



Green University of Bangladesh

*Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)*

Basic Data Communication Live Code implementation

*Course Title: Data Communication Lab
Course Code: CSE-308
Section: 221 D9*

Students Details

Name	ID
Md. Harun-Ur-Roshid	221002138
Md. Mostakim Rahman Shipon	221002098

*Submission Date: 10/06/2024
Course Teacher's Name: Md. Mamunur Rahman*

[For teachers use only: **Don't write anything inside this box**]

<u>Lab Project Status</u>	
Marks:	Signature:
Comments:	Date:

Contents

1	Introduction	2
1.1	Overview	2
1.2	Motivation	2
1.3	Problem Definition	2
1.3.1	Problem Statement	2
1.3.2	Complex Engineering Problem	3
1.4	Design Goals/Objectives	4
1.5	Application	4
2	Design/Development/Implementation of the Project	5
2.1	Introduction	5
2.2	Project Details	5
2.3	Projects Image	5
2.4	Implementation	6
2.5	Algorithms	19
3	Performance Evaluation	21
3.1	Simulation Environment/ Simulation Procedure	21
3.2	Results Overall Discussion	23
3.2.1	Complex Engineering Problem Discussion	23
4	Conclusion	24
4.1	Discussion	24
4.2	Limitations	24
4.3	Scope of Future Work	24

Chapter 1

Introduction

1.1 Overview

The Basic Data Communication Live Code Implementation System is a Java-based project aimed at providing a comprehensive understanding and practical implementation of fundamental data communication techniques. The system will incorporate key concepts such as Hamming code, bit stuffing/de-stuffing, and character stuffing/de-stuffing to facilitate real-time encoding, decoding, and transmission of data.

1.2 Motivation

In today's digital era, effective data communication is essential across various domains including telecommunications, networking, and information technology. Understanding the underlying principles and implementing techniques like Hamming code and data stuffing is crucial for ensuring reliable and efficient transmission of data. This project seeks to bridge the gap between theoretical knowledge and practical implementation, providing students and enthusiasts with hands-on experience in data communication.

1.3 Problem Definition

1.3.1 Problem Statement

The project aims to address the following challenges:

1. Implementing Hamming code for error detection and correction.
2. Designing algorithms for bit stuffing/de-stuffing and character stuffing/de-stuffing.
3. Developing an intuitive user interface for live demonstration and interaction.

1.3.2 Complex Engineering Problem

One of the complex engineering problems involves efficiently encoding and decoding data using Hamming code while ensuring error detection and correction. Additionally, designing algorithms for bit stuffing/destuffing and character stuffing/destuffing requires careful consideration of various scenarios and edge cases to maintain data integrity and synchronization.

Name of the Attributes	Explain how to address
P1: Depth of knowledge required	Understanding data encode/decode systems and also some network layers.
P2: Range of conflicting requirements	Projects that are too basic may not provide enough challenge for someone who wants to learn.
P3: Depth of analysis required	For projects simulating data transfer or basic error detection, the analysis involves comprehending the concepts themselves. You'll need to analyze how the specific encoding scheme works.
P4: Familiarity of issues	Live coding environments might not offer extensive debugging tools you'd have in a traditional development setting. Be prepared to identify and fix errors on the fly using print statements or logging mechanisms.
P5: Extent of applicable codes	Focus on core functionalities: These projects might involve short snippets of code to demonstrate specific concepts. Simulating data transfer with different encoding schemes might require a few lines of code to manipulate bits and bytes. Implementing basic error detection could involve calculating checksums using simple arithmetic operations.
P6: Extent of stakeholder involvement and conflicting requirements	Stakeholders: These projects might involve internal stakeholders like colleagues, students, or a training audience. Level of Involvement: Stakeholder involvement might be limited to defining the learning objectives or the overall theme of the live coding session.
P7: Interdependence	This is a core aspect of any coding project but especially relevant in data communication due to the layered nature of network protocols. Each layer (Physical, Data Link, Network, Transport, etc.) relies on the functionalities of the layer below it to function properly.

