Experiment No -5

AIM: To implement error detection and correction techniques.

OBJECTIVES:

- To study error detection and correction.
- To understand CRC and Hamming code techniques.
- To implement CRC and Hamming code techniques.

PROBLEM STATMENT:

Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode

OUTCOMES:

CO2: Analyze data flow between peer to peer in an IP network using Application, Transport and Network Layer Protocols

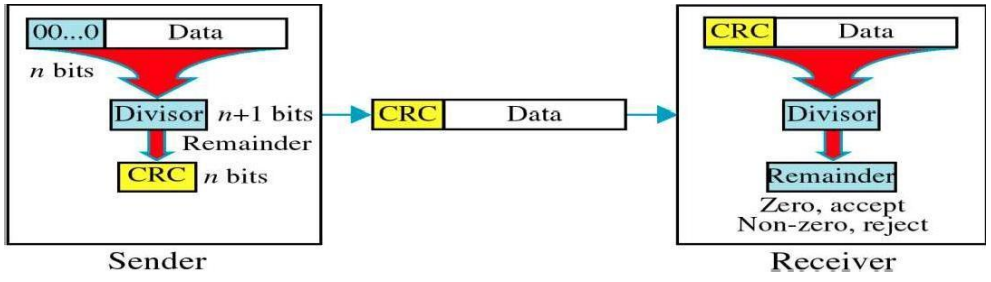
SOFTWARE & HARDWARE REQUIREMENTS:

- Software: Jdk and wireshark
- Hardware: Open source Linux operating system.

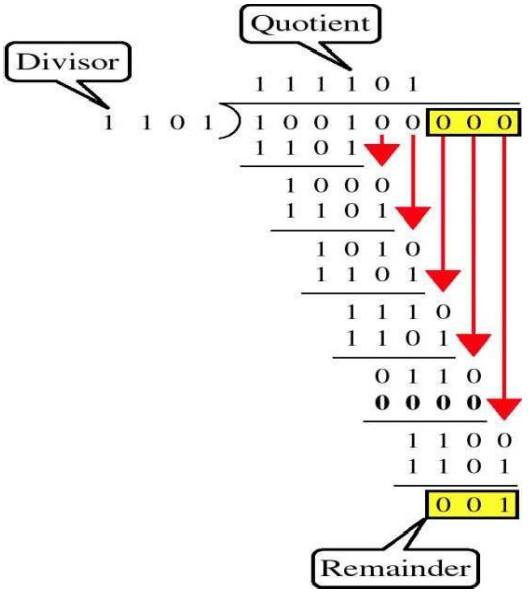
THEORY:

Cyclic Redundancy Check: CRC

- Given a k-bit frame or message, the transmitter generates an n-bit sequence, known as a *frame check sequence (FCS)*, so that the resulting frame, consisting of (k+n) bits, is exactly divisible by some predetermined number.
- The receiver then divides the incoming frame by the same number and, if there is no remainder, assumes that there was no error.



Example:



Hamming code:

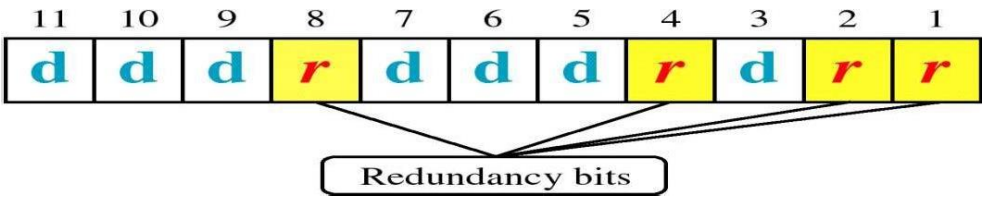
- Hamming codes are a family of linear error-correcting codes that generalize the Hamming(7,4)-code
- Invented by Richard Hamming in 1950

Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors

General algorithm:

