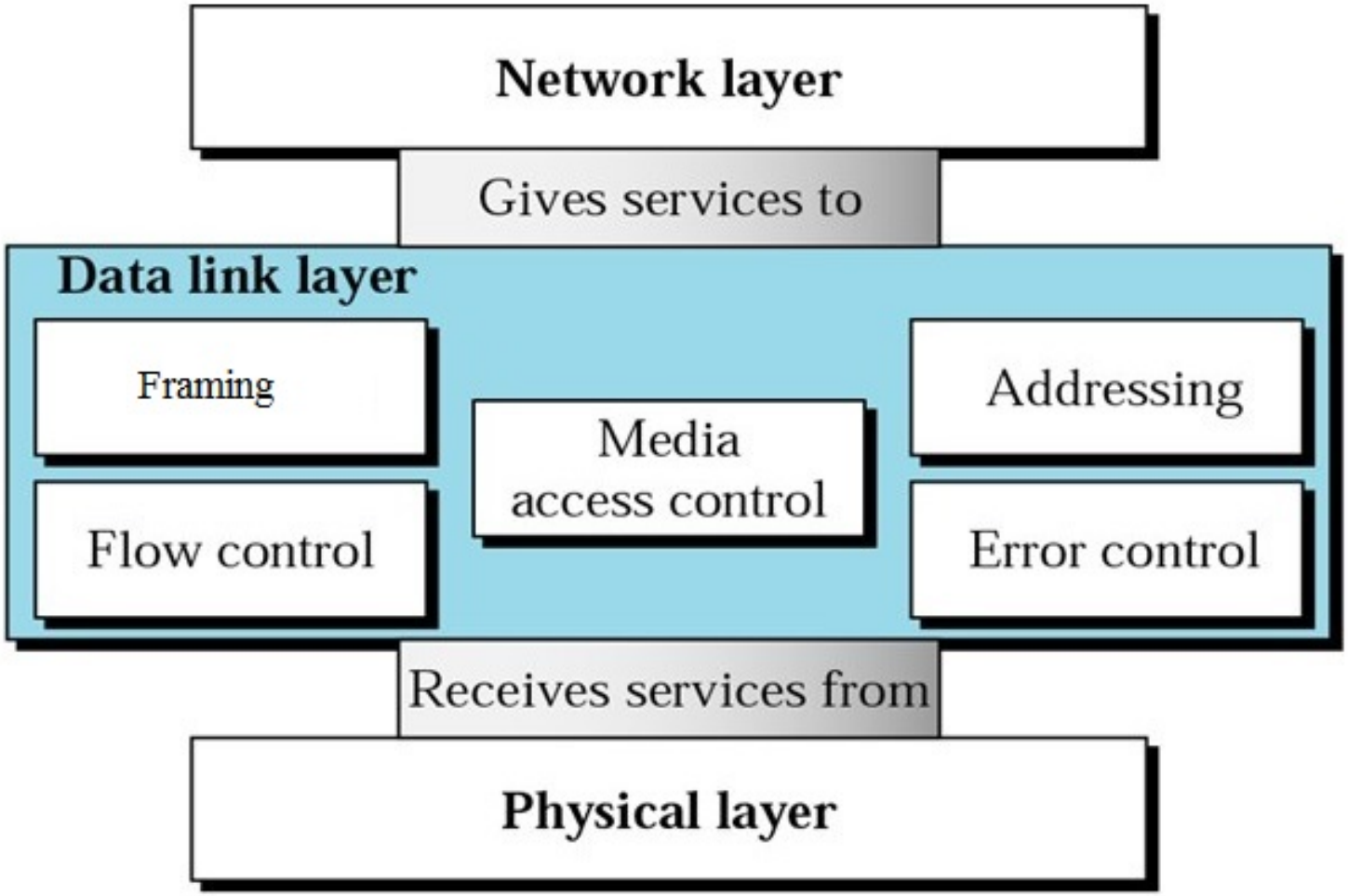
Module3
Data Link Layer

Data Link Layer



Error Detection and Correction

- Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference.
  This interference can change the shape of the signal.

Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.

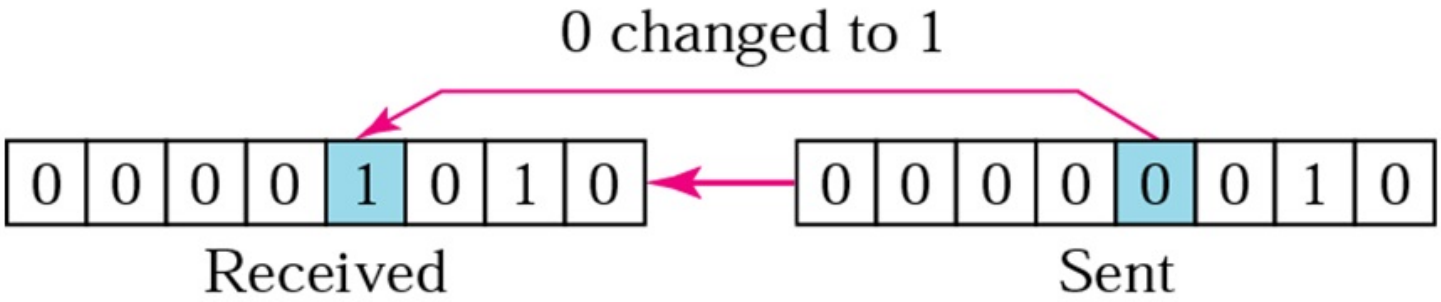
Error Detection and Correction

- Types of error:
  - Single bit error
  - Burst error

Error Detection and Correction

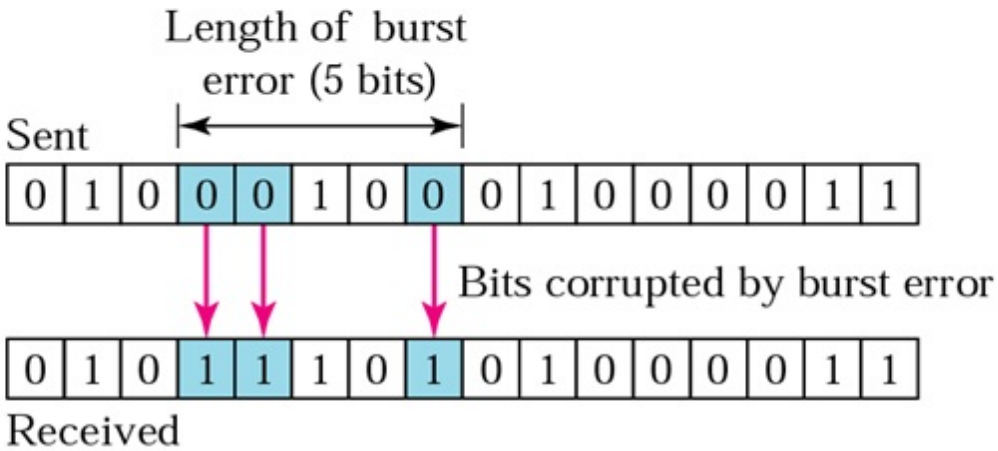
- Types of error:
  - Single bit error
  - Burst error

In a single-bit error, only one bit in the data unit has changed.



Error Detection and Correction

A burst error means that 2 or more bits in the data unit have changed.



Error Detection Techniques

- The correction of errors is more difficult than the detection.
- In error detection, we are looking only to see if any error has occurred. We are not even interested in the number of errors.
- 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.
- For example;
  if we need to correct one single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two errors in a data unit of the same size, we need to consider 28 possibilities.

