Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO #03

Course Title: Data Communication Lab

Course Code:CSE-308 Section:221-D10

Lab Experiment Name: Implementation of Hamming code Error Detection and Correction.

<u>Student Details</u>

Name		ID
1.	Masum Hossain	221902164

Lab Date : 21/05/2024

Submission Date : 26/05/2024

Course Teacher's Name : Rusmita Halim Chaity

<u>Lab Report Status</u>

Marks: ………………………………… Signature:.....................

Comments:... Date:..............................

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementing a program to first calculate the number of parity bits needed and with that, generate the hamming code. Then check if the generated hamming code has any errors in transmission and correct if there are any errors.

2. OBJECTIVES

This lab report aims at the implementation of Hamming code for both error detection and correction .But if the message contains only a single bit error and it is the main drawback of the Hamming code.

This lab mainly focuses on :

- To understand the concept of Hamming Code and its significance in digital communication.
- To calculate the number of parity bits required for a given message bit stream.
- To generate the Hamming Code for the given message bit stream.
- To detect and correct errors that may occur during transmission.

3. ANALYSIS

The Hamming Code is a technique used in digital communication to detect and correct errors in the transmitted data. It is a type of linear error-correcting code that adds redundant parity bits to the message bit stream, allowing for the detection and correction of single-bit errors.

Algorithm:

1. Input the length of the message bit stream and the message bits (0s and 1s).
2. Calculate the number of parity bits required using the formula: n = ceil(log2(m+n+1)), where m is the length of the message bit stream, and n is the number of parity bits.
3. Generate the Hamming Code by inserting parity bits at appropriate positions in the message bit stream.
4. Calculate the parity bits based on the even parity rule, ensuring that the sum of the bits in the respective parity groups is even.
5. Transmit the Hamming Code to the receiver.
6. At the receiver's end, recalculate the parity bits and compare them with the received parity bits.
7. If the calculated and received parity bits differ, an error has occurred during transmission.
8. Determine the position of the error using the binary representation of the differing parity bits.
9. Correct the error by flipping the bit at the identified position.
10. Output the corrected Hamming Code.

4. IMPLEMENTATION

In this section, we critically analyze the implementation of the Hamming code for error detection and correction.The program takes the length of the message bit stream and the message bits as input from the user. It then calculates the number of parity bits required and generates the Hamming Code by inserting parity bits at appropriate positions, following the even parity rule.

After generating the Hamming Code, the program simulates transmission by prompting the user to enter the received Hamming Code.

It recalculates the parity bits for the received code and compares them with the transmitted parity bits. If any discrepancy is found, it identifies the position of the error using the binary representation of the differing parity bits. It then corrects the error by flipping the bit at the identified position and outputs the corrected Hamming Code.

Code for both Error detection and correction:

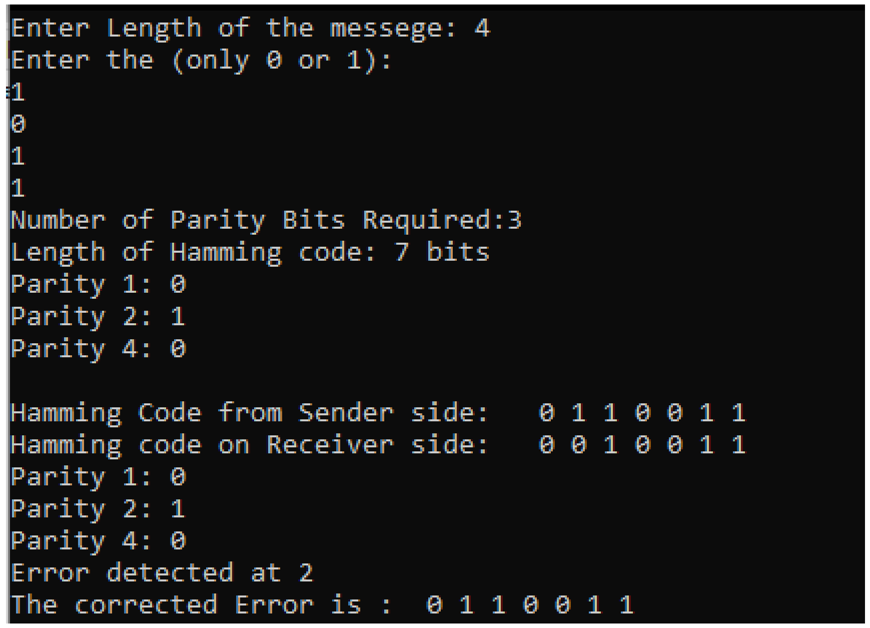
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void main()
{
   int maxp=10;
   int a[50],temp[100],temp2[100];
   int t,i,j,k,nd,n,nh,sum=0,pos=0;

