7. ERROR CONTROL CODING

7.1. Introduction

Let us consider a digital communication channel transmitting binary series ('0'-s and '1'-s) entering the channel input. The channel consists of a modulator, the physical transmission medium and a demodulator. The modulator converts input 0s and 1s to pairs of signals suitable for transmission through the medium. During transmission, these signals are distorted and disturbed by noise. The infinite set of the received signals is then converted back to 0s and 1s by the demodulator using a decision rule. These decisions, however, are not free of errors. The probability of error would certainly be reduced if the transmitted power or the duration of the signals were increased. These methods are not used because neither poor efficiency nor low transfer rate is desirable. Fortunately, there is a procedure called *error control coding* to keep the probability of transmission errors at an acceptably low level.

There are two tasks in the error control coding: error *detection* and error *correction*. When error detection is used, receiver informs the transmitter on a backward channel that an error has been found and requests the transmitter to re-send the signal. In the case of error correction, the receiver is able to correct certain errors. Hybrid coding procedures are also used where the receiver first tries to correct the error and then checks the result by error detection.

7.2. Basic Terms

The basic structure containing error control is shown in Fig. 7.1.

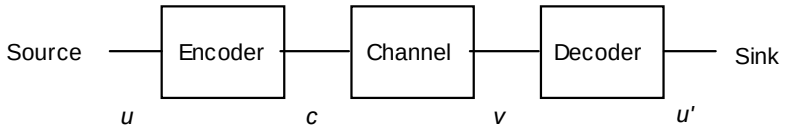


Figure. 7.1. Error Correction in the Communication Channel

The source transmits k bit long binary *messages* $u = (u_1, u_2, ..., u_k)$ through the communication channel towards the sink. Each message is converted by the encoder to an n bit long binary *code word* $c = (c_1, c_2, ..., c_n)$. The *word received* at the channel output is $v = (v_1, v_2,..., v_n)$ and is of the same length as c.

An error caused by the channel is said to occur at the *m-th* position if $c_m \neq v_m$. Let t be the *number of all errors* occurring during the transfer of c. Generally *d(c,v),* the *Hamming distance* of arbitrary words c and v is defined as the number of those positions where these two words differ, i.e. $t = d(c, v)$.

Code is defined as a 2^k set of binary vectors of length n, k is the length of the binary message. Therefore the code is usually denoted as *C(n, k).* The elements of the code are called *code words. Coding* is a reversible process which transforms messages into code words, i.e. different messages are transformed into different code words.

Decoding is performed in two consecutive steps: First, on the basis of a *decision rule*, the received word v is decoded to code word c' then -as the inverse of encoding- a u' message is assigned to the decoded code word.

The most frequently used decision rule chooses that code word *c'* which has the shortest Hamming distance to the received word *v*, i.e.

$$d(c', v) = \min d(c,v), \quad c \in C$$

As it turns out from the previous discussion, there are two main tasks of error correction coding. First, a code has to be set up which creates code words with Hamming distances as great as possible. As the second, such a decision rule has to be constructed which finds the code word *c'* being of the minimum distance from the received code in a simple way, i.e. without the need to go through and compare with all the code words. If the code is short, however, such a thorough examination is possible yet. For instance, if $n \leq 10$ then the so called *table lookup decoding* can be used.

Example 7.1. Let us choose $k = 2$, $n = 5$ and consider the following code:

u	***c***
00	00000
01	01101
10	10110
11	11011

In this case, the first ten rows of the 32-row decoding table are

v	***c'***	***u'***
00000	00000	00
10000	00000	00
01000	00000	00
11000	00000	00
00100	00000	00
10100	10110	10
01100	01101	01
11100	01101	01
00010	00000	00
10010	10110	10

Because the limited size of the memory for storing the table, table lookup decoding cannot be usually used for large values of *n*. If the value of *k* is small ($k << n$), however, the number of code words (2^k) is small, too, even if the code is long. In such a case, decoding can be performed by comparing *v(x)* to each code word and finding the most similar one.

In typical cases however, neither of the two previous methods can be used. Just imagine the case when the code length *k* is 50 bits, i.e. $2^{50} \approx 10^{15}$ code words! To show a solution for such a case, further terms have to be introduced.

The minimum Hamming distance between the code words of a code is a very important parameter. It is called *code distance* and is denoted as d_{min}.
Thus, formally $d_{min} = \min \{d(c, c')\}$, $c \neq c'$, $c, c' \in C$. It is easy to check that the code distance in the example 7.1. is $d_{min} = 3$.

The aim of *error detection* is to decide whether the received word is a code word or not. If the number of errors within one received code word is not more than *t* and $d_{min} > t$, it is certain that no combination of errors results in an other code word. This is very important, otherwise it would be impossible to detect errors in the receiver.

Theorem 7.1.: If the minimum Hamming distance of a code d_{min}, it is able to detect d_{min}-1 errors.

