Chapter 11

Error Control Codes

In digital transmission systems, an error occurs when a bit is altered between transmission and reception; that is, a binary 1 is transmitted and a binary 0 is received, or a binary 0 is transmitted and a binary 1 is received. Regardless of the design of the transmission system, there will be errors, resulting in the change of one or more bits in a transmitted block of data. Three approaches are in common use for coping with data transmission errors:

1. Error-detection codes
2. Error-correction codes, also called forward error correction (FEC) codes
3. Automatic repeat request (ARQ) protocols

All three approaches are applied to individual blocks of data, called frames, packets, or cells, depending on the protocol used for data exchange. The first two approaches use **redundancy**. In essence, for error detection and error correction, additional bits, which are a function of the data bits, are appended to the data bits by the sender. These redundant bits are used by the receiver for the purpose of error detection or correction. An error-detection code simply detects the presence of an error. Typically, such codes are used in conjunction with a protocol at the data link or transport level that uses an ARQ scheme. With an ARQ scheme, a receiver discards a block of data in which an error is detected and the transmitter retransmits that block of data. FEC codes are designed not just to detect but correct errors, avoiding the need for retransmission. FEC schemes are frequently used in wireless transmission, where retransmission schemes are highly inefficient and error rates may be high. The first two categories of controlling errors in data transmission will be discussed in this chapter.

11.1 Cyclic Redundancy Check

The cyclic redundancy check (CRC) code is an error detection code. It is used in the receiver to detect error in the received sequence. If any error is found, the receiver uses a protocol to ask the transmitter to retransmit message.

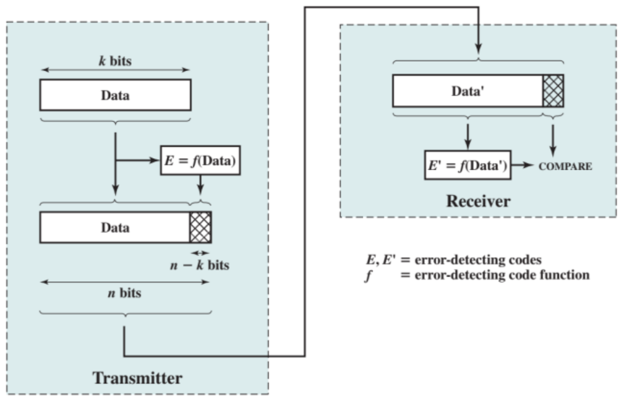


Figure 11.1 Principle of error detection codes

For a given frame of bits, additional bits that constitute an **error-detecting code** are added by the transmitter. This code is calculated as a function of the other transmitted bits. Typically, for a data block of *k* bits, the error-detecting algorithm yields an error-detecting code of *n* - *k* bits, where (*n* - *k*) 6 *k*. The error-detecting code, also referred to as the **check bits**, is appended to the data block to produce a frame of *n* bits, which is then transmitted. The receiver separates the incoming frame into the *k* bits of data and (*n* - *k*) bits of the error-detecting code. The receiver performs the same error-detecting calculation on the data bits and compares this value with the value of the incoming error-detecting code. A detected error occurs if and only if there is a mismatch.

One of the most common, and one of the most powerful, error-detecting codes is the cyclic redundancy check, which can be described as follows. Given a k-bit block of bits, or message, the transmitter generates an (n - k)-bit sequence, known as a frame check sequence (FCS), such that the resulting frame, consisting of n bits, is exactly divisible by some predetermined number. The receiver then divides the incoming frame by that number and, if there is no remainder, assumes there was no error. There are three ways in which CRC can be implemented: modulo 2 arithmetic, polynomials, and digital logic. For the sake of simplicity, we will discuss the modulo 2 arithmetic-based CRC technique.

