ERROR DETECTING CODE USING CYCLIC REDUNDANCY CHECK (CRC-CCITT 16-bit)

CCITT- CONSULTATIVE COMMITTEE FOR INTERNATIONAL TELEGRAPHY AND TELEPHONY

OBJECTIVE

- To implement error detection using Cyclic Redundancy Check (CRC- CCITT 16-bit) technique.

DATA TRANSMISSION

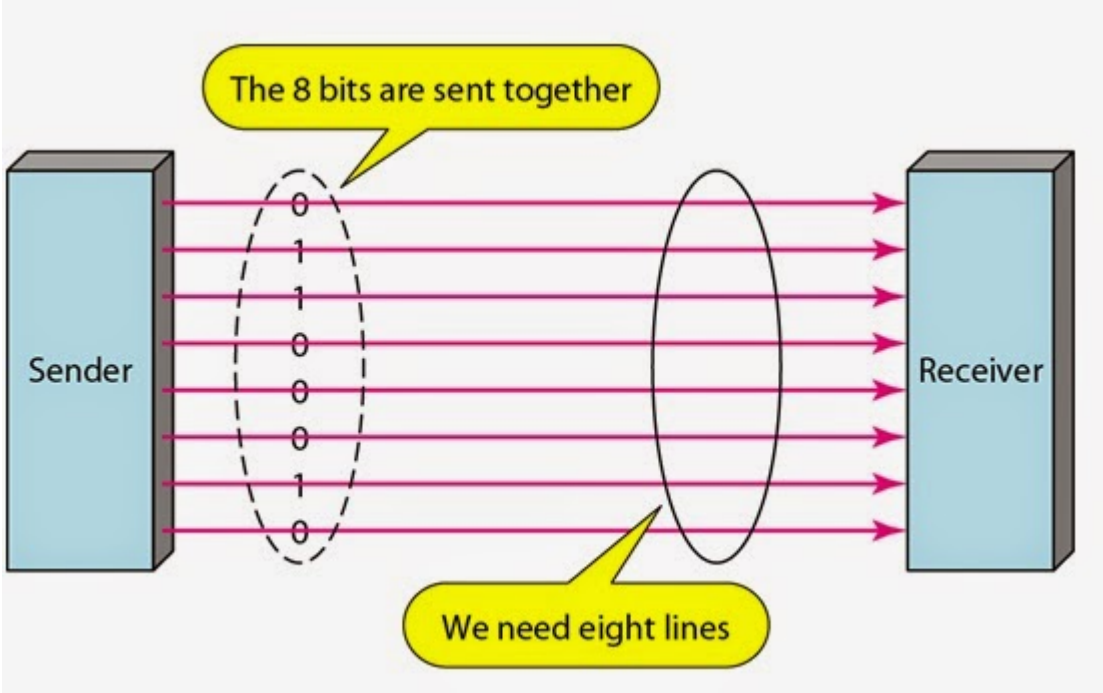
- Data transmission refers to the movement of data in the form of bits between two or more digital devices.
- This transfer of data takes place via some form of transmission media (for example, coaxial cable, fibre optics etc.)

TYPES OF DATA TRANSMISSION

PARALLEL DATA TRANSMISSION

- In parallel transmission, all the data bits are transmitted simultaneously on separate communication lines.
- In order to transmit **‘n’** bits of data, **‘n’** wires or lines are used.
- Parallel transmission is used for short distance communication.

PARALLEL DATA TRANSMISSION BLOCK DIAGRAM



ADVANTAGE / DISADVANTAGE OF PARALLEL TRANSMISSION

Advantage:

- It is fastest method of transmitting data as multiple data bits are transmitted simultaneously with a single clock pulse.

