Problem Statement

Design and implement an error detection module.
● The error detection module implements the four schemes namely LRC, VRC, Checksum and CRC
● Test the program for the following cases
  ○ Error is detected by all four schemes
  ○ Error is detected by Checksum but not by CRC
  ○ Error is detected by VRC but not by CRC
● Design an error injection module to inject random errors in input frames to test the cases.

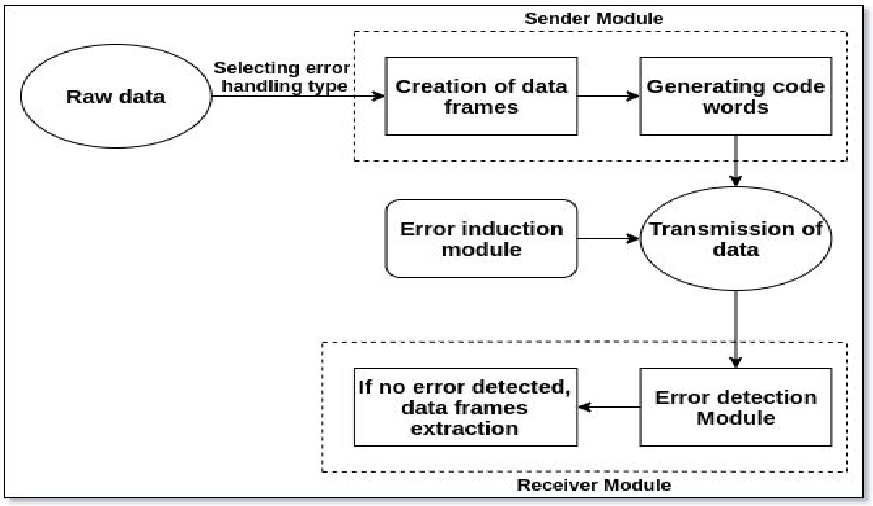
Design description

`Purpose of the program:` The program tries to simulate the environment of data communication at data link layer for analysing the performances of various error detection modules namely vertical redundancy check (VRC), longitudinal redundancy check (LRC), checksum, Cyclic redundancy check (CRC). The key points are -

1. Generating data frames from raw data.
2. Generating code words from data frames with the help of redundant bits based on the error handling method used.
3. Induction of errors into the code words while transmitting.
4. Reception of code words.
5. Error detection and if no error is detected, extracting the original data.

Now, due to simulator constraints, it is very less probable for any error to be induced in the data. Hence, an error induction module is created to induce single bit or burst errors based on the analysis requirements. To achieve a more realistic view, errors are induced randomly into the sent data. In the later sections of the report, a complete analysis of the results over various possible test cases is shown for further examination of different methods used.

`Structural design:`



1. The raw data is read from the input file.
2. Based on the error handling scheme, data frames are created followed by the creation of code words.
3. The program writes the data in an intermediate file representing data transmission.
4. Based on the choice of the user, error may be induced on the transmitted data altering data in the intermediate file.
5. Receiver module receives the data from the intermediate file.
6. Error is being detected using the scheme used to generate the code words. Information regarding the scheme is inserted as a header in the transmitted data.
7. If the program does not detect any error, supposedly valid data is produced by it.

Input and output format:

1. The input is read from the file “message.txt”.
2. The sender module asks for the scheme type to be used.
3. If CRC is chosen, the module further asks for the choice of the divisor numbered 1 to 9 serially mentioned in Appendix (i).
4. The sender module transmits data onto the intermediate file, “intermediate.txt”.
5. The error induction module asks for the choice of error to be induced - (i) Single bit Errors, (ii) Burst Errors. Note: This error induction module is made a separate unit just for the sake of ease in analysis of the program.
6. The final transmitted data is received by the receiver module and the final results are displayed in the terminal.

Code Snippet

Vertical Redundancy Check

Sender Module:

```
String VRC(String input,int frameLen)
   {
       String output="";
       int parity;
       for(int j=0;j<input.length()/frameLen;j++)
       {
           String input1=input.substring(j*frameLen,(j+1)*frameLen);
           parity=0;
           for(int i=0;i<input1.length();i++)
           {
               if(input1.charAt(i)=='1') parity++;
           }
           if(parity%2==0) output= output+input1+'0';
           else output=output+input1+'1';
       } return output;}
```

Receiver Module:

```
static boolean VRC(final String input) {
       int parity;
       for (int j = 0; j < input.length() / 9; j++) {
           final String input1 = input.substring(j * 9, (j + 1) * 9);
           parity = 0;
           for (int i = 0; i < input1.length(); i++) {
               if (input1.charAt(i) == '1')
                   parity++;
           }
           if (parity % 2 == 1)
               return false;
           out = out + input1.substring(0, input1.length() - 1);
       }
       return true;
   }
```