Error Detection Techniques

- Error Detection Techniques:
  - Redundancy
  - Parity Check
  - Cyclic Redundancy Check(CRC)
  - Checksum

Error Detection Techniques

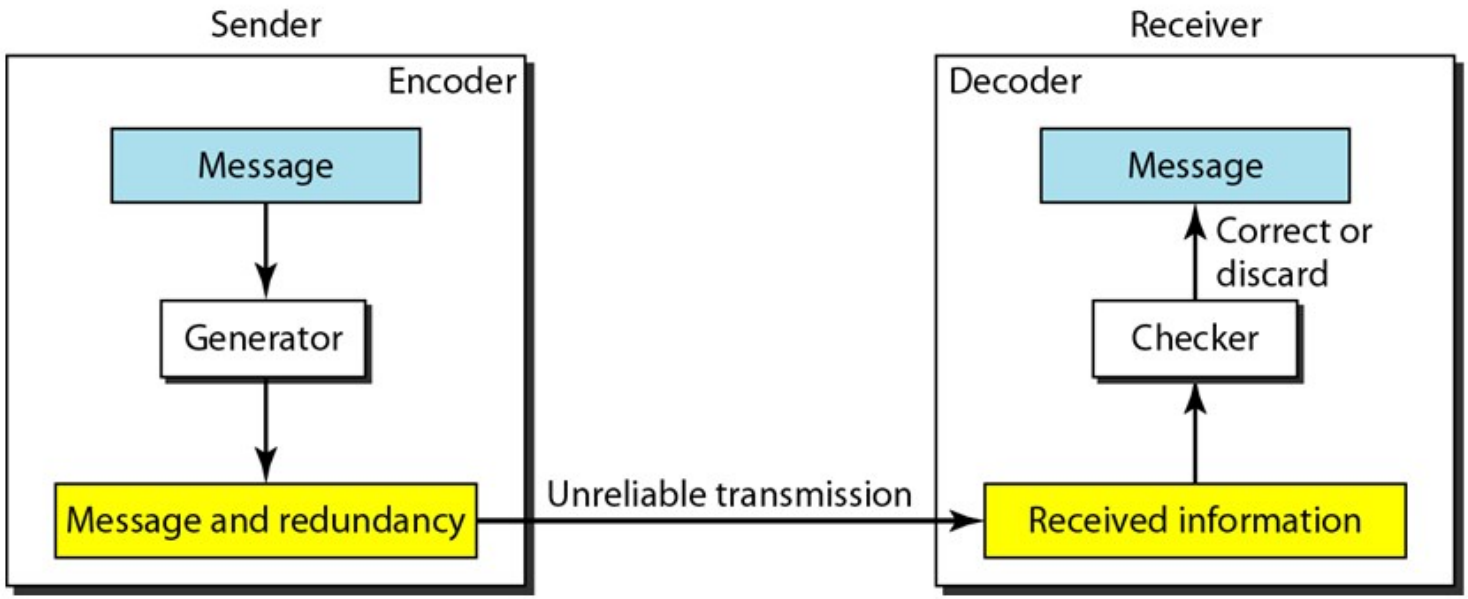
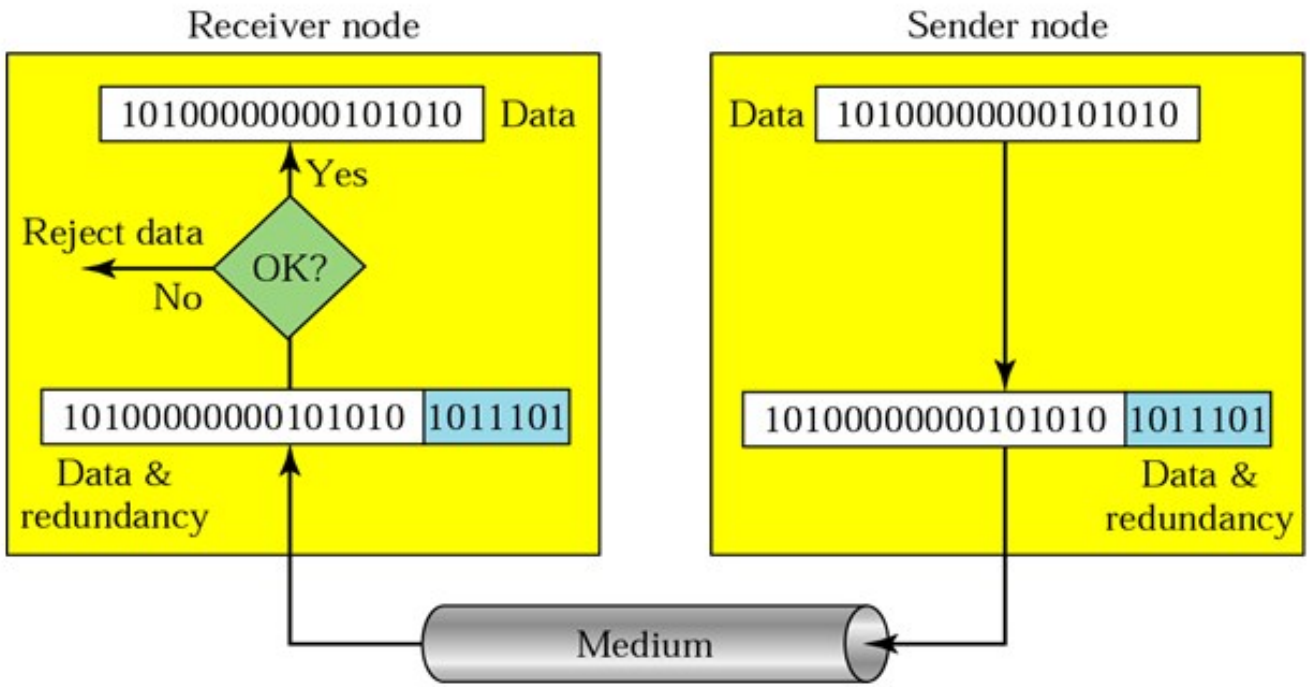


Figure 1 : Structure of encoder and decoder

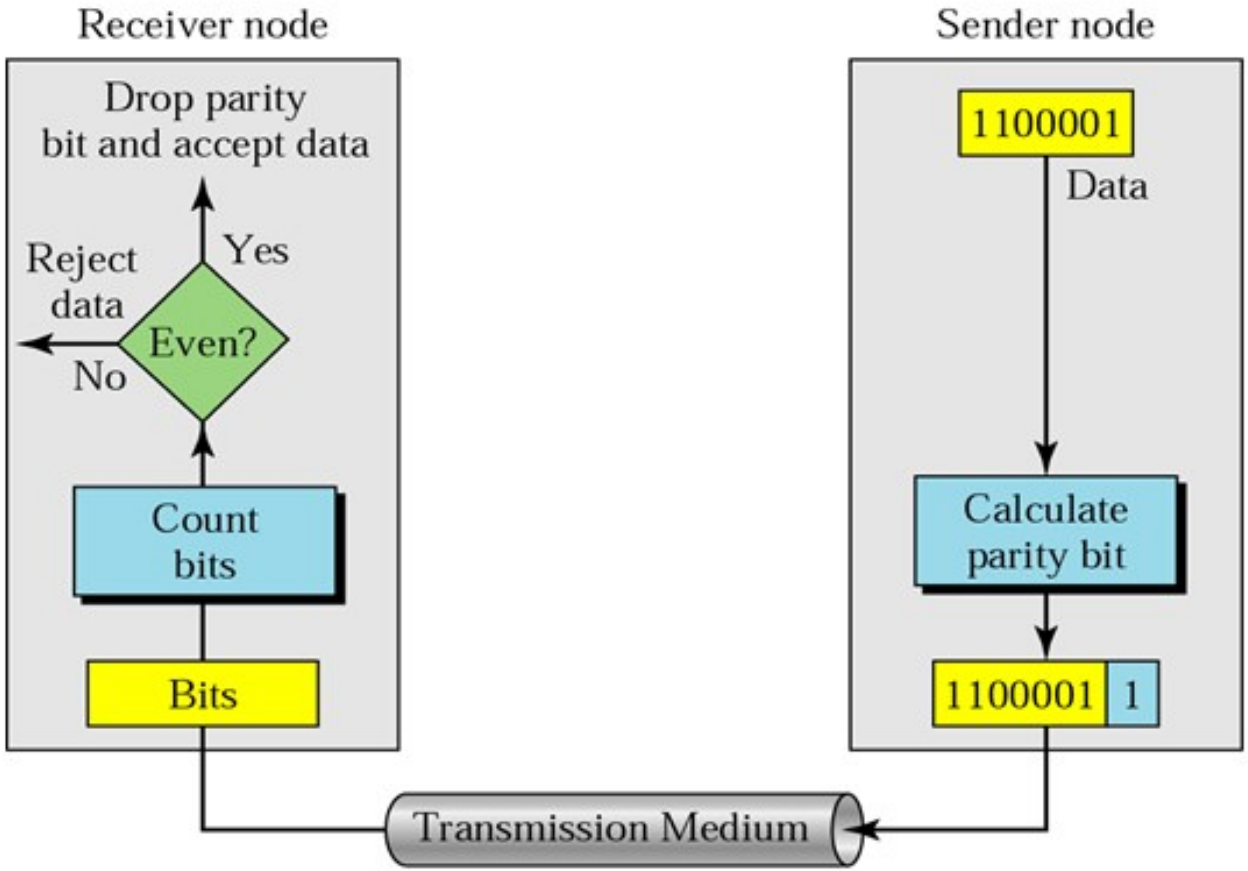
Error Detection Techniques

Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

Redundancy


Receiver node
Sender node
10100000000101010
Data
Yes
Reject data
OK?
No
10100000000101010
1011101
Data &
redundancy
Data
10100000000101010
10100000000101010
1011101
Data &
redundancy
Medium