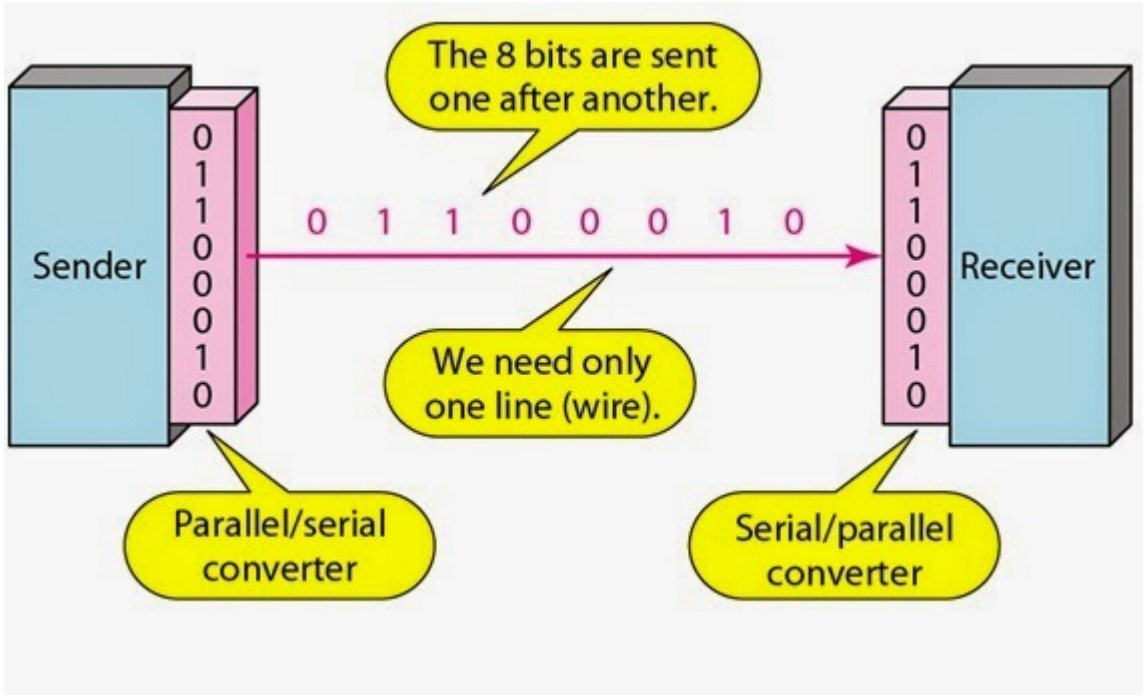
Disadvantage:

- It is costly method of data transmission as it requires **‘n’** lines to transmit **‘n’** bits at the same time.

SERIAL DATA TRANSMISSION

- In serial transmission, the data bits transmitted serially one after the other.
- It requires only one communication line to transmit data from sender to receiver.
- Serial transmission is used for long distance communication.

SERIAL DATA TRANSMISSION: BLOCK DIAGRAM



ADVANTAGE/ DISADVANTAGE OF SERIAL TRANSMISSION

Advantage:

- Use of single communication line reduces the transmission line cost by the factor of **‘n’** as compared to parallel transmission.

Disadvantages:

- Use of conversion devices at source and destination end may lead to increase in overall transmission cost.

- This method is slower as compared to parallel transmission as bits are transmitted serially one after the other.

TYPES OF SERIAL TRANSMISSION

- There are two types of serial transmission
  - **Synchronous**
  - **Asynchronous**
- Both of these transmissions use **'Bit synchronization'**
- Bit Synchronization is a function that is required to determine when the beginning and end of the data transmission occurs.
- Bit synchronization helps the receiving computer to know when data begin and end during a transmission.
- Therefore bit synchronization provides timing control.

ASYNCHRONOUS TRANSMISSION

- Asynchronous transmission sends only one character at a time where a character is either a letter of the alphabet or number or control character i.e. it sends one byte of data at a time.
- Bit synchronization between two devices is made possible using **start bit** and **stop bit.**
- Start bit indicates the beginning of data i.e. alerts the receiver to the arrival of new group of bits. A start bit usually **0** is added to the beginning of each byte.
- Stop bit indicates the end of data i.e. to let the receiver know that byte is finished, one or more additional bits are appended to the end of the byte. These bits, usually **1**s are called stop bits.

ASYNCHRONOUS TRANSMISSION: PICTORIAL REPRESENTATION

Stop bit	Data bits	Start bit
1	1 1 0 1 0 1 0 0	0

APPLICATIONS/ ADVANTAGES OF ASYNCHRONOUS TRANSMISSION

- Asynchronous transmission is well suited for keyboard type-terminals and paper tape devices.
- The advantage of this method is that it ***does not require any local storage*** at the terminal or the computer as transmission takes place character by character.
- Asynchronous transmission is best suited to Internet traffic in which information is transmitted in short bursts.
- This type of transmission is used by modems.

