PROJECT REPORT

IMPLEMENTATION OF HAMMING CODE FOR SINGLE BIT ERROR DETECTION

HAMMING CODE:

"Hamming code is a set of error-correction codes that can be used to ***detect and correct the errors*** that can occur when the data is moved or stored from the sender to the receiver. It is technique developed by ***R.W. Hamming for error correction***."

SOFTWARE:

- Visual Studio 2010 (.Net Application)

LANGUAGE:

- C#

OVERVIEW OF PROJECT:

- In our Project ***"Implementation of Hamming Code for Single Bit Error Detection",*** we are working on twenty-two bit input data bits from the sender. The Program will add parity bits itself for ensuring that data is correct. Error detects and correct at receiver side. Receiver again calculates parity bits for detecting errors and correction. We are working on even parity bit condition. The minimum input is 1 bit and the maximum input is twenty-two bit. The addition of parity bits based upon power of 2. It is ***"Windows Form Application Project"*** which consists of two forms and two classes. Class1 is ***"Hamming_Code_send"*** is receiving data bit in an array and add parity bits. The Class2 is ***"Hamming_Code_Recieving"*** when user enter receive data, class2 takes it an array and then again calculate parity bits for detection of errors, convert parity bits into decimal for finding error bit position. It consist of 53 method, because parity bit addition based on number of data bits, when you enter different data bits, different method call depending upon number of data bits. It is a ***"General Program",*** user can find hamming code for 22 data bits. When program show sending bits, parity bits show in Red Color.

DESCRIPTION

HAMMING CODE:

"Hamming code is a set of error-correction codes that can be used to ***detect and correct the errors*** that can occur when the data is moved or stored from the sender to the receiver. It is technique developed by ***R.W. Hamming for error correction***."

REDUNDANT BITS:

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.
The number of redundant bits can be calculated using the following formula:

2^r ≥ m + r + 1

Where, r = redundant bit, m = data bit

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using:
= 2^4 ≥ 7 + 4 + 1
Thus, the number of redundant bits= 4

PARITY BITS:

A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data are even or odd. Parity bits are used for error detection. There are two types of parity bits:

- **EVEN PARITY BIT**
  In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.

GENERAL ALGORITHM OF HAMMING CODE:

The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100 etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8 etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.
   a. Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant
   position (1, 3, 5, 7, 9, 11, etc).
   b. Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from
   the least significant bit (2, 3, 6, 7, 10, 11, etc).
   c. Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc).

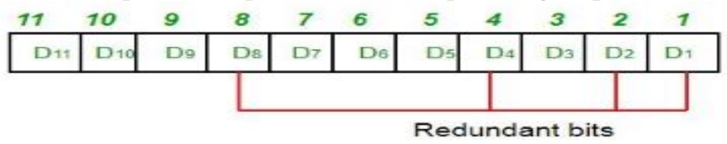
d. Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from
the least significant bit bits (8–15, 24–31, 40–47, etc).

5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is
odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

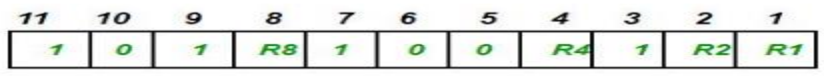
DETERMINING THE POSITION OF REDUNDANT BITS:

These redundancy bits are placed at the positions which correspond to the power of 2.
As in the above example:

1. The number of data bits = 7
2. The number of redundant bits = 4
3. The total number of bits = 11
4. The redundant bits are placed at positions corresponding to power of 2- 1, 2, 4, and 8.

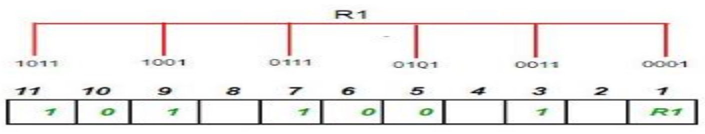


Suppose the data to be transmitted is 1011001, the bits will be placed as follows:

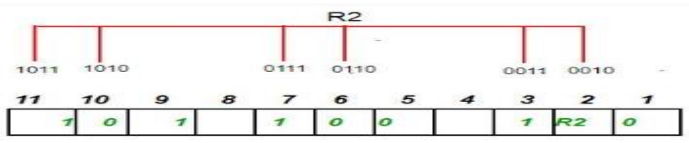


DETERMINING THE PARITY BITS:

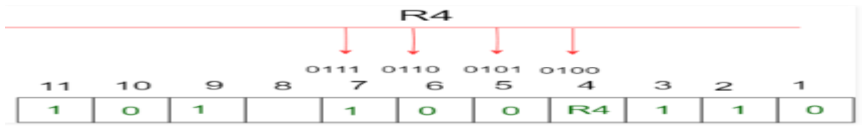
1. R1 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the least significant position.
R1: bits 1, 3, 5, 7, 9, 11.To find the redundant bit R1, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R1 is an even number the value of R1 (parity bit's value) = 0.