Generally, a code is said to have error detection (or error correction) capability *t*, if it is able to detect (or to correct) not less than *t* errors and there is at least one received word with *t*+1 errors in which the errors cannot be detected. For instance, error detection capability of the code given in example 7.1. is 2, in accordance with the above theorem.

When speaking about error *correction*, the question is what should be the condition for an *unambiguous* restoration of the transmitted code word *c* from the received word *v*. The formal condition is that for any other code word *c''*

$$d(v, c'') > d(v, c) \qquad (7.1.)$$

i.e. *c* has to be the closest to the received word. The Hamming distance is really a distance (non negative, symmetrical and it fulfills the triangle-inequality) wherefore

$$d(v, c'') \geq d(c, c'') - d(v, c) \qquad (7.2.)$$

The equation (7.1). can be satisfied if the right side of the equation (7.2.) is greater than $d(v, c)$, i.e. if

$$d(c, c'') - d(v, c) > d(v, c) \qquad (7.3.)$$

which leads to $d(c,c'') > 2d(v,c)$. This condition is certainly satisfied (taking into account also that $d(c, c'') \geq d_{min}$, $c \neq c''$) if $d_{min}/2 > d(v, c)$ is true.

Theorem 7.2.: If the minimum Hamming distance of a code d_{min}, it is able to correct int[$(d_{min}$-1)/2] errors.

Error correction ability of the code given in example 7.1. is 1, i.e. the code is able to correct one error (any damaged bit) of the received word. It follows from theorem 7.2. that for the correction of *t* errors, the minimum Hamming distance of the code must be $d_{min} \geq 2t+1$.

Now, let us face the following problem: How can a code be constructed with a sufficiently great code distance? To answer this question, let us introduce the term of linear codes.

7.3. Linear Codes

The idea of linear coding can be demonstrated by a simple example.

Example 7.2.: Let us have a look at the code used in example 7.1. again! It can be easily checked that this encoding can be interpreted as a matrix multiplication $c = u \cdot G$ where

$$\mathbf{G} = \begin{pmatrix} 10110 \\ 01101 \end{pmatrix}$$

Notice that ***G*** is a 2 x 5 (*k* x *n*) binary matrix. In this simple case the set of codes consists of the all-zero vector, the first and the second row the ***G*** matrix, and co-ordinate- by-co-ordinate modulo-2 (XOR) sum of the two rows. Thus the elements of the code *C* are generated as linear combinations of the rows of the matrix ***G***. The code -as a set of binary vectors- forms a linear space. Generalizing this remark we can understand the term of linear codes.

A binary code C is called *linear* if the set C is a *linear space*, i.e. if for all $c, c' \hat{I} C$, $c + c' \hat{I} C$ is also true. Thus the all-zero code word (0) is also an element of the linear code, because $c + c = 0$ is true for any binary code word.

Linear codes are significant since their code words are generated relatively simply, error detection and correction is also simpler than for nonlinear codes. Terms commonly used for the space of real vectors remain valid in the space of binary vectors. Suppose that vectors $g_1, g_2, .., g_k$ form a base of the linear space C, i.e. with these vectors, any $c \in C$ element can be generated as

$$c = \sum_{i=1}^{k} u_i \cdot g_i \qquad i=1, 2, ..., k$$

Let us build a k x n matrix $\boldsymbol{G}$, the rows of which are $g_1, g_2, ..., g_k$. Encoding is performed by $c = u \cdot \boldsymbol{G}$ and the matrix $\boldsymbol{G}$ is called obviously as *generator matrix.*

The code C can thus be succinctly given by an appropriate set of just k words instead of listing all the 2^k code words. Furthermore, encoding is governed by a simple rule. Notice that several generator matrices belong to the same code, i.e. as many as many different bases, the actual linear space can have. On the other hand, there is only one generator matrix which encodes given messages into given code words.

Going back to the example 7.1., one can discover that the encoding was chosen so that the first two bits of each code word corresponds to the message itself. This is advantageous because the second step of decoding, i.e. asserting the proper message to decoded code word is trivial as decoding simply means detachment of the first k bits of the code word. The principle of such encoding can be generalized as follows:

An C(n, k) code is called *systematic* if the first k bits of its code words correspond to the messages. Generator matrix of a systematic code is unambiguous and according to matrix multiplication rules, it is of the following form:

$$\boldsymbol{G} = (\boldsymbol{I}_k, \boldsymbol{B}) \qquad (7.4.)$$

$\boldsymbol{I}_k$ is a unity matrix of size k x k and $\boldsymbol{B}$ is a matrix of size k x (n-k). Structure of the code word belonging to the message u is

$$c = (u_1, u_2, ..., u_k, c_{k+1}, c_{k+2}, ..., c_n)$$

The first k co-ordinates of the code word are called the *message segment* and the following n-k co-ordinates are called the *parity segment.*

To select code words of a linear code C out of a set of 2^n binary vectors, an (n - k) x n binary matrix $\boldsymbol{H}$ of sizecan be assigned to the code. For this matrix

$$\boldsymbol{H}c^T = 0 \qquad (7.5.)$$

is true if and only if $c \in C$ where $(.)^T$ operator stands for matrix transposition. H matrix is called the *parity-check matrix*. If the code is systematic then

$$\boldsymbol{H} = (\boldsymbol{A}, \boldsymbol{I}_{n-k}), \qquad (7.6.)$$

where

$$\boldsymbol{A} = -\boldsymbol{B}_T \qquad (7.7.)$$

and $\boldsymbol{I}_{n-k}$ is an (n - k) x (n - k) unity matrix.

