over an AWGN channel. With this modulation scheme, each binary encoded symbol is mapped by the modulator into a binary waveform of duration $T = T_c = R_c/R_s$ seconds. This duration is shorter than that used in the uncoded case by a factor R_c. Denoting with $\mathcal{E}_b$ the energy per information bit, and assuming that the transmitted power is kept constant, we can conclude that coding decreases the energy per channel symbol to the value $\mathcal{E}_b R_c$. As a result, in case of hard decisions, more channel symbols will be incorrectly demodulated than with uncoded transmission. These observations about coding seem rather discouraging. In fact, bandwidth efficiency is decreased and more errors in the demodulated sequence are to be expected. Nevertheless, in a well-designed coded system, the larger number of errors at the demodulator output will be compensated for by the error-correcting capabilities of the decoder. Therefore, a coded transmission should trade bandwidth efficiency for a better overall error performance, using the same transmission power, or, equivalently, for a smaller required power for a given error performance. The decrease in the required power for the coded system is referred to as *coding gain*.

Let us describe the processing that must take place at the channel output to achieve such a result. Consider first the case in which the demodulator is used to make decisions on whether each binary waveform carries a 0 or a 1. To this purpose, the demodulator output is quantized to two levels denoted by 0 and 1 and is said to make *hard* decisions. The sequence of binary digits from the demodulator is fed into the decoder. The decoder attempts to recover the information sequence by using the code word's redundancy for either detecting or correcting the errors that are present at the demodulator output. Such a decoding process is called *hard-decision* decoding. In this model, assuming a binary antipodal coherent modulation and an AWGN channel, the combination of modulator, channel, and demodulator is equivalent to a binary symmetric channel (BSC). Its transition probability is the error probability of a binary antipodal modulation scheme (see Chapter 4). The overall error performance of the coded scheme depends on the implementation of efficient algorithms for error detection and correction.

At the other extreme, consider the case in which the unquantized output of the demodulator, the sufficient statistics, is fed to the decoder. This stores the n outputs corresponding to each sequence of n binary waveforms and builds 2^k decision variables. With the optimum decision strategy, the cascaded demodulator and decoder perform the same operation as the optimum coherent demodulator of Chapter 4, i.e., they choose the transmitted sequence corresponding to the n-bit code word which is closest, in the sense of the Euclidean distance, to the received sequence. Such a decoding process is called *unquantized soft-decision* decoding. In this model, the combination of modulator, channel, and demodulator is equivalent to a binary-input, continuous-output channel. It is intuitive that this approach presents a higher reliability than that achieved with the hard-decision scheme. In fact, the decoder can take advantage of the additional information contained in the unquantized samples that represent each individual binary transmitted waveform. An intermediate case, called *soft-decision* decoding, is represented by a demodulator whose output is quantized to Q levels, with $Q > 2$. In this case, the combination of modulator, channel, and demodulator is equivalent to a binary input, Q-ary output discrete channel. The advantage over the analog (unquantized) case is that all the processing can be accomplished with digital circuitry. Therefore, it represents an approximation of the unquantized soft-decision decoding.

The advantage of a coded transmission scheme over an uncoded one is usually measured by its *coding gain*. This is defined as the difference (in decibels) in the required value of $\mathcal{E}_b/N_0$ to achieve a given bit error probability between a binary antipodal uncoded transmission and the encoded one. This concept is represented qualitatively in Fig. 10.2, where we plot the two curves expressing the bit error probability $P_b(e)$ versus the signal-to-noise ratio per bit $\mathcal{E}_b/N_0$ for the uncoded and encoded systems. The typical behavior of the two curves of Figure 10.2 suggests two considerations:

- The coding gain, which depends on the value of the bit error probability (and thus on the signal-to-noise ratio), increases with the signal-to-noise ratio and tends (for $\mathcal{E}_b/N_0 \to \infty$ and hence for $P_b(e) \to 0$) to an asymptotic value that will be evaluated later in the chapter.

- For low values of the signal-to-noise ratio, there can be a crossing between the uncoded and coded curves, meaning that the coding gain becomes negative. In other words, there is a limit to what a code can do in terms of improving a bad channel.

To quantitatively assess the limits of the coding gain, we have plotted in Figure 10.3 the curve of the binary uncoded antipodal scheme (curve A) with the two channel capacity limits: the first (curve B), which tends to -1.6 dB, corresponding to the infinite-bandwidth capacity limit and to soft-decision decoding, and the second (curve C), which tends to 0.4 dB, the BSC capacity limit which refers to a hard-decision demodulator. These limits had been evaluated in Section 3.3.

The regions between the uncoded curve and those of the capacity limits represent the region of potential coding gains. As an example, for a bit error proba-

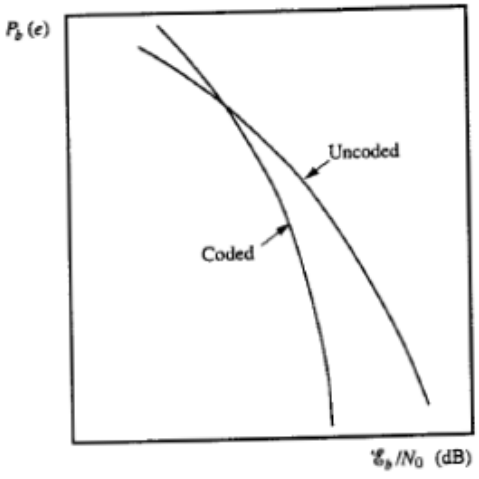

Figure 10.2: *Typical behavior of the bit error probability versus bit signal-to-noise ratio for uncoded and coded systems.*