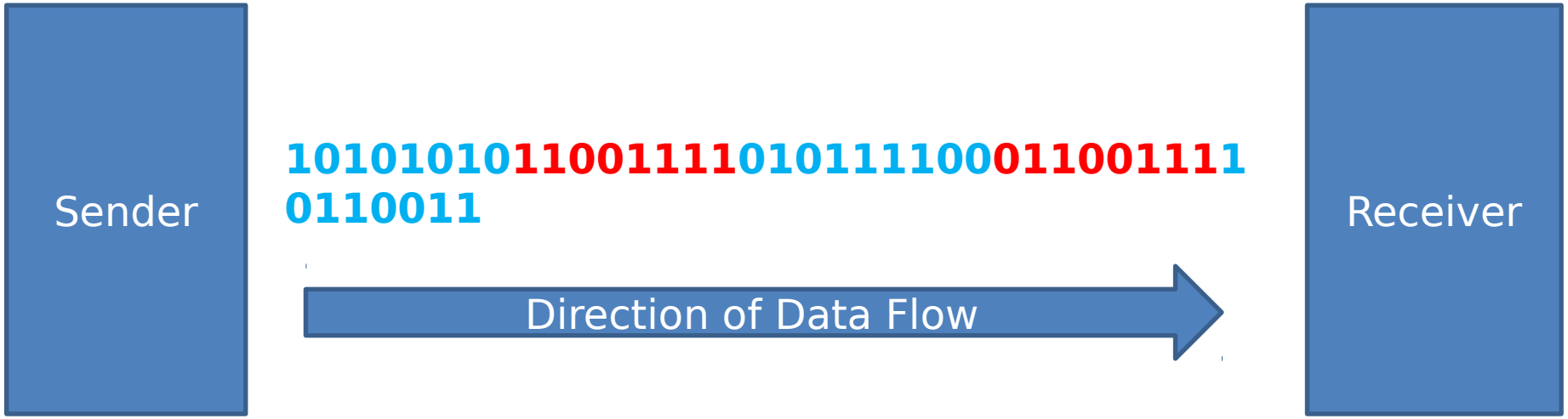
DISADVANTAGES OF ASYNCHRONOUS DATA TRANSMISSION

- This method is ***less efficient*** and ***slower*** than synchronous transmission due to the overhead of extra bits and insertion of gaps into bit stream.
- Successful transmission inevitably depends on the recognition of the start bits. These bits can be missed or corrupted.

SYNCHRONOUS TRANSMISSION

- In this method bit stream is combined into longer frames that may contain ***multiple bytes.***
- There is no gap between the various bytes in the data stream.
- In the absence of start & stop bits, bit synchronization is established between sender & receiver by '***timing***' the transmission of each bit.
- Since the various bytes are placed on the link without any gap, it is the responsibility of receiver to separate the bit stream into bytes so as to reconstruct the original information.
- In order to receive the data error free, the receiver and sender operates at the same clock frequency.

SYNCHRONOUS TRANSMISSION: BLOCK DIAGRAM



ADVANTAGE OF SYNCHRONOUS TRANSMISSION

Advantage:

- This method is ***faster*** as compared to asynchronous as there are no extra bits (start bit & stop bit) and also there is no gap between the individual data bytes.

Disadvantages:

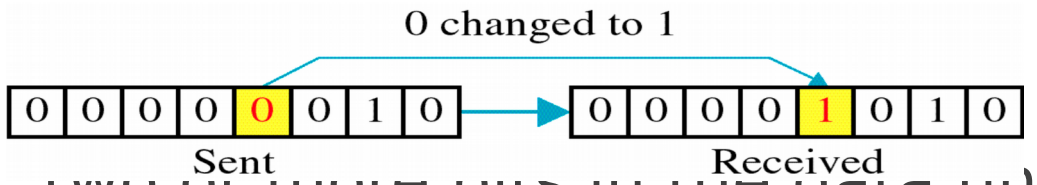
- It is costly as compared to asynchronous method. It ***requires local buffer storage*** at the two ends of line to assemble blocks
- It also requires ***accurately synchronized clocks*** at both ends. This lead to increase in the cost.

ERROR

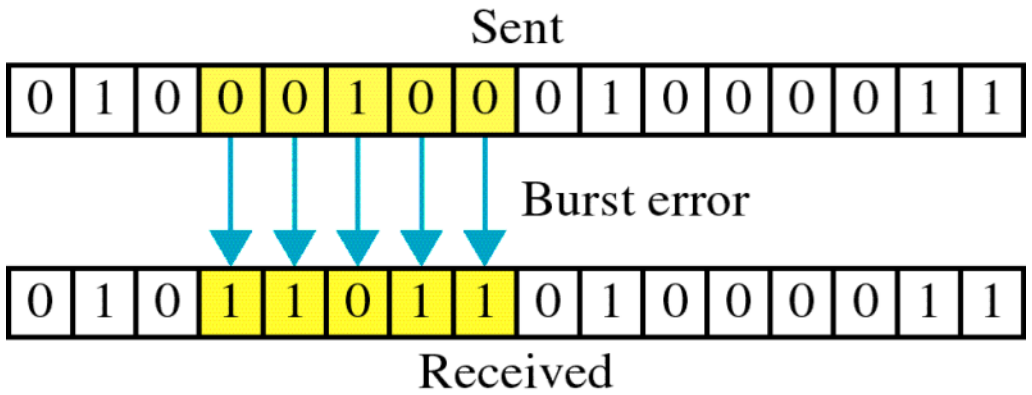
- Networks must be able to transfer data from one device to another with acceptable ***accuracy.***
- For most applications, a system must guarantee that the data received are identical to the data transmitted.
- Data can be corrupted during transmission.
- Some applications require that ***errors*** be detected and corrected.
- Small level of errors are acceptable in applications like audio or video transmissions.
- High level of accuracy is expected in text transmission.