Equations (7.6.) and (7.7.) can easily be proved: Starting from (7.3.) and (7.5.), a chain of equations can be set up for an arbitrary pair of *c* and *u*:

$$\boldsymbol{H}c^T = \boldsymbol{H}(u\boldsymbol{G})^T = \boldsymbol{H}\boldsymbol{G}^T u^T = 0$$

so that

$$\boldsymbol{H}\boldsymbol{G}^T = 0 \qquad (7.8.)$$

Substituting equations (7.4.) and (7.6.) into (7.8.)

$$\boldsymbol{H}\boldsymbol{G}^T = (\boldsymbol{A}, \boldsymbol{I}_{n-k})\cdot(\boldsymbol{I}_k, \boldsymbol{B})^T = \boldsymbol{A} + \boldsymbol{B}^T = 0$$

which validates equation (7.7.)

Example 7.3.: Suppose we have the code *C* of example 7.1. the generator matrix of which is given in example 7.2. Using the notation introduced above:

$B = \begin{pmatrix} 110 \\ 101 \end{pmatrix}$, and taking into account that -1 = 1 (modulo 2),

$A = \begin{pmatrix} 11 \\ 10 \\ 01 \end{pmatrix}$, so the parity matrix is $\boldsymbol{H} = \begin{pmatrix} 11100 \\ 10010 \\ 01001 \end{pmatrix}$

In the following, a procedure will be shown how to use the ***H*** matrix for decoding. Let c be the code word sent and v the word received. The difference of the two vectors is called the *error vector*:

$$e = v - c$$

For instance, if c = (10110) and v = (11110) then e = (01000) indicating that the 2nd bit was damaged. Notice that using (7.5.)

$$\boldsymbol{H}v^T = \boldsymbol{H}(c+e)^T = \boldsymbol{H}c^T + \boldsymbol{H}e^T = \boldsymbol{H}e^T,$$

i.e. the value of $\boldsymbol{H}v^T$ depends only on the error vector and is independent of the code word. The following quantity

$$s = e\boldsymbol{H}^T \qquad (7.9.)$$

which is just the row version of He^T is called the *syndrome* of the error vector *e*. Syndromes of the code words are **0**. (Row vector $e\boldsymbol{H}^T$ corresponds to the column vector $\boldsymbol{H}e^T$). Figure 7.2. visualizes this multiplication (7.9.)

Returning to example 7.3., syndrome of the error vector e = (01000) is s = (101). The length of the syndrome vector is $n - k$, that is 5 – 2 = 3. As the syndromes are independent of code words, a table of syndromes can be set up for table lookup decoding. This table is much shorter as if it would have been given for the code words and its structure is as follows:

syndrome	error vector with min. errors
s_0=0	e_0=0

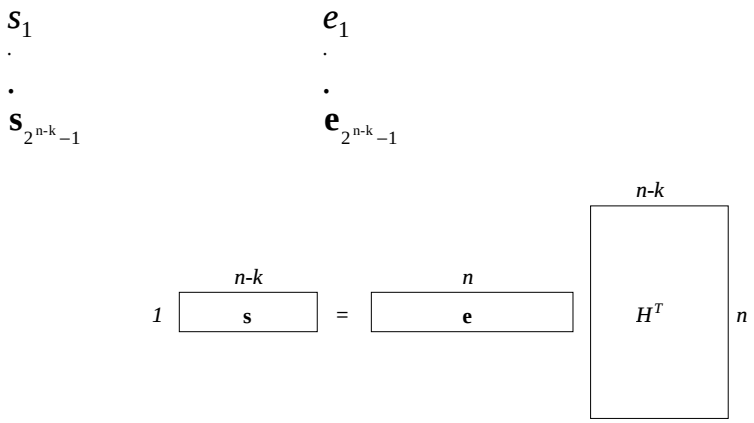


Fig. 7.2. Dimensional Presentation of the Syndrome Computation

In the first row, there is the zero syndrome and the corresponding zero-error vector (error-free case). The length of the syndrome vectors is *n-k* so that the number of the syndromes is only 2^{n-k}.

The steps of *syndrome decoding* are as follows:
1. Compute the syndrome *s* corresponding to the received word *v*.
2. Read out the predicted error vector corresponding to the computed *s* from the table.
3. Compute *c'*= *v* - *e*
4. Assign decoded message *u'* to the computed *c'*.

Let us illustrate the above procedure by an example!