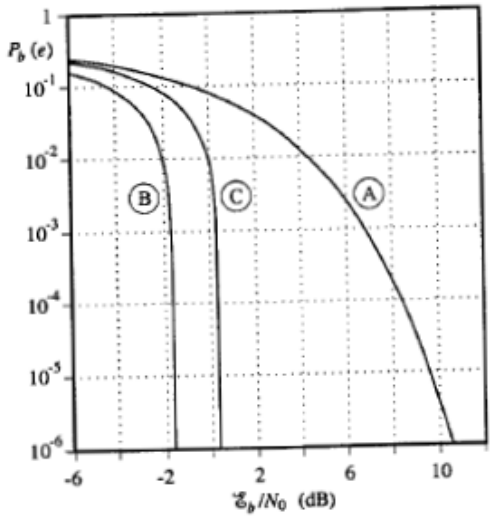

Figure 10.3: *Potential coding gains of coded transmission with respect to binary uncoded antipodal transmission.*

bility of 10^{-5}, a potential coding gain of 11.2 dB is theoretically available in the case of unquantized soft-decision decoding. Another limit, the *cutoff rate* limit, will be derived when analyzing the performance of coded transmission.

The fifty years that separate us from the channel coding theorem of Shannon has seen a great research effort aiming at filling the channel coding gap through the discovery of codes approaching the capacity limits. Until recently, these efforts had been very successful up to the *cutoff rate* limit (see Section 10.4), a couple of dB short of the capacity limit, but were unable to reach the region between cutoff rate and capacity. As we shall see at the end of next chapter, we now know a way to design codes that can approach to within 0.5 dB of the coding gain promised by the capacity limit at bit error probabilities of 10^{-5} to 10^{-7}.

10.2. Block codes

We will consider mainly *binary* codes, i.e., codes for which both the data words and the code words are formed by binary digits 0 and 1. This concept can be extended to q-ary codes, and a particularly important case occurs when $q = 2^b$ is a power of 2; in this case, q admits a binary representation with b bits, and the (n, k) code of q-ary elements is equivalent to an (nb, kb) binary code.

The basic feature of block codes is that the block of n digits (code word) generated by the encoder depends only on the corresponding block of k digits generated by the source (data word). Therefore, the encoder is memoryless. A great deal of block code theory is an extension of the notion of *parity check*. Take a sequence of k binary digits. Transform it into a sequence of length $n = k + 1$ digits by simply adding in the last position a new binary digit, following the rule that the number of ones in the new sequence must be even. This redundant digit is called a parity-check digit. In this way, any error event on the channel that changes the parity of the sequence from even to odd can be detected by the decoder.

Parity-check codes are a particular class of block codes in which the digits of the code word are a set of n parity checks performed on the k information digits. The code is usually referred to as an (n, k) code. An encoder (or, simply, a code) is called *systematic* when the first k digits in the code word are a replica of the information digits in the data word, and the remaining $(n - k)$ digits are parity checks on the k information digits. Parity checks in binary sequences are formally dealt with using modulo-2 arithmetic, in which the rules of ordinary arithmetic hold true except that the sum $(1 + 1)$ is 0, not 2. Throughout this chapter, modulo-2 arithmetic will be used unless otherwise specified. A functional block diagram of the encoder is shown in Fig. 10.4. It consists of a k-stage in-

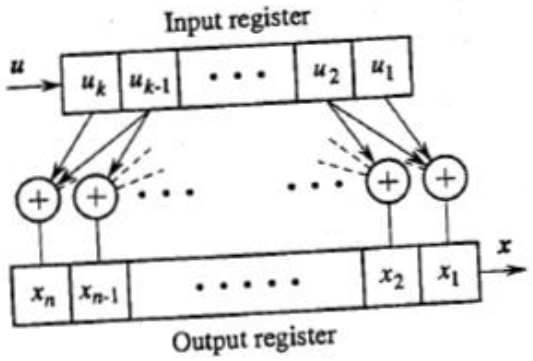

Figure 10.4: *Block diagram of the encoder for a parity-check code.*

put shift register, n modulo-2 adders, and an n-stage output shift register. Each adder is connected to a subset of stages of the input register in order to perform the desired parity checks. The vector $\mathbf{u} = [u_1, u_2, \ldots, u_k]$ of k information digits is fed into the input register. When this register is loaded, the content of each adder is fed in parallel into the corresponding stage of the output register, which shifts out the code word $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. While shifting out one code word, the input register is reloaded and the whole operation repeated. The clocks for the two registers are different, the output rate being higher by a factor $1/R_c$. The following simple examples will clarify these concepts.

Example 10.1 Repetition code (3, 1)
In this code, each code word of length $n = 3$ is defined by the relations

$$x_1 = u_1, \quad x_2 = u_1, \quad x_3 = u_1 \tag{10.1}$$

The encoder is sketched in Fig.10.5. Obviously, the adders are omitted in this case. The resulting *repetition* encoder is defined by the correspondence

Data words	Code words
0	000
1	111

Notice that the encoder is systematic, and that only two of the eight sequences of length 3 are used in the code. □

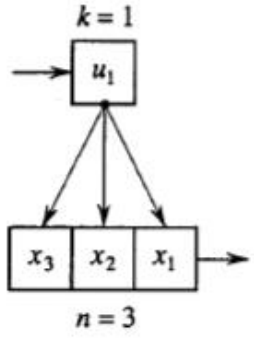

Figure 10.5: *Encoder for the repetition code* (3, 1).

