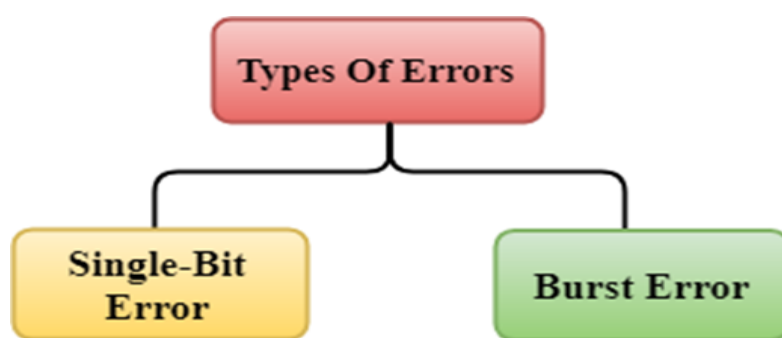


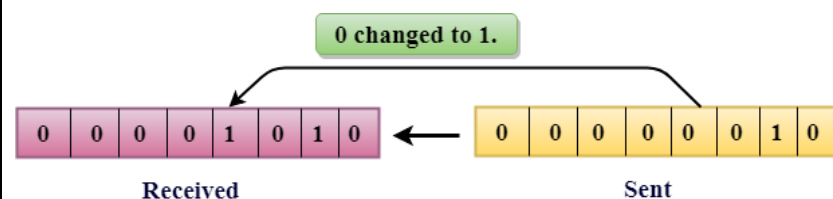
1. ERROR

- An error is a situation when the data sent by the sender and received by the receiver, but that data doesn't match the sender data.
- For example, the sender sends the 0101010 data, and the receiver receives the 1101010.
- Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Some
- applications can tolerate a small level of error. For example, random errors in audio or video transmissions may be tolerable, but when we transfer text, we expect a very high level of accuracy.

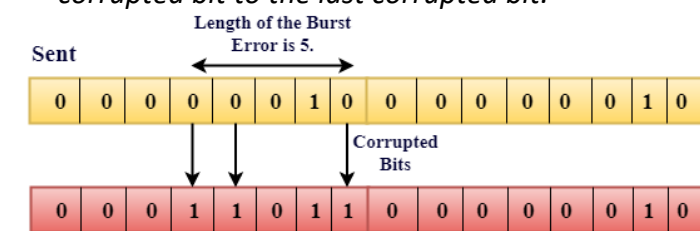
Types of Error



- The only one bit of a given data unit is changed from 1 to 0 or from 0 to 1.
- In the above figure, the message which is sent is corrupted as single-bit, i.e., 0 bit is changed to 1.
- Single-Bit Error does not appear more likely in Serial Data Transmission. For example, Sender sends the data at 10 Mbps, this means that the bit lasts only for 1 μ s and for a single-bit error to occur, a noise must be more than 1 μ s.
- Single-Bit Error mainly occurs in Parallel Data Transmission. For example, if eight wires are used to send the eight bits of a byte, if one of the wires is noisy, then single-bit is corrupted per byte.



- The two or more bits are changed from 0 to 1 or from 1 to 0 is known as Burst Error.
- The Burst Error is determined from the first corrupted bit to the last corrupted bit.

