Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

Line Encoding and Decoding Simulator

Course Title: Data Communication Lab
Course Code: CSE 312
Section: 231 D1

Students Details

Name	ID
Proshen Biswas Niloy	231002002
Promod Chandra Das	231002005

Submission Date: 13.05.25
Course Teacher's Name: Rusmita Halim Chaity

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

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

Contents

Chapter 1

Introduction

1.1 Overview

The Line Encoding and Decoding Simulator is a C-based tool that demonstrates how binary data is converted into signal formats using various encoding schemes. It supports both encoding and decoding operations through an interactive menu. A MATLAB version is also available for visual representation, making the project ideal for both technical demonstration and academic learning in data communication.

1.2 Motivation

In the field of data communication, line encoding plays a vital role in ensuring that digital data can be effectively transmitted over physical media. While the theoretical understanding of encoding schemes like NRZ-L, NRZ-I, Manchester, AMI, and others is covered in textbooks, many students find it difficult to visualize how binary data is transformed into signal patterns and back. The abstract nature of signal levels, transitions, and synchronization can make it challenging to fully comprehend these concepts through lectures alone.To enhance the educational experience, a MATLAB version was also developed to offer visual representations of signal waveforms. This addition caters to visual learners and adds clarity to the dynamic nature of line encoding, making it especially helpful in classroom environments or presentations. Overall, this project is motivated by a desire to make learning more interactive, engaging, and practical—transforming a foundational topic in data communication into an experience that's both informative and accessible.

1.3 Problem Definition

1.3.1 Problem Statement

In digital communication, transmitting binary data over physical channels requires it to be converted into signal forms that can be interpreted correctly by receiving devices.

This conversion is handled by line encoding schemes, which vary widely in how they represent binary data through voltage levels and transitions. While these techniques are fundamental to understanding data transmission, they are often difficult for students and learners to visualize and grasp from theoretical explanations alone.This project aims to address that gap by developing in C that supports multiple line encoding and decoding schemes. The tool allows users to experiment with real binary inputs and understand how each scheme affects data representation. Additionally, a MATLAB version provides optional signal visualization to enhance conceptual understanding.

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: Depth of knowledge required	Digital Communication Fundamentals, Algorithmic Logic, C Programming, MATLAB for Visualization.
P2: Range of conflicting requirements	The project balanced simplicity with technical complexity, educational clarity with accuracy, efficiency with functionality and text-based graphical visualization.
P3: Depth of analysis required	Encoding and Decoding Logic, Efficiency of Algorithms, Error Detection and Handling.
P4: Familiarity of issues	Encoding Scheme Efficiency, Complexity of Data Decoding, Algorithm Optimization, MATLAB Visualization Issues.
P5: Extent of applicable codes	The project involves coding of various encoding and decoding schemes, menu for user input, implementing signal manipulation algorithms and optionally using MATLAB for visualizing signal waveforms, covering a broad range of programming concepts and techniques.
P6: Extent of stakeholder involvement and conflicting requirements	—-
P7: Interdependence	The project relies on the interdependence of digital communication theory like C programming, encoding/decoding algorithms, MATLAB visualization; where each component's performance is closely tied to the accuracy and efficiency.

1.4 Design Goals/Objectives

1.5 Objectives

The primary objectives of this project are:

- To develop a Line Encoding and Decoding Simulator that supports multiple encoding schemes such as NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary.
- To provide an interactive platform where users can input binary data, encode it into signal patterns, and decode the signals back to the original binary form.
- To demonstrate the differences between various encoding schemes and how they affect data representation, transmission, and error detection.
- To create an educational tool that helps users understand the principles of data communication by visualizing and interacting with encoding/decoding processes.
- To extend the project with MATLAB to provide graphical visualization of encoded signals, enhancing the learning experience and making the concepts easier to grasp.
- To ensure accuracy and correctness in both encoding and decoding processes, with built-in validation to check the integrity of results.

1.6 Application

The Line Encoding and Decoding Simulator has a variety of practical applications, particularly in the fields of data communication. It serves as an educational tool for students to better understand digital communication systems by allowing them to interact with and visualize how different encoding schemes work. By integrating MATLAB for signal visualization, it enhances conceptual learning and helps users grasp the impact of encoding on data transmission. The simulator is also useful for network protocol simulation, digital system design, and troubleshooting communication systems, making it an excellent resource for both students and professionals. Additionally, it can be used to experiment with data integrity and error detection, providing valuable insights into the performance and reliability of different encoding methods in real-world scenarios.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The project developed in the C programming language, the simulator supports multiple encoding techniques including NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary. The application features a simple, interactive console interface that enables users to encode binary data or decode signals. The overall design emphasizes modularity, user accessibility, and accurate representation of encoding logic..