Table 1.1: Complex Engineering Problem Attributes

1.4 Design Goals/Objectives

Specify and discuss the goals or objectives of your project.

1. To develop a user-friendly interface for inputting and visualizing data.
2. To implement Hamming code algorithms for error detection and correction.
3. To design and implement algorithms for bit stuffing/destuffing and character stuffing/destuffing.
4. To enable real-time encoding, decoding, and transmission of data.
5. To ensure robust error handling and data integrity mechanisms.

1.5 Application

Basic Data Communication Live Code Implementation Projects has several potential applications:

- 1. Educational tool:** Provides students with hands-on experience in data communication techniques.
- 2. Research platform:** Enables researchers to experiment with and analyze different encoding and decoding algorithms.
- 3. Industry training:** Offers practical training for professionals in the field of telecommunications and networking.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The Basic Data Communication Live Code Implementation System is a Java project designed to teach and demonstrate basic data communication methods. It will include important concepts like Hamming code, bit stuffing/de-stuffing, and character stuffing/de-stuffing, allowing users to see how data is encoded, decoded, and transmitted in real-time.

2.2 Project Details

In this Basic Data Communication Live Code Implementation System projects we are gonna using Java programming language. Where we are gonna use stuffing destuffing algorithm, hamming distance algorithm, and parity check algorithm. We just make a interface with java swing. We use IntelliJ Idea IDE for doing this projects.

2.3 Projects Image

Here you can see the projects image:

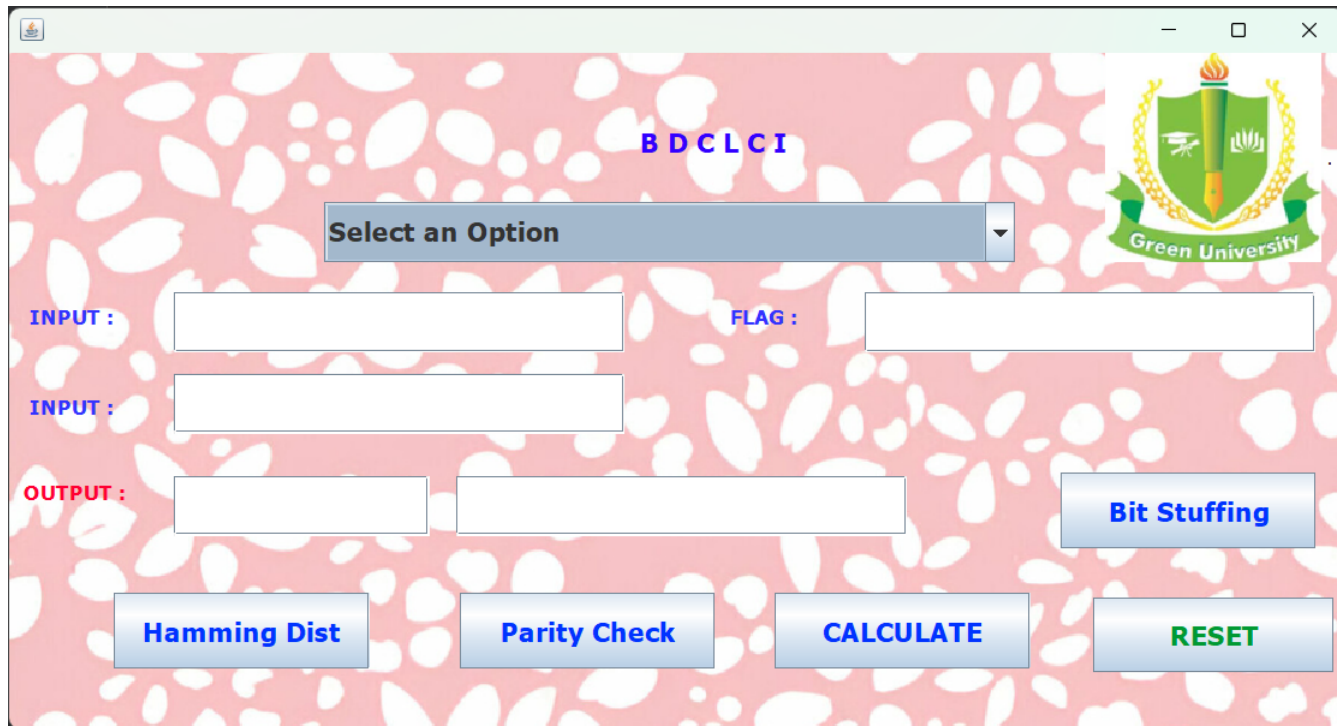


Figure 2.1: BDCLCI

2.4 Implementation

Here is the code for this projects.

```
package Data_Comes;

import java.awt.event.KeyEvent;
import java.lang.Math;
import java.util.*;
import java.io.*;

import javax.swing.JOptionPane;
public class Data_Com extends javax.swing.JFrame {

    String operation;
    String A;
    String B;
    private Object math;
    public Data_Com() {
        initComponents();
    }

    @SuppressWarnings("unchecked")

    private void initComponents() {
```

```

jPanel1 = new javax.swing.JPanel();
jPanel2 = new javax.swing.JPanel();
reset = new javax.swing.JButton();
convert = new javax.swing.JButton();
result1 = new javax.swing.JTextField();
jLabel2 = new javax.swing.JLabel();
s2 = new javax.swing.JTextField();
jLabel4 = new javax.swing.JLabel();
s3 = new javax.swing.JTextField();
jLabel3 = new javax.swing.JLabel();
combobox = new javax.swing.JComboBox<>();
jLabel1 = new javax.swing.JLabel();
jLabel6 = new javax.swing.JLabel();
jLabel5 = new javax.swing.JLabel();
result = new javax.swing.JTextField();
jLabel7 = new javax.swing.JLabel();
s1 = new javax.swing.JTextField();
convert1 = new javax.swing.JButton();
convert2 = new javax.swing.JButton();
convert3 = new javax.swing.JButton();
jLabel8 = new javax.swing.JLabel();

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 100, Short.MAX_VALUE)
);
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 100, Short.MAX_VALUE)
);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jPanel2.setLayout(null);

reset.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
reset.setForeground(new java.awt.Color(0, 153, 51));
reset.setText("RESET");
reset.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        resetActionPerformed(evt);
    }
});
jPanel2.add(reset);
reset.setBounds(722, 363, 160, 50);

```



```

convert.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
convert.setForeground(new java.awt.Color(0, 51, 255));
convert.setText("CALCULATE");
convert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        convertActionPerformed(evt);
    }
});
jPanel2.add(convert);
convert.setBounds(510, 360, 170, 50);

result1.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N
jPanel2.add(result1);
result1.setBounds(110, 282, 170, 40);

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel2.setForeground(new java.awt.Color(255, 0, 51));
jLabel2.setText("OUTPUT :");
jPanel2.add(jLabel2);
jLabel2.setBounds(10, 267, 70, 51);

s2.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N
s2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        s2ActionPerformed(evt);
    }
});
jPanel2.add(s2);
s2.setBounds(110, 214, 300, 40);

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel4.setForeground(new java.awt.Color(51, 51, 255));
jLabel4.setText(" INPUT :");
jPanel2.add(jLabel4);
jLabel4.setBounds(10, 210, 90, 51);

s3.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N
jPanel2.add(s3);
s3.setBounds(570, 160, 300, 40);

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel3.setForeground(new java.awt.Color(51, 51, 255));
jLabel3.setText("FLAG :");
jPanel2.add(jLabel3);
jLabel3.setBounds(480, 150, 70, 51);

combobox.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N