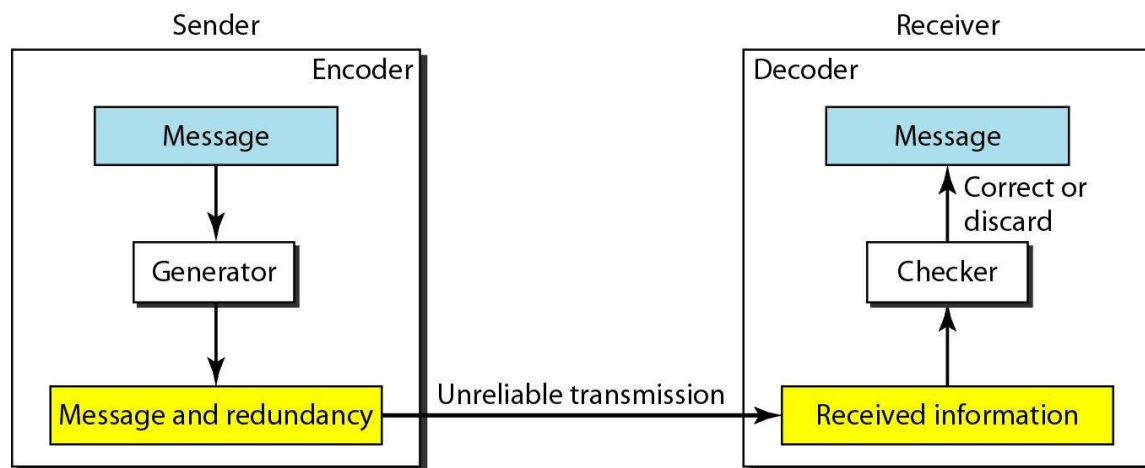


Received

- The duration of noise in Burst Error is more than the duration of noise in Single-Bit.
- Burst Errors are most likely to occur in Serial Data Transmission.
- The number of affected bits depends on the duration of the noise and data rate.

2. Redundancy

1. To detect or correct errors, we need to send extra (redundant) bits with data.
2. These redundant bits are added by the sender and removed by the receiver



3. Detection Versus Correction :

The correction of errors is more difficult than the detection. In error detection, we are looking only to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of errors. A single-bit error is the same for us as a burst error.

In error correction :

we need to know the exact number of bits that are corrupted and more importantly, their location in the message. The number of the errors and the size of the message are important factors. If we need to correct one single error in an 8-bit data unit, we need to consider eight possible error locations.

1. There are two main methods of error correction.

1. Forward error correction :
is the process in which the receiver tries to guess the message by using redundant bits. This is possible, if the number of errors is small.
2. Correction by retransmission :
is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message

4. coding

Redundancy is achieved through various coding schemes.
We can divide coding schemes into two broad categories:

1. block coding
2. convolution coding

2. NOTE—

1. In modulo-N arithmetic, we use only the integers in the range 0 to N -1, inclusive.
2. XORing of two single bits or two words

$$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0$$

a. Two bits are the same, the result is 0.

$$0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1$$

b. Two bits are different, the result is 1.

	1	0	1	1	0
\oplus	1	1	1	0	0
	0	1	0	1	0

c. Result of XORing two patterns

5. BLOCK CODING

*In block coding, we divide our message into blocks,
each of **k bits**, called **datawords**.*

*We add **r** redundant bits to each block to make the length **$n = k + r$** .*

*The resulting **n -bit** blocks are called **codewords**.*

1. With k bits, we can create a combination of 2^k data words;
2. with n bits, we can create a combination of 2^n code words.
3. The block coding process is one-to-one i.e the same dataword is always encoded as the same codeword.
4. Since $n > k$, the number of possible code words is larger than the number of possible data words.
5. This means that we have $2^k - 2^n$ codewords that are not used.



2^k Datawords, each of k bits



2^n Codewords, each of n bits (only 2^k of them are valid)

Example

$k = 4$	$n = 5$
$2^k = 16$ datawords	$2^n = 32$ codewords
Therefore :	
<ul style="list-style-type: none">• 16 out of 32 codewords are used for message transfer• the rest are either used for other purposes or unused.	

Error Detection :

1. The sender

- creates codewords out of datawords
- by using a generator that applies the rules and procedures of encoding.

2. Each codeword sent to the receiver may change during transmission.

3. If the received codeword is the same as one of the valid code words,

- the word is accepted;
- the corresponding dataword is extracted for use.

4. If the received codeword is not valid, it is discarded.

NOTE

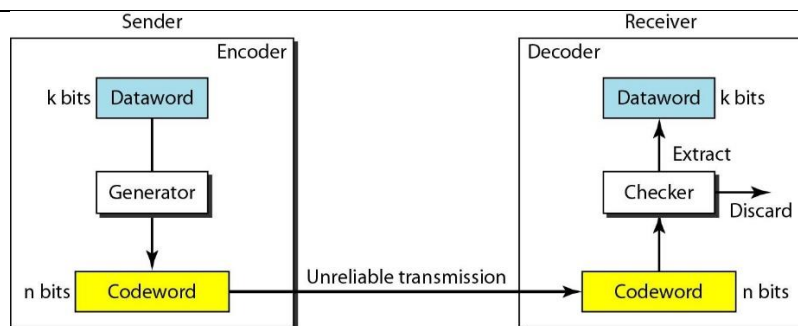
- However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.
- This type of coding can detect only single errors. Two or more errors may remain undetected.
- An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

CODE

Table 10.1 A code for error detection (Example 10.2)

Datawords	Codewords
00	000
01	011
10	101
11	110

encoder and decoder



Working Example:

Let us assume that $k = 2$ and $n = 3$.