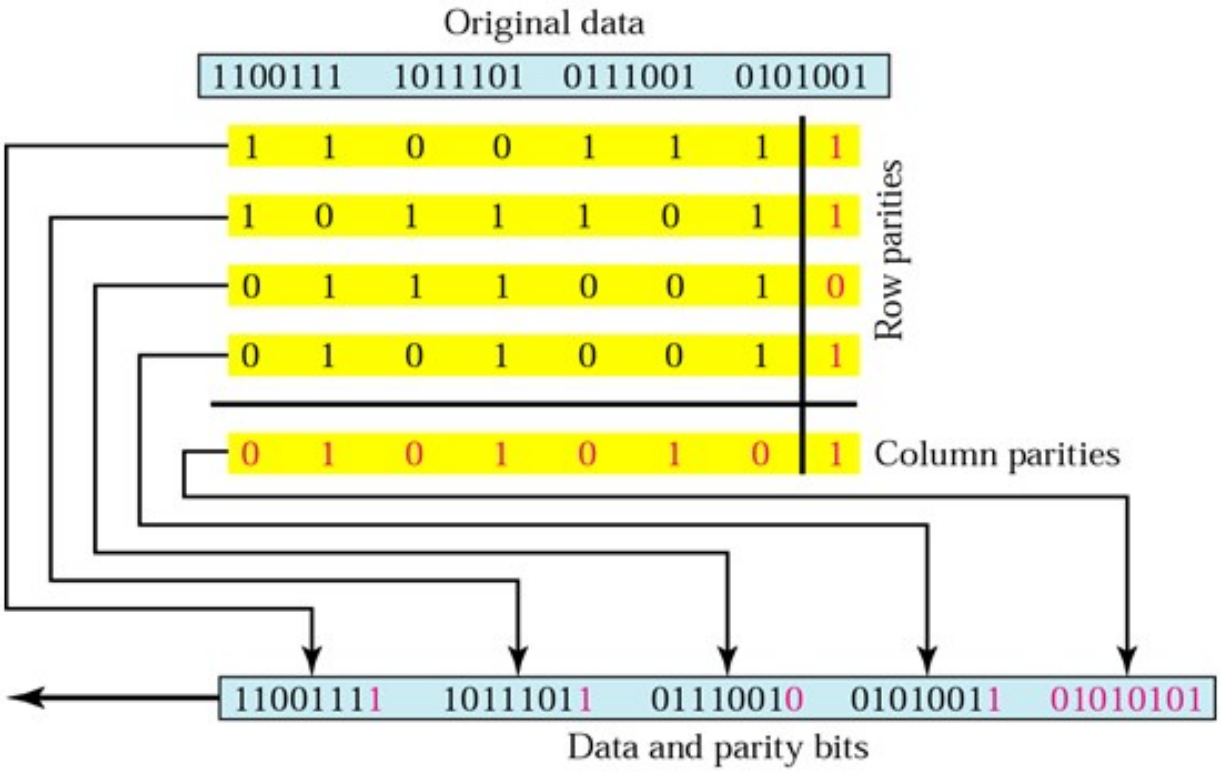
Parity Check



Parity Check

In parity check, a parity bit is added to every data unit so that the total number of 1s is even (or odd for odd-parity).

Two Dimensional Parity Check

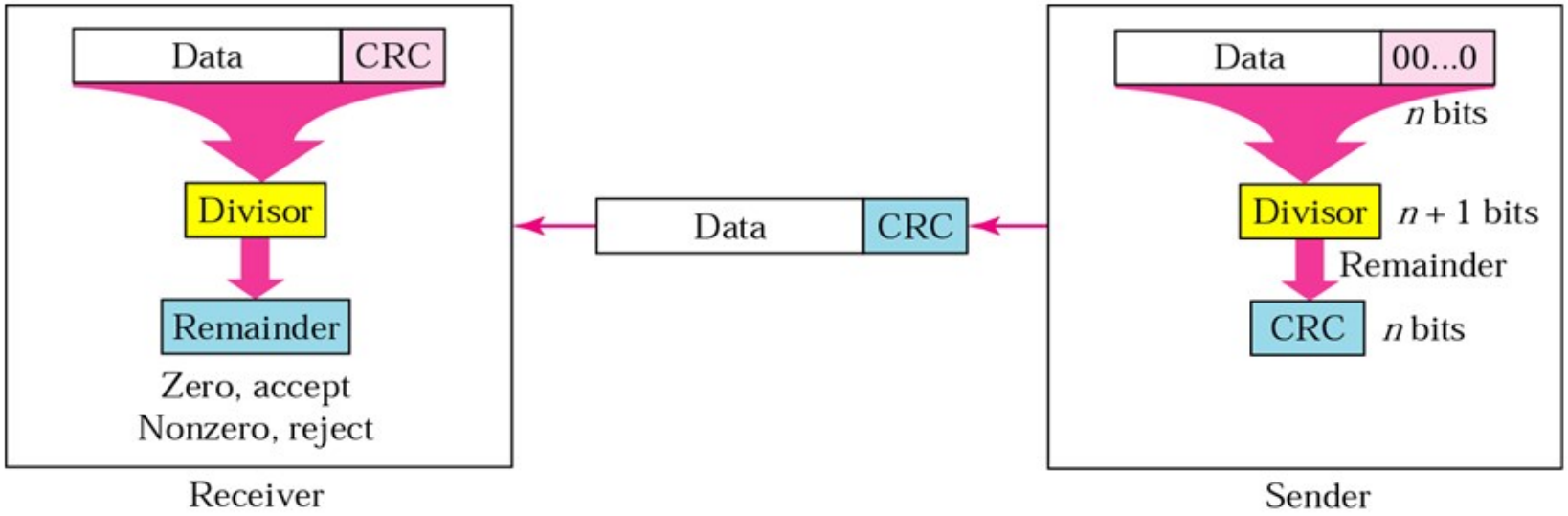


Two Dimensional Parity Check

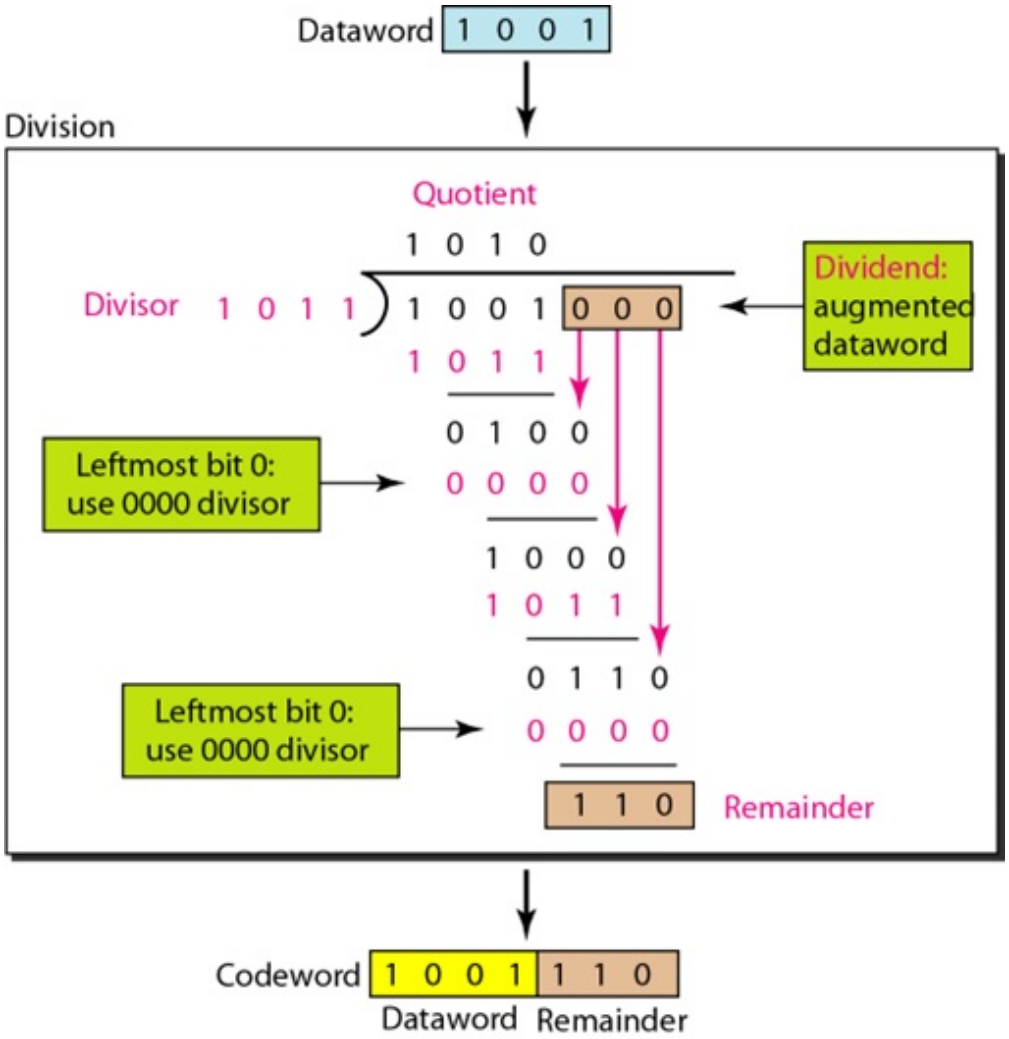
In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Cyclic Redundancy Check (CRC)