```

```

comboBox.setForeground(new java.awt.Color(204, 0, 153));
comboBox.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "S
jPanel2.add(comboBox);
comboBox.setBounds(210, 100, 460, 40);

jLabel1.setBackground(new java.awt.Color(0, 0, 255));
jLabel1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
jLabel1.setForeground(new java.awt.Color(51, 0, 255));
jLabel1.setText("                                B D C L C I");
jPanel2.add(jLabel1);
jLabel1.setBounds(230, 30, 390, 60);

jLabel6.setIcon(new javax.swing.ImageIcon("C:\\\\Users\\\\ARC\\\\Documents\\\\Tutu
jLabel6.setText("jLabel6");
jPanel2.add(jLabel6);
jLabel6.setBounds(730, 0, 150, 140);

jPanel2.add(jLabel5);
jLabel5.setBounds(10, 0, 150, 150);

result.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N
jPanel2.add(result);
result.setBounds(298, 282, 300, 40);

jLabel7.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel7.setForeground(new java.awt.Color(51, 51, 255));
jLabel7.setText(" INPUT :");
jPanel2.add(jLabel7);
jLabel7.setBounds(10, 150, 90, 51);

s1.setFont(new java.awt.Font("Tahoma", 0, 18)); // NOI18N
jPanel2.add(s1);
s1.setBounds(110, 160, 300, 40);

convert1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
convert1.setForeground(new java.awt.Color(0, 51, 255));
convert1.setText("Hamming Dist");
convert1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        convert1ActionPerformed(evt);
    }
});
jPanel2.add(convert1);
convert1.setBounds(70, 360, 170, 50);

convert2.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
convert2.setForeground(new java.awt.Color(0, 51, 255));
convert2.setText("Parity Check");

```

```

convert2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        convert2ActionPerformed(evt);
    }
});
jPanel2.add(convert2);
convert2.setBounds(300, 360, 170, 50);

convert3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
convert3.setForeground(new java.awt.Color(0, 51, 255));
convert3.setText("Bit Stuffing");
convert3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        convert3ActionPerformed(evt);
    }
});
jPanel2.add(convert3);
convert3.setBounds(700, 280, 170, 50);

jLabel8.setIcon(new javax.swing.ImageIcon("C:\\Users\\ARC\\Documents\\Tutu
jLabel8.setText("jLabel8");
jPanel2.add(jLabel8);
jLabel8.setBounds(0, 0, 890, 450);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane()
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, 890, ja
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, 450, ja
);

pack();
}

private void s2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void resetActionPerformed(java.awt.event.ActionEvent evt) {
    s3.setText(null);
    s2.setText(null);
    result1.setText(null);
    s1.setText(null);
    result.setText(null);
}

```

```

}

private void convertActionPerformed(java.awt.event.ActionEvent evt) {

    if(combobox.getSelectedItem().equals("Hamming Distance"))
    {
        if((s2.getText().equals("")) || (s1.getText().equals("")) )
        {
            JOptionPane.showMessageDialog(null, "You must enter value to compute",
                "DST System",JOptionPane.INFORMATION_MESSAGE);
        }
        else
        {
            String str1 =s1.getText();
            String str2 =s2.getText();
            if(str1.length()!=str2.length())
            {
                JOptionPane.showMessageDialog(null, "You must enter value same length",
                    "DST System",JOptionPane.INFORMATION_MESSAGE);
            }
            else
            {
                int distance = 0;

                for (int i = 0; i < str1.length(); i++) {
                    if (str1.charAt(i) != str2.charAt(i)) {
                        distance++;
                    }
                }
                String distance1 = Integer.toString(distance);
                result1.setText("Humming Distance");
                result.setText(distance1);

            }
        }
    }

    if(combobox.getSelectedItem().equals("parity check"))
    {
        if((s1.getText().equals("")) )
        {
            JOptionPane.showMessageDialog(null, "You must enter value to compute in",
                "DST System",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