Types of Errors

- **Single bit error**: only one bit in the data unit has changed.



- **Burst error**: two or more bits in the data unit have changed.



ERROR DETECTING CODES

- Whenever a message is transmitted, it may get scrambled by ***noise*** or data may get corrupted.
- Basic approach used for error detection is the use of ***redundancy bits***, where additional bits are added to facilitate detection of errors.

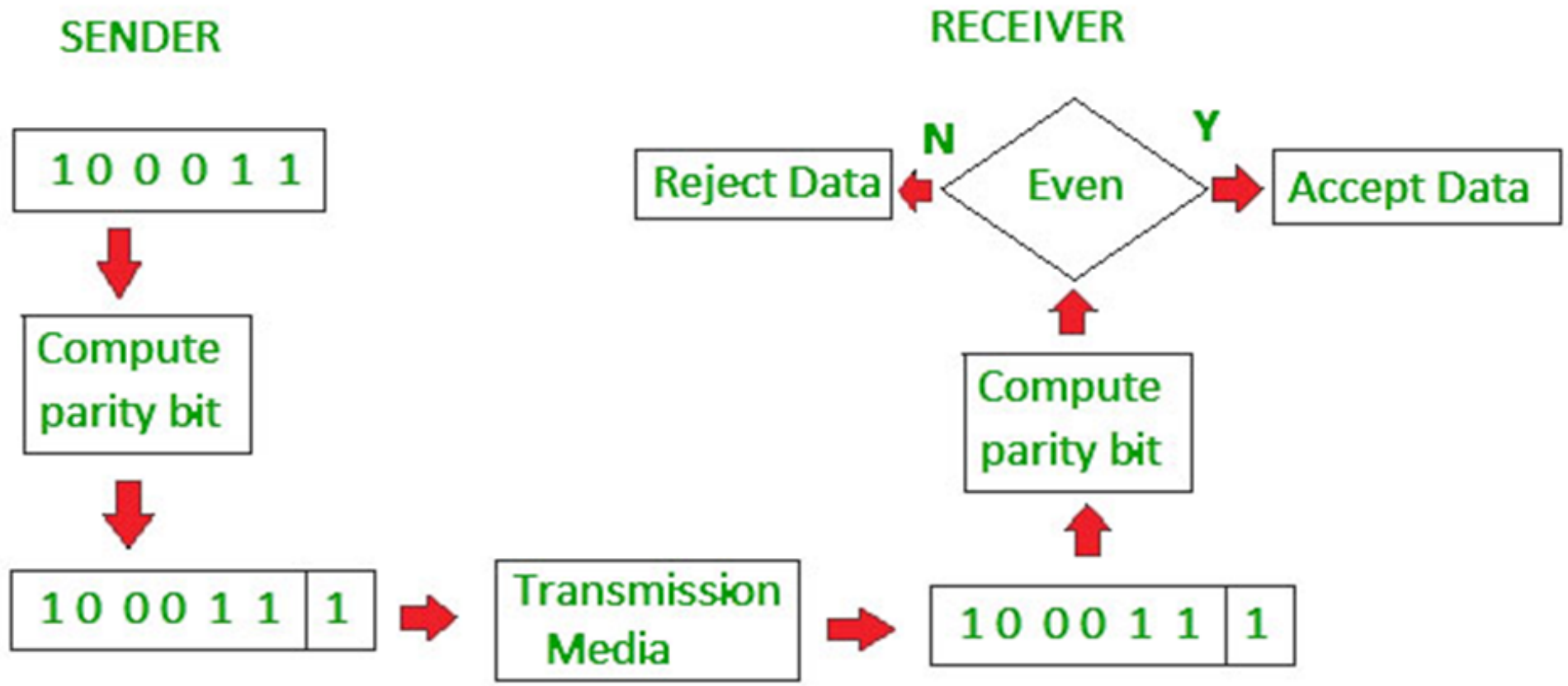
POPULAR TECHNIQUES FOR ERROR DETECTION

- Simple Parity check
- Two-dimensional Parity check
- Checksum
- Cyclic redundancy check

SIMPLE PARITY CHECK

- Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :
- 1 is added to the block if it contains odd number of 1’s, and
- 0 is added if it contains even number of 1’s
- This scheme makes the total number of 1’s even, that is why it is called ***even parity checking***.

SIMPLE PARITY CHECK: EXAMPLE



TWO-DIMENSIONAL PARITY CHECK

- Parity check bits are calculated for each row, which is equivalent to a simple parity check bit.
- Parity check bits are also calculated for all columns, then both are sent along with the data.
- At the receiving end these are compared with the parity bits calculated on the received data.