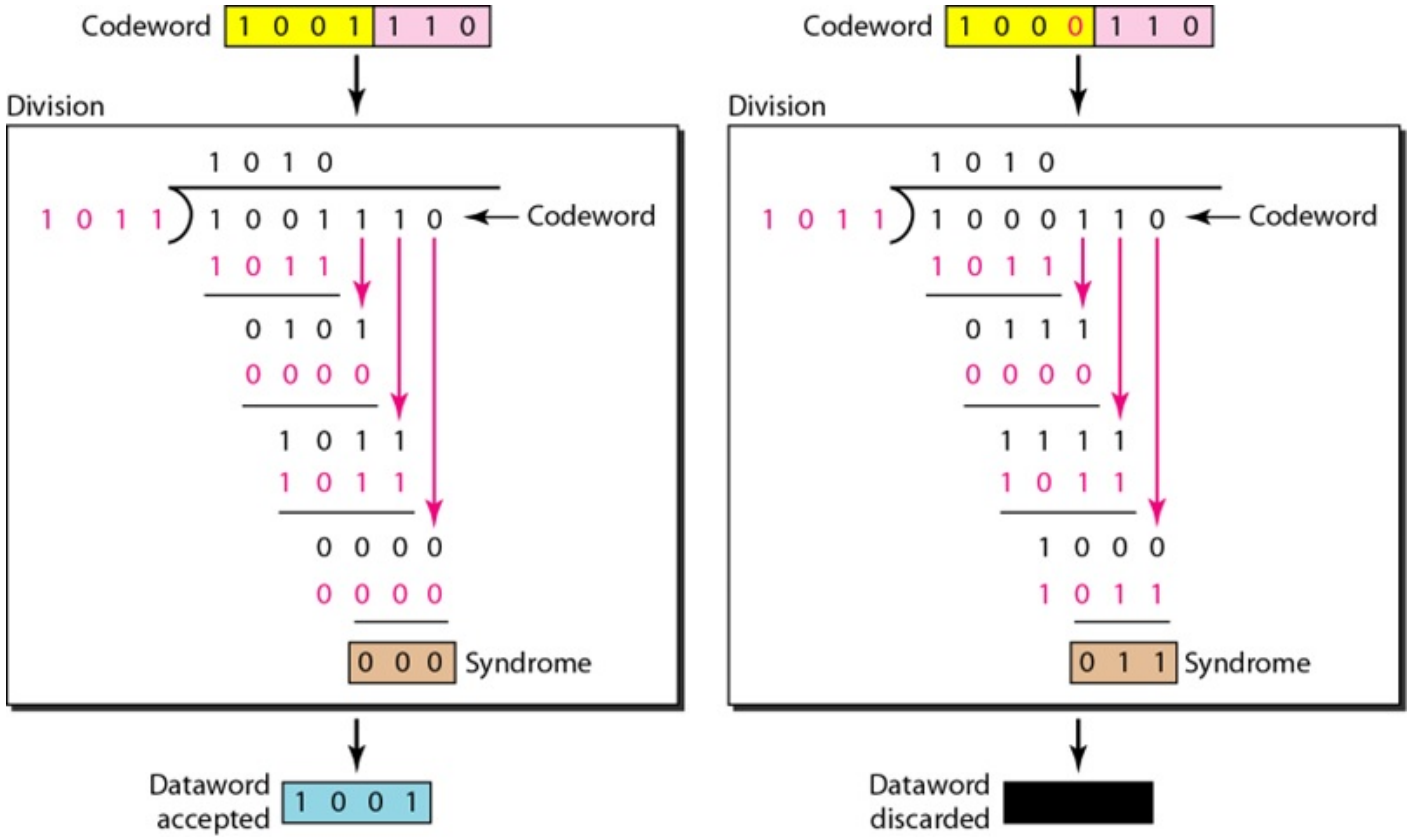
Cyclic Redundancy Check(CRC)



Cyclic Redundancy Check(CRC)



Cyclic Redundancy Check(CRC)



Cyclic Redundancy Check(CRC)

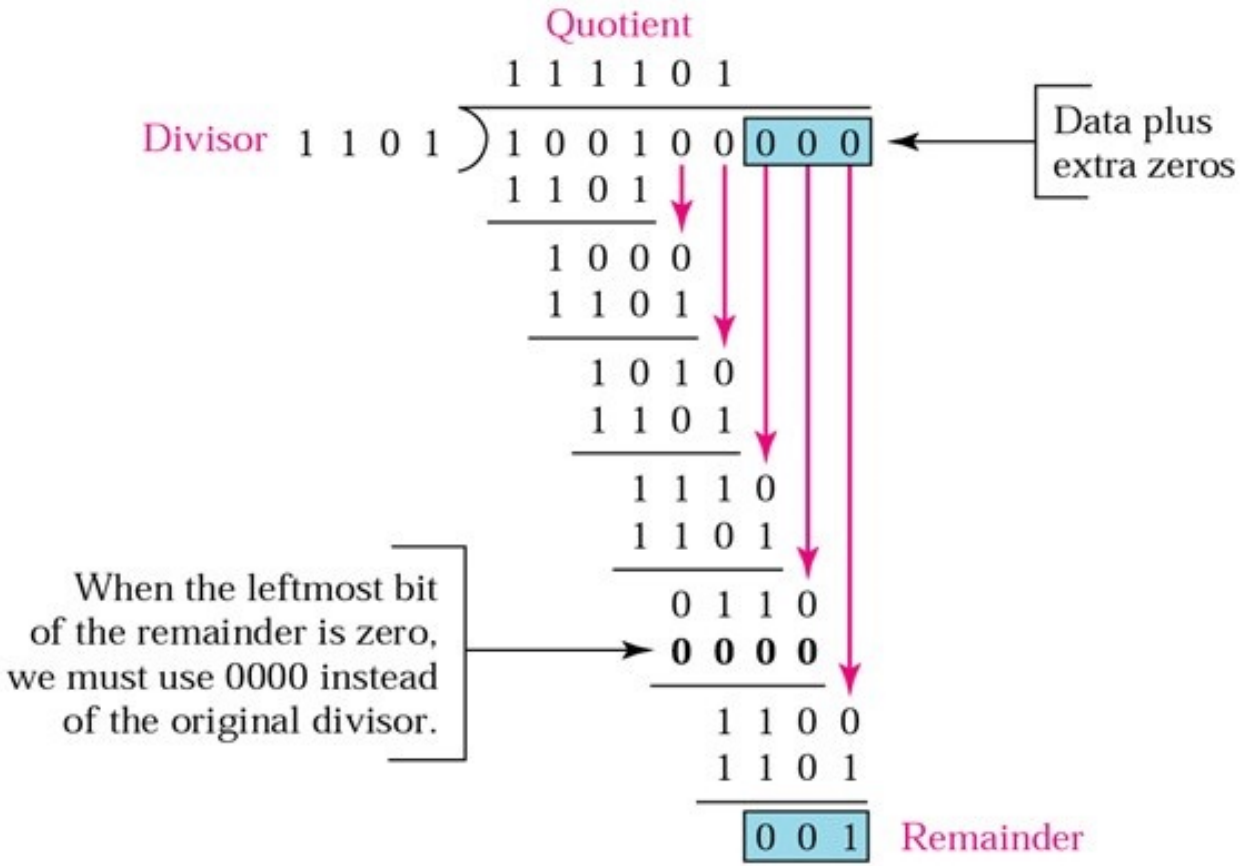


Figure 2 : Binary division in a CRC Generator

Cyclic Redundancy Check(CRC)