Example 7.4.: Using the parity matrix **H** of the example 7.3., *C*(5, 2) code will result in the following table:

s	*e*
000	00000
100	00100
010	00010
110	10000
001	00001
101	01000
011	00011
111	01010

As it can be seen from the table, the code is able to correct all the five different single errors and also two double errors. In the following let us overview the basic characteristics of some well-known and simple codes.

7.4. Simple Linear Codes

7.4.1. Repeat Code

The simplest error correction code is the repeat code. In this case, the length of the message is *k* = 1, which is repeated *n*-times, so that the resulting code is *C*(*n*, 1). According to only two possible messages (0 or 1), the code contains just two code words (000...0 and

111...1). Obviously, the code distance is n, thus (n - 1)/2 errors can be corrected (preferably, n is an odd number).

Example 7.5.: The simplest repeat code being able to correct one error is made up of code words (000) and (111). For instance, if the received word is damaged to (110) then (111) is decoded and the message is decided to be 1.

7.4.2. Single Parity-Check Code

The simplest error detection code is the single parity-check code. The code is generated in such a way that each message is extended by a single digit, i.e. parity-check code is $C(n, n - 1)$. The value of extension bit depends on the values of the message bits. Parity-check can be chosen either even or odd. Even-parity check is defined as modulo 2 sum of the elements, i.e. the number of '1'-s is even in each code word while odd-parity check is its inverse. It is easy to see that the Hamming distance is 2, i.e. if one bit of any code word is damaged at an arbitrary position, the parity becomes erroneous. By damaging of two bits, however, parity (even or odd) is restored and an *other* valid code word is received and the error is not detected. This is obvious since the Hamming distance is 2, the single parity-check code is able to detect just one error (see theorem 7.1.).

7.4.3. Two Dimensional Parity-Check Code

Let us arrange the k bits of a message into a u p x q matrix**,** i.e. $k = pq$. Extending each row and each column by a parity-check digit, a (p + 1) x (q + 1) matrix is obtained, provided the bottom right element of the matrix is also defined somehow. Let us use this element as the 'parity-check of parities', i.e. a parity-check digit for bits standing in last row and the last column, respectively. It is easy to show that this digit is the same for the rows as for the columns.

The code distance is 4 which can be proved as follows: Suppose we have an even-parity check case and a u message matrix containing just one '1'. In this case, the corresponding row and column parity bits must also be '1', same as the bottom right parity bit. That is, altogether four '1'-s are in the code, thus the distance from the all-zero code word is 4 (Hamming distance is computed by comparison of the corresponding matrix co-ordinates of the two code words).

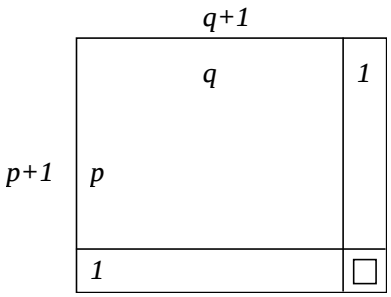


Figure 7.3 Two-Dimensional Parity Check Code

For any codeword, the Hamming distance of two arbitrary codewords is equal to the number of non-zero elements found in their difference. Furthermore, for linear codes, the difference of two codewords is also a codeword. Consequently, minimum distance between codeword pairs is equal to the number of 1s in the non-zero code word with the minimum

number of 1s. Since no such a non-zero codeword exists which contains less than four 1-s, the code distance is really 4.

7.4.4. Hamming Code

Keeping in mind that the syndrome is computed as $s = eH^T$ (see Fig. 7.2.), syndrome of the error vector e = (000...1...0) containing just a one '1' at its i^{th} coordinate is given by the i^{th} column of the matrix $\boldsymbol{H}$. Consequently, for the maximum number of syndromes (to be able to correct maximum number of errors), it is a good practice to choose the columns of the $\boldsymbol{H}$ matrix to be different. This choice guarantees that the code corresponding to the matrix $\boldsymbol{H}$ is able to correct every single error. The code corresponding to the matrix $\boldsymbol{H}$ and the generator matrix $\boldsymbol{G}$ can be easily obtained by constructing the matrix $\boldsymbol{H}$ as systematic (7.6.) and then computing (7.7.).

Example 7.6.: According to the construction rules described above, for the C(7,4) code which is able to correct single errors, let us choose the following matrix $\boldsymbol{H}$:

$$\boldsymbol{H} = \begin{pmatrix} 1101100 \\ 1011010 \\ 0111001 \end{pmatrix}$$

The code length is $n = 2^3-1 = 7$ so that $\boldsymbol{H}$ has 7 different columns each containing $n - k = 7 - 4 = 3$ bit long non-zero binary vector. The generator matrix $\boldsymbol{G}$, corresponding to this code is

$$\boldsymbol{G} = \begin{pmatrix} 1000110 \\ 0100101 \\ 0010011 \\ 0001111 \end{pmatrix}$$

The resulting code is a (7, 4) Hamming code. Notice that this code is an optimum from a certain point of view. Namely, if the task is to construct a single-error correction code with $n = 7$ then there is no such a code among them whose size is greater than $2^4 = 16$. Of course, this construction can be generalized for parameters ($n = 2^r-1$, $k = 2^r -1 - r$) where $r \geq 3$ is an integer.