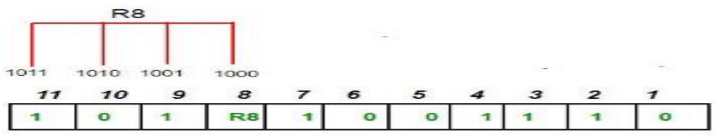
2. R2 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the second position from the least significant bit.
R2: bits 2,3,6,7,10,11. To find the redundant bit R2, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R2 is an odd number the value of R2(parity bit's value)=1



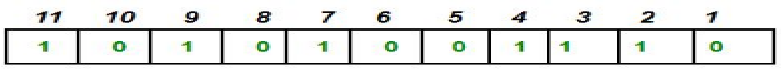
3. R4 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the third position from the least significant bit.
   R4: bits 4, 5, 6, 7.To find the redundant bit R4, we check for even parity. Since the total of 1's in all the bit positions corresponding to R4 is an odd number the value of R4 (parity bit's value) = 1.



4. R8 bit is calculated using parity check at all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit.
   R8: bit 8, 9, 10, 11 . To find the redundant bit R8, we check for even parity. Since the total number of 1's in all the bit positions corresponding to R8 is an even number the value of R8(parity bit's value)=0.



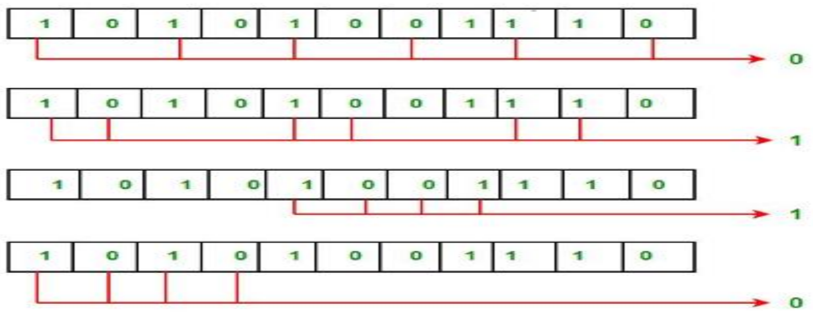
Thus, the data transferred is:



11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	0	1	1	1	0

ERROR DETECTION AND CORRECTION:

Suppose in the above example the 6th bit is changed from 0 to 1 during data transmission, then it gives new parity values in the binary number:



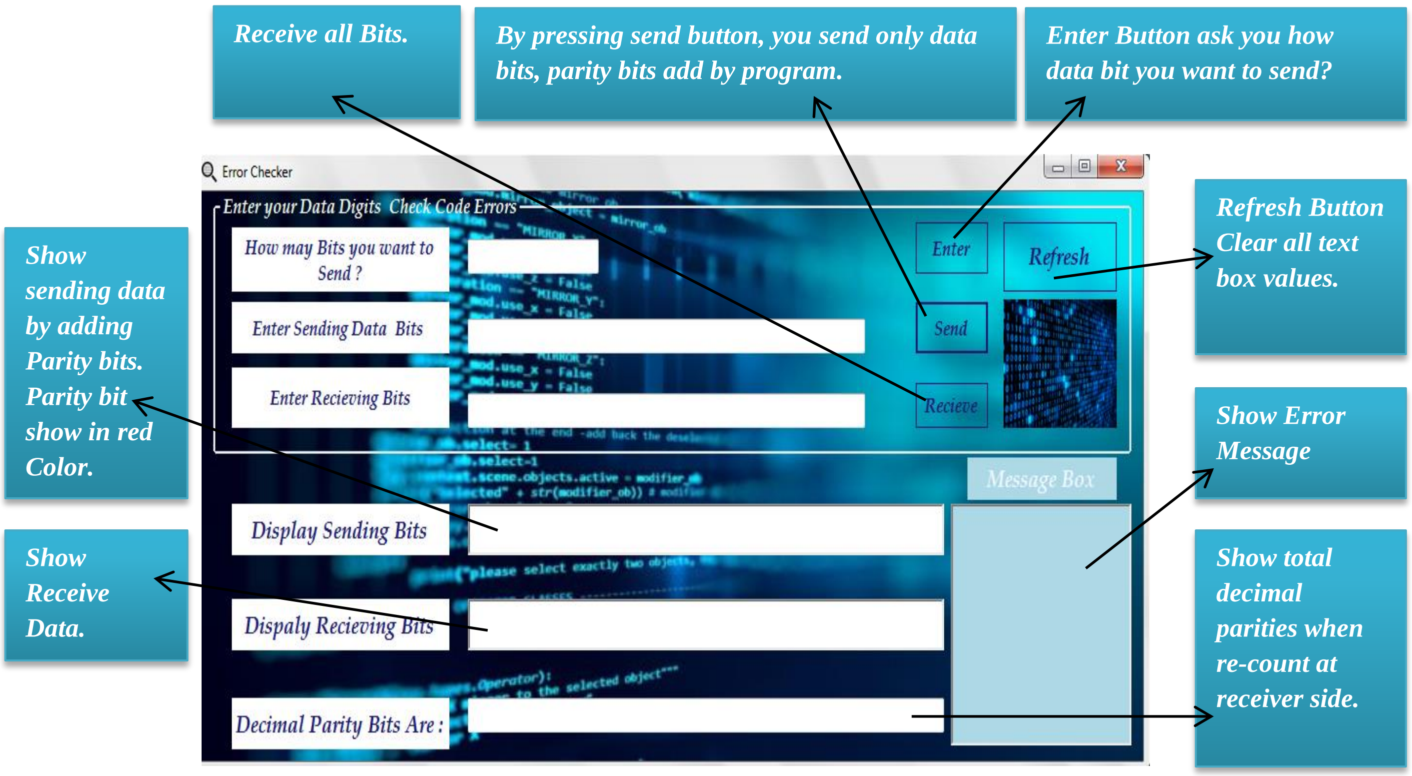
The bits give the binary number as 0110 whose decimal representation is 6. Thus, the bit 6 contains an error. To correct the error the 6th bit is changed from 1 to 0

