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

Project Report

Course Title: *Data Communication Lab*
Course Code: *CSE 308*
Section: *222 D3*

Students Details

Name	ID
Md. Zahidul Islam	221902091
Md. Mahim Hossain	221902082

Submission Date: *08 June, 2024*
Course Teacher's Name: *Mahbubur Rahman*

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

Project Report Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Basic Data Communication Live Code Implementation project implements various data communication techniques within a Java Swing application, facilitating comprehensive evaluation and analysis of their performance. Techniques such as Bit Stuffing, Bit De-stuffing, Character Stuffing, Character De-stuffing, IP address conversion(Dotted Decimal to Binary, Binary to Dotted Decimal, IPv4 to IPv6, IPv4 Information), error detection, and Hamming Code error correction are integrated into the application. Through an intuitive graphical user interface, users can input data, execute the selected technique, and view the resulting output, enabling thorough experimentation and understanding of data communication principles. The project aims to provide a practical platform for studying and assessing different aspects of data transmission, from ensuring data integrity to error detection and correction, fostering insights into their effectiveness and limitations in real-world scenarios.

1.2 Motivation

In today's digital world, understanding data communication is crucial for professionals in various fields such as computer science, telecommunications, and networking. However, many find it challenging to grasp these concepts solely through theoretical learning. Hence, there is a need for an interactive platform where individuals can visualize and experiment with different data communication protocols and algorithms.

1.3 Problem Definition

1.3.1 Problem Statement

The project addresses the lack of interactive tools for learning data communication concepts. Traditional teaching methods often rely on theoretical explanations without providing practical demonstrations or hands-on experience. This project seeks to bridge

this gap by offering a platform where users can actively engage with data communication concepts through live code implementation.

- Insufficient interactive tools hinder data communication learning.
- Complexity of concepts like Hamming Distance requires hands-on practice.
- Educational resources lack practical demonstrations and real-time feedback.
- Managing errors in algorithms like concurrent transactions poses challenges.
- Stakeholder engagement is vital for developing effective learning tools.

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: User-friendly interface	Design an intuitive GUI with clear navigation and interactive elements to ensure ease of use for users.
P2: Comprehensive Coverage	Implement a wide range of data communication techniques, ensuring that the application covers key concepts and protocols in the field
P3: Real-Time Feedback	Utilize real-time processing and feedback mechanisms to provide immediate responses to user inputs and computations.
P4: Error Handling	Implement robust error detection and handling mechanisms to gracefully manage invalid inputs and unexpected scenarios, providing informative error messages to users.
P5: Educational Value	Develop clear explanations and visualizations for each data communication concept, supplemented with educational resources and insights to enhance the learning experience.
P6: Scalability	Design the application architecture to be modular and extensible, allowing for future enhancements and updates to accommodate evolving educational needs and technological advancements.

1.4 Design Goals/Objectives

The objective of the Basic Data Communication Live Code Implementation project is to create an intuitive and secure platform that facilitates learning and experimentation in data communication concepts. By providing an easy-to-use graphical user interface

(GUI) and real-time feedback, the system aims to replace traditional learning methods with hands-on practice. Through automation, the project seeks to reduce the time and cost associated with understanding complex data communication techniques while ensuring accuracy and efficiency in computation and analysis. Key objectives include:

- Create a user-friendly interface for easy interaction.
- Cover a wide range of data communication techniques comprehensively.
- Provide immediate feedback to enhance learning
- Implement robust error handling mechanisms.
- Offer clear explanations and educational resources.
- Design the application architecture to be scalable for future updates.

1.5 Application

Basic Data Communication Live Code Implementation(BDCLCI)
The Basic Data Communication Live Code Implementation project finds applications in various educational and professional settings, including:

- Facilitating interactive learning sessions for students studying computer science, networking, and telecommunications in academic institutions.
- Providing a platform for professionals to refresh their knowledge of data communication concepts and experiment with different protocols in professional training.
- Enabling individuals interested in data communication to explore concepts at their own pace and gain practical insights through hands-on experimentation in self-study.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The project is implemented using Java, with Swing for the graphical user interface. The main components of the GUI include text fields for input and output, a dropdown menu for selecting the technique, and buttons for performing the action or resetting the fields.

2.2 Project Details

- **Input Field:** A text field for the user to input data.
- **Dropdown Menu:** A combo box listing the available techniques.
- **Output Field:** A text field to display the processed output.
- **Reset Button:** Clears all input and output fields.
- **Perform Button:** Triggers the processing of the input data using the selected technique.
- **Flag:** A text field for user input data.

2.3 Implementation

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.InetAddress;
import java.net.UnknownHostException;

import javax.swing.*;
```

```
public class Home1 extends javax.swing.JFrame {

    public Home1() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {
        jTextField1 = new javax.swing.JTextField();
        jComboBox1 = new javax.swing.JComboBox<>();
        jTextField2 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jTextField3 = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                ImageIcon img = new ImageIcon("p.jpg");
                g.drawImage(img.getImage(), 0, 0, getWidth(), getHeight(), this);
            }
        };

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new String[]{
            "Select an Option", "Bit Stuffing", "Bit De-stuffing",
            "Character Stuffing", "Character De-Stuffing",
            "Dotted Decimal to Binary", "Binary to Dotted Decimal",
            "IP Information", "IPv4 to IPv6", "Parity Check","Hamming Encode",
            "Hamming Decode","Error Detection"
        }));

        jButton1.setBackground(new java.awt.Color(255, 104, 104));
        jButton1.setFont(new java.awt.Font("Segoe UI", 1, 18));
        jButton1.setText("Reset");

        jButton2.setBackground(new java.awt.Color(195, 255, 147));
```

```
jButton2.setFont(new java.awt.Font("Segoe UI", 1, 16));
jButton2.setText("Perform");

jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel1.setForeground(Color.WHITE);
jLabel1.setText("Select Option");

jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel2.setForeground(Color.WHITE); // Set font color to white
jLabel2.setText("Input");

jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel4.setForeground(Color.WHITE);
jLabel4.setText("Output");

jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel5.setForeground(Color.WHITE);
jLabel5.setText(":");

jLabel6.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel6.setForeground(Color.WHITE);
jLabel6.setText(":");

jLabel8.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel8.setForeground(Color.WHITE);
jLabel8.setText(":");

jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 18));
jLabel9.setForeground(Color.WHITE);
jLabel9.setText("Flag");

jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 20));
jLabel3.setForeground(Color.WHITE);
jLabel3.setText(":");

jLabel7.setFont(new java.awt.Font("Segoe UI", 1, 24));
jLabel7.setForeground(Color.WHITE);
jLabel7.setText("Basic Data Communication Live Code Implementation");

jButton2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        performSelectedTechnique();
    }
});

jButton1.addActionListener(new ActionListener() {
    @Override
```

```
            public void actionPerformed(ActionEvent e) {
                jTextField1.setText("");
                jTextField2.setText("");
                jTextField3.setText("");
                jComboBox1.setSelectedIndex(0);
            }
        });

