10.4 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a_0 to a_6, and the bits in the second word b_0 to b_6, we can shift the bits by using the following:

$$b_1 = a_0 \quad b_2 = a_1 \quad b_3 = a_2 \quad b_4 = a_3 \quad b_5 = a_4 \quad b_6 = a_5 \quad b_0 = a_6$$

In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

Cyclic Redundancy Check

We can create cyclic codes to correct errors. However, the theoretical background required is beyond the scope of this book. In this section, we simply discuss a category of cyclic codes called the cyclic redundancy check (CRC) that is used in networks such as LANs and WANs.

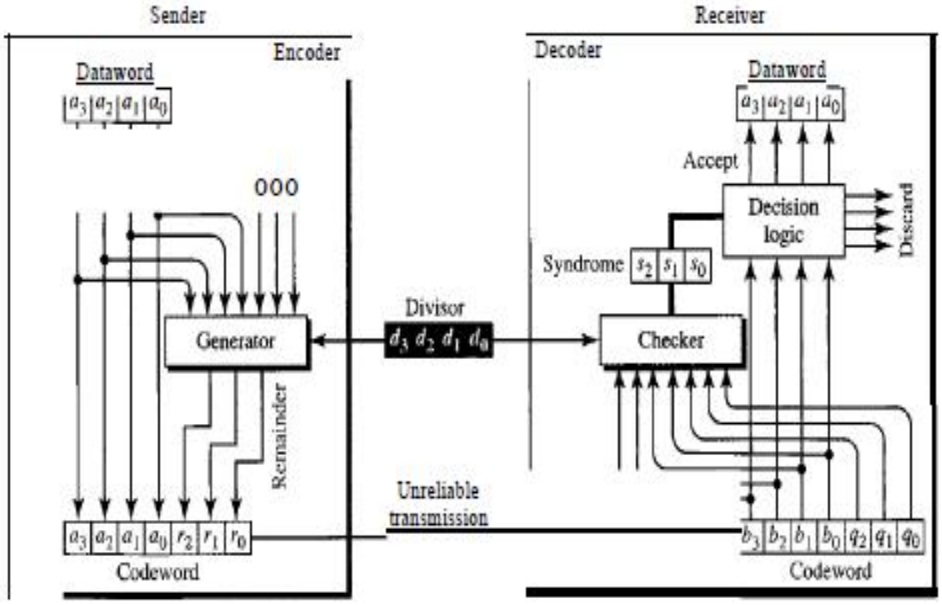
Table 10.6 shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

Table 10.6 *A CRC code with C(7, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 10.14 shows one possible design for the encoder and decoder.

Figure 10.14 *CRC encoder and decoder*



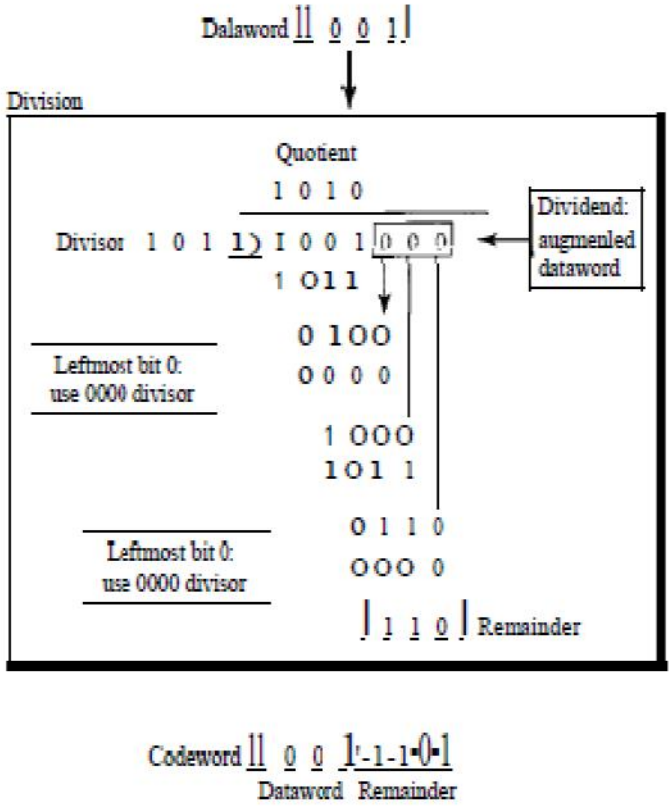
In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding $n - k$ (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size $n - k + 1$ (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient ofthe division is discarded; the remainder *(r2rl ro)* is appended to the dataword to create the codeword.

The decoder receives the possibly corrupted codeword. A copy of all n bits is fed to the checker which is a replica of the generator. The remainder produced by the checker is a syndrome of $n - k$ (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

Let us take a closer look at the encoder. The encoder takes the dataword and augments it with $n - k$ number of as. It then divides the augmented dataword by the divisor, as shown in Figure 10.15.

Figure 10.15 *Division in CRC encoder*



The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, as mentioned at the beginning of the chapter, in this case addition and subtraction are the same. We use the XOR operation to do both.