GRAPHIC USER INTERFACE OF HAMMING CODE IMPLEMENTATION:

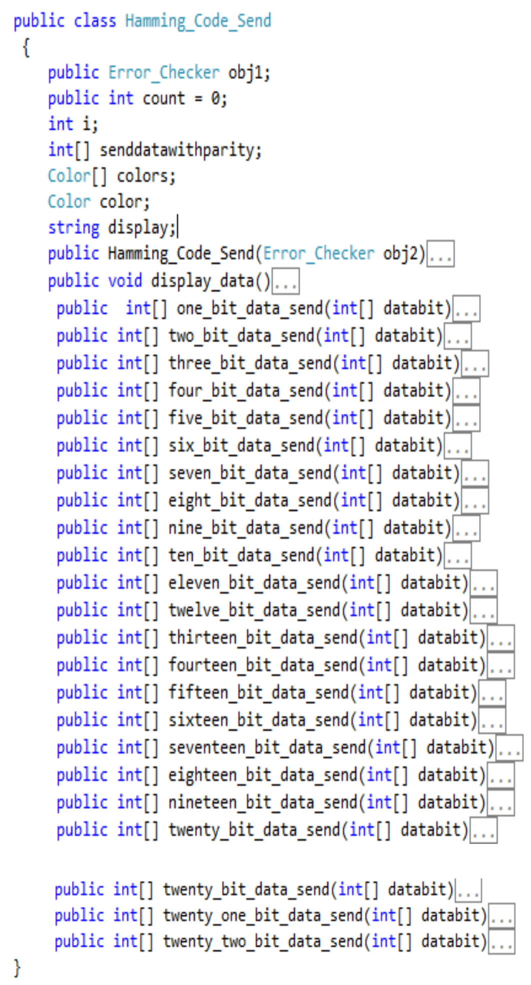
WELCOME FORM:

Welcome
IMPLEMENTATION OF HAMMING CODE FOR SINGLE BIT ERROR DETECTION AND CORRECTION.
Next

HAMMING CODE IMPLEMENTATION FORM:

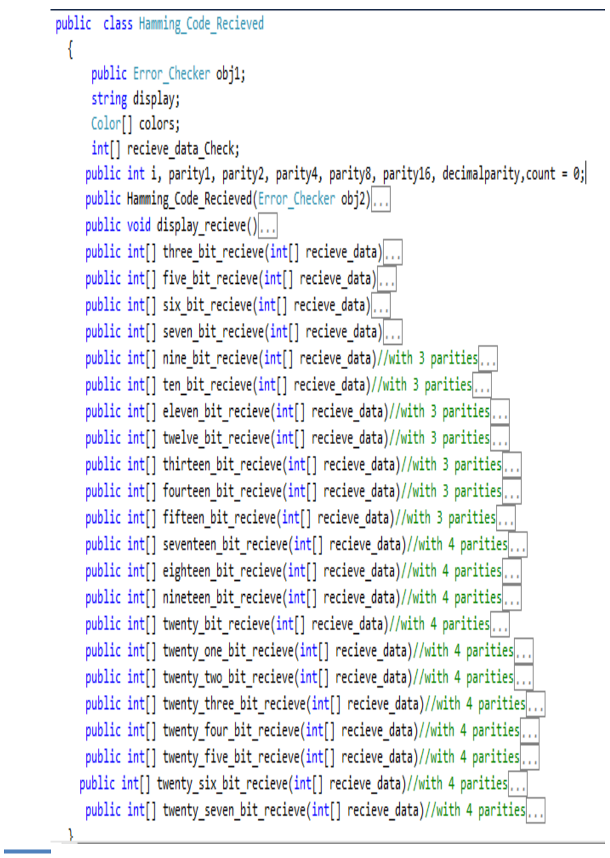

Receive all Bits.
By pressing send button, you send only data bits, parity bits add by program.
Enter Button ask you how data bit you want to send?
Error Checker
Enter your Data Digits Check Code Errors
How may Bits you want to Send ?
Enter Sending Data Bits
Enter Recieving Bits
Enter
Refresh
Send
Recieve
Message Box
Display Sending Bits
Dispaly Recieving Bits
Decimal Parity Bits Are :
Show sending data by adding Parity bits. Parity bit show in red Color.
Show Receive Data.
Refresh Button Clear all text box values.
Show Error Message
Show total decimal parities when re-count at receiver side.

METHODS IN PROJECT:

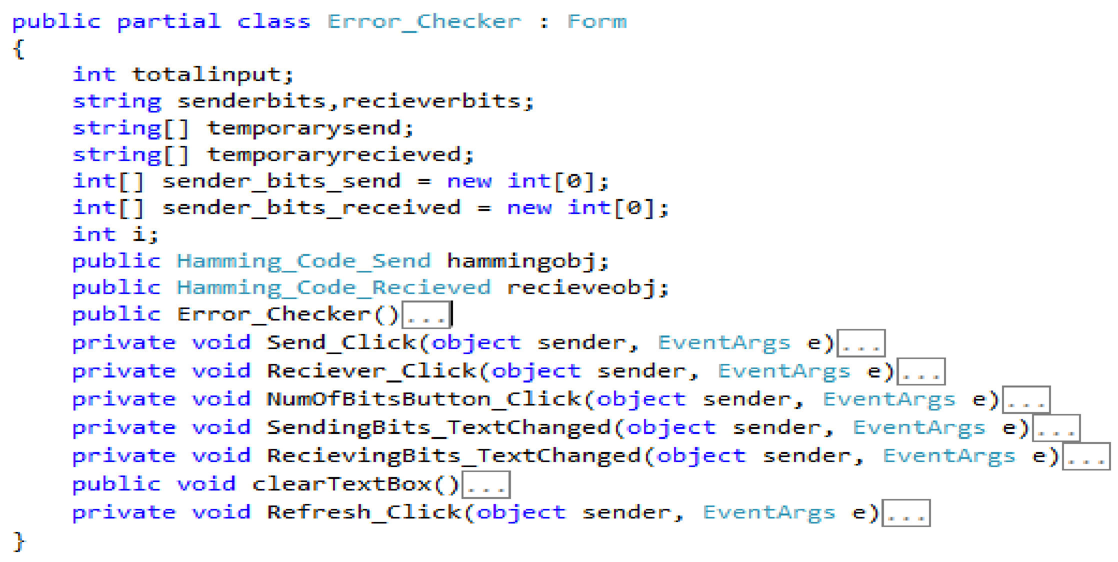


```
public class Hamming_Code_Send
 {
    public Error_Checker obj1;
    public int count = 0;
    int i;
    int[] senddatawithparity;
    Color[] colors;
    Color color;
    string display;
    public Hamming_Code_Send(Error_Checker obj2)...
    public void display_data()...
     public  int[] one_bit_data_send(int[] databit)...
     public int[] two_bit_data_send(int[] databit)...
     public int[] three_bit_data_send(int[] databit)...
     public int[] four_bit_data_send(int[] databit)...
     public int[] five_bit_data_send(int[] databit)...
     public int[] six_bit_data_send(int[] databit)...
     public int[] seven_bit_data_send(int[] databit)...
     public int[] eight_bit_data_send(int[] databit)...
     public int[] nine_bit_data_send(int[] databit)...
     public int[] ten_bit_data_send(int[] databit)...
     public int[] eleven_bit_data_send(int[] databit)...
     public int[] twelve_bit_data_send(int[] databit)...
     public int[] thirteen_bit_data_send(int[] databit)...
     public int[] fourteen_bit_data_send(int[] databit)...
     public int[] fifteen_bit_data_send(int[] databit)...
     public int[] sixteen_bit_data_send(int[] databit)...
     public int[] seventeen_bit_data_send(int[] databit)...
     public int[] eighteen_bit_data_send(int[] databit)...
     public int[] nineteen_bit_data_send(int[] databit)...
     public int[] twenty_bit_data_send(int[] databit)...

     public int[] twenty_bit_data_send(int[] databit)...
     public int[] twenty_one_bit_data_send(int[] databit)...
     public int[] twenty_two_bit_data_send(int[] databit)...
}
```



```
public  class Hamming_Code_Recieved
  {
      public Error_Checker obj1;
      string display;
      Color[] colors;
      int[] recieve_data_Check;
     public int i, parity1, parity2, parity4, parity8, parity16, decimalparity,count = 0;
     public Hamming_Code_Recieved(Error_Checker obj2)...
     public void display_recieve()...
     public int[] three_bit_recieve(int[] recieve_data)...
     public int[] five_bit_recieve(int[] recieve_data)...
     public int[] six_bit_recieve(int[] recieve_data)...
     public int[] seven_bit_recieve(int[] recieve_data)...
     public int[] nine_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] ten_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] eleven_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] twelve_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] thirteen_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] fourteen_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] fifteen_bit_recieve(int[] recieve_data)//with 3 parities...
     public int[] seventeen_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] eighteen_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] nineteen_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_one_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_two_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_three_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_four_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_five_bit_recieve(int[] recieve_data)//with 4 parities...
    public int[] twenty_six_bit_recieve(int[] recieve_data)//with 4 parities...
     public int[] twenty_seven_bit_recieve(int[] recieve_data)//with 4 parities...
  }
```