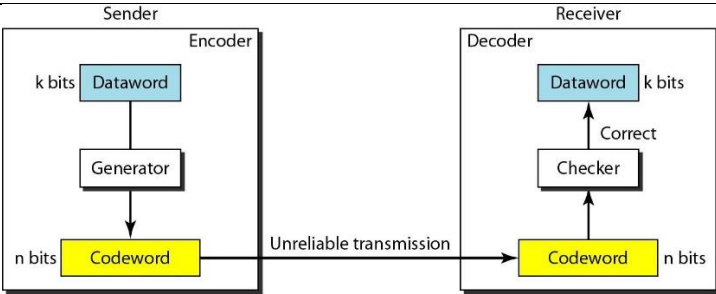
Assume the sender encodes the dataword 01 as 011 and sends it to the receiver.

Consider the following cases:

- The receiver receives 011.**
 - It is a valid codeword.
 - The receiver extracts the dataword 01 from it.
- The codeword is corrupted during transmission, and 111 is received.**
 - This is not a valid codeword
 - and is discarded.
- The codeword is corrupted during transmission, and 000 is received.**
 - This is a valid codeword.
 - The receiver incorrectly extracts the dataword 00.
- Two corrupted bits have made the error undetectable.**

Error Correction:

1. Error correction is much more difficult than error detection.
2. In error detection, the receiver needs to know only that the received codeword is invalid;
3. in error correction the receiver needs to find (or guess) the original codeword sent.
4. The receiver need to find the original codeword sent, so Need more redundant bits for error correction than for error detection.
5. The idea is the same as error detection but the checker functions are much more complex.

CODE		encoder and decoder	
Dataword	Codeword		
00	00000		
01	01011		
10	10101		
11	11110		

Working Example

1. We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords.
2. Assume the dataword is 01
Creates the codeword 01011.
3. The codeword is corrupted during transmission, and 01001 is received.
4. First, the receiver finds that the **received codeword is not in the table.**
This means an error has occurred.
(Detection must come before correction.)

The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

5. Comparing the received codeword with the codewords of the table
6. The original codeword must be the one in the table that **differs from the received codeword by 1 bit.**
7. The receiver replaces 01001 with 01011 value of the codeword and consults the table to find the dataword as 01.

6. Hamming Distance = $d(a,b)$

1. Hamming distance is a metric for comparing two binary data strings.
2. While comparing two binary strings of equal length,
3. Hamming distance is the number of bit positions in which the two bits are different.
4. The Hamming distance between two strings, a and b is denoted as $d(a,b)$.
5. In order to calculate the Hamming distance between two strings, and,
6. we perform their XOR operation, $(a \oplus b)$, and then count the total number of 1s in the resultant string.

Example

Suppose there are two strings 1101 1001 and 1001 1101.

$11011001 \oplus 10011101 = 01000100$.

Since, this contains two 1s, the Hamming distance,

Therefore $d(11011001, 10011101) = 2$.

Minimum Hamming Distance :

1. The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words
2. We use d_{min} to define the minimum Hamming distance in a coding scheme.
3. To find this value, we find the Hamming distances between all words and select the smallest one.
4. Minimum Hamming Distance used to find out the no of error bits that can be corrected .

Table 10.1 A code for error detection

Datawords	Codewords
00	000
01	011
10	101
11	110

Find the minimum Hamming distance of the coding scheme in **Table 10.1**.

Solution

We first find all Hamming distances.

$$\begin{array}{llll} d(000, 011) = 2 & d(000, 101) = 2 & d(000, 110) = 2 & d(011, 101) = 2 \\ d(011, 110) = 2 & d(101, 110) = 2 & & \end{array}$$

The d_{min} in this case is 2.

Dataword	Codeword
00	00000
01	01011
10	10101
11	11110

Find the minimum Hamming distance of the coding scheme in **Table 10.2**.