Before starting discussion on CRC, let's first have a review of modulo 2 arithmetic. Modulo 2 arithmetic uses binary addition with no carries, which is just the exclusiveOR (XOR) operation. Binary subtraction with no carries is also interpreted as the XOR operation: For example

$$\begin{array}{r} 1111 \\ +1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 1111 \\ -0101 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 11001 \\ \times 11 \\ \hline 11001 \\ 11001 \\ \hline 101011 \end{array}$$

Suppose that we want to send the bit sequence M=101011 having length m= 6. We will not send the message bit directly. Instead, M will be accompanied by some extra bits, which we call frame check sequence (FCS), R. Then, we will send both M and R. The combination of M and R is called codeword because it does not represent the original message bits only. The codeword can also be called as the transmitted bit sequence, T. Now a question is how we can generate the FCS. To generate the FCS, we need another sequence, named pattern sequence, P, which is known to both the sender and receiver. The pattern sequence is also known as generator sequence. The length of the pattern sequence depends on the FCS length. Longer the FCS length, better is the error detection accuracy. To generate a FCS of k bits, we need a pattern sequence of k+1 bits. The steps of the FCS generation and the codeword generation using the generated FCS is given below:

1. Decide how many FCS bits, *K*, you are going to use.
2. Append K zeros at the end of the message bits to generate *M*+*K* bits long sequence *S*.
3. Select a *K*+*1* bits long pattern sequence, *P*.
4. Divide the sequence *S* by the pattern sequence *P* to find the *K* bits of the remainder, R.
5. Remove the appended zeros from S and append the calculated remainder *R*. Thus, the *N* bits message bits and *K* bits remainder constitutes the transmitting sequence, *T*.

Once we have generated the transmitting sequence or codeword, *T*, we will send it to the destination. Please notice that we are sending M+K bits, NOT the message M-bits only. The K bits are extra bits. Upon reception of the codeword, the receiver will check whether the received sequence is a valid codeword or erroneous sequence. To check this, the receiver follows the following steps:

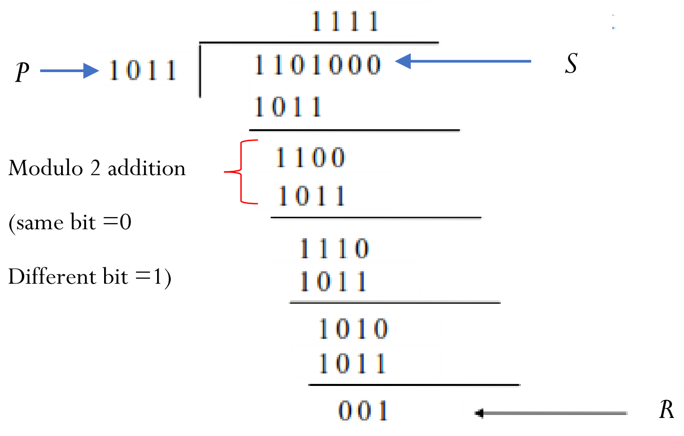
1. At the destination, the received sequence, T', is divided by the same patter sequence, *P*.
2. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
3. A remainder indicates that the data unit has been damaged on the way and therefore must be rejected.

Let's now see an example of codeword generation at the sender and validity checking of the received sequence as a codeword at the receiver. Before starting the example, we will discuss a technique of representing a binary sequence using a polynomial. Although such polynomial representation is not essential, it is quite common to use in error control coding. Suppose that we have a message polynomial, $M = X^5 + X^3 + X^1 + 1$. How can we find the corresponding bit sequence? Since the power of the first term of the polynomial is 5, so there are 5+1 bits in the sequence: one for each term of the polynomial. Since the largest term is X^5, the possible terms are X^5, X^4, X^3, X^2, X^1, $X^0 = 1$. If any of this term exists in the polynomial, there will be a 1 in the corresponding position in the binary sequence. Since X^5 exist in the sequence, the left-most bit will be 1, so, M=1. X^4 does not exist, so the next bit is 0; M=10. X^3 exists, thus, the next term is 1; M=101. X^2 does not exist, the next bit is 0; M=1010. The next two terms X^1 and X^0 (that is, 1) exist, so the last two terms will be 11. Thus, M=101011. Similarly, the bit sequence M=1001110 corresponds to the polynomial $M = X^6 + X^3 + X^2 + X^1$. That's all about the polynomial representation of a bit sequence.

Generate FCS if the message polynomial and generator polynomial are $X^3 + X^2 + 1$ And $X^3 + X + 1$, respectively.

After converting the message polynomial and generator polynomial, we get M=1 1 0 1 and P = 1 0 1 1. Let's follow the steps of the FCS generation discussed before.

1. Decide the FCS length. Since the patter sequence length is 4, the FCS length must be 4-1= 3. That is, k=3.
2. Append 3 zeros at the end of the message sequence. Thus, S=1101000
3. Select the pattern sequence. Since we have been given the pattern sequence in question, we do not need to select it. P= 1 0 1 1.
4. Divide S by P using modulo 2 arithmetic.