        GroupLayout jPanel1Layout = new GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(81, 81, 81)
                    .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                    Alignment.TRAILING)
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addGroup(jPanel1Layout.createParallelGroup
                            (GroupLayout.Alignment.TRAILING, false)
                                .addComponent(jLabel4, GroupLayout.Alignment.
                                LEADING, GroupLayout.DEFAULT_SIZE, 70,
                                Short.MAX_VALUE)
                                .addComponent(jLabel2, GroupLayout.Alignment.
                                LEADING,GroupLayout.DEFAULT_SIZE,GroupLayout.
                                DEFAULT_SIZE,Short.MAX_VALUE))
                            .addGap(18, 18, 18)
    .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
        .addComponent(jLabel6, GroupLayout.PREFERRED_SIZE, 69, GroupLayout.
        PREFERRED_SIZE)
        .addComponent(jLabel8, GroupLayout.Alignment.TRAILING, GroupLayout.
            PREFERRED_SIZE, 69, GroupLayout.PREFERRED_SIZE)))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(jLabel1)
                    .addGap(18, 18, 18)
                        .addComponent(jLabel5, GroupLayout.PREFERRED_SIZE, 68,
                        GroupLayout.PREFERRED_SIZE)))
                    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                    Alignment.LEADING, false)
                        .addComponent(jTextField2)
                        .addComponent(jComboBox1, 0, 271, Short.MAX_VALUE)
                        .addComponent(jTextField1))
                    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED,
                    GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jLabel9)
                    .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jLabel3, GroupLayout.PREFERRED_SIZE, 37,
                    GroupLayout.PREFERRED_SIZE)
```

```
                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTextField3, GroupLayout.PREFERRED_SIZE, 141,
                GroupLayout.PREFERRED_SIZE)
                .addGap(82, 82, 82))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(125, 125, 125)
                .addComponent(jLabel7, GroupLayout.PREFERRED_SIZE, 622,
                GroupLayout.PREFERRED_SIZE)
                .addContainerGap(131, Short.MAX_VALUE))
            .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
            createSequentialGroup()
                .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(jPanel1Layout.createParallelGroup
                (GroupLayout.Alignment.TRAILING)
                .addComponent(jButton1, GroupLayout.PREFERRED_SIZE,
                101, GroupLayout.PREFERRED_SIZE)
                .addComponent(jButton2, GroupLayout.PREFERRED_SIZE,
                101, GroupLayout.PREFERRED_SIZE))
                 .addGap(78, 78, 78)));

        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
                createSequentialGroup()
                    .addGap(47, 47, 47)
                    .addComponent(jLabel7, GroupLayout.PREFERRED_SIZE, 75,
                    GroupLayout.PREFERRED_SIZE)
                        .addGap(40, 40, 40)
                        .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING, false)
                            .addComponent(jLabel1, GroupLayout.DEFAULT_SIZE,
                            GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jLabel5, GroupLayout.DEFAULT_SIZE,
                            GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jComboBox1, GroupLayout.DEFAULT_SIZE,
                            41, Short.MAX_VALUE))
                        .addGap(18, 18, 18)
                        .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
                            .addComponent(jLabel2, GroupLayout.DEFAULT_SIZE,
                            GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jTextField1, GroupLayout.DEFAULT_SIZE,
                            41, Short.MAX_VALUE)
                            .addComponent(jLabel8, GroupLayout.DEFAULT_SIZE,
                            GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jTextField3)
                            .addComponent(jLabel9, GroupLayout.DEFAULT_SIZE,
                            GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
                        .addComponent(jLabel3, GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGap(18, 18, 18)
                    .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                    Alignment.BASELINE)
                    .addComponent(jTextField2, GroupLayout.PREFERRED_SIZE, 41,
                    GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel6, GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jLabel4, GroupLayout.DEFAULT_SIZE,
                        GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGap(5, 5, 5)
                    .addComponent(jButton2, GroupLayout.PREFERRED_SIZE, 42,
                    GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jButton1, GroupLayout.PREFERRED_SIZE, 42,
                    GroupLayout.PREFERRED_SIZE)
                    .addGap(46, 46, 46))
        );

        getContentPane().add(jPanel1);
        pack();
    }
    private static String bitStuffing(int[] arr) {
        int N = arr.length;
        int[] brr = new int[2 * N];
        int i, j, k; // Variables to traverse arrays
        i = 0;
        j = 0;
        int count = 0;
        // Loop to traverse in the range [0, N)
        while (i < N) {
            // If the current bit is a set bit
            if (arr[i] == 1) {
                brr[j++] = arr[i];

                    if (count == 5) {
                    brr[j++] = 0;
                    count = 0;
                }
            } else {
                  brr[j++] = arr[i];
                count = 0;
            }
            i++;
        }

        StringBuilder output = new StringBuilder();
```

```
    for (i = 0; i < j; i++) {
        output.append(brr[i]);
    }

    return output.toString();
}

// Function for bit de-stuffing
private static String bitDestuffing(int[] arr) {
    int N = arr.length;
    int[] brr = new int[N];

    int i = 0, j = 0, count = 0;

    // Loop to traverse in the range [0, N)
    while (i < N) {
        // If the current bit is a set bit
        if (arr[i] == 1) {
            brr[j++] = arr[i];
            count++;

            if (count == 5) {
                i++;
                count = 0;
            }
        } else {
            // Otherwise insert arr[i] into the array brr
            brr[j++] = arr[i];
            count = 0;
        }
        i++;
    }

    // Build string output
    StringBuilder output = new StringBuilder();
    for (i = 0; i < j; i++) {
        output.append(brr[i]);
    }

    return output.toString();
}

public static String characterStuffing(String data, String flag) {
    String ESCAPE = "ESC";
    StringBuilder stuffedData = new StringBuilder(flag);

    for (int i = 0; i < data.length(); i++) {
        if (data.substring(i).startsWith(flag) ||
```

```
        data.substring(i).startsWith(ESCAPE)) {
            stuffedData.append(ESCAPE);
        }
        stuffedData.append(data.charAt(i));
    }

    stuffedData.append(flag);
    return stuffedData.toString();
}

public static String characterDeStuffing(String data, String flag) {
    StringBuilder destuffedData = new StringBuilder();
    String ESCAPE = "ESC";
    int i = flag.length();
    while (i < data.length() - flag.length()) {
        if (data.substring(i).startsWith(ESCAPE)) {
            i += ESCAPE.length();
        }
        destuffedData.append(data.charAt(i));
        i++;
    }

    return destuffedData.toString();
}

private String dottedDecimalToBinary(String input) {
    String[] octets = input.split("\\.");
    if (octets.length != 4) return "Invalid IPv4 address.";

    StringBuilder binaryIP = new StringBuilder();
    for (String octet : octets) {
    try {
        int decimal = Integer.parseInt(octet);
        if (decimal < 0 || decimal > 255) return "Invalid IPv4 address.";
        String binary = Integer.toBinaryString(decimal);
        binaryIP.append(String.format("%08d", Integer.parseInt(binary)));
        binaryIP.append(".");
    } catch (NumberFormatException e) {
            return "Invalid IPv4 address.";
        }
    }

    return binaryIP.substring(0, binaryIP.length() - 1);
}

private String binaryToDottedDecimal(String input) {
    String[] octets = input.split("\\.");
    if (octets.length != 4) return "Invalid binary IP address.";
```