   printf("Enter Length of the message: ");
   scanf("%d",&nd);
   printf("Enter the (only 0 or 1)");
   for(i=0;i<nd;i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0,j=0;i<nd;i++)
   {
      for(k=0;k<maxp;k++)
      {
         t=pow(2,k)-1;
         if(j==t)
         {
            temp[j]=0;
            j++;
         }
      }
      temp[j]=a[i];
      j++;
   }
   nh=j;
    n=nh-nd;
    printf("Number of Parity Bits Required:%d \n",n);
   printf("Length of Hamming code: %d bits\n",nh);
   int b[n];
   int m=n-1;
   for(k=0;k<n;k++)
   {
      t=pow(2,k)-1;

      for(i=t;i<nh;)
      {
         for(j=0;j<=t;j++)
         {
            sum=sum+temp[i];
            i++;
            if(i>=nh)
               break;
         }
         if(i>=nh)
            break;

         for(j=0;j<=t;j++)
         {
            i++;
            if(i>=nh)
               break;
         }
         if(i>=nh)
            break;
      }
      temp[t]=sum%2;
      sum=0;
      printf("Parity %d: %d\n",t+1,temp[t]);
   }
   printf("\nHamming Code from Sender side:   ");
   for(i=0;i<nh;i++)
   {
      printf("%d ",temp[i]);
   }
   printf("\nHamming code on Receiver side:   ");
   for(i=0;i<nh;i++)
   {
      scanf("%d",&temp2[i]);
   }
```

```
sum=0;
for(k=0;k<n;k++)
{
   t=pow(2,k)-1;

   for(i=t;i<nh;)
   {
      for(j=0;j<=t;j++)
      {
         sum=sum+temp2[i];
         i++;
         if(i>=nh)
            break;
      }
      if(i>=nh)
         break;

      for(j=0;j<=t;j++)
      {
         i++;
         if(i>=nh)
            break;
      }
      if(i>=nh)
         break;
   }
   b[m]=sum%2;
   sum=0;
   printf("P%d: %d\n",t+1,b[m]);
   m--;
}
for(m=0;m<n;m++)
{
   pos=pos+b[n-m-1]*pow(2,m);
}
printf("Error detected at %d :\n",pos);
if(temp2[pos-1]==0)
   temp2[pos-1]=1;
else
   temp2[pos-1]=0;

printf("The corrected Error is :  ");
for(i=0;i<nh;i++)
{
   printf("%d ",temp2[i]);
}}
```

5. OUTPUT

The program has been run several times and it successfully passed all the cases.

Here is a case:



```
Enter Length of the messege: 4
Enter the (only 0 or 1):
1
0
1
1
Number of Parity Bits Required:3
Length of Hamming code: 7 bits
Parity 1: 0
Parity 2: 1
Parity 4: 0

Hamming Code from Sender side:   0 1 1 0 0 1 1
Hamming code on Receiver side:   0 0 1 0 0 1 1
Parity 1: 0
Parity 2: 1
Parity 4: 0
Error detected at 2
The corrected Error is :  0 1 1 0 0 1 1
```

Fig01: Showing the Hamming code error detection and correction.

6. ANALYSIS AND DISCUSSION

The program generates Hamming Codes for a message bit stream, efficiently detecting and correcting errors using even parity. At the receiver's end, it recalculates parity bits, comparing them with the received ones. Discrepancies reveal error positions via the binary form of differing parity bits, allowing error correction by flipping the bit at that position.

The algorithm has a time complexity of O(n log n) and a space complexity of O(n) due to the necessary bit stream and parity calculations.

7.SUMMARY

This lab experiment successfully implements the Hamming Code technique for error detection and correction in digital communication, providing a practical understanding of this important concept.