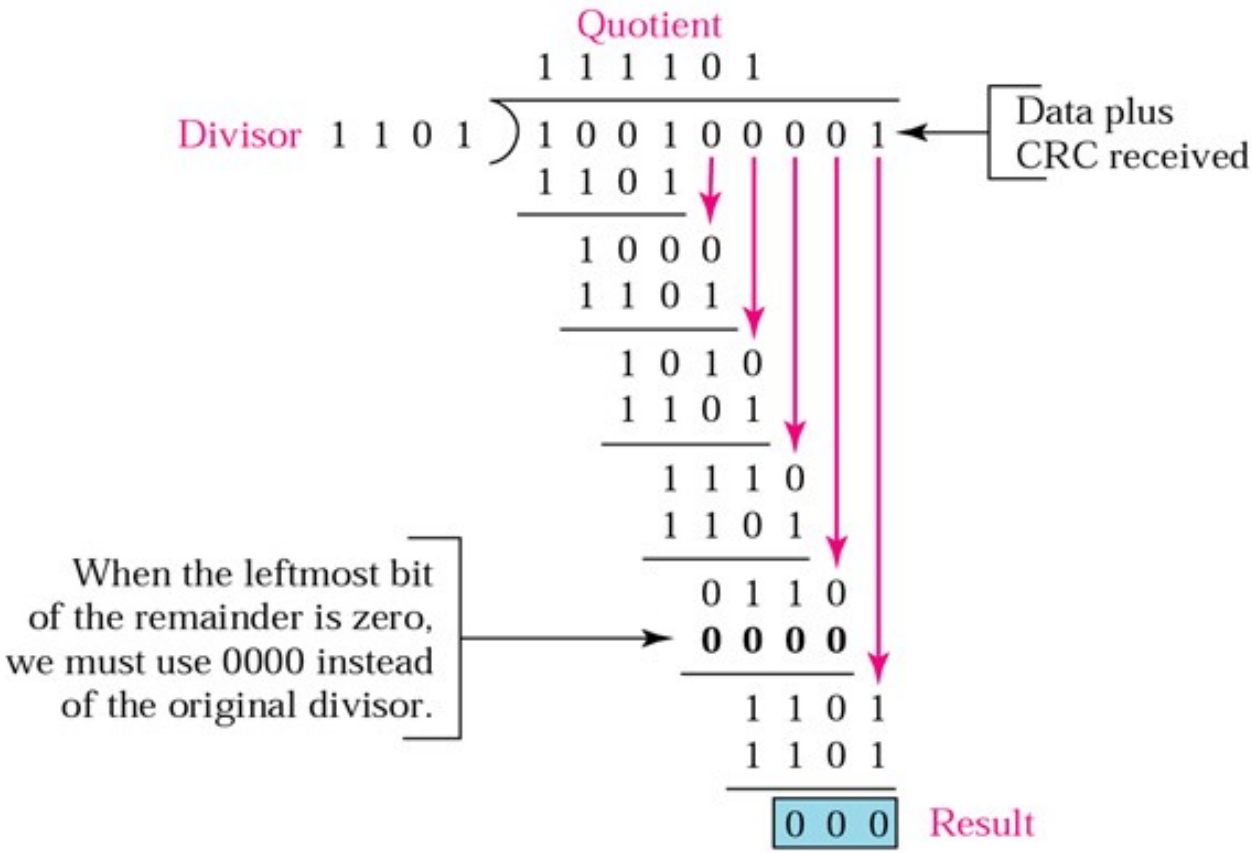


Figure 3 : Binary division in a CRC Checker

Cyclic Redundancy Check(CRC)

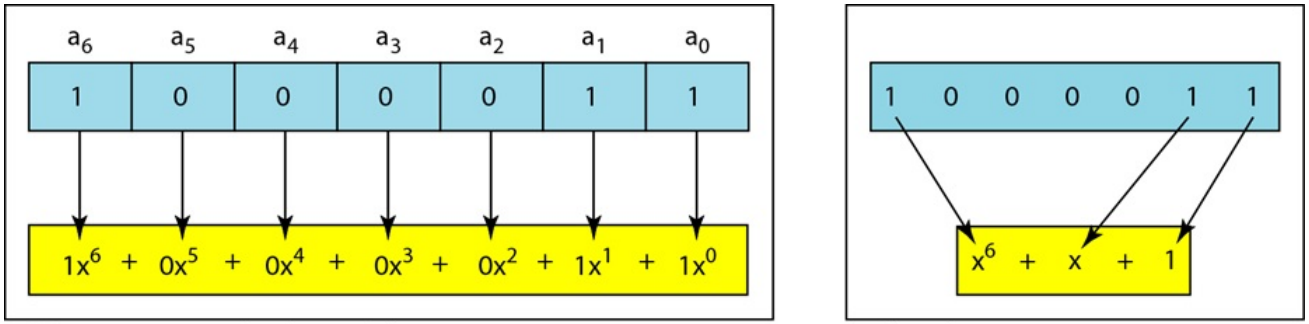


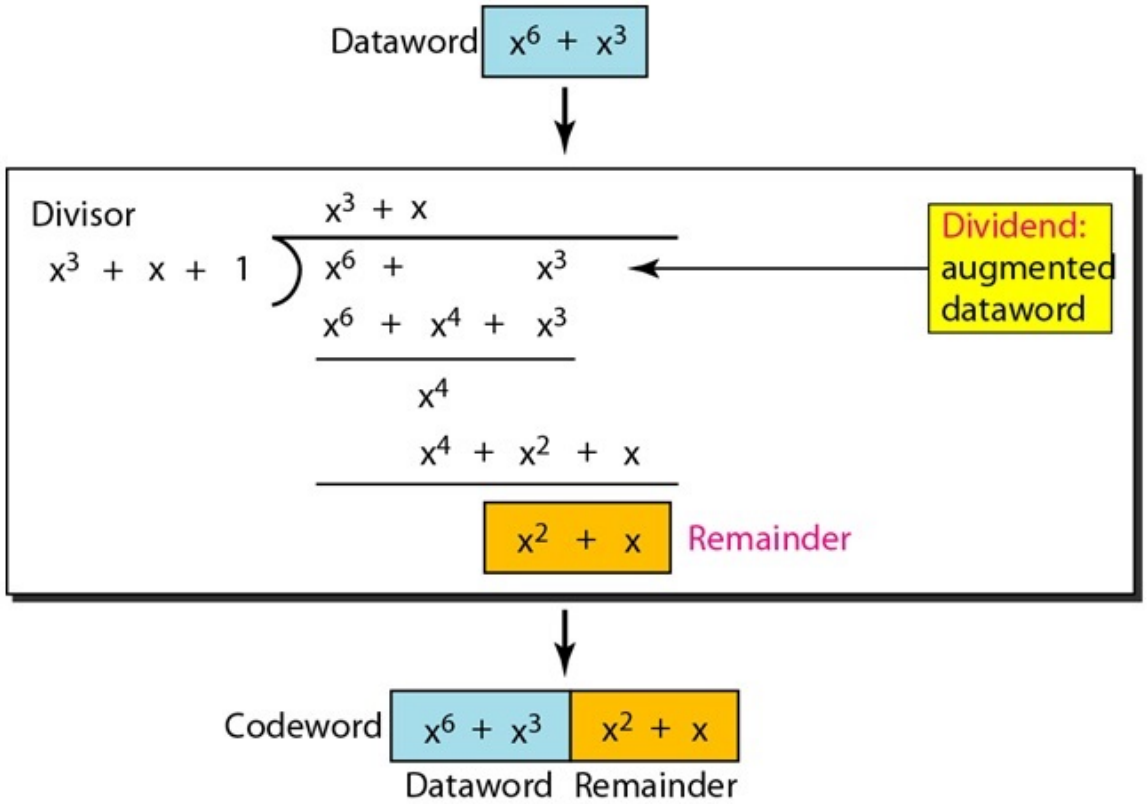
Figure 4 : Polynomial to represent a binary word

Cyclic Redundancy Check(CRC)

- Exercise1: For the given $M(x) = x^3 + 1$ and generator polynomial $x^3 + x + 1$, show the generation of codeword at sender site and receiver site. Assume that no error in codeword.