Solution

We first find all the Hamming distances.

$$\begin{array}{lll} d(00000, 01011) = 3 & d(00000, 10101) = 3 & d(00000, 11110) = 4 \\ d(01011, 10101) = 4 & d(01011, 11110) = 3 & d(10101, 11110) = 3 \end{array}$$

The d_{min} in this case is 3.

Three Parameters :

- Any coding scheme needs to have at least three parameters:
 - the codeword size n ,
 - the dataword size k ,
 - the minimum Hamming distance d_{min} .
- A coding scheme C is written as $C(n, k)$ with a separate expression for d_{min} .
- For example, we can call our first coding scheme $C(3, 2)$ with $d_{min} = 2$ and our second coding scheme $C(5, 2)$ with $d_{min} = 3$.

Minimum Distance for Error Detection :

- To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} + 1$
- Minimum Hamming Distance used to find out the no of error bits that can be corrected .
- The Number of Error Bits a Coding Scheme can detect with a Minimum Hamming Distance of d is $d-1$

Numericals

1. The minimum Hamming distance for our first code scheme (Table 10.1) is 2. This code guarantees detection of only a single error.

For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

2. Our second block code scheme (Table 10.2) has $d_{min} = 3$. This code can detect up to two errors. Again, we see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.

However, some combinations of three errors change a valid codeword to another valid codeword. The receiver accepts the received codeword and the errors are undetected.

Minimum Distance for Error Correction :

- *To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{\min} = 2t+1$*
- *The Number of Error Bits a Coding Scheme can correct with a Minimum Hamming Distance of d is $(d-1)/2$.*

Numericals

Example 10.9 *A code scheme has a Hamming distance $d_{\min} = 4$. What is the error detection and correction capability of this scheme?*

Solution : *This code guarantees the detection of upto three errors ($s = 3$), but it can correct up to one error.*

7. LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called linear block codes.

A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword

Both table belongs to linear block code because XORing gives the another valid codeword

Table 10.1 *A code for error detection (Example 10.2)*

<i>Datawords</i>	<i>Codewords</i>	Dataword	Codeword
00	000	00	00000
01	011	01	01011
10	101	10	10101
11	110	11	11110

Minimum Distance for Error Correction :

- In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.*
- In our second code (Table 10.2), the numbers of 1s in the nonzero codewords are 3, 3, and 4. So in this code we have $d_{min} = 3$.*

8. Types of linear Block Codes

1.SIMPLE PARITY-CHECK CODE:

1. the most familiar error- detecting code is the simple parity-check code.
2. In this code, a k -bit dataword is changed to an n -bit codeword where $n = k + 1$
3. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even.
4. A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.

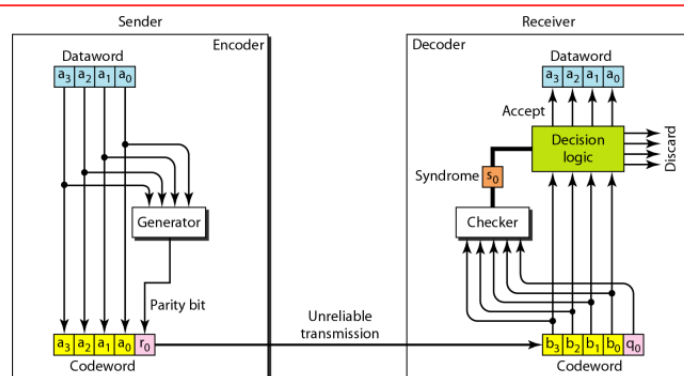
CODES

Table 10.3 Simple parity-check code $C(5, 4)$

Datawords	Codewords	Datawords	Codewords
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

ENCODER-DECODER DIAGRAM

Figure 10.10 Encoder and decoder for simple parity-check code



Working Example:

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011.

The codeword created from this dataword is 10111, which is sent to the receiver.

We examine five cases:

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a_1 . The received codeword is 10011. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0 . The received codeword is 10110. The syndrome is 1. No dataword is created.
4. An error changes r_0 and a second error changes a_3 . The received codeword is 00110. The syndrome is 0. The dataword 0011 is created at the receiver.
5. Note that here the dataword is wrongly created due to the syndrome value.
6. Three bits- a_3 , a_2 , and a_1 —are changed by errors. The received codeword is (01011). The syndrome is 1. The dataword is not created.