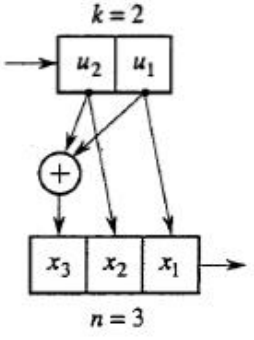

Figure 10.6: *Encoder for the parity-check code* (3, 2).

Example 10.2 Parity-check code (3, 2)
This is a code in which the third digit is a parity check on the first two digits. The code word is defined by the relations

$$x_1 = u_1, \quad x_2 = u_2, \quad x_3 = u_1 + u_2 \tag{10.2}$$

The systematic encoder is shown in Fig. 10.6. It is defined by the correspondence

Data words	Code words
00	000
01	011
10	101
11	110

Notice that only four of the eight sequences of length 3 are used in the code. □

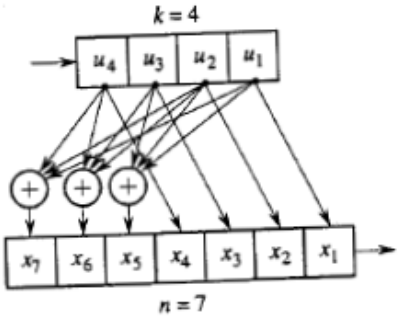

Figure 10.7: *Encoder for the Hamming code* (7, 4).

Example 10.3 Hamming code (7, 4)
The Hamming code (7, 4) is defined by the relations

$$\begin{aligned} x_i &= u_i, \qquad i = 1,2,3,4 \\ x_5 &= u_1 + u_2 + u_3 \\ x_6 &= u_2 + u_3 + u_4 \\ x_7 &= u_1 + u_2 + u_4 \,. \end{aligned} \tag{10.3}$$

The corresponding systematic encoder is shown in Fig. 10.7. It is defined by the correspondence

Data words	Code words
0000	0000 000
0001	0001 011
0010	0010 110
0011	0011 101
0100	0100 111
0101	0101 100
0110	0110 001
0111	0111 010
1000	1000 101
1001	1001 110
1010	1010 011
1011	1011 000
1100	1100 010
1101	1101 001
1110	1110 100
1111	1111 111

Notice that only 16 of the 128 sequences of length 7 are used in the code. □

These examples show that all the information required to specify the encoder operation is contained in relations of the type of (10.1), (10.2), and (10.3). With reference to Figs. 10.5, 10.6 and 10.7, these relations specify the connections between the input register cells and the adders. If the encoder is systematic, only the $(n-k)$ parity-check equations of the redundant digits must be assigned.

The information that specifies the encoding rule, and thus the structure of the encoder itself, can be concisely represented by the *generator* matrix G of the code. It is a $k \times n$ matrix whose (i, j) entry is 1 if the i-th cell of the input register is connected to the j-th adder, and 0 otherwise. Using the row-vector notation for the data word u and the code word x, the encoding rule is described by the equation

$$\mathbf{x} = \mathbf{u}\mathbf{G} \tag{10.4}$$

It is easily seen that obtaining a code word x through (10.4) is equivalent to summing the rows of the matrix G corresponding to the ones contained in the information sequence u.

Example 10.4 For the (7, 4) Hamming code of Example 10.3, the generator matrix G can be found by inspection of the encoder of Fig. 10.7 as follows:

$$\mathbf{G} = \left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \tag{10.5}$$

If we want the code word corresponding to the data word $\mathbf{u} = [1100]$, we must add the first two rows of G, obtaining

$$\begin{array}{r} 1000101 + \\ 0100111 = \\ \hline 1100010 \end{array}$$

and the result agrees with the code table given in Example 10.3. □

For systematic encoders, the first k columns of G form a $k \times k$ identity matrix, so that G assumes the form

$$\mathbf{G} = [\mathbf{I}_k \vdots \mathbf{P}] \tag{10.6}$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{P}$ is a $k \times (n-k)$ matrix containing the information regarding the parity checks. The knowledge of $\mathbf{P}$ completely defines the encoding rule for a systematic encoder.

The following important properties of parity check codes can be proved.

Property 1 The block code consists of all possible sums of the rows of the generator matrix.

Property 2 The sum of two code words is still a code word.

Property 3 The n-tuple of all zeros is always a code word.

Because of these properties, parity-check codes are also called *linear* codes. Linear block codes can be interpreted as being a subspace of the vector space containing all 2^n binary n-tuples. From this algebraic viewpoint, the rows of the generator matrix $\mathbf{G}$ are a basis of the subspace and consist of k linearly independent code words. In fact, all their 2^k linear combinations generate the entire subspace, that is, the code. Note that any k linearly independent code words of an (n, k) linear code can be used to form a generator matrix for the code. For these reasons, it is straightforward to conclude that any generator matrix of an (n, k) block code can be reduced, by means of row operations and column permutations, to the systematic form (10.6), which is also called *reduced-echelon* form. However, while row operations do not alter the code, column permutations may lead to a different set of code words, i.e., to a code that differs from the original one in the arrangement of its binary symbols. Two codes whose generator matrices can be obtained from each other by row operations and column permutations have the same *word* error probability, and, because of that, are said to be *equivalent*. Note, however, that their *bit* error probabilities (it will be defined later in the chapter, together with the word error probability) may be different, because equivalent codes can admit different encoders, and hence different mappings between data words and code words.