Cyclic Redundancy Check(CRC)

- Exercise1: For the given $M(x) = x^3 + 1$ and generator polynomial $x^3 + x + 1$, show the generation of codeword at sender site and receiver site. Assume that no error in codeword.
- Solution: In polynomial CRC, dataword is multiplied with order of generator polynomial
  That is, $(x^3 + 1) \times x^3 = x^6 + x^3$

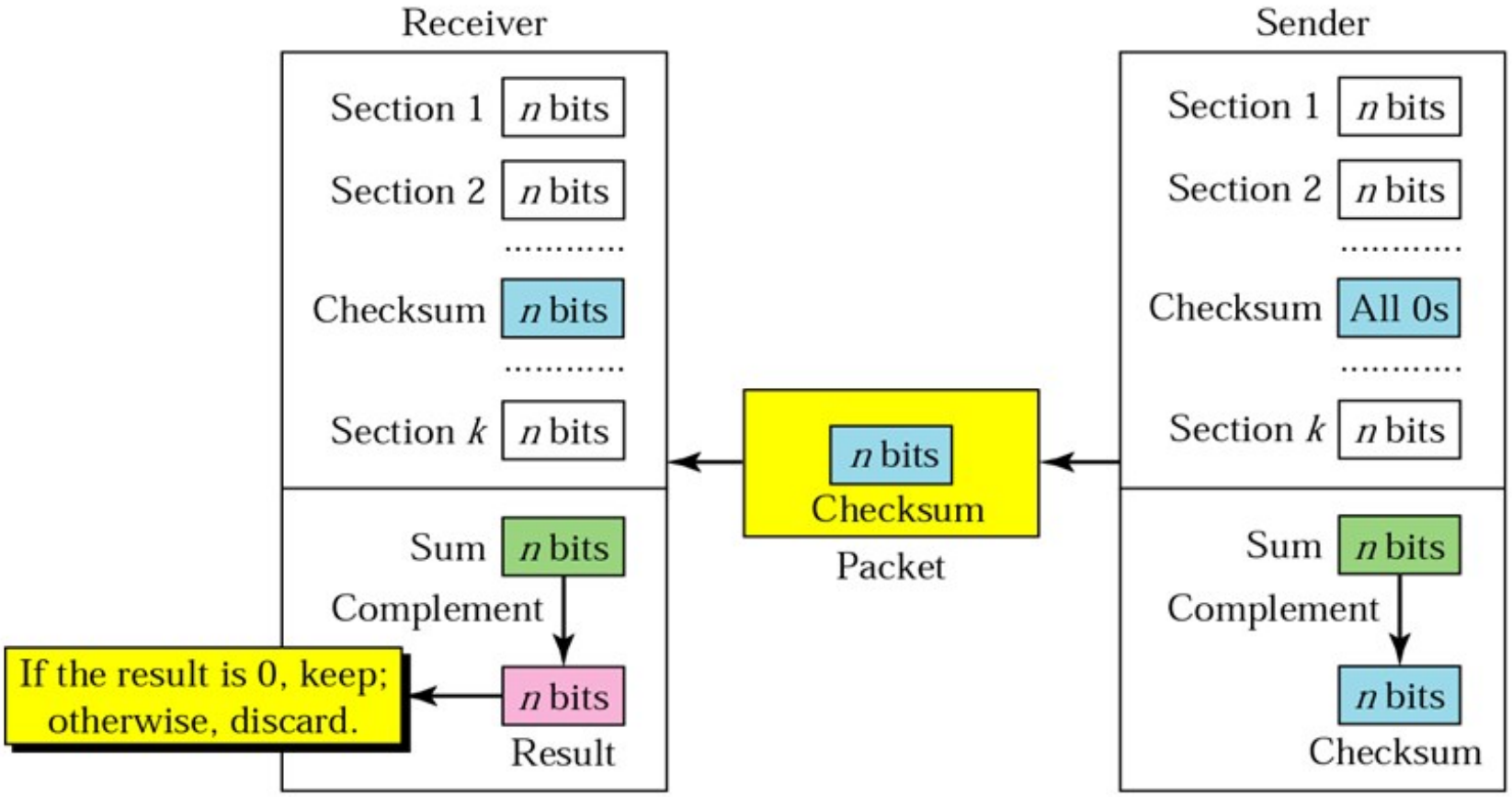


Cyclic Redundancy Check(CRC)

- Exercise2: Given the dataword 1010011110 and the divisor 10111
  (a) Show the generation of the codeword at the sender site (using binary division).
  (b) Show the checking of the codeword at the receiver site (assume no error).

Checksum

Checksum


Receiver
Section 1 n bits
Section 2 n bits
Checksum n bits
Section k n bits
Sum n bits
Complement
n bits
Result
If the result is 0, keep; otherwise, discard.
n bits
Checksum
Packet
Sender
Section 1 n bits
Section 2 n bits
Checksum All 0s
Section k n bits
Sum n bits
Complement
n bits
Checksum

Checksum

- Sender Site:
  - The message is divided into 16-bit words
  - The value of the checksum word is set to 0.
  - All words including the checksum are added using one's complement addition
  - The sum is complemented and becomes the checksum
  - The checksum is sent with the data

Checksum

- Receiver Site:
  - The message (including checksum) is divided into 16-bit words
  - All words are added using one's complement addition.
  - The sum is complemented and becomes the new checksum
  - If the value of checksum is 0, the message is accepted; otherwise, it is rejected

- Exercise1: Let us calculate the checksum for the given four frames and do the generation of codeword at sender and receiver site
  Frame1:11001100; Frame2:10101010; Frame3:11110000; Frame4:11000011

- Exercise1: Let us calculate the checksum for the given four frames and do the generation of codeword at sender and receiver site
  Frame1:11001100; Frame2:10101010; Frame3:11110000; Frame4:11000011
- Solution:

Sender's End

Frame 1:	11001100
Frame 2:	+ 10101010
Partial Sum:	1 01110110
	+ 1
	01110111
Frame 3:	+ 11110000
Partial Sum:	1 01100111
	+ 1
	01101000
Frame 4:	+ 11000011
Partial Sum:	1 00101011
	+ 1
Sum:	00101100
Checksum:	11010011

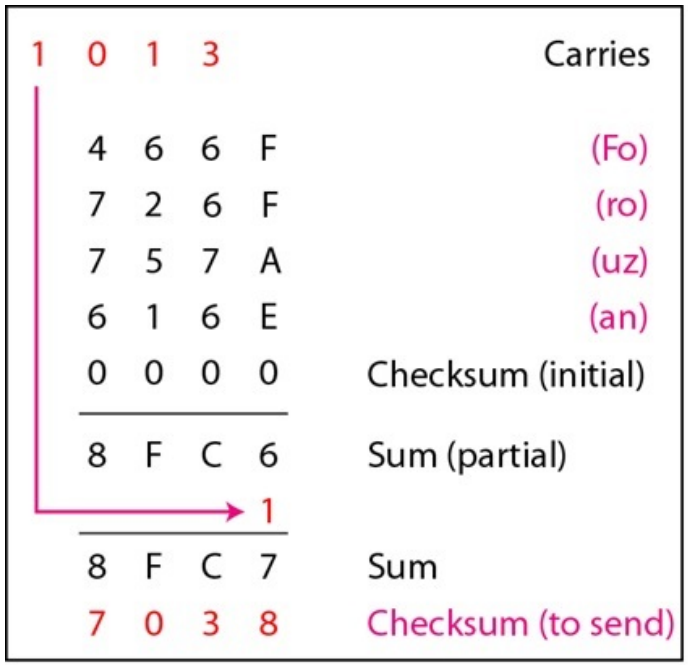
Receiver's End

Frame 1:	11001100
Frame 2:	+ 10101010
Partial Sum:	1 01110110
	+ 1
	01110111
Frame 3:	+ 11110000
Partial Sum:	1 01100111
	+ 1
	01101000
Frame 4:	+ 11000011
Partial Sum:	1 00101011
	+ 1
Sum:	00101100
Checksum:	11010011
Sum:	11111111
Complement:	00000000