It would be pertinent to mention here that it does not matter whether the dividend is greater than divisor. However, the dividend length must be at least equal to the length of the divisor.

5. Pick the K=3 bits remainder, R, from the right side. This time, the remainder is 0 0 0 1. We have picked up 3 bits from the right side; hence the FCS, R = 0 0 1. Let's now form the codeword or transmitting sequence, T = [M R] =[1 0 1 1 0 0 1]. This sequence will be sent to the destination.

When the sequence, T = [1011001] is sent to the destination, the sequence may either reach the destination intact or it may get contaminated in the communication channel (the path through which the signal travels to the destination). It is up to the receiver to detect whether there is any error in the received sequence. Pertinently, we should remember that the pattern sequence is known to both sender and receiver. We consider both scenarios:

(a) the receiver receives the intact message sequence
(b) the receiver receives the erroneous sequence.

We will see both scenario one by one.

(a) Correct Reception Scenario:

The receiver follows the steps we discussed before:

i) Divide the received sequence by the pattern sequence.

```
1011 | 1 1 0 1 0 0 1
       1 0 1 1
       ----------
         1 1 0 0
         1 0 1 1
       ----------
           1 1 1 0
           1 0 1 1
       ----------
             1 0 1 1
             1 0 1 1
           ----------
             0 0 0 0
```

ii) Check whether there is any remainder. Since there is no remainder, there is no error in the reception. That is, the received sequence is the bit sequence sent from the sender. There was no alteration of any bit over the channel.

b) Let’s now consider a scenario where one of the transmitted bits get altered on the channel. Instead of receiving the sent sequence, 1 1 0 1 0 0 1, the receiver has received 1 0 0 1 0 0 1. It is seen that the second bit from left has got altered from 1 to 0. How can the receiver know that the received sequence 1 0 0 1 0 0 1 is correct or not? To know this, it follows the same procedure what we discussed before.

i) Divide the received sequence, 1 0 0 1 0 0 1, by the pattern sequence.

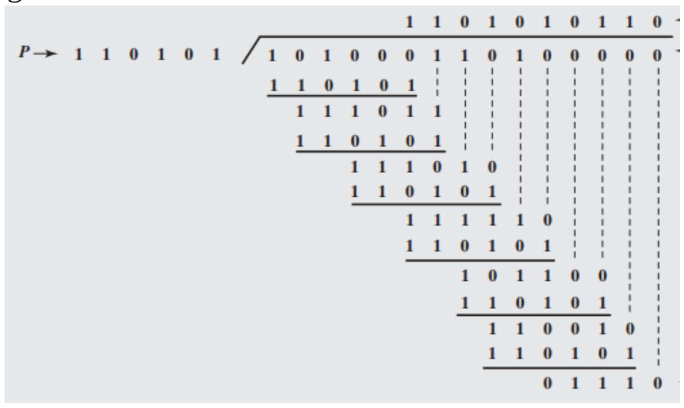
```
1011 | 1 0 0 1 0 0 1
       1 0 1 1
       ----------
           1 0 0 0
           1 0 1 1
       ----------
               1 1 1
```

ii) Check whether there is any remainder. Yes, the remainder exists. Thus, there is an error in the received sequence. The receiver now requests the sender to resends the sequence.

Let’s see another example. Suppose that the message and pattern sequences are given as follows: Message *M* = 1010001101, Pattern *P* = 110101. Generate the codeword.

1. Since the pattern sequence length is 6, the FCS length is 6-1= 5. That is, K=5.
2. Append 5 zeros at the end of the message sequence. Thus, S=101000110100000
3. Select the pattern sequence. Since we have been given the pattern sequence in question, we do not need to select it. P= 110101

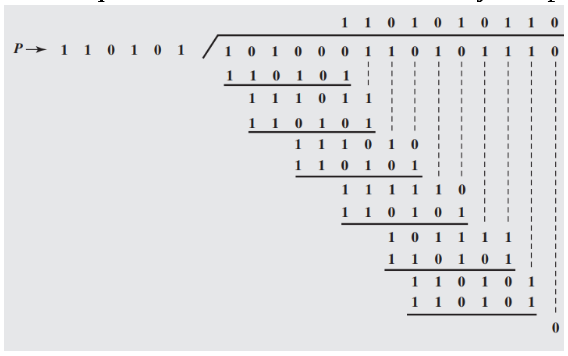
4. Divide S by P using modulo 2 arithmetic.