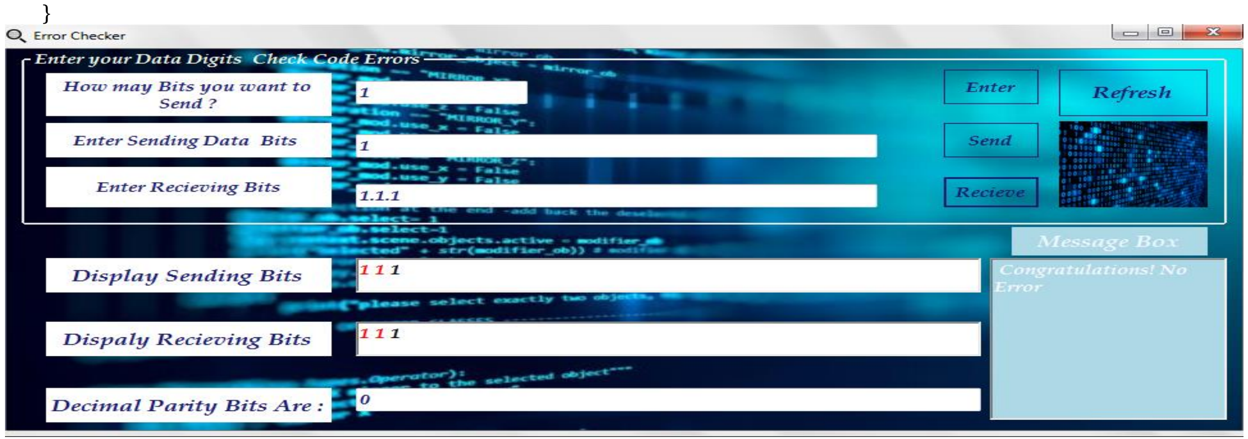
```
public partial class Error_Checker : Form
{
    int totalinput;
    string senderbits,recieverbits;
    string[] temporarysend;
    string[] temporaryrecieved;
    int[] sender_bits_send = new int[0];
    int[] sender_bits_received = new int[0];
    int i;
    public Hamming_Code_Send hammingobj;
    public Hamming_Code_Recieved recieveobj;
    public Error_Checker()...
    private void Send_Click(object sender, EventArgs e)...
    private void Reciever_Click(object sender, EventArgs e)...
    private void NumOfBitsButton_Click(object sender, EventArgs e)...
    private void SendingBits_TextChanged(object sender, EventArgs e)...
    private void RecievingBits_TextChanged(object sender, EventArgs e)...
    public void clearTextBox()...
    private void Refresh_Click(object sender, EventArgs e)...
}
```

SENDING DATA BITS:

Taking data bits and checking total number of bits and calling method based on condition:

```
private void Send_Click(object sender, EventArgs e)
    {
       int count = 0;
       if (senderbits == null)
       {
          MessageBox.Show("Please Enter a Value");
       }
       else
       {
          sender_bits_send = new int[totalinput];
          for (i = 0; i < totalinput; i++)
          {
             temporarysend = senderbits.Split('.');
             sender_bits_send[i] = Convert.ToInt32(temporarysend[i]);
             count++;
          }
          for(int  j  = 0; j < totalinput; j++)
          {
             if(sender_bits_send[j] < 0 || sender_bits_send[j] > 1)
             {
                MessageBox.Show("Invalid Input");
                return;
             }

          }
          if (totalinput == 1)
          {
             hammingobj.one_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 2)
          {
             hammingobj.two_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 3)
          {
             hammingobj.three_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 4)
          {
             hammingobj.four_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 5)
          {
             hammingobj.five_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 6)
          {
             hammingobj.six_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 7)
          {
             hammingobj.seven_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 8)
          {
             hammingobj.eight_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 9)
          {
             hammingobj.nine_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 10)
          {
             hammingobj.ten_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 11)
          {
             hammingobj.eleven_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 12)
          {
             hammingobj.twelve_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 13)
          {
             hammingobj.thirteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 14)
          {
             hammingobj.fourteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 15)
          {
             hammingobj.fifteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 16)
          {
             hammingobj.sixteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 17)
          {
             hammingobj.seventeen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 18)
          {
             hammingobj.eighteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 19)
          {
             hammingobj.nineteen_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 20)
          {
             hammingobj.twenty_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 21)
          {
             hammingobj.twenty_one_bit_data_send(sender_bits_send);
          }
          else if (totalinput == 22)
          {
             hammingobj.twenty_two_bit_data_send(sender_bits_send);
          }
      }
  }
```



Error Checker

Enter your Data Digits Check Code Errors

How may Bits you want to Send ? 1

Enter Sending Data Bits 1

Enter Recieving Bits 1.1.1

Display Sending Bits 1 1 1

Dispaly Recieving Bits 1 1 1

Decimal Parity Bits Are : 0

Message Box

Congratulations! No Error

RECEIVING DATA:

Taking receiving bits and checking total number of bits and calling method based on condition:

```
private void Reciever_Click(object sender, EventArgs e)
    {
      if (recieverbits == null)
      {
        MessageBox.Show("Please Enter a Value");
      }
      else
      {
        int count2 = hammingobj.count;
        sender_bits_received = new int[count2];
        int count = 0;
        for (i = 0; i < count2; i++)
        {
          temporaryrecieved = recieverbits.Split('.');
          sender_bits_received[i] = Convert.ToInt32(temporaryrecieved[i]);
          count++;
        }
        MessageDisplay.Text = "Recieving Values Count = " + count.ToString();
        for (int j = 0; j < count2 ; j++)
        {
          if (sender_bits_received[j] < 0 || sender_bits_received[j] > 1)
          {
            MessageBox.Show("Invalid Input");
            return;
          }
        }
        if (count == 3)
        {
          recieveobj.three_bit_recieve(sender_bits_received);
        }
        else if (count == 5)
        {
          recieveobj.five_bit_recieve(sender_bits_received);
        }
        else if (count == 6)
        {
          recieveobj.six_bit_recieve(sender_bits_received);
        }
        else if (count == 7)
        {
          recieveobj.seven_bit_recieve(sender_bits_received);
        }
        else if (count == 9)
        {
          recieveobj.nine_bit_recieve(sender_bits_received);
        }
        else if (count == 10)
        {
          recieveobj.ten_bit_recieve(sender_bits_received);
        }
        else if (count == 11)
        {
          recieveobj.eleven_bit_recieve(sender_bits_received);
        }
        else if (count == 12)
        {
          recieveobj.twelve_bit_recieve(sender_bits_received);
        }
        else if (count == 13)
        {
          recieveobj.thirteen_bit_recieve(sender_bits_received);
        }
        else if (count == 14)
```

```
            {
                recieveobj.fourteen_bit_recieve(sender_bits_received);
            }
            else if (count == 15)
            {
                recieveobj.fifteen_bit_recieve(sender_bits_received);
            }
            else if (count == 17)
            {
                recieveobj.seventeen_bit_recieve(sender_bits_received);
            }
            else if (count == 18)
            {
                recieveobj.eighteen_bit_recieve(sender_bits_received);
            }
            else if (count == 19)
            {
                recieveobj.nineteen_bit_recieve(sender_bits_received);
            }
            else if (count == 20)
            {
                recieveobj.twenty_bit_recieve(sender_bits_received);
            }
            else if (count == 21)
            {
                recieveobj.twenty_one_bit_recieve(sender_bits_received);
            }
            else if (count == 22)
            {
                recieveobj.twenty_two_bit_recieve(sender_bits_received);
            }
            else if (count == 23)
            {
                recieveobj.twenty_three_bit_recieve(sender_bits_received);
            }
            else if (count == 24)
            {
                recieveobj.twenty_four_bit_recieve(sender_bits_received);
            }
            else if (count == 25)
            {
                recieveobj.twenty_five_bit_recieve(sender_bits_received);
            }
            else if (count == 26)
            {
                recieveobj.twenty_six_bit_recieve(sender_bits_received);
            }
            else if (count == 27)
            {
                recieveobj.twenty_seven_bit_recieve(sender_bits_received);
            }
            else if (count == 1 || count == 2 || count == 4 || count == 8 || count == 16)
            {
                MessageBox.Show("Recieved Data cannot be equal to Parity Bit values Positions");
            }
            else if (count <= 0 || count > 27)
            {
                MessageBox.Show("Recieved Data cannot be <=0 or >27");
            }
        }
    }
```