7.5. Cyclic Codes

7.5.1. Basic Terms

Definition: a linear code is cyclic, if any cyclic shift of any of the codewords results in a codeword, too. The term *cyclic shift* is explained by the following example. Let us have a series of c_i, $i = 0, ..., n-1$ elements (the length of the series is n):

$$c = (c_{n-1}, c_{n-2}, c_{n-3}, ...c_3, c_2, c_1, c_0).$$

Cyclic shift to left by two results then in

$$\bar{c} = (c_{n-3}, c_{n-4}, ... c_3, c_2, c_1, c_0, c_{n-1}, c_{n-2})$$

The following discussion is limited to binary numbers that is to series consisting only of binary digits as elements: $c_i \in (0,1)$. Repeating the previous example e.g. for a 9-bit series, if the original code is (100111000) then after a two-step left cyclic shift we have (011100010).

The usage of cyclic codes is motivated by several factors:
- an *n*-bit long shift register with feedback loop can store *n* codewords, each of the length *n*,
- it can be shown that both the encoding and the syndrome generation can be performed by shift registers having apropriate loop back,
- cyclic codes can be well handled in mathematical way, too,

Keeping in mind the definition of the cyclic code, it is important to note, however, that the cyclic shifts of one codeword usually do not generate all the codewords of a cyclic code.

7.5.2. Mathematical Representation of Cyclic Shift

Let us assign a polynomial to a code word *c* of a cyclic code as follows:

$$c = (c_{n-1}, c_{n-2}, ... c_2, c_1, c_0)$$
$$\downarrow$$
$$c(x) = (c_{n-1}x^{n-1} + c_{n-2}x^{n-2} +,... + c_2x^2 + c_1x + c_0) \qquad (7.10)$$

This assignment restores the position of an element by the order of the *x* value, e.g. the coefficient of x^j (c_j) is the $(j + 1)^{th}$ element of the series counted from the right side.

A code word containing *n* elements is decribed by a polynomial of $(n - 1)^{th}$ order. For instance, using polynomials of 5^{th} order, the $c(x) = x^5 + x$ polynomial describes the code word (100010) while the $c(x) = x^4 + x^3 + 1$ polynomial describes the code word (011001).

Note that the knowledge of the maximum order of the polynomial is important, without that the leftmost 'zero' element(s) would not be recognized. In the following, we show that the polynomial desribing the code word obtained by *k* left shifts, a series of n elements decribed by a $c(x)$ polynomial can be determined as follows:

$$\bar{c}(x) = (x^k c(x)) \text{ -mod-} (x^n + 1) \qquad (7.11)$$

(Read as: $x^k c(x)$ modulo $(x^n + 1)$). According to the rules of modulo-polynomial-algebra, the above expression is the remainder of the $(x^k c(x))$ polynomial after being divided by the $(x^n + 1)$ polynomial, while the result must not contain negative exponent and rules of modulo-2 algebra have to be used for the binary coefficients.

Example 7.7.: A single left shift performed on c = (1100). The corresponding polynomial is $c(x) = x^3 + x^2$ thus $xc(x) = x^1c(x) = x^4 + x^3$. For the polynomial division, it is a good practice to present the zero-coefficient members, too:

$$(1x^4 + 1x^3 + 0x^2 + 0x^1 + 0x^0) : (x^4 + 1) = 1$$
$$1x^4 \qquad\qquad 1x^0$$
$$\text{---------------------}$$
$$0x^4 + 1x^3 + 0x^2 + 0x^1 + 1x^0$$

Note: because of the mod-2 algebra applied for the coefficients (0-1), the result of the division is +1. The division can not be continued without obtaing quotient with negative exponent thus the remainder is :

$$\bar{c}(x) = 1x^3 + 0x^2 + 0x^1 + 1x^0 = x^3 + 1,$$

Indeed, the corresponding code word, $\bar{c}$ = (1001) is the one, obtained by a one-step left shift of the original c = (1100) word.

First, let us prove the rule given in (7.11.) for $k = 1$. If

$$c(x)=c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ... + c_0 \quad (7.12)$$

then

$$xc(x)= c_{n-1}x^{n} + c_{n-2},x^{n-2} + ...+ c_0x \quad (7.13)$$

Dividing this by $(x^n + 1)$, the result will be c_{n-1} and the remainder

$$\bar{c}(x) = xc(x) + c_{n-1}(x^n + 1) \quad (7.14)$$

In ordinary polynomial division the remainder is $xc(x)-c_{n-1}(x^n+1)$ but now the coefficients are handled according to the mod-2 algebra thus '+' has to be written instead of '-' and this shall always be done in the following.