```
    StringBuilder decimalIP = new StringBuilder();
    for (String octet : octets) {
        try {
            if (octet.length() != 8) return "Invalid binary IP address.";
            int decimal = Integer.parseInt(octet, 2);
            decimalIP.append(decimal);
            decimalIP.append(".");
        } catch (NumberFormatException e) {
            return "Invalid binary IP address.";
        }
    }

    return decimalIP.substring(0, decimalIP.length() - 1);
}


private String ipv4ToIpv6(String input) {
    String[] octets = input.split("\\.");
    if (octets.length != 4) return "Invalid IPv4 address.";

    StringBuilder ipv6 = new StringBuilder("::ffff:");
    for (int i = 0; i < octets.length; i++) {
        try {
            int value = Integer.parseInt(octets[i]);
            if (value < 0 || value > 255) return "Invalid IPv4 address.";
            ipv6.append(String.format("%02x", value));
            if (i % 2 == 1 && i < octets.length - 1) {
                ipv6.append(":");
            }
        } catch (NumberFormatException e) {
            return "Invalid IPv4 address.";
        }
    }

    return "IPv6 (short)--->  " + ipv6.toString();
}
private String ipInformation(String input) {
    try {
        InetAddress inetAddress = InetAddress.getByName(input);
        byte[] addressBytes = inetAddress.getAddress();

        if (addressBytes.length == 4) {  // IPv4
            int firstByte = addressBytes[0] & 0xFF;
            String type = "Public";
            if ((firstByte >= 10 && firstByte <= 10) ||
                    (firstByte >= 172 && firstByte <= 172 &&
                    (addressBytes[1] & 0xF0) == 16) ||
```

```
                    (firstByte >= 192 && firstByte <= 192 &&
                    addressBytes[1] == 168)) {
                type = "Private";
            }
            return "IPv4 Address\nClass: " + getIPv4Class(firstByte) +
            "\nType: " + type;
        } else {  // IPv6
            return "IPv6 Address";
        }
    } catch (UnknownHostException e) {
        return "Invalid IP address.";
    }
}

private String getIPv4Class(int firstByte) {
    if (firstByte >= 0 && firstByte <= 127) return "A";
    if (firstByte >= 128 && firstByte <= 191) return "B";
    if (firstByte >= 192 && firstByte <= 223) return "C";
    if (firstByte >= 224 && firstByte <= 239) return "D";
    return "E";
}


private String parityCheck(String inputData) {
    char[] data = new char[100];
    for (int i = 0; i < inputData.length(); i++) {
        data[i] = inputData.charAt(i);
    }

    // Finding the user data length
    int length = inputData.length();
    int count = 0;

    // Checking even parity
    for (int i = 0; i < length; i++) {
        if (data[i] == '1') {
            count++;
        }
    }

    // Increasing the array for adding the parity bit.
    int c = length + 1;
    char[] newData = new char[c];

    // Initializing the parity
    if (count % 2 == 0) {
        for (int i = c - 1, j = length - 1; j >= 0; i--, j--) {
            newData[i] = data[j];
```

```
        }
        newData[0] = '0';
    } else {
        for (int i = c - 1, j = length - 1; j >= 0; i--, j--) {
            newData[i] = data[j];
        }
        newData[0] = '1';
    }

    // Build string output
    StringBuilder output = new StringBuilder();
    for (char c1 : newData) {
        if (c1 != '\0') {
            output.append(c1);
        }
    }
    return output.toString();
}

class HammingCode {

    public static String encode(String data) {
        int r = 0;
        while (Math.pow(2, r) < data.length() + r + 1) {
            r++;
        }

        int length = data.length() + r;
        char[] result = new char[length];

        int j = 0;
        for (int i = 1; i <= length; i++) {
            if (Math.pow(2, j) == i) {
                result[i - 1] = 'P';
                j++;
            } else {
                result[i - 1] = data.charAt(i - j - 1);
            }
        }

        for (int i = 0; i < r; i++) {
            int parityBitPos = (int) Math.pow(2, i);
            int parity = 0;

            for (int k = 1; k <= length; k++) {
                if (((k >> i) & 1) == 1 && k != parityBitPos) {
                    parity ^= result[k - 1] - '0';
                }
```

```
        }

        result[parityBitPos - 1] = (char) (parity + '0');
    }

    return new String(result);
}

public static String detectError(String senderData, String flagData) {
    StringBuilder errorBits = new StringBuilder();

    // Check for errors
    for (int i = 0; i < senderData.length(); i++) {
        if (senderData.charAt(i) != flagData.charAt(i)) {
            if (errorBits.length() > 0) {
                errorBits.append(", ");
            }
            errorBits.append(i + 1);
        }
    }

    // Construct output
    if (errorBits.length() > 0) {
        return "Error in bit number : " + errorBits.toString();
    } else {
        return "No error detected";
    }
}

public static String decode(String data) {
    int r = 0;
    while (Math.pow(2, r) < data.length() + 1) {
        r++;
    }

    int length = data.length();
    int errorPosition = 0;

    for (int i = 0; i < r; i++) {
        int parityBitPos = (int) Math.pow(2, i);
        int parity = 0;

        for (int k = 1; k <= length; k++) {
            if (((k >> i) & 1) == 1) {
                parity ^= data.charAt(k - 1) - '0';
            }
        }
```

```
            if (parity != 0) {
                errorPosition += parityBitPos;
            }
        }

        if (errorPosition > 0 && errorPosition <= length) {
            char[] correctedData = data.toCharArray();
            correctedData[errorPosition - 1] =
            (data.charAt(errorPosition - 1) == '0') ? '1' : '0';
            data = new String(correctedData);
        }

        StringBuilder originalData = new StringBuilder();
        int j = 0;
        for (int i = 1; i <= length; i++) {
            if (Math.pow(2, j) != i) {
                originalData.append(data.charAt(i - 1));
            } else {
                j++;
            }
        }

        return originalData.toString();
    }
}



private void performSelectedTechnique() {
    String input = jTextField1.getText();
    String flag = jTextField3.getText();
    String selectedOption = (String) jComboBox1.getSelectedItem();
    String output = "";

    if (selectedOption == null) {
        JOptionPane.showMessageDialog(this, "Please select an option.");
        return;
    }

    switch (selectedOption) {
        case "Bit Stuffing":
            // Convert input string to integer array
            int[] arr = new int[input.length()];
            for (int i = 0; i < input.length(); i++) {
                arr[i] = Character.getNumericValue(input.charAt(i));
            }
            output = bitStuffing(arr);
            break;
```