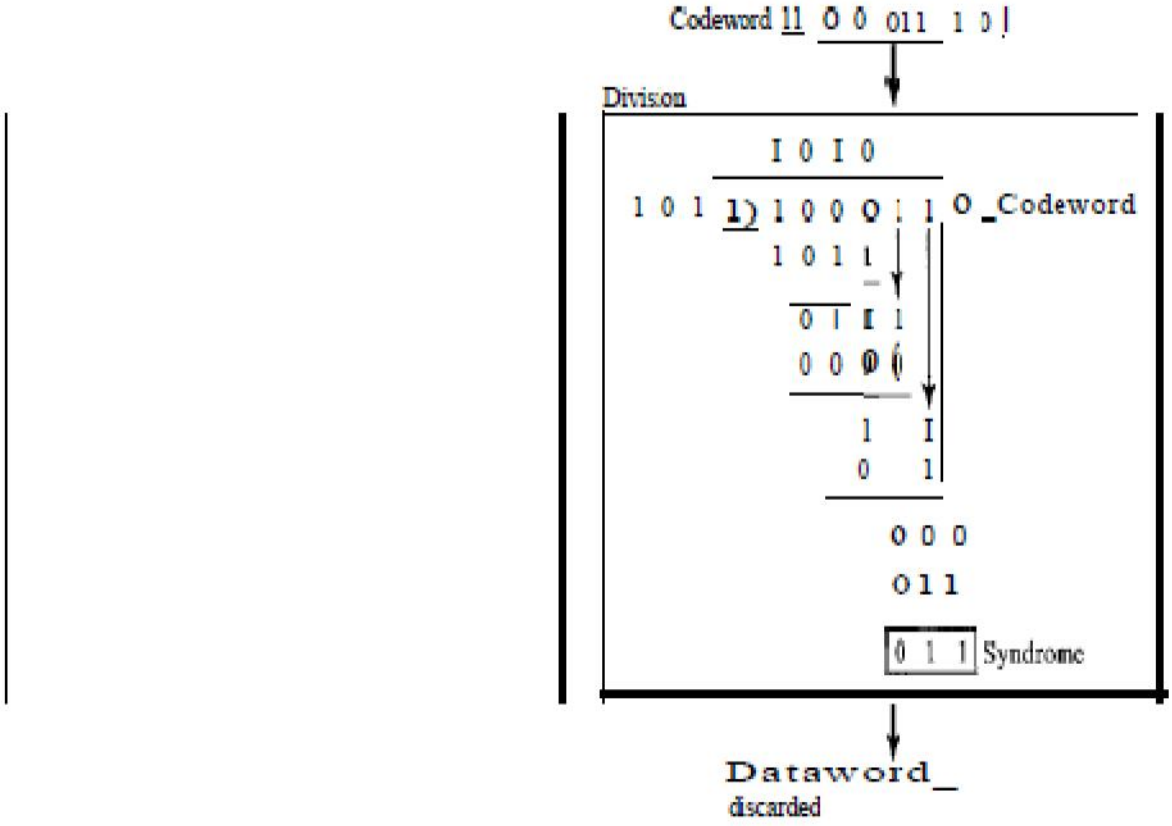
As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.

When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits *(r2' rl'* and *ro).* They are appended to the dataword to create the codeword.

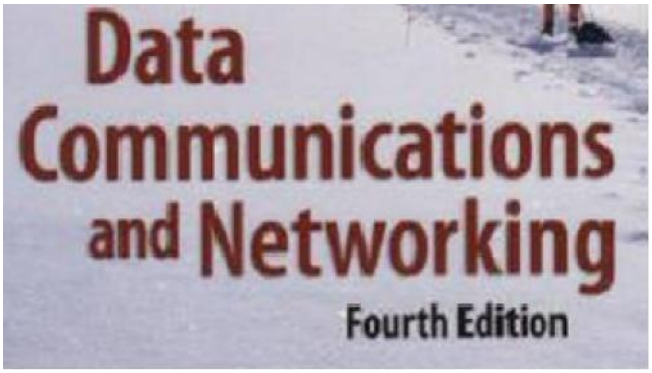
Decoder

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure 10.16 shows two cases: The left-hand figure shows the value of syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is one single error. The syndrome is not all 0s (it is 011).

Figure 10.16 *Division in the CRC decoder for two cases*



Reference:



Behrouz A. Forouzan

Expected Output:

```
sr@msritcse:~$ ./a.out

Enter data : 101

---------------------------------------
Generatng polynomial : 10001000000100001
---------------------------------------
Modified data is : 10100000000000000000
---------------------------------------
 CRC checksum is : 01010000010100101
---------------------------------------
Final codeword transmitted is : 10101010000010100101
---------------------------------------
Test error detection 0(yes) 1(no)? : 1

 CRC checksum is : 00000000000000000
No error detected


---------------------------------------
sr@msritcse:~$ ./a.out

Enter data : 101

---------------------------------------
Generatng polynomial : 10001000000100001
---------------------------------------
Modified data is : 10100000000000000000
---------------------------------------
 CRC checksum is : 01010000010100101
---------------------------------------
Final codeword transmitted is : 10101010000010100101
---------------------------------------
Test error detection 0(yes) 1(no)? : 0

Enter the position where error is to be inserted : 3

---------------------------------------
Erroneous data : 10001010000010100101

 CRC checksum is : 00010000000100001
Error detected


---------------------------------------
```