ADDING PARITY BIT AT SENDER SIDE:

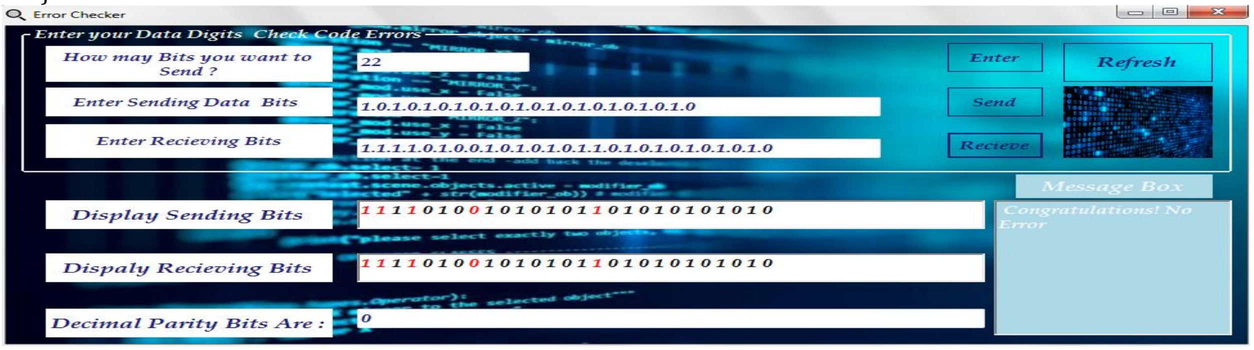
We are showing method here when user send six bit data:

```
public int[] six_bit_data_send(int[] databit)
      {
         senddatawithparity = new int[(databit.Length + 5)];
         senddatawithparity[1] = databit[0] ^ databit[1] ^ databit[3] ^ databit[4];    //parity bit 3 , 5 , 7 & 9
         senddatawithparity[2] = databit[0] ^ databit[2] ^ databit[3] ^ databit[5];     //parity bit 3 , 6 , 7 & 10
         senddatawithparity[3] = databit[0];                        //data bit 3
         senddatawithparity[4] = databit[1] ^ databit[2] ^ databit[3];    //parity bit 5 , 6, 7
         senddatawithparity[5] = databit[1];                        //data bit 5
         senddatawithparity[6] = databit[2];                     //data bit 6
         senddatawithparity[7] = databit[3];  // data bit 7
         senddatawithparity[8] = databit[4] ^ databit[5]; // parity bit 8....9,10
         senddatawithparity[9] = databit[4];//data bit 9
         senddatawithparity[10] = databit[5];//data bit 10
         colors = new Color[] { Color.Red, Color.Red, Color.Red, Color.Black, Color.Red, Color.Black, Color.Black,
Color.Black, Color.Red, Color.Black, Color.Black };
        display_data();
        return senddatawithparity;
      }
```

CHECKING PARITY BIT AT RECIEVER SIDE:

We are showing method here when user recieve six bit data:

```
public int[] six_bit_recieve(int[] recieve_data)
      {
        recieve_data_Check = new int[(recieve_data.Length + 1)];
        recieve_data_Check[1] = recieve_data[0];
        recieve_data_Check[2] = recieve_data[1];
        recieve_data_Check[3] = recieve_data[2];
        recieve_data_Check[4] = recieve_data[3];
        recieve_data_Check[5] = recieve_data[4];
        recieve_data_Check[6] = recieve_data[5];
        parity1 = recieve_data_Check[1] ^ recieve_data_Check[3] ^ recieve_data_Check[5];
        parity2 = recieve_data_Check[2] ^ recieve_data_Check[3] ^ recieve_data_Check[6];
        parity4 = recieve_data_Check[4] ^ recieve_data_Check[5] ^ recieve_data_Check[6];
        decimalparity = parity1 * 1 + parity2 * 2 + parity4 * 4;
        obj1.displayparitybits.Text = decimalparity.ToString();
        colors = new Color[] { Color.Red, Color.Red, Color.Red, Color.Black, Color.Red, Color.Black, Color.Black };
        display_recieve();
        return recieve_data_Check;
      }
```