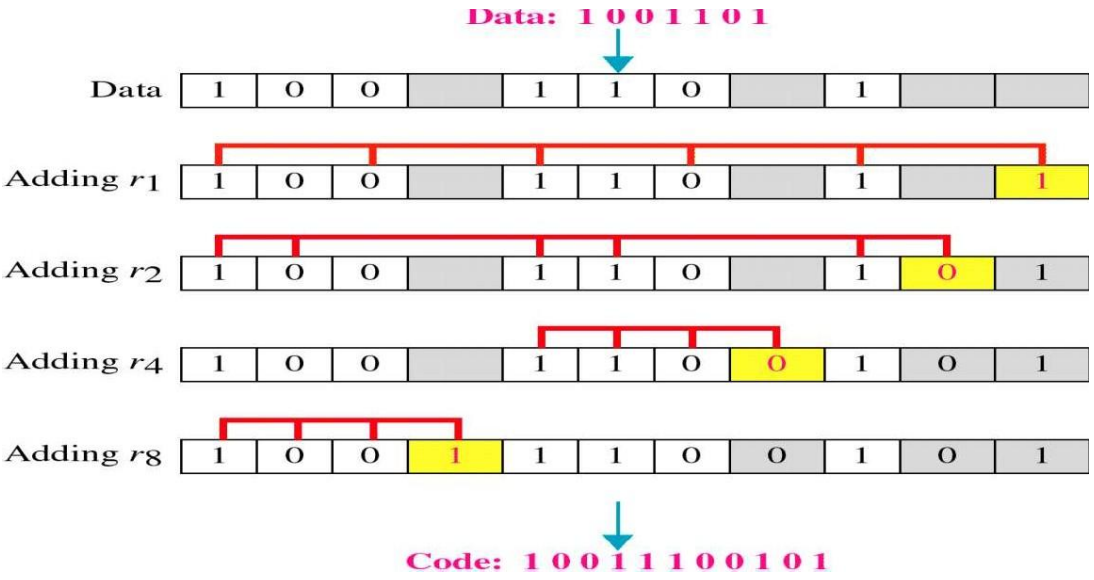
- The following general algorithm generates a single-error correcting (SEC) code for any number of bits.
- Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
- Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
- All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)
- All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
- Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

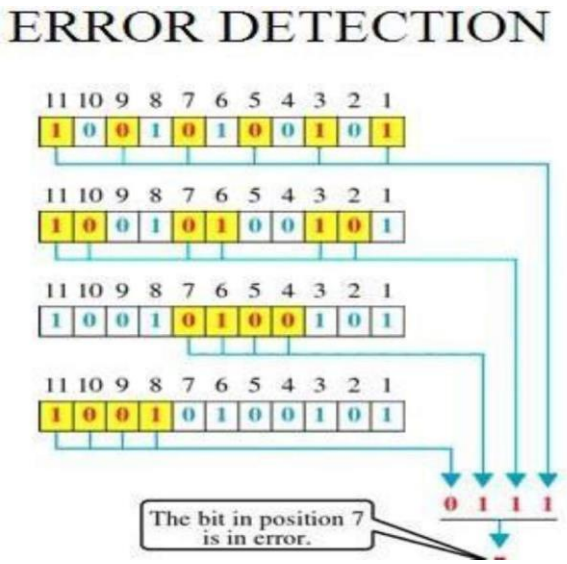
- Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.
- Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have 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.
- In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.



Example

Error detection & correction:





Example of Hamming Code Generation

Suppose a binary data 1001101 is to be transmitted. To implement hamming code for this, following steps are used:

1. Calculating the number of redundancy bits required. Since number of data bits is

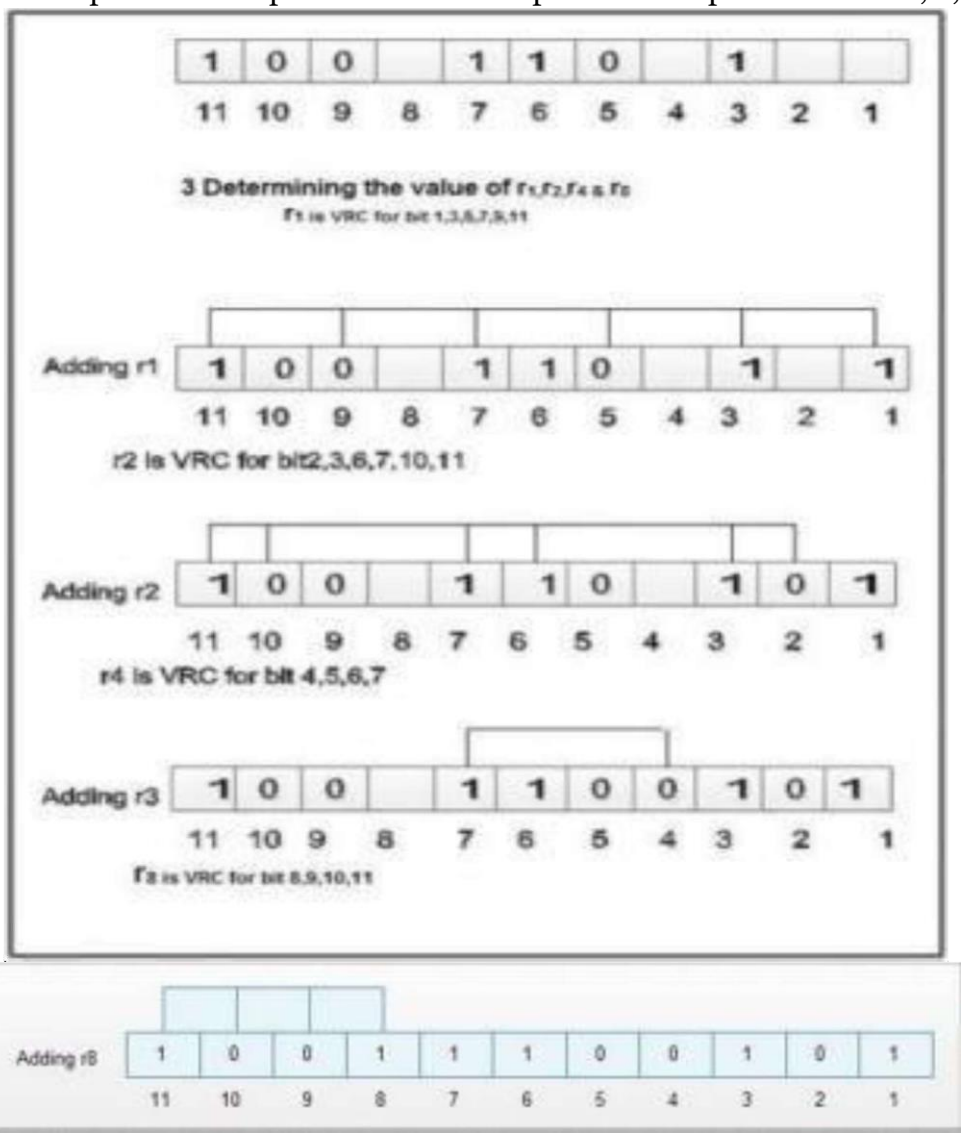
7, the value of r is calculated as

$$2^r \geq m + r + 1$$

$$2^4 \geq 7 + 4 + 1$$

Therefore no. of redundancy bits = 4

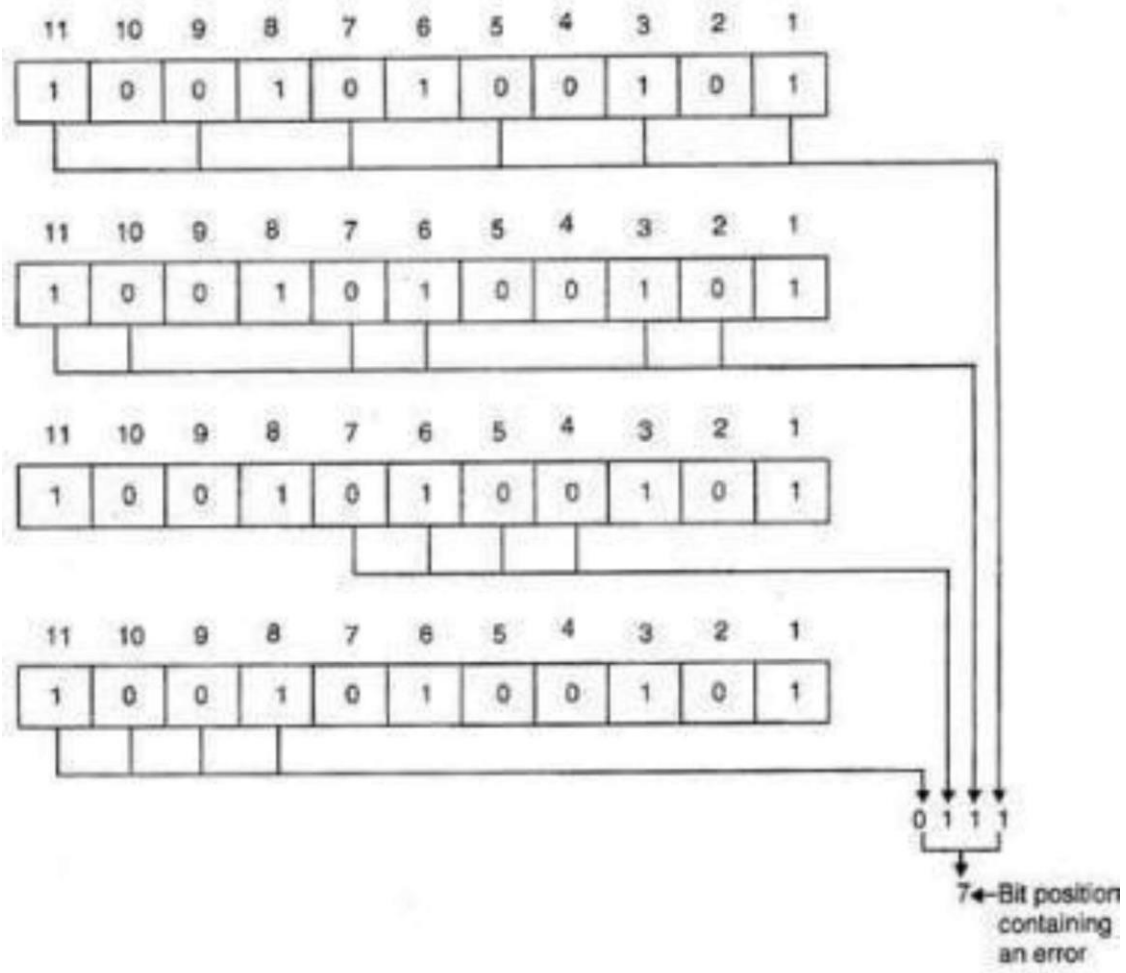
3. Determining the positions of various data bits and redundancy bits. The various r bits are placed at the position that corresponds to the power of 2 *i.e.* 1, 2, 4, 8



4. Thus data 1 0 0 1 1 1 0 0 1 0 1 with be transmitted.

Error Detection & Correction

Considering a case of above discussed example, if bit number 7 has been changed from 1 to 0.The data will be erroneous.



Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes

```
#include<stdio.h>
//#include<conio.h>

void main() {     int data[10];
   int dataatrec[10],c,c1,c2,c3,i;
  //  clrscr();
   printf("Enter 4 bits of data one by one\n");        scanf("%d",&data[0]);
scanf("%d",&data[1]);      scanf("%d",&data[2]);      scanf("%d",&data[4]);

   //Calculation      of      even      parity                    data[6]=data[0]^data[2]^data[4];
data[5]=data[0]^data[1]^data[4];       data[3]=data[0]^data[1]^data[2];

   printf("\nEncoded data is\n");