```
            case "Bit De-stuffing":
                int[] deStuffingInputArray = new int[input.length()];
                for (int idx = 0; idx < input.length(); idx++) {
                    deStuffingInputArray[idx] = Character.
                    getNumericValue(input.charAt(idx));
                }
                output = bitDestuffing(deStuffingInputArray);
                break;
            case "Character Stuffing":
                output = characterStuffing(input, flag);
                break;
            case "Character De-Stuffing":
                output = characterDeStuffing(input, flag);
                break;
            case "Dotted Decimal to Binary":
                output = dottedDecimalToBinary(input);
                break;
            case "Binary to Dotted Decimal":
                output = binaryToDottedDecimal(input);
                break;
            case "IPv4 to IPv6":
                output = ipv4ToIpv6(input);
                break;
            case "IP Information":
                output = ipInformation(input);
                break;
            case "Parity Check":
                output = parityCheck(input);
                break;
            case "Hamming Encode":
                output = HammingCode.encode(input);
                break;
            case "Hamming Decode":
                output = HammingCode.decode(input);
                break;
            case "Error Detection":
                output = HammingCode.detectError(input, flag);
                break;
            default:
                JOptionPane.showMessageDialog(this, "Invalid option selected.");
                return;
        }

        jTextField2.setText(output);
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
```

```
                new Home1().setVisible(true);
            }
        });
    }

    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JComboBox<String> jComboBox1;
    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.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
}
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The simulation for evaluating the performance of various data communication techniques was conducted using a custom-built Java application with a graphical user interface (GUI). The application, developed in Java Swing, provides options for simulating and visualizing different data communication protocols and techniques, including Bit stuffing, Bit De-stuffing, Character stuffing, Character De-Stuffing, Dotted Decimal to Binary, Binary to Dotted Decimal, IP Information, IPv4 to IPv6 conversion, Single Parity Check, Hamming Encoding, Hamming Decoding, and error detection/correction mechanisms.

Key components of the simulation environment include:

- **Java Swing GUI:** The user interface allows easy selection of techniques and input of data, facilitating quick and intuitive simulations.
- **Technique Implementation:** Each communication technique is implemented as a separate method within the application, ensuring modularity and ease of testing.
- **Input Handling:** The application takes various inputs, such as binary sequences, IP addresses, and plain text, depending on the selected technique.
- **Output Display:** Results of the simulations, including processed data and error detection/correction outcomes, are displayed directly on the GUI for immediate review.

The simulation procedure involves the following steps:

1. **Selection of Technique:** Users select a data communication technique from a dropdown menu.
2. **Input Data Entry:** Users input the required data in the provided text fields.
3. **Execution:** Upon pressing the 'Perform' button, the selected technique is executed with the provided input.

4. **Result Display:** The application processes the data and displays the output in the designated area of the GUI.

5. **Reset:** Users can reset the inputs and outputs using the 'Reset' button to perform new simulations.

This environment allows for comprehensive testing and evaluation of various data communication techniques in a controlled, repeatable manner.

3.2 Results Analysis/Testing

3.2.1 Graphical User Interface

The GUI provides a simple interface for users to interact with the basic data communication live code implementation.

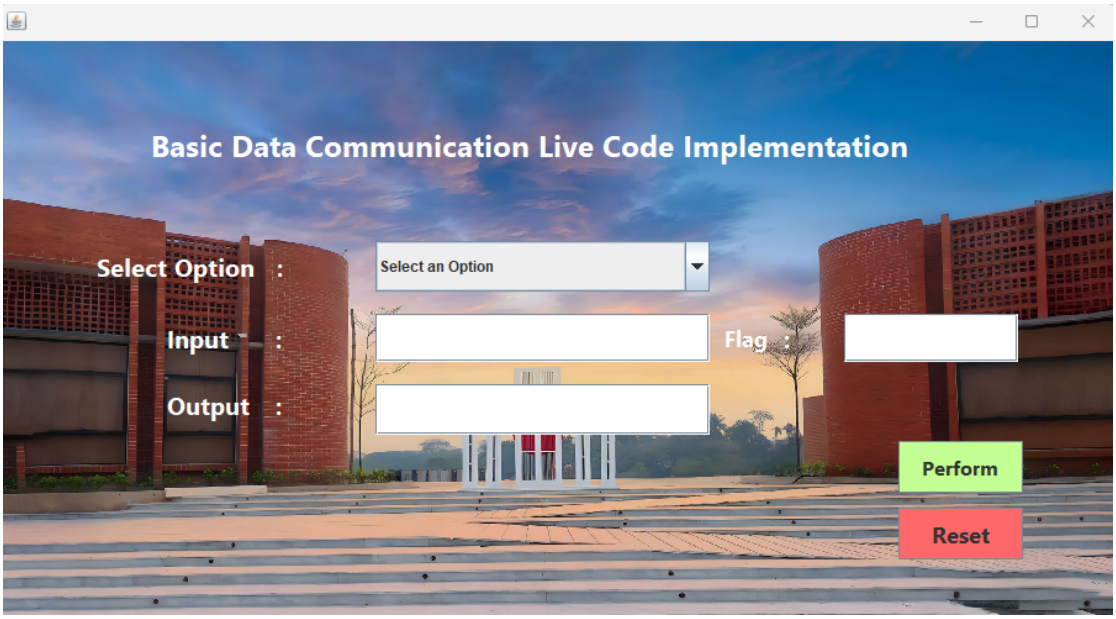

Figure 3.1: First Graphical User Interface.

3.2.2 Bit Stuffing

The bit stuffing algorithm scans the input bit sequence and inserts a '0' after every sequence of five consecutive '1's to avoid confusion with control flags.

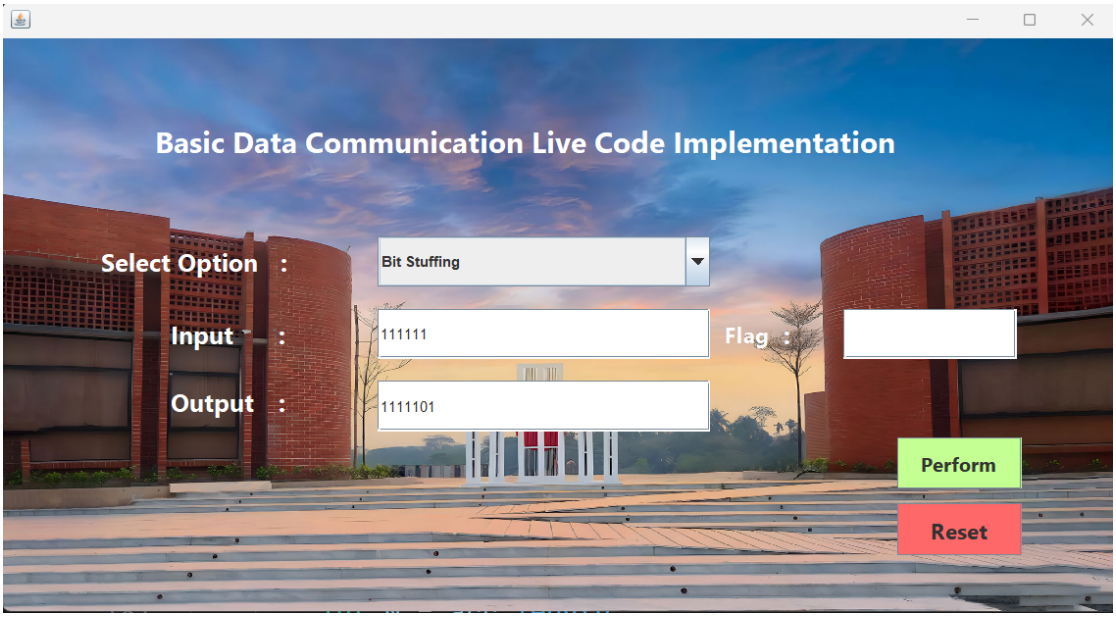

Figure 3.2: Bit Stuffing Algorithm

3.2.3 Bit De-stuffing

The bit de-stuffing algorithm reverses the bit stuffing process by removing the '0' that follows every sequence of five consecutive '1's.