How may Bits you want to Send ? 22

Enter Sending Data Bits 1.0.1.0.1.0.1.0.1.0.1.0.1.0.1.0.1.0.1.0.1.0

Enter Recieving Bits 1.1.1.1.0.1.0.0.1.0.1.0.1.0.1.1.0.1.0.1.0.1.0.1.0.1.0

Display Sending Bits 1 1 1 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0

Dispaly Recieving Bits 1 1 1 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0

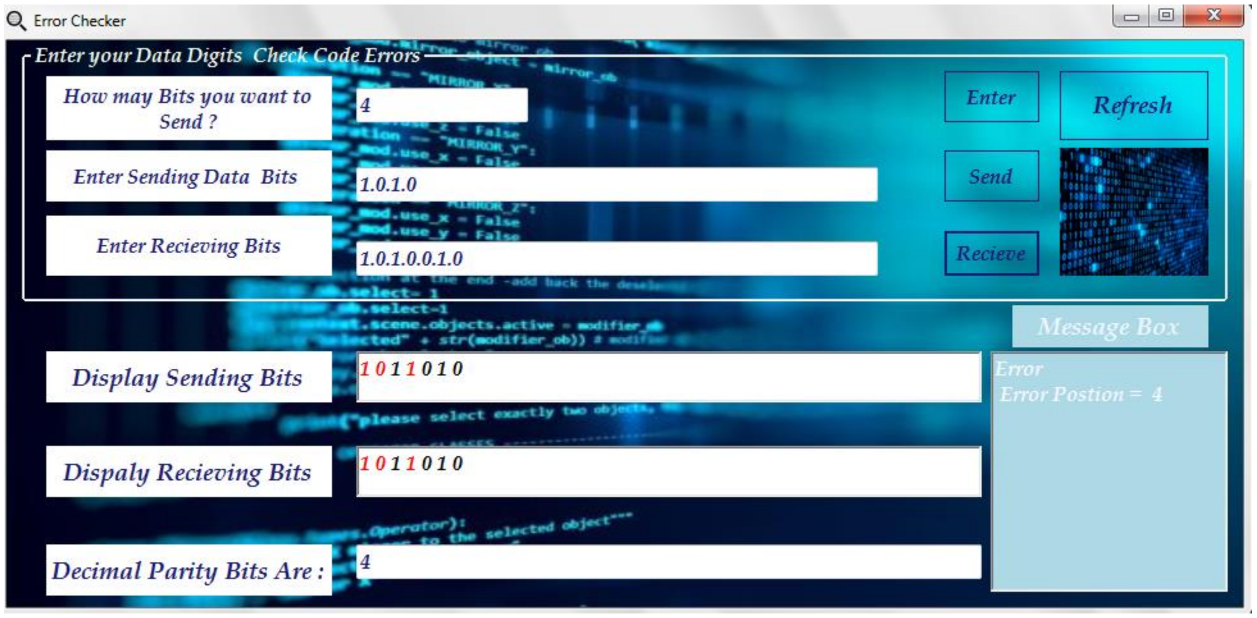
Decimal Parity Bits Are : 0

Congratulations! No Error

ERROR DETECTION AND CORRECTION:

```
public void display_recieve()
    {
      if (decimalparity == 0)
      {
        //MessageBox.Show("No Error");
        obj1.MessageDisplay.Text = " Congratulations! No Error";
      }
      else
      {
       // MessageBox.Show("There is an error in position " + decimalparity);
        obj1.MessageDisplay.Text = "Error \n Error Postion =  " + decimalparity;
        if (recieve_data_Check[decimalparity] == 0)
        {
          recieve_data_Check[decimalparity] = 1;
        }
        else
        {
          recieve_data_Check[decimalparity] = 0;
        }
      }

      for (i = 1; i < recieve_data_Check.Length; i++)
      {
         display = Convert.ToString(recieve_data_Check[i]);
        Color color = colors[i];
        obj1.displayrecievingbit.SelectionColor = color;
        obj1.displayrecievingbit.AppendText(display);
        obj1.displayrecievingbit.AppendText(" ");
      }
    }
```



How may Bits you want to Send ? 4

Enter Sending Data Bits 1.0.1.0

Enter Recieving Bits 1.0.1.0.0.1.0

Display Sending Bits 1 0 1 1 0 1 0

Dispaly Recieving Bits 1 0 1 1 0 1 0

Decimal Parity Bits Are : 4

Error
Error Postion = 4