Thus, every (n, k) block code is equivalent to a systematic (n, k) block code (see Problem 10.5). Therefore, if the word error probability is the parameter of interest, we can consider only systematic codes without loss of generality.

An important parameter of a code word is its *Hamming weight*, that is, the number of ones that it contains. The set of all distinct weights in a code, together with the number of code words of that weight, is the *weight distribution* of the code. Owing to the previous definition, equivalent codes have the same weight distributions. Given two code words $\mathbf{x}_i$ and $\mathbf{x}_j$, it is useful to define a quantity to measure their difference. This quantity is the *Hamming distance* d_{ij} between the two code words, defined as the number of positions in which the two code words differ. Clearly, d_{ij} satisfies the condition $0 \leq d_{ij} \leq n$. The smallest

among the Hamming distances between distinct code words ($i \neq j$) is called the *minimum distance* $d_{\min}$ of the code. The following property allows an easy computation of $d_{\min}$ for linear codes.

Property 4 The minimum distance of a linear block code is the minimum weight of its nonzero code words.

In fact, the distance between two binary sequences is equal to the weight of their modulo-2 sum, and the sum of two code words is still a code word (Property 2).

Example 10.5 Consider again the (7, 4) Hamming code of Example 10.3. From the code table, we obtain the following weight distribution

Weight	Number of code words
0	1
3	7
4	7
7	1

Using Property 4, we get $d_{\min} = 3$. □

10.2.1. Error-detecting and error-correcting capabilities of a block code

Assume that the demodulator makes hard decisions so that the discrete channel between the channel encoder and decoder can be modeled as a binary symmetric channel (BSC). Each transmitted code word $\mathbf{x}$ is received at the decoder input as a sequence $\mathbf{y}$ of n binary digits (Fig. 10.1). The encoder is systematic. Therefore, the first k digits of $\mathbf{y}$ are the received information digits, while the remaining $(n-k)$ digits are the received parity-check digits. The sequence $\mathbf{y}$ can contain independent random errors caused by the channel noise. Let us define a binary vector $\mathbf{e}$ called an *error* vector:

$$\mathbf{e} = [e_1, \ldots, e_n] \tag{10.7}$$

Each component e_i is 1 if the channel has changed the i-th transmitted digit; otherwise, it is 0. The received vector is then

$$\mathbf{y} = \mathbf{x} + \mathbf{e} \tag{10.8}$$

where $\mathbf{x}$ is the transmitted code word. The decoder recomputes the $(n-k)$ parity-checks using the first k received bits, and compares them with the $(n-k)$

received parity-checks. If they match, the received sequence is a code word. Otherwise, an error is detected. Therefore, at least for error detection, the decoding rule is very simple: an error pattern is detected whenever at least one of the $(n-k)$ controls on parity checks fails. Let us define a vector $\mathbf{s}$ that contains the parity checks performed on the received word $\mathbf{y}$. Its $(n-k)$ binary digits are zeros for all parity checks that are satisfied, and ones for those that are not. The vector $\mathbf{s}$ is called the *syndrome* of the received vector $\mathbf{y}$. Recalling the definition (10.6) of the generator matrix $\mathbf{G}$ of a systematic code, it is easily verified that the syndrome can be obtained from the equation

$$\mathbf{s} = \mathbf{y}\mathbf{H}' \tag{10.9}$$

where the prime means transpose, and where we have introduced the *parity-check* matrix $\mathbf{H}$, defined as

$$\mathbf{H} \triangleq [\mathbf{P}' \vdots \mathbf{I}_{n-k}] \tag{10.10}$$

It is an $(n-k) \times n$ matrix, whose rows represent the parity-check symbols computed by the decoder. A direct calculation using (10.6) shows that

$$\mathbf{G}\mathbf{H}' = \mathbf{0} \tag{10.11}$$

where $\mathbf{0}$ is a $k \times (n-k)$ matrix all of whose elements are zero.

Example 10.6 Consider again the (7, 4) Hamming code of Example 10.3. The three parity-check symbols computed by the decoder on the received sequence $\mathbf{y}$ can be written by inspection of (10.3) as follows:

$$\begin{aligned} s_1 &= (y_1 + y_2 + y_3) + y_5 \\ s_2 &= (y_2 + y_3 + y_4) + y_6 \\ s_3 &= (y_1 + y_2 + y_4) + y_7 \end{aligned} \tag{10.12}$$

The parity check matrix is therefore

$$\mathbf{H} = \left[\begin{array}{cccc:ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{10.13}$$

It can be verified that (10.13) is also obtained from (10.5) using the definition (10.10). The property (10.11) can also be verified. □

From the definition of the syndrome associated with a received sequence $\mathbf{y}$, the following two properties can be verified:

Property 5 The syndrome associated with a sequence $\mathbf{y}$ is a zero vector if and only if $\mathbf{y}$ is a code word.

Property 6 The decoder can detect all channel errors represented by vectors $\mathbf{e}$ that are not code words.

Since the channel can introduce 2^n different error vectors, only 2^k of them are not detected by the decoder, that is, those corresponding to the set of code words. Finally, since no code word exists with a weight less than $d_{\min}$ (except, of course, the all-zero code word), the following theorem can be proved.