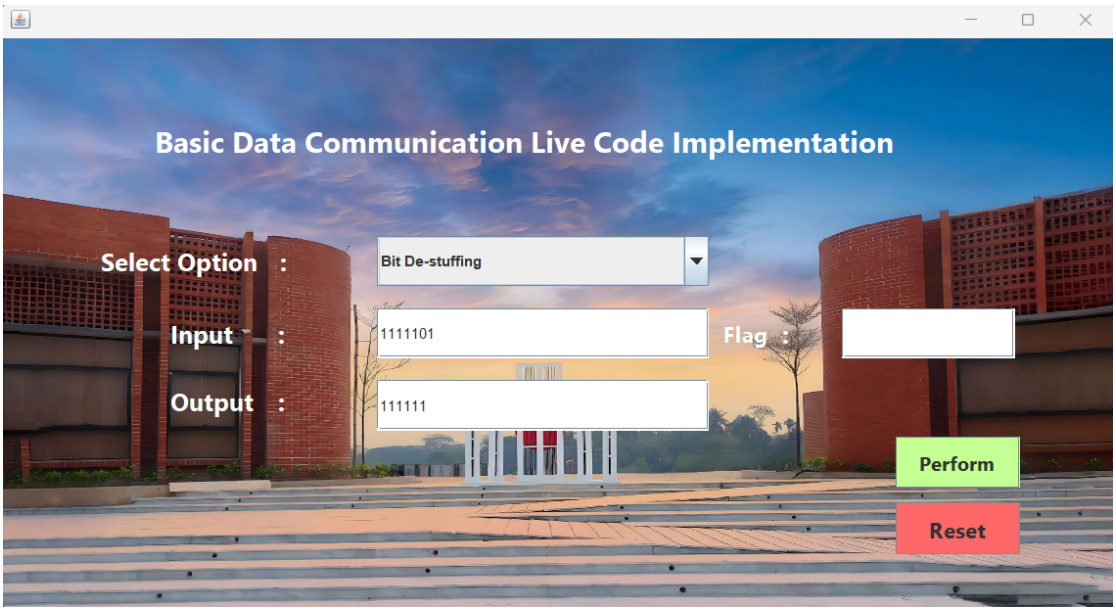

Figure 3.3: Bit De-stuffing Algorithm

3.2.4 Character Stuffing

Character stuffing replaces specific characters in the input sequence with escape sequences to differentiate data from control information.

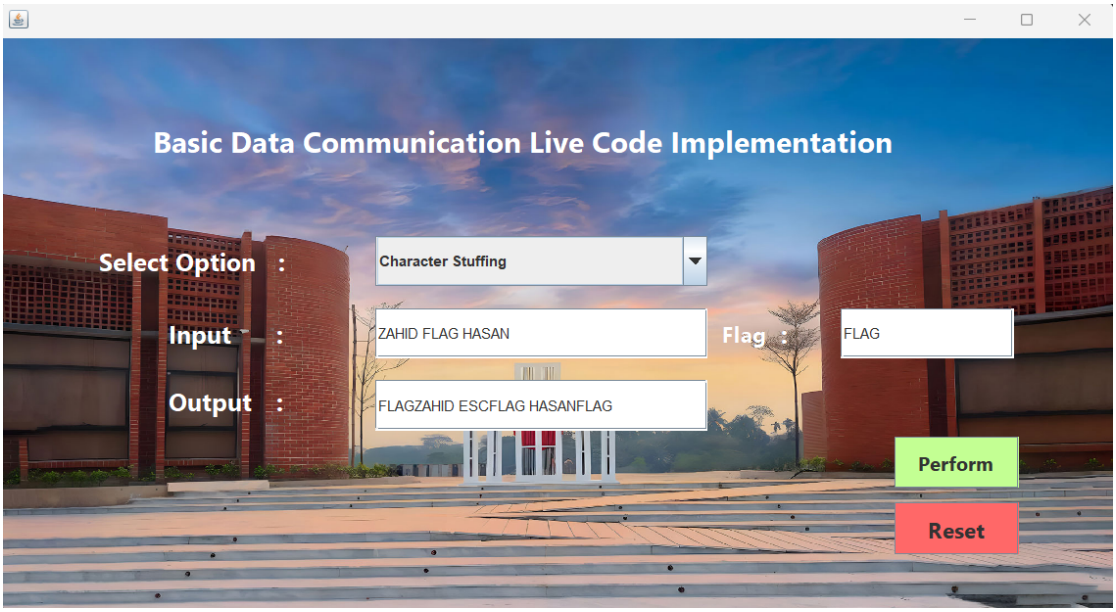

Figure 3.4: Character stuffing algorithm

3.2.5 Character De-stuffing

Character de-stuffing reverses the character-stuffing process by replacing escape sequences with the original characters.

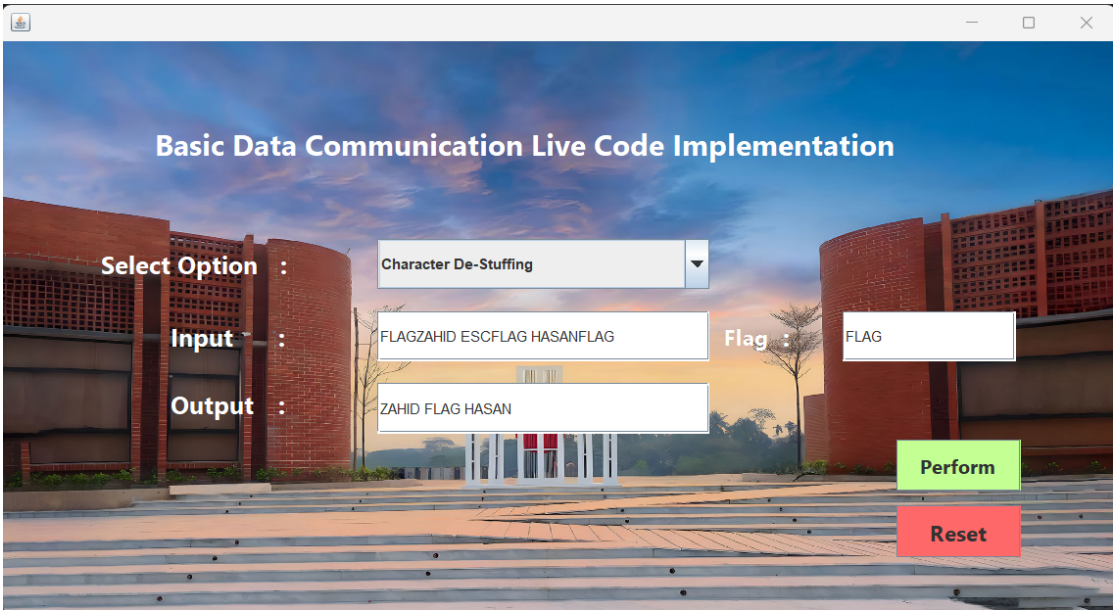

Figure 3.5: Character De-stuffing algorithm

3.2.6 Dotted Decimal to Binary

This algorithm converts each octet of a dotted decimal IPv4 address to its binary equivalent.