5. Pick the k=5 bits remainder, R, from the right side. This time, the remainder is 0001110. We have picked up 5 bits from the right side; hence the FCS, R = 01110. Let's now form the codeword or transmitting sequence, T = [M R] =[101000110101110]. This sequence will be sent to the destination.

Suppose that the receiver receives the correct sequence, 101000110101110. But it needs to be sure that it has received the correct sequence. To do this, it follows the validation procedure:

i) Divide the received sequence, 101000110101110, by the pattern sequence.



i) Check whether is any remainder. Since there is no remainder, there is no error in the reception. That is, the received sequence is the bit sequence sent from the sender. There was no alteration of any bit over the channel.

Exercise

1. Detect whether the received sequence 101110101 is error free if the pattern sequence is 1010.

11.2 Linear Block Code

In the previous section, we discussed CRC which is applicable where retransmission of an erroneous frame is possible. There are some application, such as wireless transmission, where the retransmission is practically very inefficient to implement. In such scenario, it is necessary to correct the erroneous frame by the receiver itself instead of asking the sender to resend the frame. Since CRC can only detect error, we need another technique that can detect and correct error. An example of such kind of code is linear block code which can correct error. In the linear block code, if any two codewords are added together, the result will also be a codeword. Next, we will discuss the codeword generation, which will be followed by error detection at the receiver and error correction.

Before going to the main discussion, let's see which notations we are going to use. Please note that we are going to use different notations in this discussion compared to the CRC code. The message sequence and its length are represented by M and k, respectively, while the redundant sequence, which is derived from the message sequence, and its length are represented by Q and q. The codeword will be generated combining the message sequence and the redundant sequence and will be represented by N of length k+q. In short,

Message, *M*:	*k* bits long
Redundant bits, *Q*:	*q b*its long
Codeword length, *N*:	*k*+*q* bits long

Suppose that we want to send the message M= [0 1 1]. We have to generate the redundant bits Q from the message sequence. To generate the redundant bits, CRC uses a pattern sequence. However, linear block code does not use the pattern sequence; rather it uses a generator matrix, G. The size of the generator matrix is k×n, where n = k+q. G is formed by combining a parity matrix, $P_{k\times q}$, and an identity matrix, I_k. Just to remind you, an identity is a matrix which has the same number of rows and columns and the diagonal elements are 1 while the other elements are 0. Once we have both matrices, we form the generator matrix as $G = \left[P_{k\times q}\ \ I_k\right]$. For example, if the message length, k=4 and the redundant bit length, q=3 (i.e., n = k+q = 4+3 = 7). The sizes of P and I will be 4×3 and 3×3. Then, $G = [P_{4\times 3}\ \ I_4]$. However, if the message length k = 3 and the redundant bit length, q =3, the size of the P and I will be 3×3 and 3×3, respectively.

Suppose that the parity and identity matrices are as follows:

$$P_{3\times 3} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus, k = 3 and q = 3; hence, n = 3+3 = 6. In words, we will derive three redundant bits (as q = 3) from 3 message bits (as k = 3) to produce 6 transmit sequence or codeword (as n = 6). Combining P and I, we can form G,

$$G = [P_{3\times3} \quad I_3]$$

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

A linear block code is named based on the size of the generator matrix. Since the size of the generator matrix is 3×6, we call this code (n,k) = (6, 3) block code. In words, it is a linear block code that takes 3 message bits and generates 6 bits codeword. A question may arise where we will get the parity matrix. The parity matrix design is another issue which is out of scope of this discussion. In short, each column of the parity matrix determines the corresponding element of the redundant bits. The i-th redundant bit will be generated based on the i-th column of P. For example, the first element of the redundant bit sequence will be generated by those elements of the message bits for which the corresponding location of the first column of P has 1. Since first and third elements of the first column of P contains 1, so the first redundant bits will be created combining (modulo 2 addition) the first and third elements of the message bits.