```

```

        }
        else
        {
int a=Integer.parseInt(s1.getText());

String str3 =Integer.toBinaryString(a);
s2.setText("Binary: "+str3);
int counter = 0;

        for (int i = 0; i < str3.length(); i++)
        {
            if (str3.charAt(i) == '1')
            {
                counter++;
            }
        }
        if(counter%2==0)
        {
            result1.setText("Parity");
            result.setText("Even Parity");
        }
        else

        {
            result1.setText("Parity");
            result.setText("Odd parity");
        }
    }
}

if(combobox.getSelectedItem().equals("Bit Stuffing"))
{
    if((s1.getText().equals(""))))
    {
        JOptionPane.showMessageDialog(null, "You must enter value to compute",
            "DST System",JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
        String flag=s3.getText();
        String data = s1.getText();
        String res = new String();
        String out=new String();
        int counter = 0,i;
    }
}

```

```

        for(i=0;i<data.length();i++)
        {
            if(data.charAt(i) == '1')
            {
                counter++;
                res = res + data.charAt(i);
            }
            else
            {
                res = res + data.charAt(i);
                counter = 0;
            }
            if(counter == 5)
            {
                res = res + '0';
                counter = 0;
            }
        }
        result1.setText("Bit Stuffing");
        result.setText(flag+res+flag);
    }
}

if(combobox.getSelectedItem().equals("Bit Destuffing"))
{
    if((s1.getText().equals("")) || (s3.getText().equals("")) )
    {
        JOptionPane.showMessageDialog(null, "You must enter value to compute in
        "DST System",JOptionPane.INFORMATION_MESSAGE);
    }

    else
    {
        String res = s1.getText();
        String flag=s3.getText();
        //String res = new String();
        String out=new String();
        int counter=0,i;
        for(i=0;i<res.length();i++)
        {

            if(res.charAt(i) == '1')
            {

                counter++;
                out = out + res.charAt(i);

            }

```

```

        else
        {
            out = out + res.charAt(i);
            counter = 0;
        }
        if(counter == 5)
        {
            if((i+2)!=res.length())
                out = out + res.charAt(i+2);
            else
                out=out + '1';
            i=i+2;
            counter = 1;
        }
    }
    result1.setText("Bit Destuffing");
    result.setText(flag+out+flag);
}
}
if(combobox.getSelectedItem().equals("Character Stuffing"))
{
    if((s1.getText().equals(""))))
    {
        JOptionPane.showMessageDialog(null, "You must enter value to compute",
            "DST System",JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
        String flag=s3.getText();
        String data = s1.getText();
        String data1 = s2.getText();
        String res = new String();
        String out=new String();
        int i,j;
        for(i=0,j=0;i<data.length();i++)
        {
            if(data.charAt(i) ==data1.charAt(j))
            {
                res = res + data.charAt(i);
                res = res + 'k';
            }
            else
            {
                res = res + data.charAt(i);
            }
        }
    }
}

```

```

        }
        result1.setText("Character Stuffing");
        result.setText(flag+res+flag);
    }
    }
    if(combobox.getSelectedItem().equals("Character Destuffing"))
    {
        if(!"".equals(s1.getText()) || !"".equals(s2.getText()) || !"".equals(s3.getText()))
        {
            JOptionPane.showMessageDialog(null, "wait till update again. This option is not available",
                "DST System",JOptionPane.INFORMATION_MESSAGE);
        }
        else if((s1.getText().equals("")) && (s2.getText().equals("")) && (s3.getText().equals("")))
        {
            JOptionPane.showMessageDialog(null, "You must enter value to compute",
                "DST System",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

private void convert1ActionPerformed(java.awt.event.ActionEvent evt) {

    if(combobox.getSelectedItem().equals("Hamming Distance"))
    {
        if((s2.getText().equals("")) )
        {
            JOptionPane.showMessageDialog(null, "You must enter value to compute",
                "DST System",JOptionPane.INFORMATION_MESSAGE);
        }
        else
        {
            String str1 =s1.getText();
            String str2 =s2.getText();
            if(str1.length()!=str2.length())
            {
                JOptionPane.showMessageDialog(null, "You must enter value same length",
                    "DST System",JOptionPane.INFORMATION_MESSAGE);
            }
            else
            {
                int distance = 0;

                for (int i = 0; i < str1.length(); i++) {
                    if (str1.charAt(i) != str2.charAt(i)) {