Longitudinal Redundancy Check

Sender Module:

```
String LRC(String input,int frameLen)
   {
       String output="";
       String input1[]=new String[4];

       for(int i=0;i<input.length()/(4*frameLen);i++)
       {
           output+=input.substring(4*i*frameLen,4*(i+1)*frameLen);
           for(int j=0;j<4;j++)
           {
               input1[j]=input.substring((4*i+j)*frameLen,(4*i+j+1)*frameLen);
           }

           String intermediate="";

           for(int j=0;j<frameLen;j++)
           {
               int parity=0;
               for(int k=0;k<4;k++)
               {
                   if(input1[k].charAt(j)=='1') parity++;
               }
               if(parity%2==0) intermediate=intermediate+'0';
               else intermediate=intermediate+'1';
           }
           output+=intermediate;
       }
       return output;
   }
```

Receiver Module:

```
static boolean LRC(String input) {
        int parity;

        String input1[]=new String[5];
        for(int i=0;i<input.length()/(5*8);i++)
        {
            parity=0;
            out=out+input.substring(4*i*8,4*(i+1)*8);
            for(int j=0;j<5;j++)
            {
                input1[j]=input.substring((5*i+j)*8,(5*i+j+1)*8);
            }
            for(int j=0;j<8;j++)
            {
                for(int k=0;k<5;k++)
                {
                    if(input1[k].charAt(j)=='1') parity++;
                }
                if(parity%2==1) return false;
            }
        }
        return true;
    }
```

CheckSum:

Sender Module:

```
String checksum(String input,int frameLen)
    {
        String output=input;
        int sum=0;

        for(int i=0;i<input.length()/frameLen;i++)
        {
            sum+=Integer.parseInt(input.substring(i*frameLen,(i+1)*frameLen),2);
        }
        String x=Integer.toBinaryString(sum);
        int
y=Integer.parseInt(x.substring(0,x.length()-frameLen),2),z=Integer.parseInt(x.s
ubstring(x.length()-4),2);
        x=Integer.toBinaryString(y+z);
        for(int i=0;i<4-x.length();i++) x='0'+x;
        for(int i=0;i<x.length();i++)
        {
            if(x.charAt(i)=='0') output+='1';
            else output+='0';
        }
        return output;
    }
```

Receiver Module:

```
static boolean checksum(String input)
    {
        out=input.substring(0,input.length()-4);
        int sum=0;
        for(int i=0;i<(input.length()/4);i++)
        {
            sum+=Integer.parseInt(input.substring(i*4,(i+1)*4),2);
        }
        String x=Integer.toBinaryString(sum);
        int
y=Integer.parseInt(x.substring(0,x.length()-4),2),z=Integer.parseInt(x.substrin
g(x.length()-4,x.length()),2);
```

```
        x=Integer.toBinaryString(y+z);
        if(Integer.parseInt(x,2)==15) return true;
        return false; }
```

Cyclic Redundancy Check:

Sender Module:

```
static String cp[]={"11", "10011", "110101", "100101", "1000011", "10001001",
"100000111", "110001101", "100110001"};

   static int[] CRC(int old_data[], int choice) {
       if(cp[choice-1].length()>=old_data.length) return null;
       int divisor[]=new int[cp[choice-1].length()];
       for(int i=0;i<cp[choice-1].length();i++) divisor[i] = (int)
       cp[choice-1].charAt(i) - 48;
       int remainder[] , i;
       int data[] = new int[old_data.length + divisor.length];
       System.arraycopy(old_data, 0, data, 0, old_data.length);
       remainder = new int[divisor.length];
       System.arraycopy(data, 0, remainder, 0, divisor.length);
       for(i=0 ; i < old_data.length ; i++) {
           if(remainder[0] == 1) {
               for(int j=1 ; j < divisor.length ; j++)
                   remainder[j-1] = exor(remainder[j], divisor[j]);
           }
           else {
               for(int j=1 ; j < divisor.length ; j++)
                   remainder[j-1] = exor(remainder[j], 0);
           }
       remainder[divisor.length-1] = data[i+divisor.length];
       }
       return remainder;
   }

   static int exor(int a, int b) {
       if(a == b) return 0;
       return 1;
   }
```

Receiver Module:

```
static boolean CRC(int data[],int choice)
   {
       out="";
       int remainder[] = error_module.CRC(data, choice);

       for(int i=0 ; i < remainder.length ; i++)
           if(remainder[i] != 0) return false;
       for(int i=0;i<data.length-error_module.cp[choice-1].length();i++)
out=out+(char)(data[i]+48);
       return true;
   }
```