TWO-DIMENSIONAL PARITY

Original Data

10011001	11100010	00100100	10000100

Row parities

10011001	0
11100010	0
00100100	0
10000100	0
11011011	0

Column parities

100110010	111000100	001001000	100001000	110110110

Data to be sent

CHECKSUM

- In checksum error detection scheme, the data is divided into ***‘k’*** segments each of ***‘m’*** bits.
- In the sender’s end the segments are added using 1’s complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.
- At the receiver’s end, all received segments are added using 1’s complement arithmetic to get the sum. The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded.

CHECKSUM: EXAMPLE

Original Data

10011001	11100010	00100100	10000100
1	2	3	4

k=4, m=8

Sender

```
1        10011001
2        11100010
        (1)01111011
                  1
         01111100
3        00100100
         10100000
4        10000100
        (1)00100100
                  1
Sum:     00100101
CheckSum:11011010
```

Reciever

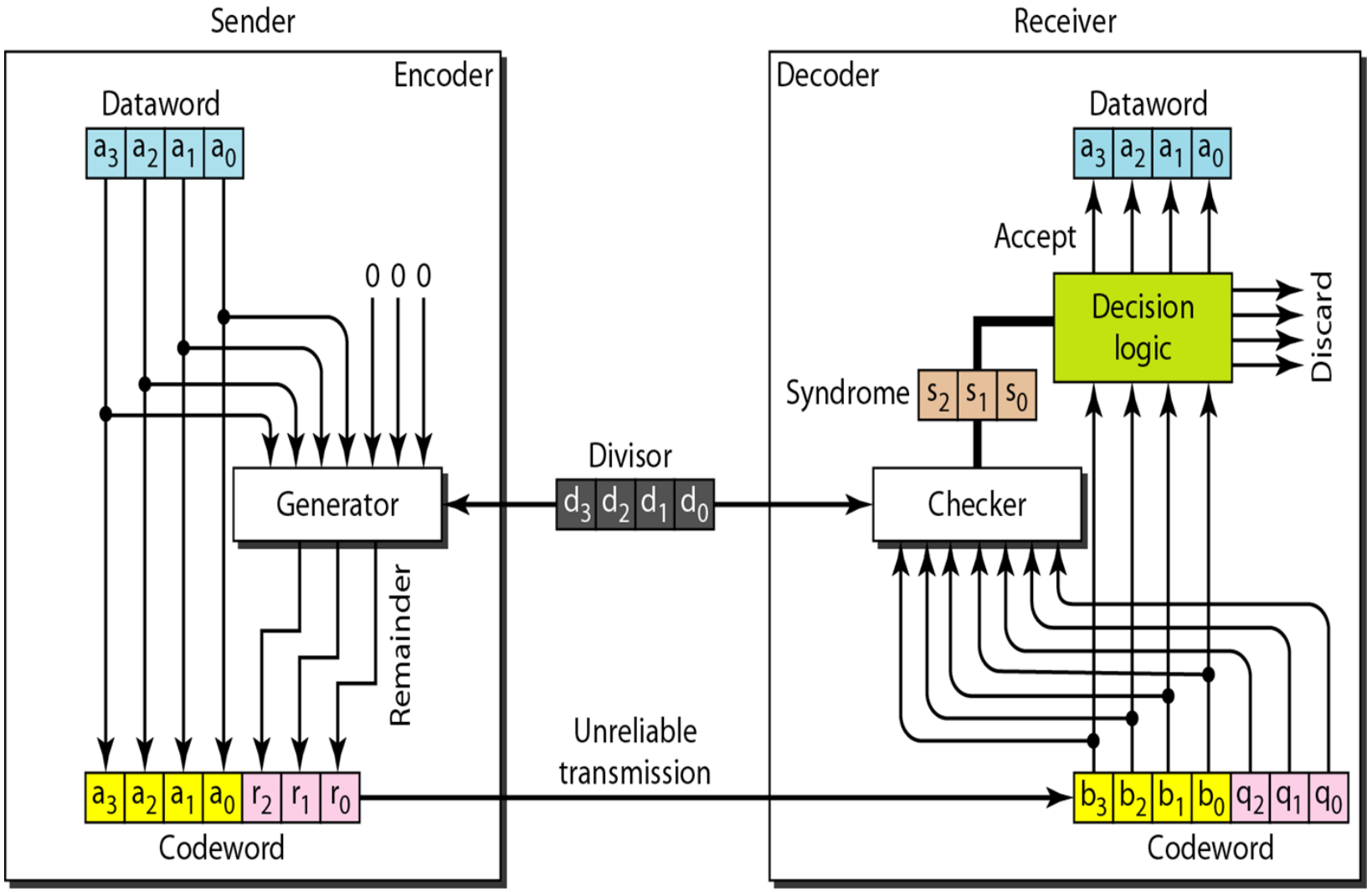
```
1        10011001
2        11100010
        (1)01111011
                  1
         01111100
3        00100100
         10100000
4        10000100
        (1)00100100
                  1
         00100101
         11011010
Sum:     11111111
Complement:00000000
```