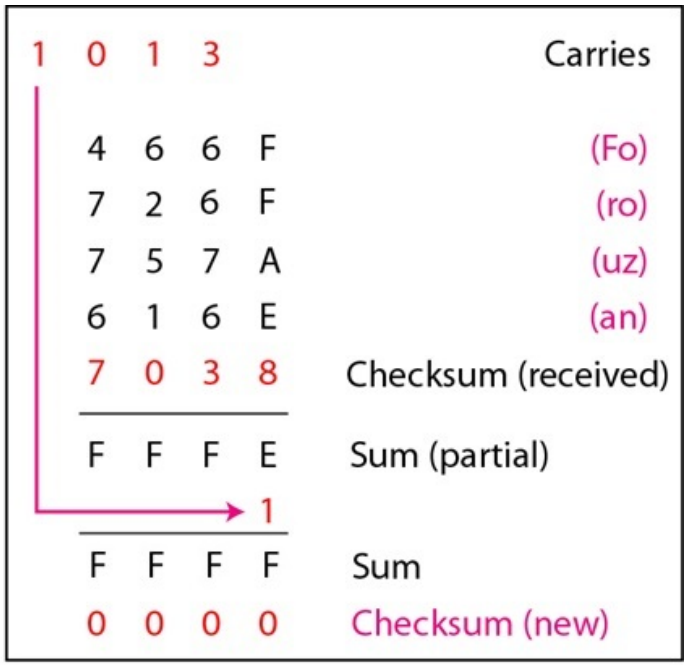
Hence accept frames.

- Exercise2: Let us calculate the checksum for a text of 8 characters (Forouzan). The text needs to be divided into 2-byte (16-bit) words. Use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as 0x46, o is represented as 0x6F, r is represented as 0x72, u is represented as 0x75, z is represented as 0x7A, a is represented as 0x61 and n is represented as 0x6E.

- Exercise2: Let us calculate the checksum for a text of 8 characters (Forouzan). The text needs to be divided into 2-byte (16-bit) words. Use ASCII (see Appendix A) to change each byte to a 2-digit hexadecimal number. For example, F is represented as 0x46, o is represented as 0x6F, r is represented as 0x72, u is represented as 0x75, z is represented as 0x7A, a is represented as 0x61 and n is represented as 0x6E.
- Solution:



a. Checksum at the sender site



a. Checksum at the receiver site

- Exercise3:

A sender needs to send the four data items Ox3456, OxABCC, Ox02BC, and OxEEEE. Answer the following:

a. Find the checksum at the sender site.

b. Find the checksum at the receiver site if there is no error.

c. Find the checksum at the receiver site if the second data item is changed to OxABCE.

d. Find the checksum at the receiver site if the second data item is changed to OxABCE and the third data item is changed to Ox02BA.

Hamming Code

Hamming Code

- The most common type of error correction are
  (1) Error Correction by retransmission
  (2) Forward error correction
- In error correction by retransmission, when an error is discovered, the receiver can have the sender retransmit the entire data unit.
- In Forward error correction, a receiver can use an error correcting code, which automatically corrects certain errors.

Hamming Code

- Hamming code is a Forward Error Correcting Code(FEC) that uses redundant bits to correct a single bit error.
- For 4 bit codes, 3 redundant bits are needed, total = 7 bits.
- For 8 bit codes, 4 redundant bits are needed, total =12 bits.
- All bit positions that are powers of two are parity bits.
- All other bit positions, with two or more bits in the binary form of their position, are data bits.

Hamming Code

- Hamming codes use the redundant bits to check the data bits using the concept of one redundant parity bit covering a number of data bits according to the binary code table.
- Bit errors can be found by identifying which parity bits are incorrect and determining which data bit was covered by those parity bits.

Hamming Code

- Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
- Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4-7, 12-15, 20-23, etc.
- Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8-15, 24-31, 40-47, etc.

Hamming Code

- Let m be the message bits and r be the check bits, that allow all single bit errors to be corrected, then this requirements requires

$$(m + r + 1) \leq 2^r$$

Number of data bits m	Number of redundancy bits r	Total bits m + r
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Figure 5 : Data and redundancy bits