Theorem 10.1

A linear block code (n, k) with minimum distance $d_{\min}$ can detect all error vectors of weight not greater than $(d_{\min} - 1)$. ▽

Until now, we have only explored the error-detection capabilities of a hard-decision decoder. The problem of error correction is more complicated, since the syndrome does not contain sufficient information to locate the errors. Using (10.8), the expression (10.9) for the syndrome can be rewritten as

$$\mathbf{s} = \mathbf{y}\mathbf{H}' = (\mathbf{x} + \mathbf{e})\mathbf{H}' \tag{10.14}$$

where $\mathbf{x}$ is a code word. Since $\mathbf{x}\mathbf{H}' = \mathbf{0}$ (Property 5), there are 2^k different sequences $\mathbf{y}$ that generate the same syndrome. They are obtained by summing to a given error vector $\mathbf{e}$ the 2^k code words. Therefore, given a transmitted code word $\mathbf{x}$, there are 2^k error vectors that give the same syndrome. Which one actually occurred is an uncertainty that cannot be removed by using only the syndrome.

A suitable decoding algorithm must be elaborated. Assume that maximum likelihood (ML) hard decisions are taken by the decoder. This means that it achieves minimum word error probability on the received code words when they are equally likely. If p is the transition probability of the equivalent BSC implied by hard decisions, we have

$$P(\mathbf{y} \mid \mathbf{x}_i) = p^{d_i}(1-p)^{n-d_i} \tag{10.15}$$

where n is the block length and d_i is the Hamming distance between the received sequence $\mathbf{y}$ and the transmitted code word $\mathbf{x}_i$. Assuming, without loss of generality, $p < 1/2$, the probability $P(\mathbf{y} \mid \mathbf{x}_i)$ is a monotonic decreasing function of d_i. Therefore, ML decoding is accomplished with minimum Hamming-distance decisions. The "best" decoding algorithm decides for the code word $\mathbf{x}_i$ which is closest to $\mathbf{y}$. Recalling the discussion regarding (10.14), we can conclude that

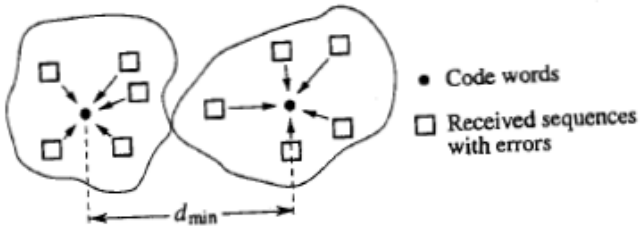

Figure 10.8: *Qualitative representation of the decision regions assigned to code words.*

the minimum-distance decoding algorithm assumes that the error vector e that actually occurred is the minimum-weight error vector in the set of the 2^k error vectors yielding the syndrome associated with the received sequence y. Before considering this decoding rule in detail using the minimum-distance decoding algorithm, let us relate the error-correcting capabilities of a block code to the code parameter d_{min}.

Theorem 10.2

A linear block code (n, k), with minimum distance d_{min}, can correct all error vectors containing no more than $t = \lfloor (d_{min} - 1)/2 \rfloor$ errors, where $\lfloor a \rfloor$ (the "floor" of a) denotes the largest integer contained in a. The code is then a t-error-correcting code, and is often denoted as an (n, k, t) code. ∇

Proof of Theorem 10.2

The decoding algorithm is implemented by assigning to each code word a decision region containing the subset of all the received sequences that are closer to it than to any other (*minimum distance decoding*, see Fig. 10.8). An error vector with no more than $\lceil (d_{min} - 1)/2 \rceil$ errors produces a received sequence lying inside the correct decision region. Error correction is therefore possible. **QED**

The results of Theorems 10.1 and 10.2 are summarized in Table 10.1.

Based on previous Theorems 10.1 and 10.2, a design goal for a block code (n, k) is to use its redundancy to achieve the largest possible minimum distance d_{min}. So far, no general solution to this problem is known. Instead, upper and lower bounds to d_{min} are used. Some of them will be described in Section 10.4.2.

d_{min}	Errors detected	Errors corrected
2	1	0
3	2	1
4	3	1
5	4	2
6	5	2
7	6	3
8	7	3
9	8	4

Table 10.1: *Error correction and error detection capabilities of linear block codes as a function of* d_{min}.

10.2.2. Decoding table and standard array of a linear block code

Using the minimum-distance hard decoding algorithm just described, the decoding operation can be performed by looking for the code word nearest to the received sequence. This approach requires the storage of the 2^k code words and repeated comparisons with the received sequence. The total storage requirement is on the order of $n \times 2^k$ bits. Hence, the approach becomes rapidly impractical even for moderately-sized codes. Also, the comparison process is unacceptably long when n and k are large.

A more efficient approach is to evaluate the syndrome associated with the received sequence y by assuming that the error vector e that actually occurred is the minimum-weight vector in the set of the 2^k vectors that generate the same syndrome. With this approach, we can build a decoding table by associating with each syndrome the error vector of minimum weight that generated it. The positions of the ones in the error vector indicate the digits that must be corrected in the received sequence y. This approach is better clarified by the following example.