   for(i=0;i<7;i++)
      printf("%d",data[i]);

   printf("\n\nEnter received data bits one by one\n");
   for(i=0;i<7;i++)          scanf("%d",&dataatrec[i]);

   c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];         c=c3*4+c2*2+c1 ;

   if(c==0) {
      printf("\nNo error while transmission of data\n");
   }    else {
      printf("\nError on position %d",c);

      printf("\nData sent : ");         for(i=0;i<7;i++)
         printf("%d",data[i]);

      printf("\nData  received  :  ");                    for(i=0;i<7;i++)
printf("%d",dataatrec[i]);

      printf("\nCorrect message is\n");

      //if errorneous bit is 0 we complement it else vice versa          if(dataatrec[7-c]==0)
dataatrec[7-c]=1;          else
         dataatrec[7-c]=0;

      for (i=0;i<7;i++) {            printf("%d",dataatrec[i]);
//getch();
      }
   }
}
```

/* output

```
proglab@proglab:~$ ./a.out Enter 4 bits of data one by
one
1
```

```
0
1
1

Encoded data is
1010101

Enter received data bits one by one
0
1
1
0
1
1
0

Error on position 2
Data sent : 1010101
Data received : 0110110
Correct message is
0110100
```

CONCLUSION: - Hence we have implemented CRC and Hamming code.