Conclusion: Accept Data

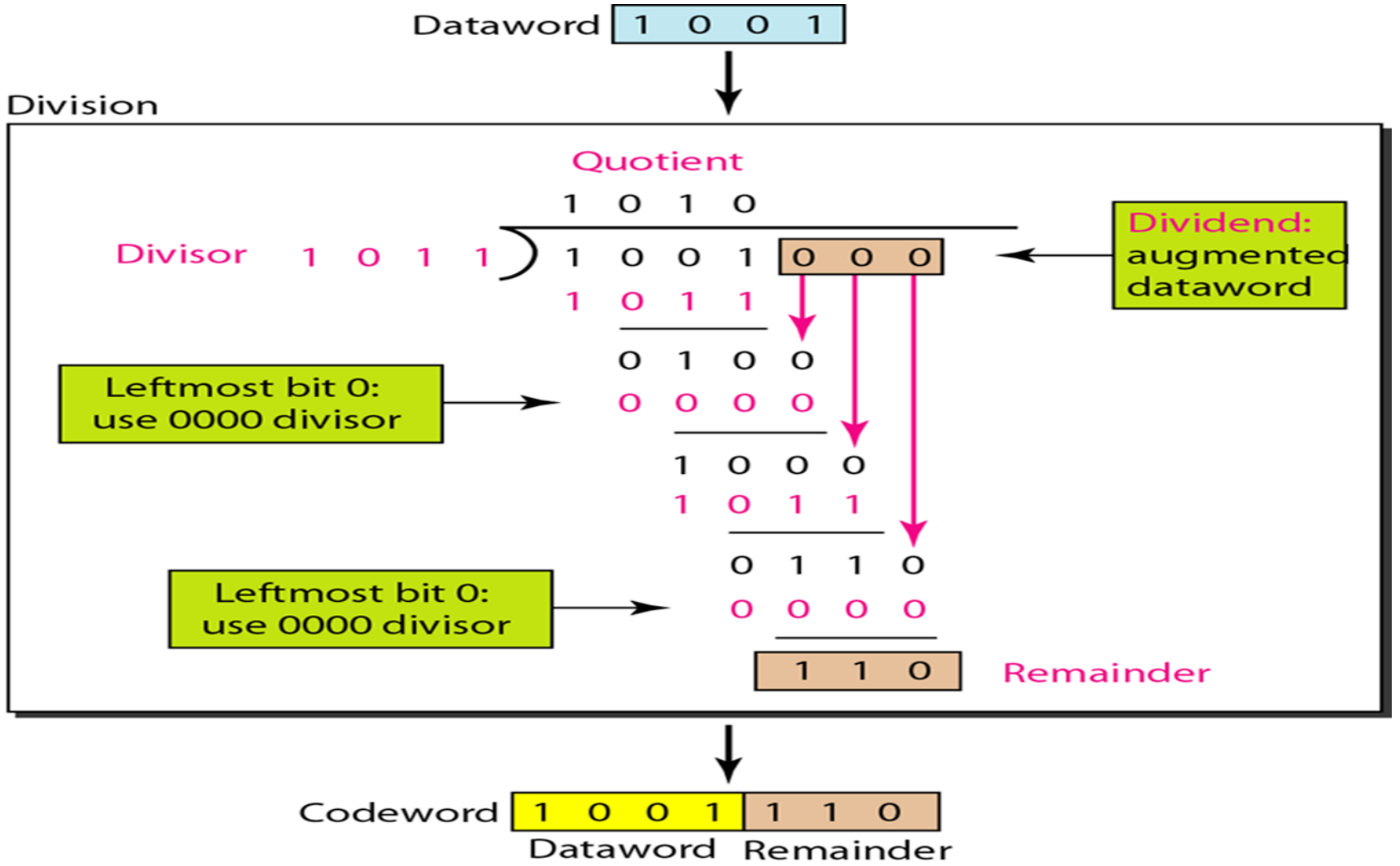
CYCLIC REDUNDANCY CHECK (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