To generate the codeword corresponding to the message M = [0 1 1], we have to multiply G by M.

$$C = M \times G$$

$$C = [0 \quad 1 \quad 1] \times \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$C = [\underbrace{1 \quad 1 \quad 0}_{Q} \quad \underbrace{0 \quad 1 \quad 1}_{M}]$$

The multiplication of M and G is done using modulo 2 addition. Let’s first consider the third element of C. Multiply the third column of G by the M. Instead of doing the multiplication as 0×0+1×1+1×1=10 (binary addition), we will use modulo 2 addition, 0×0⊕1×1⊕1×1 = 0⊕1⊕1=0. Why is it 0? The reason is that we have even number of 1 (two 1s), the result is 0. If we had odd number of 1s, the result would be 1. For example, consider the fifth element of C. It is 0×0⊕1×1⊕1×0 = 0⊕1⊕0=1. Following these rules, we can compute other elements of C. If we carefully observe C, we can see that last k-elements is the message sequence M. That is, the codeword consists of the message sequence and q extra digits. This is the sequence we will send to the receiver instead of the three bits message sequence.

At the destination, how can the receiver decide whether the received sequence is correct or not? If it finds any error, it will take further action to correct the error. To detect the validity of the received sequence, we need a new matrix called parity check matrix, H. Please notice that it is parity check matrix, not parity matrix only. To create H, we need the parity matrix and a different identity matrix. H is formed as

$$\mathrm{H} = \left[I_q \;\; P_{k\times q}^T\right]$$

Where $P_{k\times q}^T$ is the transpose of $P_{k\times q}$. Pertinently, the transpose of a matrix is obtained by making each row a column. For example, the transpose of

$$P_{3\times 3} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

is

$$P_{3\times 3}^T = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

.We see that the first row of $P_{3\times 3}$ becomes the first column of $P_{3\times 3}^T$ and the second row of $P_{3\times 3}$ becomes the second column of $P_{3\times 3}^T$. Let's go back to the discussion of the parity check matrix, H. Since

$$\mathrm{H} = \left[I_q \;\; P_{k\times q}^T\right]$$

$$\mathrm{H} = \left[I_3 \;\; P_{3\times 3}^T\right]$$

$$\mathrm{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

, the transpose of H is

$$H^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

It is pertinent to mention here that the parity matrix P, k and n are known to both sender and receiver. Once we have the parity check matrix in hand, we can start checking whether the received sequence is correct or erroneous.

Suppose that the received sequence is r. If it is a correct sequence, it becomes identical to the transmitted sequence C; that is, r = C. However, if the received sequence r is contaminated, r ≠ C, we have to try to correct the error. Let's first consider the scenario where r = C. How can we validate this? To check whether r is correct or not, multiply the transpose of the parity check matrix by r following the rules of modulo 2 addition (discussed before).

$$s = rH^T$$

$$s = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$s = [0 \quad 0 \quad 0]$$

The result of the multiplication is called syndrome. A question may arise why it is called syndrome. In medical science, a disease is diagnosed considering a set of symptoms (called syndrome). Similarly, we use the syndrome, calculated above, to diagnose the received sequence. It the syndrome is all-zero, there is no error in the received sequence. However, a non-zero syndrome indicates the reverse. Since we have found all-zero syndrome, we can conclude that there is no error in the received sequence.

Now consider an erroneous reception scenario. Suppose that there is an error in the received sequence. The second bit (from left side) has altered from 1 to 0. Hence r becomes [1 0 0 0 1 1] instead of [1 1 0 0 1 1]. How can the receiver detect that the received sequence is erroneous? It follows the same procedure as it has used above. It multiplies the H^T by the received sequence.

$$s = rH^T$$

$$s = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$s = [0 \quad 1 \quad 0]$$

We see that the syndrome is non-zero this time, which indicates that r $\neq$ C. Error is detected!

Since the linear block code is an error correction code, it should have the capability to correct the error. The question is how it can correct the error. To correct the error, we need the syndrome and H^T. Let's locate the syndrome s in H^T. We got s in H^T! *Eureka* ! The second row of H^T matches with s. Hence, the second element of the received sequence $r = [1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1]$ is erroneous. Then, alter the second bit. Since it is 0 now, alter it to 1. Then, $r_{corrected} = [1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1]$. Now, $r_{corrected} = C$. That's all about the error detection and correction using the linear block code.

Exercise

❖ Consider a (7, 4) code whose generator matrix is given by

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) Find all the codewords of the code
(b) Find the parity-check matrix
(c) Find the syndrome for the received vector [1 1 0 1 0 1 0]. Is it a valid codeword?