Error Induction Module

```
static void insert() throws IOException
   {
       files f=new files();
       String msg=f.readFile("./ass1/intermediate.txt");
       System.out.println("Read message : " + msg);
       if(msg.length()<=2) return;
       System.out.println("Choose the type of error to insert.\n 1. Single
Bit\n 2. Burst\n");
       Scanner s=new Scanner(System.in);
       int type=s.nextInt();
       Random r=new Random();
       int pos=r.nextInt(msg.length()-2)+2;
       System.out.println("Position of error : " + Integer.toString(pos));
       switch(type)
       {
           case 1:
               String msg2 = msg.substring(0,pos) +
Integer.toString(error_module.exor((int)msg.charAt(pos)-48,1)) +
msg.substring(pos+1);
               msg=msg2;
               break;

           case 2:
               System.out.println("Length of Burst Error: ");
               int length=s.nextInt();
               while(length--!=0)
               {
msg2=msg.substring(0,pos) +
Integer.toString(error_module.exor((int)msg.charAt(pos)-48,1)) +
msg.substring(pos+1);
                   msg=msg2;
                   pos++;
               }
               break;
       }
       f.WriteFile("./ass1/intermediate.txt",msg);
   }
```

Test Cases

VRC

Input

001101011010101001011100

Intermediate

I. Without Error - 100110101**0**10101010**0**01011100**0**
II. With Single Bit Error - 100110101**0**10101010**0**01011110**0**
III. With Burst Error (even length) - 100110101**0**10101010**0**010111**11**0
IV. With Burst Error (odd length) - 100110101**0**10101010**0**010111**111**

Output

I. Without Error - “Message Received: 001101011010101001011100”
II. With Single Bit Error - “Error detected!!”
III. With Burst Error (even length) - “Message Received: 001101011010101001011111”
IV. With Burst Error (odd length) - “Error detected!!”

LRC

Input

00110101101010100101110011100100

Intermediate

I. Without Error - 200110101101010100101110011100100**00100111**
II. With Single Bit Error - 200110101101010101101110011100100**00100111**
III. With Burst Error - 20011010110101010010111001110010**111010111**
IV. With Burst Error (II) - 200110101101010100101111100011011**110110**11

Output

I. Without Error - “Message Received: 00110101101010100101110011100100”
II. With Single Bit Error - “Error detected!!”
III. With Burst Error - “Error detected!!”
IV. With Burst Error (II) - “Message Received: 00110101101010100101111100011011”

Checksum

Input

00110101101010100101110011100100

Intermediate

I. Without Error - 3001101011010101001011100111001001**100**
II. With Single Bit Error - 3000101011010101001011100111001001**100**
III. With Burst Error (I) - 30011010110101010010111001110010**10010**
IV. With Burst Error (II) - 3001101011010101001011100110001101**100**

Output

I. Without Error - “Message Received: 00110101101010100101110011100100”
II. With Single Bit Error - “Error detected!!”
III. With Burst Error (I) - “Error detected!!”
IV. With Burst Error (II) - “Message Received: 00110101101010100101110011000110”

CRC

Input

00110111 (Generator - 10011)

Intermediate

I. Without Error - 4200110111**01100**
II. With Single Bit Error - 4200110110**01100**
III. With Burst Error (I) - 4200010001**01100**
IV. With Burst Error (II) - 4201001011**01100**

Output

I. Without Error - “Message Received: 00110111”
II. With Single Bit Error - “Error detected!!”
III. With Burst Error (I) - “Message Received: 00010001”
IV. With Burst Error (II) - “Error detected!!”

Note : (i) Green represents header information of the packet.
(ii) Yellow represents error injected.
(iii) Red represents erroneous reception of codeword.
(iv) Bold represents redundant bits.

Edge Case Analysis

Error detected by checksum but not CRC

Input

00110111

Intermediate

Method	**CRC (Generator - 10011)**	**Checksum**
Without Error	4200110111**01100**	300110111**0101**
With Error	4200010001**01100**	300010001**0101**

Output

Method	**CRC (Generator - 10011)**	**Checksum**
Without Error	Message Received: 00110111	Message Received: 00110111
With Error	Message Received: 00010001	Error detected!!
Result	Fail	Pass

Error detected by VRC but not CRC

Input
00110111

Intermediate

Method	**CRC (Generator - 10011)**	**VRC**
Without Error	4200110111**01100**	100110111**1**
With Error	4200010001**01100**	100010001**1**

Output

Method	**CRC (Generator - 10011)**	**VRC**
Without Error	Message Received: 00110111	Message Received: 00110111
With Error	Message Received: 00010001	Error detected!!
Result	Fail	Pass

Discussion

1. All the modules were able to detect single bit errors.
2. VRC was able to detect burst errors of odd length but fails to do so for even length errors.
3. LRC was able to detect burst error of frame length size but fails otherwise.
4. CheckSum was able to detect burst errors with greater probability but fails in few conditions where the corruption of bits does not affect the total sum of the data words.
5. CRC was able to detect burst errors of length L < r+1 but fails for burst errors of length L=r+1 and L>r+1 with probabilities $(½)^{r-1}$ and $(½)^{r}$ respectively. Where, r is the degree of generator.

Comments

❖ The assignment helps the student to understand the flow of the transmission of data in a network. It also helps the student to understand pros and cons of using one error detection module and helps choose the required module appropriately in real life scenarios.
❖ The assignment was not too hard nor too easy from a student's point of view. Considering one's fluency in a certain language, the main task was to think on how to apply those techniques which had a productive outcome.