```



```

        distance++;
    }
}
String distance1 = Integer.toString(distance);
result1.setText("Humming Distance");
result.setText(distance1);

    }
    }
}

private void convert2ActionPerformed(java.awt.event.ActionEvent evt) {
if(combobox.getSelectedItem().equals("parity check"))
{
    if((s1.getText().equals("")) )
    {
        JOptionPane.showMessageDialog(null, "You must enter value to compute in
        "DST System",JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
int a=Integer.parseInt(s1.getText());

String str3 =Integer.toBinaryString(a);
s2.setText("Binary: "+str3);
int counter = 0;

        for (int i = 0; i < str3.length(); i++)
        {
            if (str3.charAt(i) == '1')
            {
                counter++;
            }
        }
        if(counter%2==0)
        {

            result1.setText("Parity Check");
            result.setText("Even Parity");

        }
        else
        {
            result1.setText("Parity Check");
            result.setText("Odd parity");

        }
    }
}

```

```

        }
    }

    private void convert3ActionPerformed(java.awt.event.ActionEvent evt) {
if(combobox.getSelectedItem().equals("Bit Stuffing"))
{
    if((s1.getText().equals("")) && (s3.getText().equals("")) )
    {
        JOptionPane.showMessageDialog(null, "You must enter value to compute",
            "DST System",JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
String data = s1.getText();
String res = new String();
String out=new String();
int counter = 0,i;
for(i=0;i<data.length();i++)
{
    if(data.charAt(i) == '1')
    {
        counter++;
        res = res + data.charAt(i);
    }
    else
    {
        res = res + data.charAt(i);
        counter = 0;
    }
    if(counter == 5)
    {
        res = res + '0';
        counter = 0;
    }
}
result1.setText("Bit Stuffing");
result.setText(res);
    }
}

}

/**
 * @param args the command line arguments
 */

```

```

public static void main(String args[])throws IOException {

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the def
    * For details see http://download.oracle.com/javase/tutorial/uiswing/look
    */

    //</editor-fold>
    Scanner sn=new Scanner(System.in);

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Data_Com().setVisible(true);
        }
    });

    private javax.swing.JComboBox<String> combobox;
    private javax.swing.JButton convert;
    private javax.swing.JButton convert1;
    private javax.swing.JButton convert2;
    private javax.swing.JButton convert3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JButton reset;
    private javax.swing.JTextField result;
    private javax.swing.JTextField result1;
    private javax.swing.JTextField s1;
    private javax.swing.JTextField s2;
    private javax.swing.JTextField s3;
}

```

2.5 Algorithms

Here is the Algorithm for this projects.

- Initialize Components:

Initialize UI components such as buttons, text fields, and labels.
Set up the layout for the JPanel and JFrame.

- Reset Button Action:

Clear the text fields: s1, s2, s3, result, and result1.

- Calculate Button Action:

Check the selected item in the combobox.
Perform the corresponding operation based on the selected item.

- Hamming Distance Calculation:

Check if both input fields (s1 and s2) are filled.
Validate that both input strings have the same length.
Calculate the Hamming distance by comparing each character in the strings.
Display the result in the result and result1 text fields.

- Parity Check:

Check if the input field (s1) is filled.
Convert the input integer to its binary representation.
Count the number of 1s in the binary string.
Determine if the number of 1s is even or odd.
Display the parity result in the result and result1 text fields.