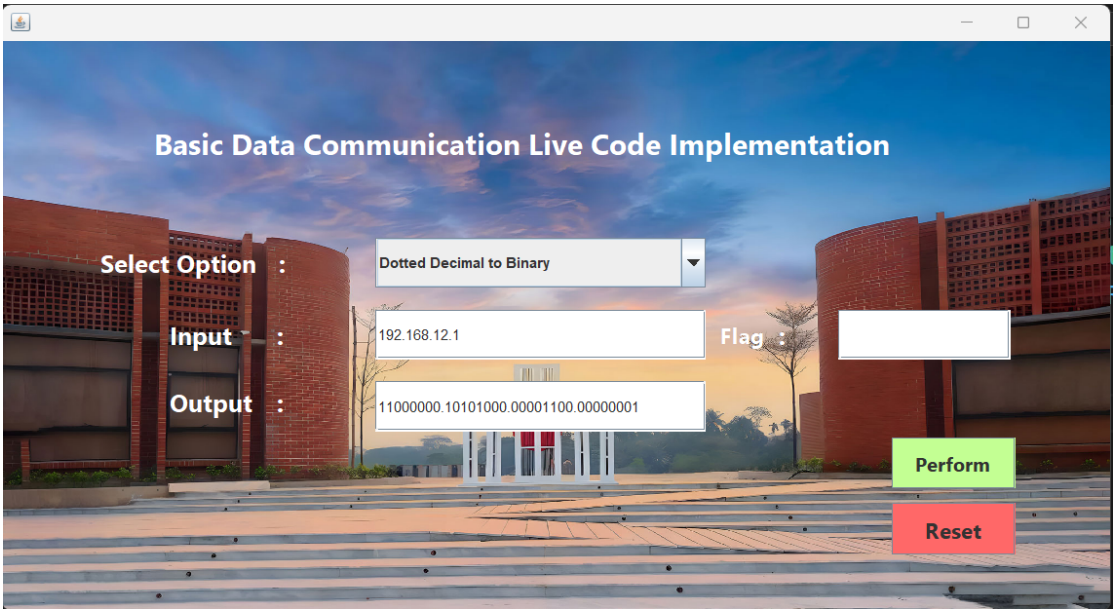

Figure 3.6: Dotted Decimal to Binary Algorithm

3.2.7 Binary to Dotted Decimal

This algorithm converts a binary IP address to its dotted decimal form.

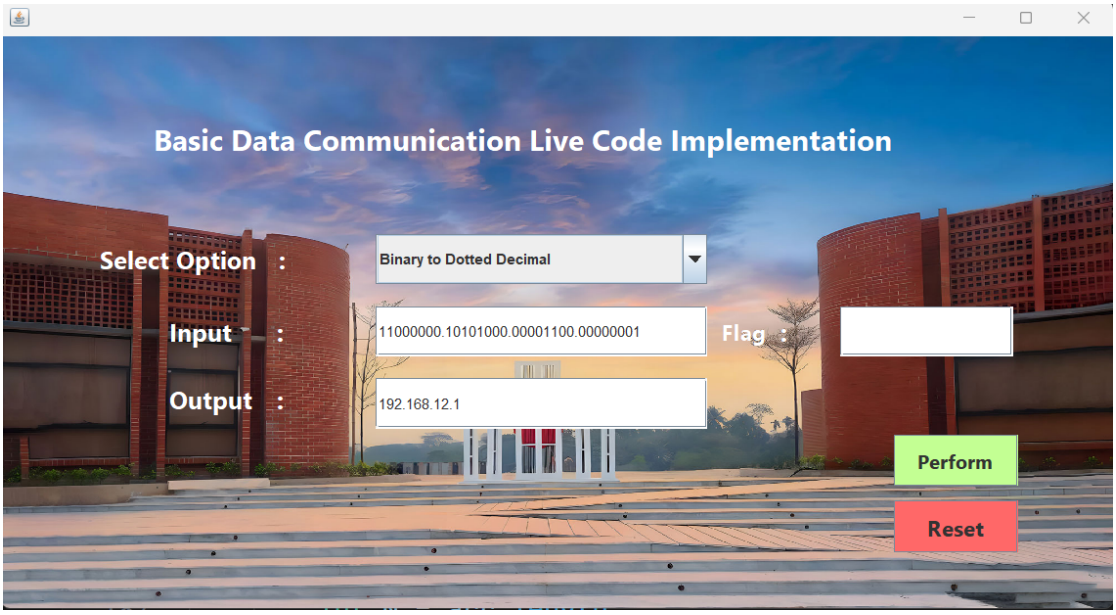

Figure 3.7: Binary to Dotted decimal algorithm

3.2.8 IPv4 Information

Detailed information retrieval for IPv4 addresses.

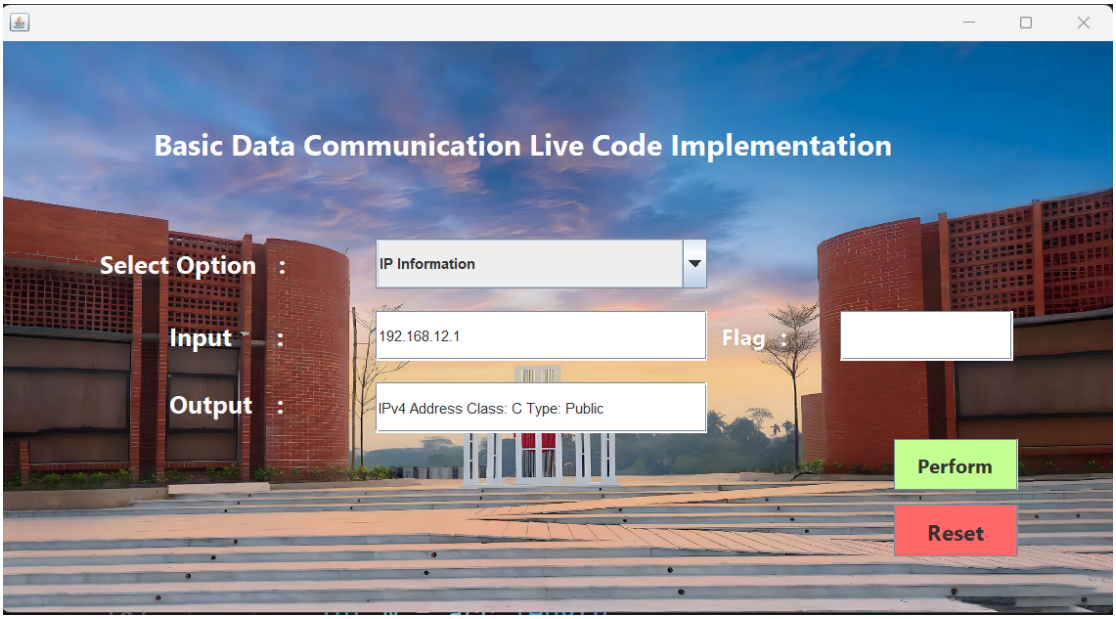

Figure 3.8: IPv4 Information

3.2.9 IPv4 to IPv6 Conversion

Converts an IPv4 address to its IPv6 equivalent using the IPv4-mapped IPv6 address format.