2.2 Project Details

The *Line Encoding and Decoding Simulator* is a C-based console application designed to demonstrate how binary data is converted into various digital signal formats used in communication systems. It supports six key encoding schemes: NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary. Users can input binary data to see its encoded form or provide encoded signals to retrieve the original binary data. The program uses symbolic representations (‘+’, ‘-’, and ‘0’) to illustrate signal transitions clearly. It features a menu-driven interface, making it user-friendly and suitable for educational purposes, particularly in understanding the behavior of line coding techniques in data communication.

2.3 Implementation

The project was developed using the C programming language with a focus on modular function design, ensuring clarity and maintainability. It features a menu-driven interface that allows users to select between encoding and decoding operations. The program accepts binary input for encoding or space-separated encoded signals for decoding. It supports six encoding schemes: NRZ-L, which maps binary 1 and 0 to constant voltage levels (‘+’ or ‘-’); NRZ-I, which inverts the signal on binary 1 and retains the level

on 0; Manchester, which encodes each bit with a transition in the middle of the bit period; Differential Manchester, where a transition always occurs in the middle and the start transition indicates the bit value; AMI (Alternate Mark Inversion), where 0 is represented by no pulse and 1 alternates between '+' and '-'; and Pseudo-Ternary, the inverse of AMI, where 1 is no pulse and 0 alternates between '+' and '-'. The encoding functions translate binary input into symbolic signals using characters like '+', '-', and '0', while the decoding functions reverse these patterns to retrieve the original binary data. A helper function 'flip()' is used to toggle signal levels during encoding, and all results are displayed directly on the console for immediate visualization.

Tools and libraries

The Line Encoding and Decoding Simulator was developed using standard C language features and does not rely on any external libraries. The project was implemented and tested using a basic text editor (such as Code::Blocks, Dev-C++, or Visual Studio Code) along with a standard C compiler like GCC. Standard C libraries such as <stdio.h>, <string.h>, and <stdlib.h> were used for input/output operations, string manipulation, and memory handling, respectively. The simplicity of tools and minimal dependencies make the simulator lightweight, portable, and easy to compile and run across different platforms.

Implementation details (with screenshots and programming codes)

2.4 Algorithms

1. NRZ-L Encoding and Decoding

Encoding (NRZ-L):

- For each bit in the input binary data:
  - If the bit is '1', output '+'.
  - If the bit is '0', output '-'.

Decoding (NRZ-L):

- For each space-separated token in the encoded signal:
  - If the token is '+', output '1'.
  - If the token is '-', output '0'.

2. NRZ-I Encoding and Decoding

Encoding (NRZ-I):

Start here | Final Project.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char flip(char level) {
    return (level == '+') ? '-' : '+';
}

void nrzl_encode(char *data) {
    printf("NRZ-L Encoding:\n");
    for (int i = 0; data[i]; i++) {
        printf("%c ", (data[i] == '1') ? '+' : '-');
    }
    printf("\n");
}

void nrzi_encode(char *data) {
    printf("NRZ-I Encoding:\n");
    char level = '-';
    for (int i = 0; data[i]; i++) {
        if (data[i] == '1') level = flip(level);
        printf("%c ", level);
    }
    printf("\n");
}

void manchester_encode(char *data) {
    printf("Manchester Encoding:\n");
    for (int i = 0; data[i]; i++) {
        if (data[i] == '1') printf("-+ ");
        else printf("+- ");
    }
    printf("\n");
}