- Bit Stuffing:

Check if the input field (s1) is filled.
Initialize a counter to track consecutive 1s.
Iterate through the input string and add a 0 after every sequence of five 1s.
Display the stuffed bit string in the result and result1 text fields.

- Bit Destuffing:

Check if the input fields (s1 and s3) are filled.
Initialize a counter to track consecutive 1s.
Iterate through the input string and remove the extra 0 after every sequence of five 1s.
Display the destuffed bit string in the result and result1 text fields.

- Character Stuffing:

Check if the input fields (s1 and s2) are filled.
Iterate through the input string and insert a special character after every sequence of five characters.
Display the stuffed character string in the result and result1 text fields.

- Character Destuffing:

Display a message indicating that this functionality is not yet implemented.

- Helper Methods:

Include additional helper methods to handle specific button actions and events

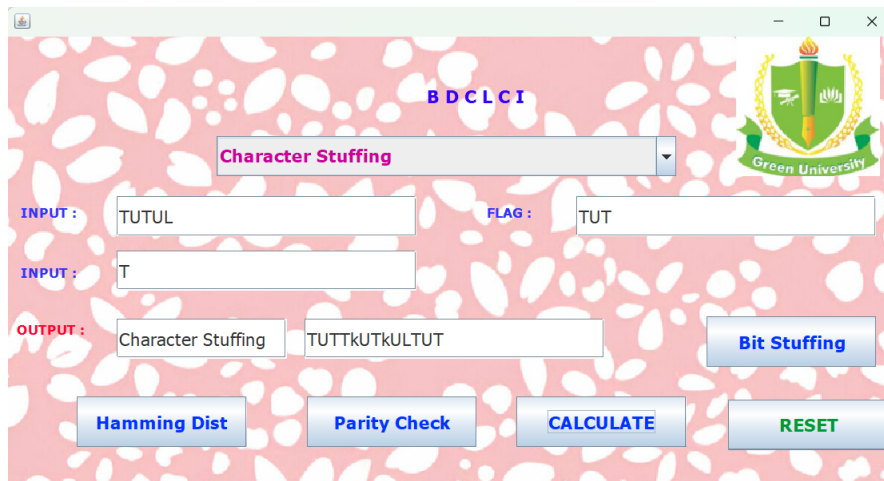
- Main Method:

Create and display the Data_Com JFrame.

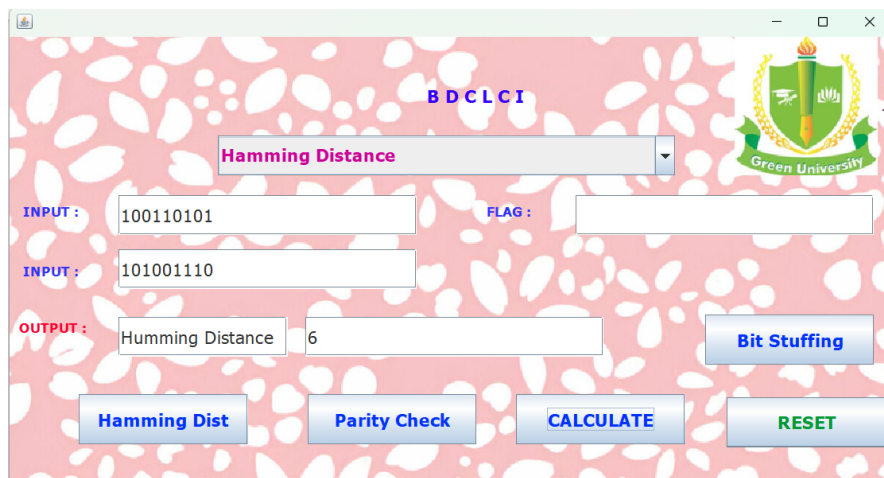
Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure




The screenshot shows a web-based simulation interface for Character Stuffing. The title bar indicates the application is 'BDCLCI'. The interface has a pink background with a white floral pattern. At the top right is the Green University logo. A dropdown menu is set to 'Character Stuffing'. There are two input fields: 'INPUT : TUTUL' and 'INPUT : T'. A 'FLAG : TUT' field is also present. The output field shows 'Character Stuffing' and 'TUTTKUTKULTUT'. At the bottom, there are four buttons: 'Hamming Dist', 'Parity Check', 'CALCULATE', and 'RESET'. A 'Bit Stuffing' button is also visible on the right side.