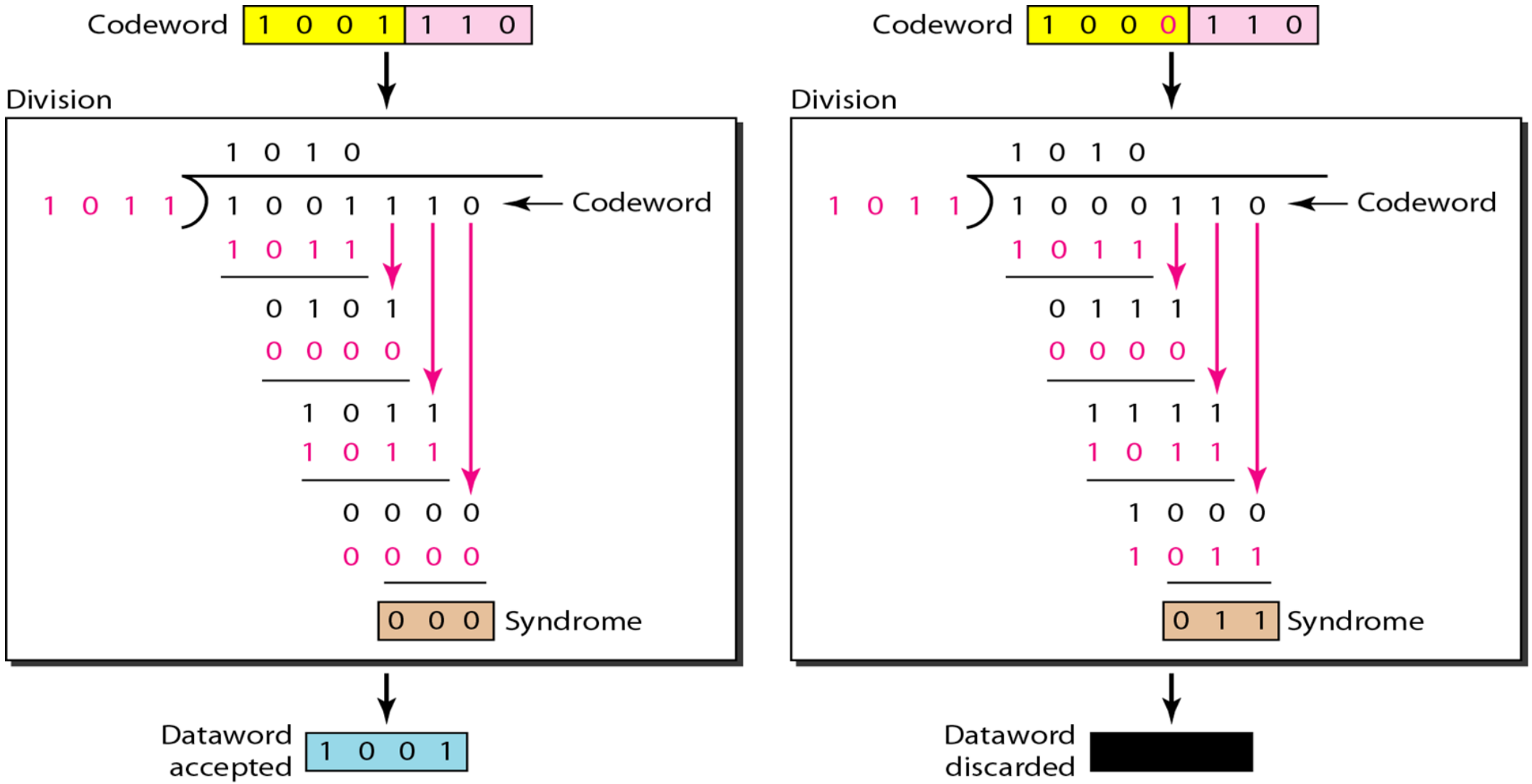
CRC ENCODER AND DECODER



DIVISION IN CRC ENCODER



DIVISION IN CRC DECODER



STANDARD POLYNOMIALS

NAME	POLYNOMIAL	APPLICATION
CRC-8	$x^8 + x^2 + x + 1$	ATM Header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs
CRC-32	$x^8 + x^2 + x + 1$	ATM Header
	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
	$x^{16} + x^{12} + x^5 + 1$	HDLC
	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

ADVANTAGES OF CYLIC CODES

- Cyclic codes have a very good performance in detecting **single-bit errors**, **double bit errors**, an **odd number of errors**.
- They can easily be implemented in ***hardware*** and ***software.***
- They are especially ***fast*** when implemented in ***hardware.***

ALGORITHM

- The message bits to be transmitted are appended with **'c' (highest degree of generator polynomial-1)** zero bits; this ***augmented message*** is the dividend.
- A predetermined **c+1**-bit binary sequence, called the ***generator polynomial***, is the divisor.
- The checksum is the ***c-bit remainder*** that results from the division operation.
- Consider a message represented by the polynomial **M(x).**
- Consider a generating polynomial **G(x).**
- This is used to generate a **CRC = C(x) to be appended to M(x). Note this G(x) is prime.**
- **Multiply M(x) by highest power in G(x)**. i.e. **Add So much zeros to M(x).**
- **Divide the result by G(x). The remainder = C(x).**
- **Transmit: T(x) = M(x) + C(x)**
- Receiver end: **Receive T(x). Divide by G(x), should have remainder 0.**