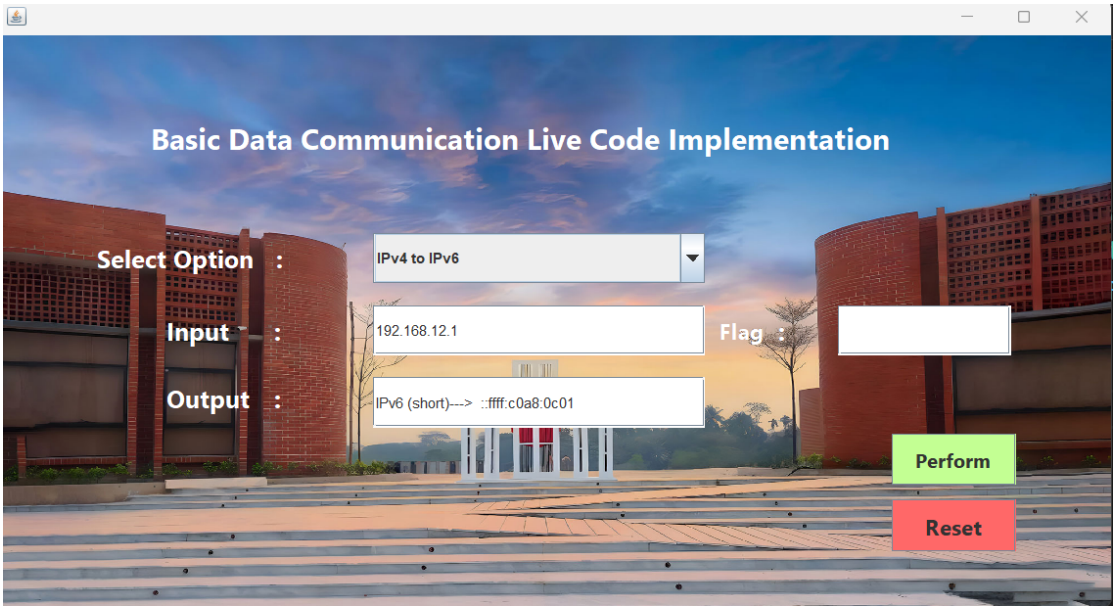

Figure 3.9: Converting IPv4 to IPv6

3.2.10 Calculate Parity for Even Parity

Calculates the parity of a given integer by converting it to binary and counting the number of '1's to determine if the parity is even or odd.

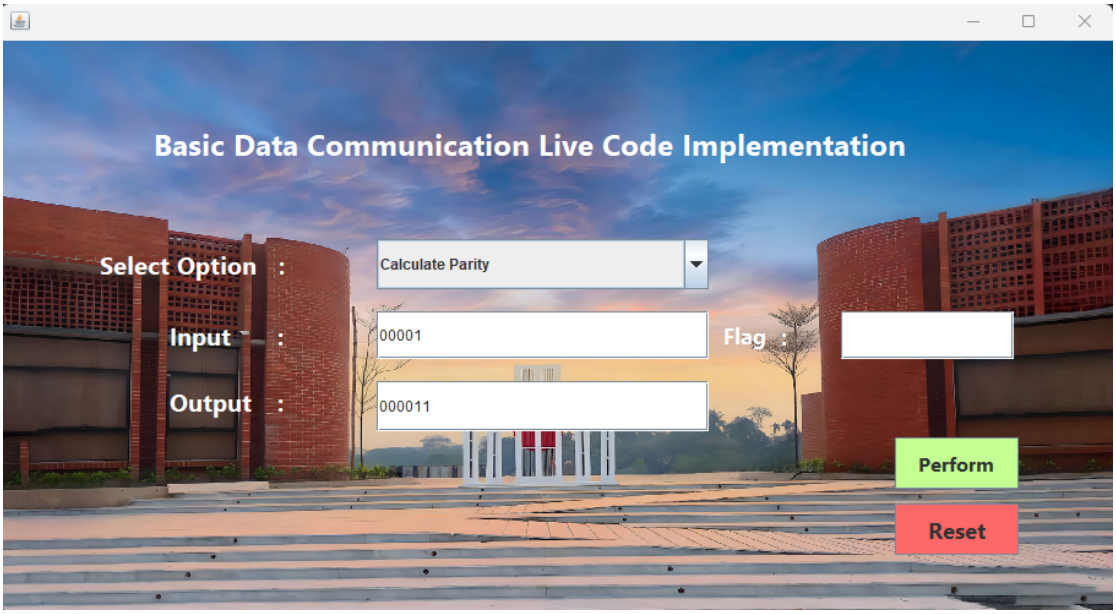

Figure 3.10: Checking Data and add single Parity bit

3.2.11 Hamming Encoding

This is a method of adding redundancy to data by inserting parity bits, enabling the correction of single-bit errors.

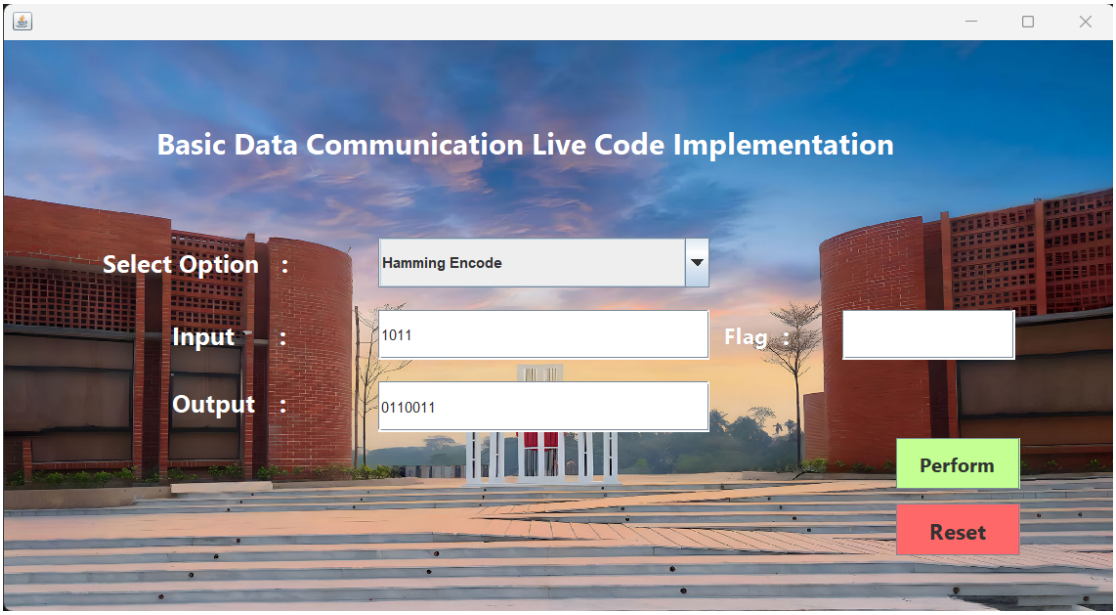

Figure 3.11: Data bit to Hamming encoding

3.2.12 Hamming Decoding

This is a technique to identify and correct single-bit errors in encoded data using the parity bits added during Hamming encoding.