Writing the $\bar{c}(x)$ in detail and grouping reasonably the polynomial members:

$$\bar{c}(x)=(c_{n-1}+ c_{n-1})x^n+ c_{n-2}x^{n-1} +.....+ c_0x + c_{n-1}$$

In this expression, the coefficient of x^n is zero (mod-2!) thus

$$\bar{c}(x)=c_{n-2}x^{n-1} + ...+ c_0x + c_{n-1} \quad (7.15)$$

and the corresponding

$$\bar{c} =(c_{n-2},..... c_0, c_{n-1}) \quad (7.16)$$

word is really a one-step left-shifted equivalent of the original word given in (7.12.).
So the rule (7.11.) is proved in general sense for k = 1. Since the procedure can be applied for the resulting shifted word, too, the rule is proved for any value of k > 1!
The cyclic shift described by equation (7.11.) is frequently written as

$$\bar{c}(x) = \text{rem}\ \frac{x^k \cdot c(x)}{x^n + 1} \quad (7.17)$$

where the 'rem' refers to remainder and its meaning is the same as in equation (7.11.).

7.5.3. Basic Theorem of Cyclic Codes

Theorem 7.3.: every cyclic code (*n*, *k*) is unabiguously described by a $g(x)$ generator polynomial which has the order (*n-k*) and divides the $(x^n + 1)$ polynomial without a remainder.
Note: since $(x^n + 1)$ can usually be divided by more than one polynomial of the order (*n - k*), several cyclic codes can be assigned to the (*n, k*) number pair.
Since a code is a set of codewords, according to the theorem, the generator polynomial determines all (2^k) codewords of the code. To see this, we make use of the fact that the cyclic codes are a subclass of the linear codes. Linear code is unambiguously defined by its generator matrix. Let us remind that the rows of the generator matrix are mutually independent codewords, linear combinations of which generates the code (i.e. the full set of codewords).

Let us show that out of a $g(x)$ polynomial, a G generator matrix can be constructed! First, let's see the non-systematic case. Since the rows of G have n elements they can be described by polynomials of at most (n-1)th order.

Let us choose the polynomial describing the lowermost row of the generator matrix as $g_k(x) = g(x)$, then continuing upwards with $g_{k-1}(x) = xg(x)$, $g_{k-2}(x) = x^2g(x)$ ..... $g_1(x) = x^{k-1}g(x)$. Notice that with this choice, each row of the generator matrix is a one-step cyclic shift of the previous one. Since the order of g(x) is exactly (n-k), the resulting generátor-matrix is of the following form:

$$\mathbf{G} = \left[\begin{array}{cccc|c} 1 & g_{n-k-1} & \cdots\cdots & g_0 & 0\cdots\cdots0 \\ & & & & \\ 0 & \cdots\cdots & 1 & & 0\ 0 \\ 0 & \cdots\cdots & 0\ 1 & g_{n-k-1} & g_0\ 0 \\ 0\ 0 & \cdots\cdots & 0\ 0\ 1 & & g_{n-k-1}\cdots g_1\ g_0 \end{array}\right]$$

k n-k

(Here $g_i \in (0,1)$, is the coefficient of the x^{i+1} member of the generator polynomial.)
It can be seen that the rows are linearly independent thus the result is really a generator matrix! It is a matrix, each rows of which (when written polynomial form) can be divided by g(x). Consequently, every linear combination of any of the rows of the generator matrix can also be divided by g(x). In other words, every codeword (more precisely: the corresponding code polynomial) can be given as

$$c(x) = g(x)\, q(x) \qquad (7.18)$$

where $q(x)$ is the quotient. Since $c(x)$ is at most of the order (n-1) and the order of the $g(x)$ is (n-k), $q(x)$ can be at most of the order of (k-1). Using binary coefficients, the number of different $q(x)$ quotients is 2^k, which –being multiplied by $g(x)$- produce exactly 2^k of different $c(x)$, i.e. the whole set of code polynomials is obtained thus $g(x)$ fully determines the code.
Let us continue the discussion with the systematic code, in which the left partition of the generator matrix is a k x k unity matrix. The last (k-th) row of the generator matrix can be the $g_k(x) = g(x)$ polynomial again. The (k-1)-th row depends on the value of g_{n-k-2} in $g(x)$. If $g_{n-k-2} = 0$ then $g_{k-1}(x) = xg(x)$, if $g_{n-k-2} = 1$ then $g_{k-1}(x) = (x+1)g(x)$ is the proper choice to have unity matrix the left partition. For better understanding, let us see the following generator matrix:

$$\mathbf{G} = \left[\begin{array}{cc|ccc} & & & & \\ & & & & \\ 0\cdots\cdots & & 1\ 0 & g_{n-k-2} & \\ 0\cdots\cdots & & 0\ 1 & g_{n-k-1}\cdots & g_0 \end{array}\right] \begin{array}{l} \\ \\ \leftarrow \{ \text{xg(x) or} \\ \leftarrow \ \text{(x+1)g(x)} \\ \leftarrow \ \text{g(x)} \end{array}$$

k n-k

If we continue the following procedure, the (*k-i*)th row of the generator matrix (*i* = 0, 1, ... (*k*-1)) is either $g_{(k-i)}(x) = g_{(k-i+1)}x$ or $g_{(k-i)}(x) = g_{(k-i+1)}(x+1)$. Thus generally: $g_{(k-i)}(x) = g(x)x^p(1 + x)^q$, where $(p + q) = i$.