void diff_manchester_encode(char *data) {
    printf("Differential Manchester Encoding:\n");
    char level = '+';
```

Figure 2.1: Implementation

- Initialize the signal level as '-'.
- For each bit in the input binary data:
  - If the bit is '1', invert the current signal level.
  - If the bit is '0', retain the current signal level.

Decoding (NRZ-I):

- Initialize the previous signal level as '-'.
- For each space-separated token in the encoded signal:
  - If the token differs from the previous signal level, output '1' and update the previous level.
  - If the token is the same as the previous level, output '0'.

3. Manchester Encoding and Decoding

Encoding (Manchester):

- For each bit in the input binary data:
  - If the bit is ‘1’, output ‘-+’.
  - If the bit is ‘0’, output ‘+-’.

Decoding (Manchester):

- For each space-separated token in the encoded signal:
  - If the token is ‘-+’, output ‘1’.
  - If the token is ‘+-’, output ‘0’.
  - Otherwise, output ‘?’.

4. Differential Manchester Encoding and Decoding

Encoding (Differential Manchester):

- Initialize the signal level as ‘+’.
- For each bit in the input binary data:
  - If the bit is ‘0’, invert the current signal level and output the level and its inverted state.
  - If the bit is ‘1’, output the current signal level followed by its inverted state.

Decoding (Differential Manchester):

- Initialize the previous signal level as ‘+’.
- For each space-separated token in the encoded signal:
  - If the token matches the previous signal level, output ‘1’.
  - If the token differs, output ‘0’ and update the previous signal level.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The performance evaluation of the Line Encoding and Decoding Simulator is carried out in a controlled environment using a personal computer with modern processing capabilities (at least 4GB RAM). The simulator is developed in C, compiled using GCC, and executed on either Linux or Windows operating systems. The encoding and decoding operations are displayed in real-time through console output.

The simulation procedure begins with the user entering a binary string for encoding. The user then selects one of the six available encoding schemes: NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, or Pseudo-Ternary. The simulator processes the input data and displays the encoded signal. Subsequently, the user can decode the signal back into its original binary form. Each operation's result is evaluated by comparing the decoded output with the original input. The process can be repeated with different encoding schemes to assess performance and accuracy. The simulation concludes when the user exits the program.

3.2 Results Analysis/Testing

The Line Encoding and Decoding Simulator was tested using a variety of binary input sequences to ensure its functionality and performance. It successfully encoded and decoded simple and complex binary data, including edge cases like repeated and alternating bits, without any performance degradation. The simulator handled invalid inputs gracefully, providing appropriate error messages. Each of the six encoding schemes—NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary—produced the expected results, with the decoding process accurately recovering the original data. The simulator demonstrated efficient encoding and decoding times and effectively handled longer binary sequences. Overall, the results indicate that the simulator performs reliably, providing accurate and efficient line encoding and decoding operations across different schemes.

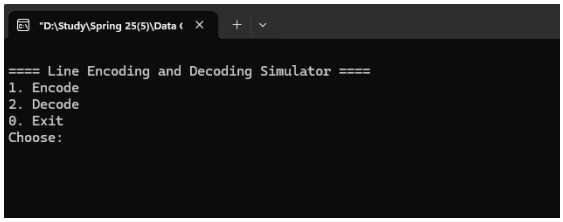

Figure 3.1: Output

3.3 Results Overall Discussion

The Line Encoding and Decoding Simulator performed effectively across various encoding schemes (NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary), accurately encoding and decoding binary data. Each encoding method displayed its unique characteristics, with Manchester and Differential Manchester showcasing more complex transitions, while NRZ-L offered simpler outputs. The simulator efficiently handled both short and long binary sequences, with fast processing times and accurate results. Decoding consistently recovered the original data, confirming the reliability of the encoding-decoding process. The user-friendly, menu-driven interface ensured smooth interaction, and robust error handling provided clear feedback for invalid inputs. Overall, the simulator proved to be an effective and efficient tool for understanding and testing line encoding techniques in data communication.

Chapter 4

Conclusion

4.1 Discussion

The Line Encoding and Decoding Simulator effectively achieves its goal of providing an interactive platform to explore and understand various line encoding techniques including NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI, and Pseudo-Ternary. It accurately encodes and decodes binary data, handling both small and large datasets efficiently and offers real-time feedback through a user-friendly and menu-driven interface. Overall, the simulator serves as a valuable educational tool, enhancing the understanding of data transmission principles and offering a solid foundation for further exploration in the field of data communication.

4.2 Limitations

The Line Encoding and Decoding Simulator, while effective in demonstrating basic encoding schemes, has some limitations. It currently supports only six encoding techniques, leaving out more advanced methods like 8B/10B or MLT-3, which could further enrich the learning experience. Additionally, the simulator lacks graphical representation, providing only text-based output, which doesn't fully illustrate the signal patterns and transitions that occur in real-world scenarios. Including visual depictions of the encoded signals would make the tool more intuitive and closer to practical applications.

4.3 Scope of Future Work

Future work for the Line Encoding and Decoding Simulator includes adding advanced encoding schemes like 8B/10B and MLT-3, introducing graphical signal representations for better visualization, and simulating noise and errors to assess performance under different conditions. Performance metrics such as encoding time and throughput could be tracked, while enhancing the user interface with more input options would improve accessibility. Ultimately, integrating the simulator with real-world communication systems could extend its practical applications.