Note:

- This shows that the simple parity check, guaranteed to detect one single error,
- A simple parity-check code can detect an odd number of errors

A better approach is the two-dimensional parity check.

1. In this method, the data to be sent is organized in a table (rows and columns).
2. For each row and each column, 1 parity-check bit is calculated.
3. The whole table is then sent to the receiver, which finds the syndrome for each row and each column.
4. The two-dimensional parity check can detect up to three errors that occur anywhere in the table
(arrows point to the locations of the created nonzero syndromes).
5. However, errors affecting 4 bits may not be detected.

Figure 10.11 Two-dimensional parity-check code

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1
Column parities							1
							Row parities

a. Design of row and column parities

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1
							Row parities

b. One error affects two parities

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1
							Row parities

c. Two errors affect two parities

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1
							Row parities

d. Three errors affect four parities

1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1
							Row parities

e. Four errors cannot be detected

2.HAMMING CODES:

1. They are error-correcting codes.
2. These codes were originally designed with $d_{min} = 3$, which means that they can detect up to two errors or correct one single error.
3. Note: some Hamming codes that can correct more than one error, our discussion focuses on the single-bit error-correcting code.
4. First let us find the relationship between n and k in a Hamming code.
5. We need to choose an integer $m \geq 3$.
6. The values of n and k are then calculated from m as $n = 2^m - 1$ and $k = n - m$.
7. The number of check bits $r = m$.
8. For example,
 - If $m = 3$, then $n = 7$ and $k = 4$. This is a Hamming code $C(7, 4)$ with $d_{min} = 3$.
 - Table 10.4 shows the datawords and codewords for this code
 - All Hamming codes discussed in this book have $d_{min} = 3$.
 - The relationship between m and n in these codes is $n = 2^m - 1$.

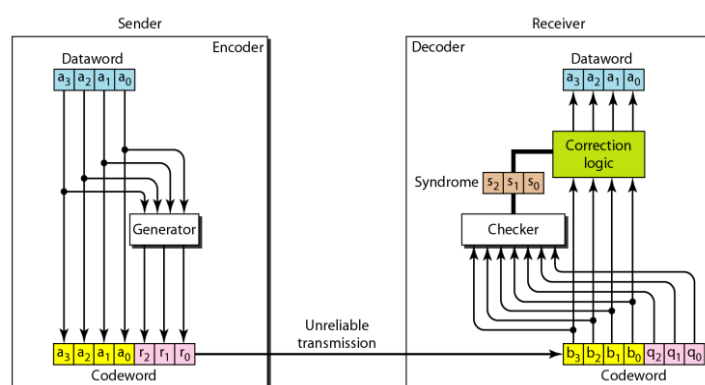
CODES

Table 10.4 Hamming code $C(7, 4)$

Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

ENCODER-DECODER DIAGRAM

Figure 10.12 The structure of the encoder and decoder for a Hamming code



WORKING

- A copy of a 4-bit dataword is fed into the generator that creates three parity checks R_1, R_2, R_0 as shown below:
 1. $R_0 = A_2 + A_1 + A_0 \quad \text{MODULO-2}$
 2. $R_1 = A_3 + A_2 + A_1 \quad \text{MODULO-2}$
 3. $R_2 = A_1 + A_0 + A_3 \quad \text{MODULO-2}$
- The checker in the decoder creates a 3-bit syndrome ($S_2S_1S_0$) in which each bit is the parity check for 4 out of the 7 bits in the received codeword:
 1. $S_0 = b_2 + b_1 + b_0 + q_0 \quad \text{MODULO-2}$
 2. $S_1 = b_3 + b_2 + b_1 + q_1 \quad \text{MODULO-2}$
 3. $S_2 = b_1 + b_0 + b_3 + q_2 \quad \text{MODULO-2}$
- The 3-bit syndrome creates eight different bit patterns (000 to 111) that can represent eight different conditions. These conditions define a lack of error or an error in 1 of the 7 bits of the received code word

Table 10.5 Logical decision made by the **correction logic analyzer**

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Working Example:

1. We need a dataword of at least 7 bits. Calculate values of k and n that satisfy this requirement.

Solution

We need to make $k=n-m$ greater than or equal to 7.