The screenshot shows a web-based simulation interface for Hamming Distance. The title bar indicates the application is 'BDCLCI'. The interface has a pink background with a white floral pattern. At the top right is the Green University logo. A dropdown menu is set to 'Hamming Distance'. There are two input fields: 'INPUT : 100110101' and 'INPUT : 101001110'. A 'FLAG :' field is empty. The output field shows 'Hamming Distance' and '6'. At the bottom, there are four buttons: 'Hamming Dist', 'Parity Check', 'CALCULATE', and 'RESET'. A 'Bit Stuffing' button is also visible on the right side.

B D C L C I



parity check

INPUT : 96 FLAG :


INPUT : Binary: 1100000

OUTPUT : Parity Check Even Parity

Bit Stuffing

Hamming Dist Parity Check CALCULATE RESET

B D C L C I



Bit Stuffing

INPUT : 1001101010100110 FLAG : 1101010


INPUT :

OUTPUT : Bit Stuffing 0101010011010101001101101010

Bit Stuffing

Hamming Dist Parity Check CALCULATE RESET

B D C L C I



Bit Destuffing

INPUT : 1010001110101110 FLAG : 1101010

INPUT :

OUTPUT : Bit Destuffing 0101010100011101011101101010

Bit Stuffing

Hamming Dist Parity Check CALCULATE RESET

3.2 Results Overall Discussion

The Data Communication Live Code Implementation System is a Java-based project designed to provide practical insights into fundamental data communication techniques. The primary goal of this project is to facilitate an understanding of various encoding, decoding, and error-checking methods through a user-friendly graphical interface.

3.2.1 Complex Engineering Problem Discussion

This project addresses complex engineering problems in data communication by implementing fundamental techniques like Hamming code, bit stuffing, and character stuffing. These methods are crucial for ensuring data integrity and error detection in transmission. The project simulates real-time encoding, decoding, and transmission processes, providing a practical learning tool. However, it faces challenges such as incomplete features, limited input validation, and a basic user interface. These limitations highlight the need for more robust error handling, comprehensive feature sets, and enhanced user experience to fully replicate real-world communication systems and provide a deeper understanding of data communication complexities.

Chapter 4

Conclusion

4.1 Discussion

The Data Communication Live Code Implementation System is a valuable project for learning and demonstrating essential data communication techniques. By providing real-time encoding, decoding, and error-checking functionalities, it offers a practical and interactive way to grasp fundamental concepts in data communication. With planned future enhancements, it has the potential to become an even more robust educational tool.

4.2 Limitations

This project has several limitations: it currently lacks the full implementation of the character destuffing feature, and input validation is basic, which can lead to errors or crashes with incorrect data. The user interface, while functional, could be more intuitive and visually appealing. Additionally, the project only covers a limited set of data communication techniques, omitting more advanced methods like cyclic redundancy check (CRC) or forward error correction (FEC). Lastly, there is minimal documentation and no tutorials, which can hinder users' understanding and effective use of the system.

4.3 Scope of Future Work

Character Destuffing Implementation: Completing the functionality for character destuffing to provide a comprehensive set of tools for both stuffing and de-stuffing processes.

Enhanced Error Handling: Improving input validation and error messages to guide users more effectively.

Additional Communication Techniques: Introducing more advanced data communication methods, such as cyclic redundancy check (CRC) or forward error correction (FEC).

User Interface Enhancements: Refining the GUI for better usability and visual appeal,

possibly including graphical representations of the data transformation processes.