Example 10.7 The (7, 4) Hamming code has minimum distance 3. Thus, it is expected to correct all single errors. There are, of course, 128 possible received words and only 8 different syndromes. All these sequences are included in Table 10.2. They are grouped in rows containing all sequences that share the same syndrome. The syndrome is shown as the first entry in each row. The first column of the table contains all error vectors of minimum weight. It can be verified by inspection that each error vector containing only one error has a different syndrome, and hence it can be corrected. Therefore, the decoding table for this code is the following:

Syn-dromes	Error patterns															
	Coset leaders	Other errors														
000	0000000	1000101	0100111	0010110	0001011	1100010	1010011	1001110	0110001	0101100	0011101	0111010	1011000	1101001	1110100	1111111
001	0000001	1000100	0100110	0010111	0001010	1100011	1010010	1001111	0110000	0101101	0011100	0111011	1011001	1101000	1110101	1111110
010	0000010	1000111	0100101	0010100	0001001	1100000	1010001	1001100	0110011	0101110	0011111	0111000	1011010	1101011	1110110	1111101
011	0001000	1001101	0101111	0011110	0000011	1101010	1011011	1000110	0111001	0100100	0010101	0110010	1010000	1100001	1111100	1110111
100	0000100	1000001	0100011	0010010	0001111	1100110	1010111	1001010	0110101	0101000	0011001	0111110	1011100	1101101	1110000	1111011
101	1000000	0000101	1100111	1010110	1001011	0100010	0010011	0001110	1110001	1101100	1011101	1111010	0011000	0101001	0110100	0111111
110	0010000	1010101	0110111	0000110	0011011	1110010	1000011	1011110	0100001	0111100	0001101	0101010	1001000	1111001	1100100	1101111
111	0100000	1100101	0000111	0110110	0101011	1000010	1110011	1101110	0010001	0001100	0111101	0011010	1111000	1001001	1010100	1011111

Table 10.2: *Standard array of the* (7, 4) *Hamming code.*

Syndrome	Error vector	Digit in error
000	0000000	None
001	0000001	7
010	0000010	6
011	0001000	4
100	0000100	5
101	1000000	1
110	0010000	3
111	0100000	2

Obviously, the syndrome 000 corresponds to the set of the 16 code words. The syndrome 111 locates an error in the second position of the received sequence, and so on. Table 10.2 can also be interpreted as follows. Assume that the sequence 1101010 is received. The corresponding syndrome is 011. Therefore, an error in position 4 is assumed and corrected. The code word obtained, which is 1100010, appears at the top of the column containing the received sequence. □

A table such as Table 10.2, containing all the 2^n n-tuples (the possible received words) of length n organized in that order, is called the *standard array* of the code. It has 2^k columns and 2^{n-k} rows. The rows are called *cosets*. The first word in each row is nominated a *coset leader*. The top word in a column is a code word, and each coset leader is the minimum-weight word that generates the syndrome common to all words of that coset.

The decoding table is built by simply associating with each syndrome the corresponding coset leader of the standard array. The coset leaders are therefore the correctable error vectors; if the error vector is not a coset leader, then an incorrect decoding will be performed. To minimize the average word error probability, the coset leaders must be the error vectors that are the most likely to occur. For a BSC, the coset leaders are the minimum-weight words associated with a given syndrome. Therefore, the decoding algorithm works as follows:

1. Compute the syndrome for the received sequence.
2. Find the correctable error vector (coset leader) in the decoding table.
3. Get the estimated code word by adding the correctable error vector to the received word.

The decoding table requires the storage of 2^{n-k} syndromes of length $(n-k)$ and of 2^{n-k} error patterns of length n: a total of $2^{n-k} \times (2n-k)$ bits. For high rate codes ($k \simeq n$), the storage requirement is close to $n \times 2^{n-k}$, considerably less as compared to the $n \times 2^k$ bits required by an exhaustive search. In spite of

l	(n, k)
2	(3, 1)
3	(7, 4)
4	(15, 11)
5	(31, 26)
6	(63, 57)
7	(127, 120)

Table 10.3: *Parameters of the first Hamming codes.*

this interesting result, the decoding table, too, becomes impractical when n and k are large numbers. In that case a more elaborate algebraic structure must be assigned to the code in order to employ decoding strategies based on computational algorithms, rather than on look-up tables.

10.2.3. Hamming codes

Equation (10.14) can be rewritten in the form

$$\mathbf{s} = \mathbf{e}\mathbf{H}' \tag{10.16}$$

Therefore, the syndrome of a given sequence is the sum of the columns of $\mathbf{H}$ corresponding to the position of the ones in the error vector. Consequently, if a column of $\mathbf{H}$ is zero, an error in that position cannot be detected. Furthermore, if two columns of $\mathbf{H}$ are equal, a single error in one of those two positions cannot be corrected since the two syndromes are not distinct. We can conclude that a block code can correct all single errors if and only if the columns of its parity-check matrix $\mathbf{H}$ are nonzero and distinct.

Hamming codes are characterized by a matrix $\mathbf{H}$ whose columns are all the possible sequences of $(n-k)$ binary digits except the zero sequence. For every $l = 2, 3, 4, \ldots$, there is a $(2^l - 1, 2^l - 1 - l)$ Hamming code. These codes have $d_{\min} = 3$ and are thus capable of correcting all single errors. Their rate $R_c = (2^l - 1 - l)/(2^l - 1)$ increases with l and approaches 1 for $l \to \infty$. The parameters of the first six Hamming codes are listed in Table 10.3.