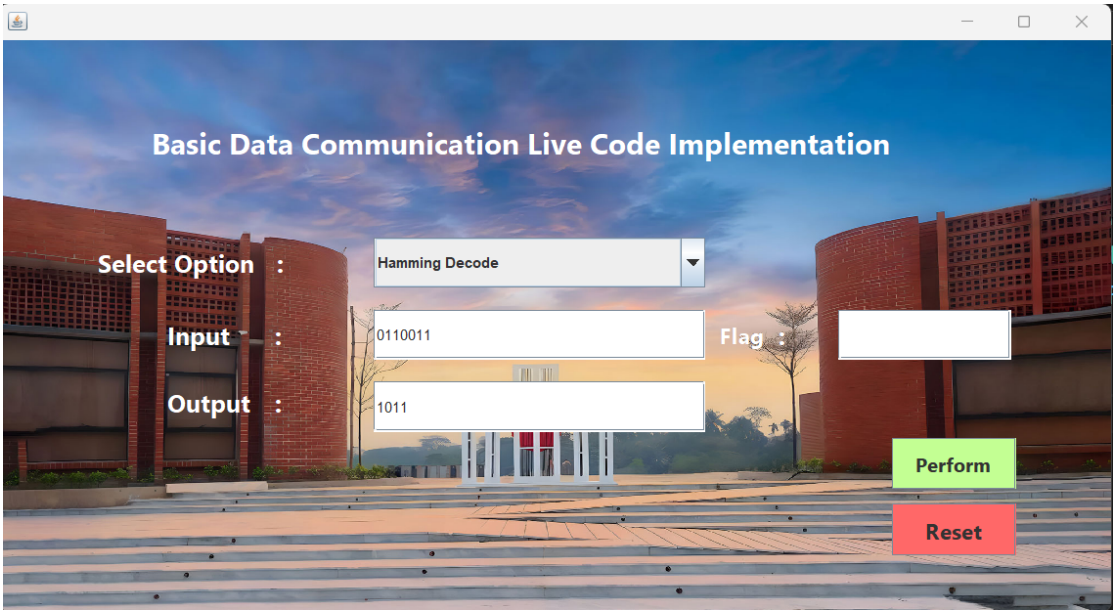

Figure 3.12: Decoding the Hamming Encoded data

3.2.13 Error Detection

General techniques to identify errors in data transmission. Here we have to input the sender data bit and receiver data bit to find out whether data transmission is correct or not.

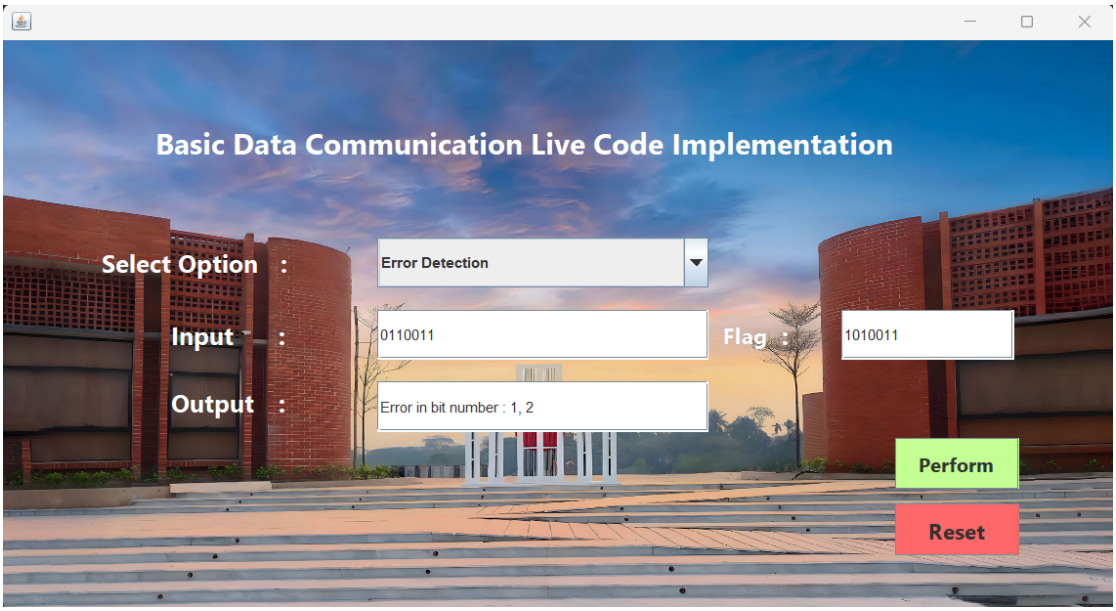

Figure 3.13: Detecting Error between sender and receiver data

Chapter 4

Conclusion

4.1 Discussion

The project successfully implemented a GUI application that enables users to apply and understand fundamental data communication techniques. Developed using Java Swing, the application offers an intuitive interface with a dropdown menu for selecting techniques such as Bit Stuffing, Bit De-stuffing, Character Stuffing, Character De-stuffing, IP address conversions, IPv4 Information, IPv4 to IPv6 Conversion, Hamming Encoding, Hamming Decoding, Error Detection, and Parity Check. Input fields allow users to enter necessary data and flags, while output fields display the processed results. Control buttons for resetting and performing actions enhance usability.

4.1.1 Effectiveness of Techniques

- Each technique's effectiveness in achieving its intended purpose is evaluated based on simulation results.
- Bit and Character Stuffing proved effective in ensuring data integrity during transmission by preventing flag or control character conflicts.
- IP address conversion techniques demonstrated accurate conversion between different formats, facilitating interoperability.
- Error detection and correction mechanisms such as Parity Check and Hamming Code showed promising error identification and correction capabilities.

4.2 Limitations

- Despite their effectiveness, some techniques have limitations. For instance, Bit Stuffing and Character Stuffing may introduce overhead, impacting efficiency.
- Error correction techniques like Hamming Code have limitations in correcting multiple-bit errors or burst errors.

- IP Information retrieval may be limited by network configurations, firewalls, or privacy settings, affecting the accuracy of results.

4.3 Scope of Future Work

1. **Enhanced Validation:** Improve input validation to handle more complex scenarios and edge cases.
2. Additional Techniques: Implement more advanced data communication techniques and protocols.
3. **Improved UI:** Enhance the user interface for better user experience and accessibility.

This project provides a strong foundation for understanding and applying fundamental data communication techniques, serving both educational and practical purposes

References

- For further references see overleaf
- Code Helping Website :www.geeksforgeeks.org
- Inspired by :Green University Of Bangladesh
- The advisor of the project Mahbubur Rahman