SOURCE CODE

```
#include<stdio.h>
 int message[100];
 int b[100];
 int j;
 int length;
// Generator Polynomial: g(x) = x^16 + x^12 + x^5 + 1
// here x^16 , means x to the power of 16,
// to remember degree of x, array gp is used
// x to the power of 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 int gp[17]={ 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1};
 void divide( int k )
  {
    int i;
    int j;
    int count=0;
```

```
for( i=0; i<k; i++ )
    {
     if( message[i] == gp[0] )
      {
       for( j=i; j<17+i; j++ )
        {
         message[j] = message[j] ^ gp[count++];
        }
      }

     count=0;
    }
  }

int main()
  {
    int i; // index, to iterate through message

    printf("\n Enter the length of Data Frame : ");
    scanf("%d",&length);

    printf("\n Enter the Message (in bits, i.e. 0's and 1's)");
    printf("\n But each bit separated by space or a new line: ");
```

```
for( i=0; i < length; i++ )
     {
      scanf("%d",&message[i]);
     }
for( i=0; i < 16; i++ ) // Append r(16) degree zeros to message bits
     {
      message[length++] = 0;
     }                  // Now it is content of message followed by 16 0's

    for( i=0; i < length; i++ )    // Xr.M(x) (ie. Msg+16 Zeros)
     {
      b[i] = message[i];
     }

    divide( length - 16 ); //No of times to be divided i.e. message length

    for( i=0; i < length; i++ )
     {
      b[i] = b[i] ^ message[i];     //MOD 2 Substraction
     }
```

```
printf("\n Data to be transmitted : ");
   for( i=0; i < length; i++ )
    {
      printf("%2d", b[i]);
    }

   printf("\n\n Enter the Received Data : ");
   for( i=0; i < length; i++ )
    {
      scanf("%d", &message[i] );
    }

   divide( length - 16 ); //No of times to be divided i.e. message length
```

```
for( i=0; i < length; i++ )
     {
       if( message[i] != 0 )
        {
          printf("\n ERROR in Received Data\n");
          return 0;
        }
     }

    printf("\n Data Received is ERROR FREE\n");
    return 0;
  }
```

EXPECTED OUTPUT
CASE:1

Enter the length of Data Frame : 8

Enter the Message (in bits, i.e. 0's and 1's)

But each bit separated by space or a new line: 1 1 0 0 1 0 1 0

Data to be transmitted : 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0

On receiver end, enter the Received Data : 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0

Data Received is ERROR FREE

EXPECTED OUTPUT
CASE:2

Enter the length of Data Frame : 8

Enter the Message (in bits, i.e. 0's and 1's)

And each bit separated by space or a new line: 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0

Data to be transmitted : 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0

On receiver end, enter the Received Data : 0 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0

ERROR in Received Data

EXPECTED OUTCOME

Students will be able to implement error detection code using CRC-CCITT 16 bit technique.

```
/*
        Computer Networks Laboratory (Lab) 15CSL77
        6. Write a program for error detecting code using CRC-CCITT (16-bits).
*/

#include<stdio.h>

int message[100];
int dataToBeSent[100];
int length;

// Generator Polynomial: g(x) = x^16 + x^12 + x^5 + 1
// x to the power of          16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
int generatorPolynomial[17]={ 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1};
```

```
void divide( int k )
 {
   int i, j, count=0;

   for( i=0; i < k; i++ )          // Till message
    {
      if( message[i] == generatorPolynomial[0] )
         for( j=i; j < 17+i; j++ )      // XOR message and generator polynomial
            message[j] = message[j] ^ generatorPolynomial[count++];

      count=0;
    }
 }
```

```c
int main()
 {
   int i; // index, to iterate through message
   // Data Frame = message
   printf("\n Enter the length of Data Frame : ");     scanf("%d",&length);

   printf("\n Enter the Message (in bits, i.e. 0's and 1's)");
   printf("\n And each bit separated by space or a new line: ");
   for( i=0; i < length; i++ )        scanf("%d",&message[i]);

   for( i=0; i < 16; i++ ) // Append r(16) degree zeros to message bits
      message[length++] = 0;  // update length while appending zeros

   for( i=0; i < length; i++ )    // Copy message
      dataToBeSent[i] = message[i];

   divide( length - 16 ); // Perform XOR operation    ( why length - 16? )

   for( i=0; i < length; i++ )           // message . crc
      dataToBeSent[i] = dataToBeSent[i] ^ message[i];
```

```
  printf("\n Data to be transmitted : ");
  for( i=0; i < length; i++ )        printf("%2d", dataToBeSent[i]);

  printf("\n\n On receiver end, enter the Reveived Data : ");
  for( i=0; i < length; i++ )        scanf("%d", &message[i] );

  divide( length - 16 ); // Perform XOR operation

  for( i=0; i < length; i++ )
     if( message[i] != 0 )
      {
        printf("\n ERROR in Recived Data\n");
        return 0;
      }

  printf("\n Data Recived is ERROR FREE\n");
  return 0;
}
```