Example 10.8 The parity-check matrix of the Hamming code (15,11) is the following:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{10.17}$$

Notice that $\mathbf{H}$ is not written in the systematic form of (10.10), its columns being in lexicographical order. It can be reduced to systematic form by a simple rearrangement of columns. The interesting property of (10.17) is that the 4-tuple in each column, as a binary number, identifies the column position. Therefore, an error vector with a single error will generate a syndrome that gives, in binary form, the position of the error in the received sequence. This information can be used for correction. □

Hamming codes have an interesting property that can be verified by inspection of the standard array (see Table 10.2 for the (7, 4) code). All possible received sequences have Hamming distance 1 from one of the code words. Codes of this type are called *perfect codes*. Another property of the Hamming codes is that they are one of the few classes of codes for which the complete weight distribution is known. The weight distribution of a code can be represented in a compact form as a polynomial, called the *weight enumerating function* (WEF) of the code. It is a polynomial in the indeterminate D

$$A(D) = \sum_{d=0}^{n} A_d D^d \tag{10.18}$$

where A_d is the number (multiplicity) of code words in the code with weight (or, equivalently, Hamming distance from the all-zero code word) d. For the Hamming codes, the WEF can be shown to be

$$A(D) = \frac{1}{n+1}[(1+D)^n + n(1+D)^{(n-1)/2}(1-D)^{(n+1)/2}] \tag{10.19}$$

The result of Example 10.5 can be checked against (10.19).

Each Hamming code can be converted to a new code by adding one parity digit that checks all previous n digits of the code word. This results in a class of $(2^l, 2^l - 1 - l)$ block codes called *extended* Hamming codes. Their parity-check matrix $\mathbf{H}_{\text{ext}}$ is obtained by adding a new row to the Hamming parity-check matrix $\mathbf{H}$ as follows:

$$\mathbf{H}_{\text{ext}} = \begin{bmatrix} & & & & \vdots & 0 \\ & & & & \vdots & 0 \\ & \mathbf{H} & & & \vdots & \vdots \\ & & & & \vdots & 0 \\ 1 & 1 & \ldots & 1 & \vdots & 1 \end{bmatrix} \tag{10.20}$$

The last row represents the overall parity-check digit. Since, with an overall parity-check, the weight of every code word must be even, the extended Hamming codes have $d_{\min} = 4$. Their particular structure makes it possible to detect

all double errors while simultaneously correcting all single errors (as in the original Hamming codes). In fact, the syndromes for double errors form a subset distinct from that of the syndromes for single errors. The decoding algorithm works as follows:

1. If the last digit of the syndrome is 1, then the number of errors must be odd. Using the minimum-distance algorithm, correction can be performed as for the Hamming codes.

2. If the last digit of the syndrome is 0, but the syndrome is not all-zero, then no correction is possible since at least two errors must have occurred. A double error is therefore detected.

This property of extending a code by the addition of an overall parity check can be applied to any linear block code other than the Hamming codes. In particular, any linear (n, k) block code with an odd minimum distance can be converted into an extended $(n+1, k)$ block code with a minimum distance increased by one.

10.2.4. Dual codes

The generator matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$ of a linear (n, k) block code are related by (10.11). This relation can be rewritten as

$$\mathbf{H}\mathbf{G}' = \mathbf{0} \tag{10.21}$$

Thus, the two matrices can be interchanged and the $\mathbf{H}$ matrix can be the generator matrix of a new $(n, n-k)$ block code. Codes that are so related are said to be *dual* codes. There is a very interesting relationship between the weight distributions of two dual codes. Let $A(D)$ be the weight enumerating function of the (n, k) block code and $A_{\text{dual}}(D)$ the weight enumerating function of its $(n, n-k)$ dual code. Then, the two weight enumerating functions are related by the identity (MacWilliams and Sloane, 1977)

$$A_{\text{dual}}(D) = 2^{-k}(1+D)^n A\left(\frac{1-D}{1+D}\right) \tag{10.22}$$

This relationship is very useful in determining the weight structure of high-rate block codes through an exhaustive computer search performed on their low-rate dual codes.

10.2.5. Maximal-length codes

The duals of the Hamming codes are called *maximal-length* codes. Therefore, for every $l = 2, 3, 4, \ldots$ there is a $(2^l - 1, l)$ maximal-length code. Its generator matrix is the parity-check matrix of the corresponding Hamming code.

The weight distribution of these codes can be easily determined by introducing (10.19) into (10.22). The weight enumerator $A(D)$ for the maximal-length codes is thus found to be

$$A(D) = 1 + (2^l - 1)D^{2^{l-1}} \tag{10.23}$$

Hence, all nonzero code words have identical weight 2^{l-1}. Also, this is the minimum distance of the code. These codes are also called *equidistant* or *simplex* codes. Additional insight into the properties of these codes will be obtained later in connection with the description of cyclic codes.

10.2.6. Reed-Muller codes

The *Reed-Muller* codes are a class of linear block codes covering a wide range of rates and minimum distances. They present very interesting properties, among them, the fact that they can be soft-decoded by using a simple trellis (see Forney, 1988b).