1. If we set $m = 3$, the result is $n=2^3-1=7$ and $k = 7-3$, or 4, which is not acceptable.
2. If we set $m = 4$, then $n = 2^4 - 1 = 15$ and $k = 15-4 = 11$, which satisfies the condition. So the code is $C(15, 11)$

2. Let us trace the path of three datawords from the sender to the destination:

1. The dataword 0100 becomes the codeword 0100011. The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.
2. The dataword 0111 becomes the codeword 0111001. The syndrome is 011. After flipping b_2 (changing the 1 to 0), the final dataword is 0111.
3. The dataword 1101 becomes the codeword 1101000. The syndrome is 101. After flipping b_0 , we get 0000, the wrong dataword. This shows that our code cannot correct two errors.

3.CYCLIC CODES

1. 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.
2. 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:
 $b_1 = a_0, b_2 = a_1, b_3 = a_2, b_4 = a_3, b_5 = a_4, b_6 = a_5, b_0 = a_6$
3. 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. A category of cyclic codes called the cyclic redundancy check (CRC) is used in networks such as LANs and WANs.
- Table below shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

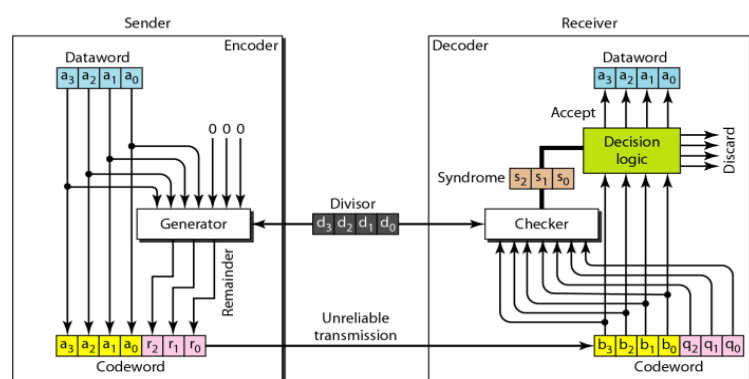
CODES

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

ENCODER-DECODER DIAGRAM

Figure 10.14 CRC encoder and decoder



Working of the Encoder :

1. 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) 0's to the right-hand side of the word.
2. 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).
3. The quotient of the division is discarded; the remainder ($R_2R_1R_0$) is appended to the dataword to create the codeword.

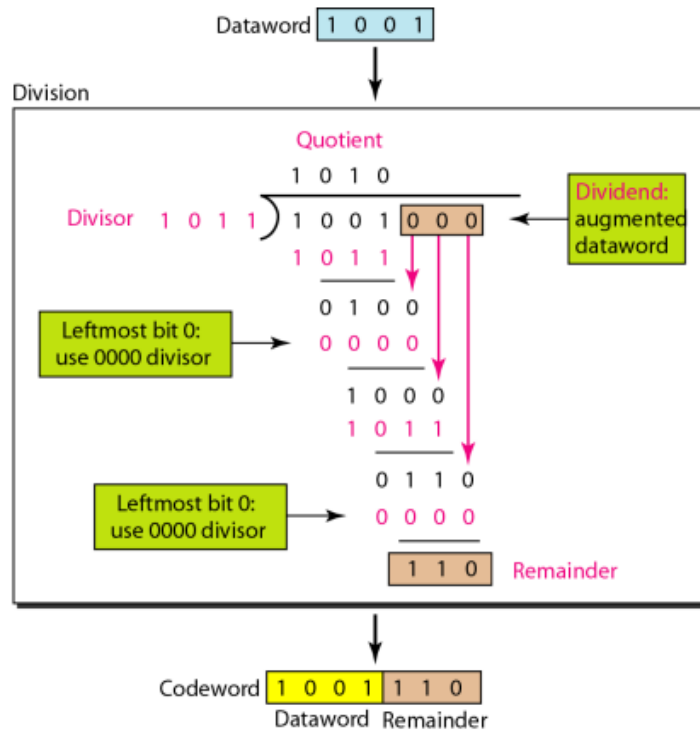
Working of the Decoder :

4. 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.
5. 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.
6. If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Working Example:

Encoder - Modulo 2 Division

Figure 10.15 Division in CRC encoder



Decoder - Modulo 2 Division