It is important to remark that both the rows of the generator matrix and their linear combinations are divisible by *g(x)*, i.e. *g(x)* is the divisor of all codewords. Therefore, equation (7.18.) and all its consequences apply to the generator matrix, too. The *g(x)* polynomial thus fully determines both the non systematic and the systematic codes.

As the last step, we prove that the code determined by the g(x) polynomial is a cyclic one. For this we show that if c(x) is a code polynomial then

$$\bar{c}(x) = \mathrm{rem}\frac{x \cdot c(x)}{x^n + 1}$$

is a code polynomial, too, i.e. it is divisible by g(x). Earlier we have seen that

$$\mathrm{rem}\frac{x \cdot c(x)}{x^n + 1} = x \cdot c(x) + c_{n-1}\left(x^n + 1\right), \qquad (7.19)$$

furthermore *c(x)* and *xc(x)* is divisible by *g(x)*, too. Expression in eq. (7.19.) is divisible by *g(x)*, if (x^n+1) is divisible, too. With that we came to the conclusion that for the cyclic property, (x^n+1) must be divisible by *g(x)*.

7.5.4. Cyclic Code Generation

Let us have a *k* bit long message described by the polynomial *u(x)* (order of which is *k*-1 at maximum) and the generator polynomial *g(x)*. For the non systematic case, the code polynomial *c(x)* can be determined trivially by equation (7.18.)

$$c(x) = g(x)u(x).$$

In the case of systematic code, the first *k* positions of the codeword are occupied by the message, i.e. by $x^{n-k}u(x)$ followed by the parity segment:

$$c(x) = x^{n-k}u(x) + p(x) \qquad (7.20)$$

Since the code polynomial must be divisible by *c(x)*,

$$\mathrm{rem}\frac{x^{n-k} \cdot u(x) + p(x)}{g(x)} = 0.$$

and because maximum order of *p(x)* and that of the *g(x)* is (*n-k*-1) and (*n-k*), respectively,

$$p(x) = \mathrm{rem}\frac{x^{n-k} \cdot u(x)}{g(x)} \qquad (7.21)$$

With that we obtained a simple rule to determine the parity segment. The whole codeword is then

$$c(x) = x^{n-k} \cdot u(x) + rem\frac{x^{n-k} \cdot u(x)}{g(x)} \qquad (7.22)$$

7.5.5. Cyclic Code Syndrome Calculation

Let us restrict ourselves only to the systematic cyclic codes. It is reasonable to divide the *n* elements of the received polynomial *v(x)* to a *k* element long predicted message segment and to a (n-k) element long predicted parity segment:

$$v(x) = \bar{u}(x) + \bar{p}(x) \tag{7.23}$$

The term 'predicted' refers to the fact that either the $x^{n-k}u(x)$ message or the *p(x)* parity segment of the code polynomial *c(x)* (or both) could be damaged during the transmission.

In decomposion given by eq.(7.23.), the order of $\bar{p}(x)$ is max. (*n-k*-1), order of $\bar{u}(x)$ is max. (*n*-1), min. (*n-k*). As it was earlier discused, the 'received' parity $\tilde{\tilde{p}}(x)$ is to be created from the received message and this has to be compared with 'arrived' parity $\bar{p}(x)$. Syndrome is then obtained as the result of the comparison of these two:

$$s(x) = \tilde{p}(x) + \tilde{\tilde{p}}(x) \tag{7.24}$$

Parity of the received word has to be created exactly as it was done at the encoder:

$$\tilde{p}(x) = rem\frac{\tilde{u}(x)}{g(x)} \tag{7.25}$$

Here $\tilde{u}(x)$ starts already with the (*n*-1)th order element since the received word has *n* elements. Thus it is not needed to multiply by x^{n-k}, as it had to be done in equation (7.20.). Since the order of $\tilde{\tilde{p}}(x)$ is not greater than (*n-k*-1), we can write

$$\tilde{\tilde{p}}(x) = rem\frac{\tilde{\tilde{p}}(x)}{g(x)}$$

Substituting into equation (7.24.)

$$s(x) = \tilde{p}(x) + \tilde{\tilde{p}}(x) = rem\frac{\tilde{u}(x)}{g(x)} + rem\frac{\tilde{p}(x)}{g(x)}$$

Making use of the disjunct feature of the polynomial orders

$$s(x) = rem\frac{\tilde{u}(x) + \tilde{p}(x)}{g(x)} = rem\frac{v(x)}{g(x)}$$

Thus the syndrome is simply given by the remainder of the division of received polynomial *v(x)* by *g(x)*.