For any m and $r < m$, there is a Reed-Muller code with parameters given by

$$n = 2^m, \quad k = \sum_{i=0}^{r} \binom{m}{i}, \quad d_{\min} = 2^{m-r} \tag{10.24}$$

The generator matrix $\mathbf{G}$ of the rth-order Reed-Muller code is defined by assigning a set of vectors as follows. Let $\mathbf{v}_0$ be a vector whose 2^m elements are all ones, and let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ be the rows of a matrix with all possible m-tuples as columns. The rows of the rth-order generator matrix are the vectors $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_m$ and all the products of $\mathbf{v}_1, \ldots, \mathbf{v}_m$ two at a time, three at a time, up to r at a time. Here the product vector $\mathbf{v}_i\mathbf{v}_j$ has components given by the products of the corresponding components of $\mathbf{v}_i$ and $\mathbf{v}_j$.

Example 10.9 In this example, we show how to generate the Reed-Muller codes with $m = 3$. There are two codes. They have the following parameters:

r	n	k	$d_{\min}$
1	8	4	4
2	8	7	2

The vectors used for building the generator matrices are given in Table 10.4. The first-order code ($r = 1$) is generated by using the vectors $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ as rows of the generator matrix. The second-order code ($r = 2$) is generated by augmenting this matrix with the additional three rows of Table 10.4. □

The first-order Reed-Muller codes are closely related to the maximal-length codes. If a maximal-length code is extended by adding an overall parity check,

$\mathbf{v}_0$	1	1	1	1	1	1	1	1
$\mathbf{v}_1$	0	0	0	0	1	1	1	1
$\mathbf{v}_2$	0	0	1	1	0	0	1	1
$\mathbf{v}_3$	0	1	0	1	0	1	0	1
$\mathbf{v}_1\mathbf{v}_2$	0	0	0	0	0	0	1	1
$\mathbf{v}_1\mathbf{v}_3$	0	0	0	0	0	1	0	1
$\mathbf{v}_2\mathbf{v}_3$	0	0	0	1	0	0	0	1

Table 10.4: *Vectors for constructing the generator matrix of Reed-Muller codes with* $m = 3$.

we obtain an *orthogonal* code. This code has 2^m code words. Each has weight 2^{m-1}, except for the all-zero code word. Therefore, every code word agrees in 2^{m-1} positions and disagrees in 2^{m-1} positions with every other code word. If this code is transmitted using an antipodal signaling scheme, each code word is represented by one out of 2^m orthogonal signals. This explains the name "orthogonal" code. For the case $m = 3$, the code generator matrix consists of the three rows $\mathbf{v}_1, \mathbf{v}_2$, and $\mathbf{v}_3$ of Table 10.4. In fact, the first column represents the overall parity-check digit, whereas the other columns are all the seven possible triples of binary digits. The first-order Reed-Muller code is obtained from this code (the orthogonal code) by adding to the generator matrix the all-ones vector $\mathbf{v}_0$. In terms of transmitted signals, this operation adds to the original orthogonal signal set the opposite of each signal. For this reason, the code is also called a *biorthogonal* code. Finally, notice that the rth-order Reed-Muller code is the dual of the Reed-Muller code of order $(m - r - 1)$.

10.2.7. Cyclic codes

The *cyclic codes* are parity-check codes that present a large amount of mathematical structure. These codes share, of course, all the properties previously described for parity-check codes, but, in addition, have peculiar properties that allow easy encoding operations and simple decoding algorithms. Cyclic codes are, for this reason, of great practical interest.

Definition 10.1

An (n, k) linear block code is a cyclic code if and only if any cyclic shift of a code word produces another code word.

Example 10.10 It can be verified that the (7, 4) Hamming code of Example 10.3 is a cyclic code. Take, for instance, the code word 0111010. There are six different cyclic shifts of this code word.

1110100 1101001 1010011 0100111 1001110 0011101

They all belong to the set of code words. The same is true for all the code words. □

In dealing with cyclic codes, it is useful to represent a binary sequence of n bits as a polynomial in the indeterminate Z of degree not greater than $(n - 1)$ with binary (0 and 1) coefficients. The binary digits of a code word will be numbered in decreasing order from $(n - 1)$ to 0, so that each index matches the exponent of Z. A code word $\mathbf{x} = [x_{n-1}, x_{n-2}, \ldots, x_0]$ is then represented by the code polynomial $x(Z)$ as follows:

$$x(Z) = x_{n-1}Z^{n-1} + x_{n-2}Z^{n-2} + \ldots + x_1 Z + x_0 \tag{10.25}$$

The binary coefficients of this polynomial will be manipulated with the rules of modulo-2 arithmetic. In this new notation, the code words of an (n, k) linear block code are in a one-to-one correspondence with code polynomials of degree not greater than $(n - 1)$.

By definition of cyclic code, if $x(Z)$ is the code polynomial of a cyclic code, then a cyclic shift of the code word (say to the left) of i positions generates another code polynomial that we denote by $x^{(i)}(Z)$. Theorem 10.3 relates the polynomial representation of a cyclically shifted n-tuple to the binomial $Z^n + 1$, which will be shown to play a crucial role for cyclic codes.

Theorem 10.3

The code polynomial $x^{(i)}(Z)$ is the remainder resulting from dividing $Z^i x(Z)$ by $(Z^n + 1)$; that is,

$$Z^i x(Z) = q(Z)(Z^n + 1) + x^{(i)}(Z) \tag{10.26}$$

where $q(Z)$ is the quotient polynomial of degree not greater than $(i - 1)$. ▽

Proof of Theorem 10.3

Let us write explicitly $Z^i x(Z)$

$$Z^i x(Z) = x_{n-1}Z^{n-1+i} + x_{n-2}Z^{n-2+i} + \ldots + x_{n-i}Z^n + \ldots + x_0 Z^i$$