Hamming Code

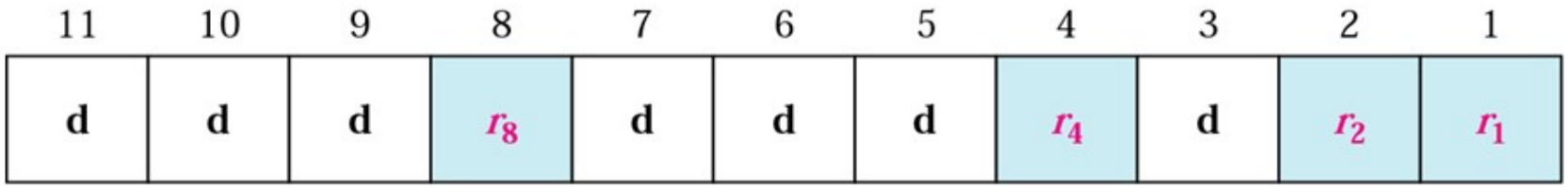


Figure 6 : Positions of redundancy bits in Hamming code

Hamming Code

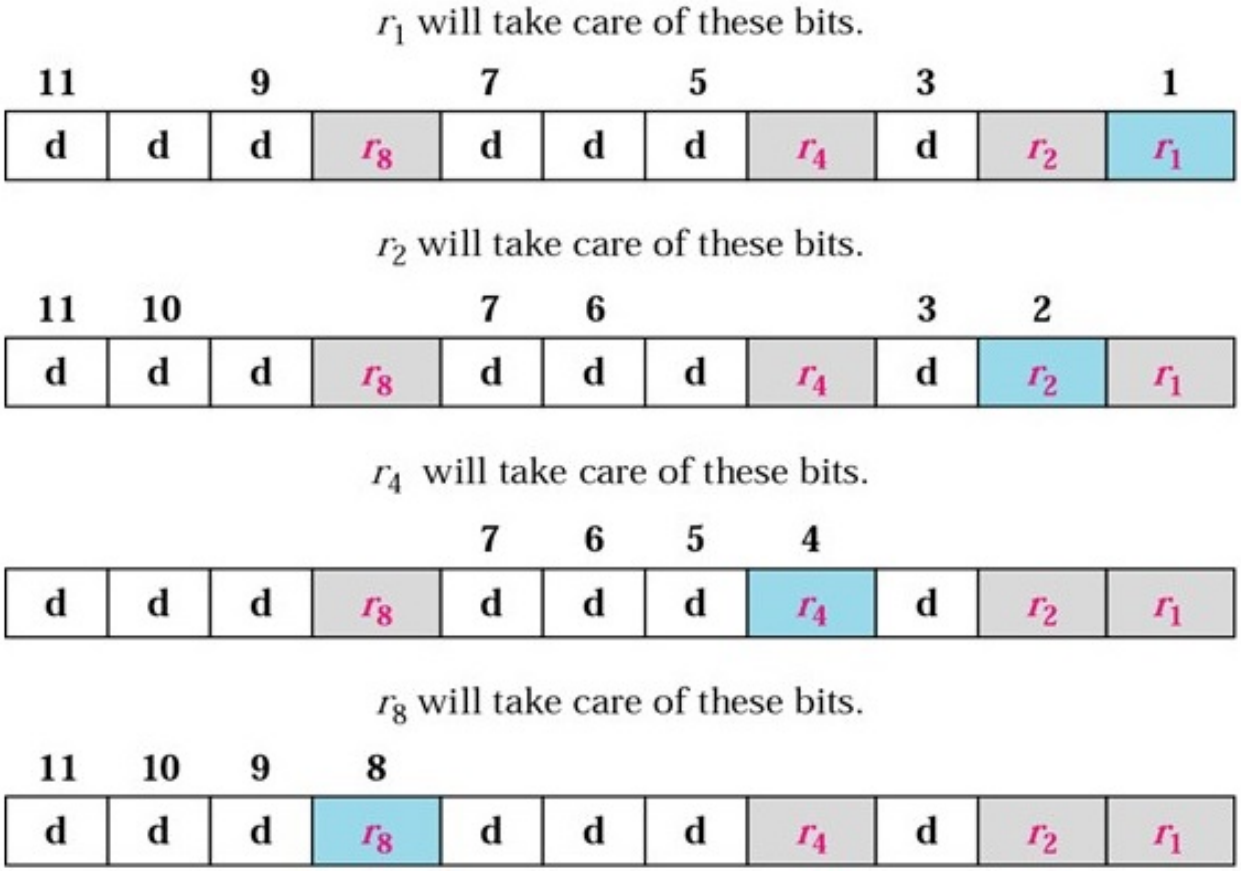


Figure 7 : Redundancy bits calculation

Hamming Code

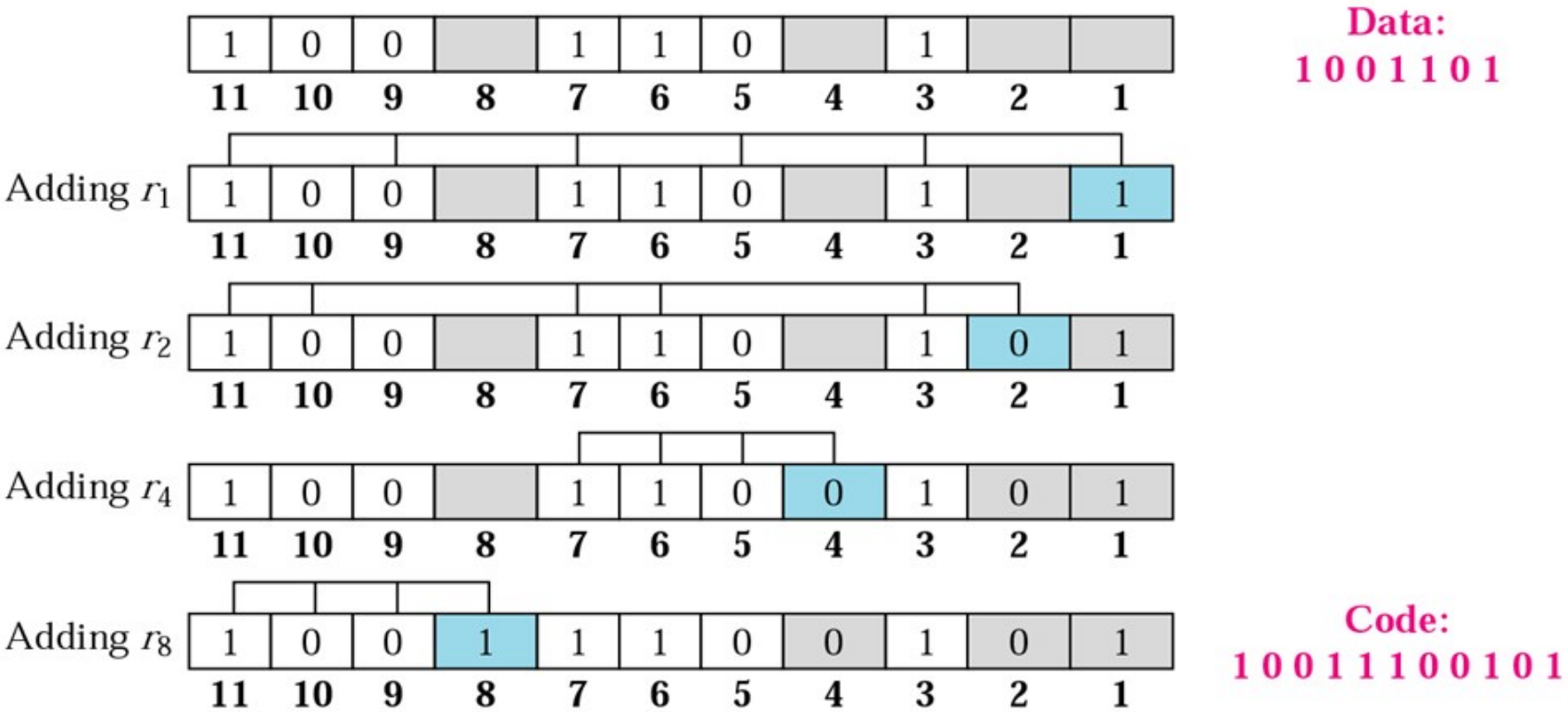
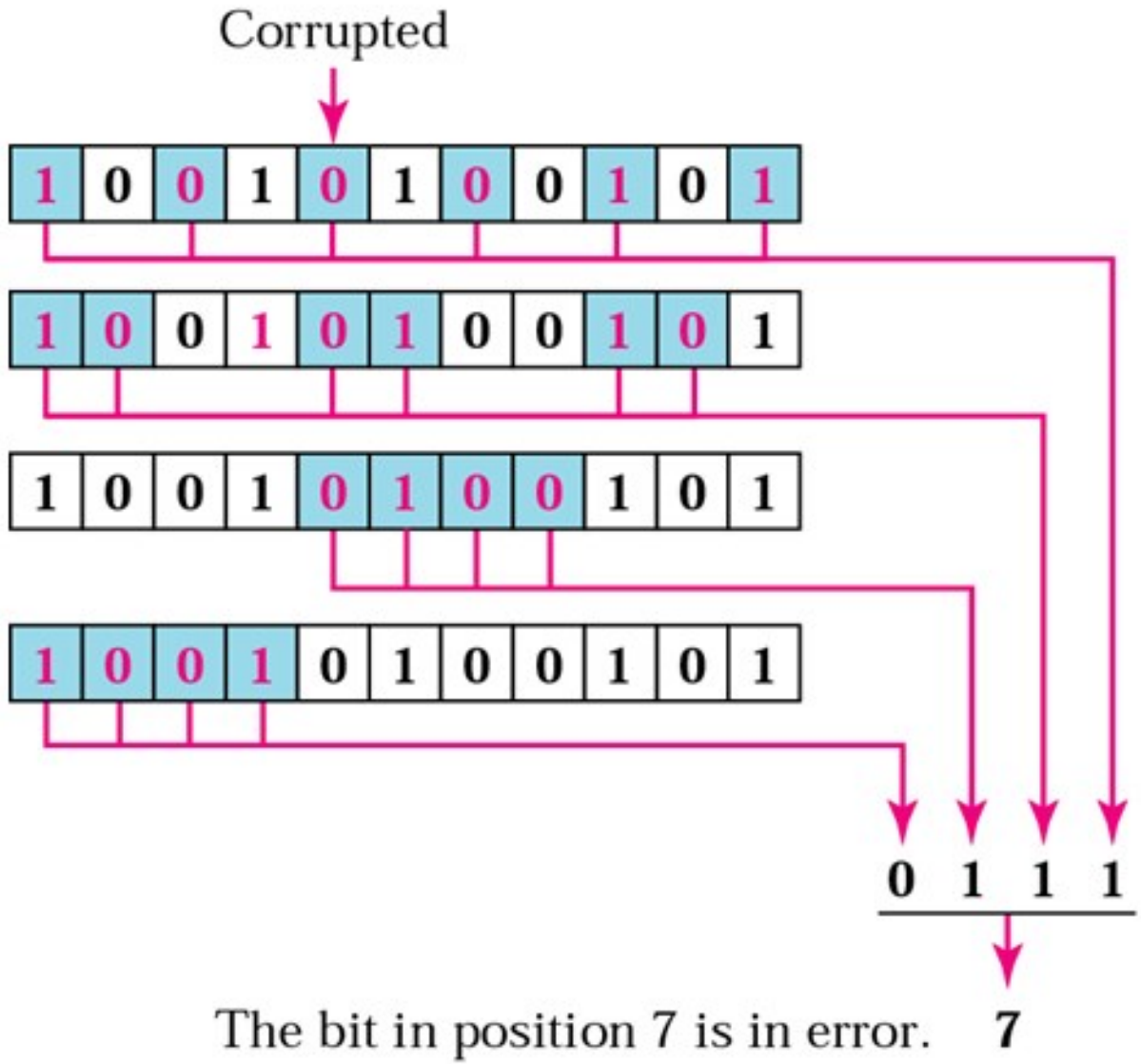


Figure 8 : Example of Redundancy bits calculation

Hamming Code



Hamming Code

- Exercise1: The code 11110101101 was received. Using the Hamming code, what is the original code sent?