7.5.6. Résumé

The *C(n, k)* cyclic code is determined by g(x) generator polynomial, *g(x)* is the divisor of $(x^n + 1)$. We restricted to systematic codes only. Each message word *u(x)* is of the (*k* -1)th order and is encoded in

$$c(x) = u(x)x^{n-k} + rem\frac{u(x)x^{n-k}}{g(x)}$$

code polynomial. During the transmission, ***c(x)*** can become faulty, e.g. it can change to a ***v(x)***. The syndrome of the error can be determined from the received ***v(x)*** as

$$s(x) = rem\frac{v(x)}{g(x)}$$

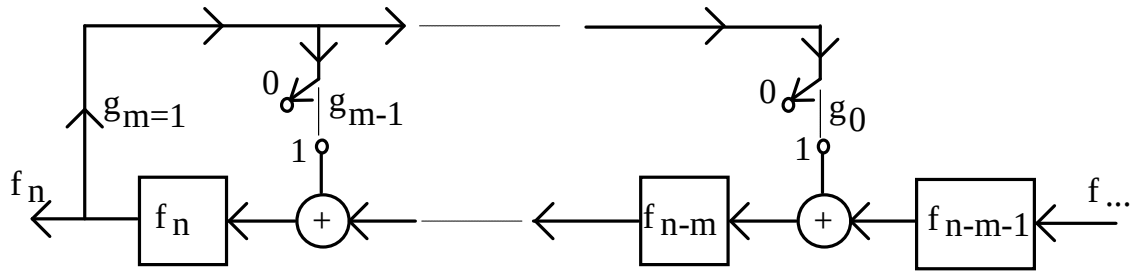
7.5.7. Practical Example of Cyclic Codes

ITU (International Telecommunication Union) recommends a following code: first 4 bits are the service bits, next 240 or 480 or 960 bits is the message and the last 16 bits are 'protective' bits, guarding the previous ones, generated by the $g(x) = x^{15} + x^{12} + x^5 + 1$ polynomial. The code is used for error detection ($s(x) \neq 0$). The code guarantees to detect any odd number of errors and any error sequence shorter than 16 bits and many other combinations of errors.

7.5.8. Feed-back shift-register

A shift register with feed-back according to divisor polynomial $g(x) = 1 \cdot x^m + g_{m-1} \cdot x^{m-1} + ...+ g_0 x^0$ performs a polynomial division (in case of binary polynomials). Before the shift, the dividend is in the shift register, after the shift the result of the division appears at the output and the new content of the shift register is the remainder of the division (see Figure 7.4.)

Before shift:



After shift:

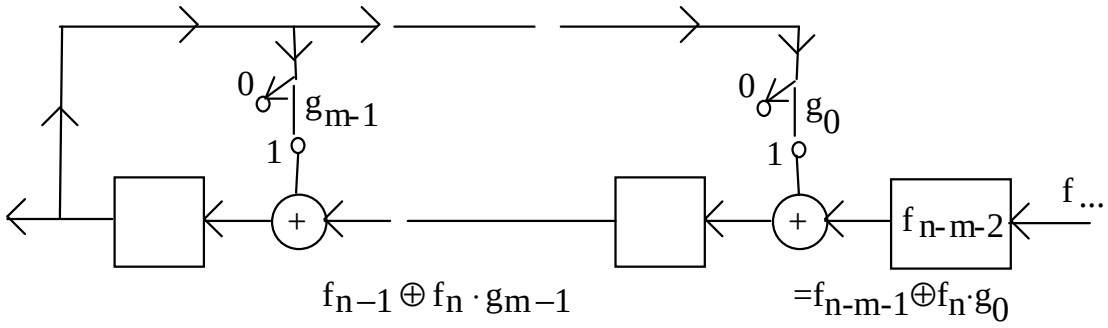


Figure 7.4. Cyclic Code Generation with Feed-Back Shift Register

Exercises:

1. Let the generator matrix of a linear code be $\mathbf{G} = \begin{pmatrix} 010111 \\ 111010 \\ 110100 \end{pmatrix}$.

Determine: a) the code words,
b) the systematic generator matrix,
c) the error correction ability,
d) the syndrome decoding table of the code.

2. Let us examine the (7,4) Hamming code. Is it true that if two errors have occurred at the transmission of a code word there is always a code word which differs from the received word in only one bit? Give an explanation!
3. Compute the probability of error for the decoding of (7,4) Hamming code if the probability of faulty bits at the channel input is p and the bit failures are independent.

Questions

1. How can the error ratio of the received bits be improved when transmitting information through a noisy communication channel?
2. How can error detection be used to improve quality of the transmission?
3. What is the algebraic structure of a linear code?
4. How many generator matrices can be assigned to a linear code?
5. List the methods used for decoding!
6. What would be the procedure of error correction for the two-dimensional parity code?
7. Determine a code which is able to correct one error by constructing its parity matrix!

References

[1] Győrfi L.-Vajda I.: A hibajavító kódolás és a nyilvános kulcsú titkosítás elemei. Jegyzet, 1991
[2] Vajda I.: Hibajavító kódolás mûszaki alkalmazásai. Jegyzet, Mérnöki Továbbképző Intézet, 1982
[3] Fritz J.-Csiszár I.: Információelmélet. Tankönyvkiadó, 1983