Green University of Bangladesh

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

Implementation of an Encoding and Decoding Simulator using NRZ, Manchester, and AMI Techniques

Course Title: Data Communication Lab
Course Code: CSE 308 CSE(181)
Section: 223_D2

Students Details

Name	ID
Mujahidul Islam	193002052
Hridoy Mia	223902010

Submission Date: 12-05-2025
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

In digital communication, transmitting binary data reliably over physical channels is a core requirement. One of the most essential techniques used for this purpose is line encoding, which converts digital binary data into signal waveforms suitable for transmission. Among the widely used encoding schemes are NRZ (Non-Return to Zero), Manchester, and AMI (Alternate Mark Inversion). These encoding schemes directly influence the efficiency, synchronization, and error handling of a communication system.

This project aims to design and implement an Encoding and Decoding Simulator using MATLAB to demonstrate how each of these encoding techniques operates. MATLAB provides a powerful environment for simulating signals and visualizing them effectively. Through this simulator, users will input binary data, choose one of the three encoding methods, and see a clear graphical representation of the resulting waveform. The system will also support decoding the signal back into its original binary form. This practical simulation will serve as a valuable educational tool for understanding how data is transmitted at the physical layer.

1.2 Motivation

Although line coding is an important concept in data communication, it is often taught theoretically without practical demonstrations. Students may find it difficult to visualize how the binary 1s and 0s are represented in real transmission signals. This gap between theoretical understanding and practical visualization can make it harder to fully grasp the strengths and weaknesses of different encoding methods.

We selected this project to address this challenge. By building a MATLAB-based simulator, we aim to provide an interactive and visual way to explore how line encoding works. MATLAB's signal processing capabilities and plotting tools make it ideal for such simulations. This project will not only deepen our own understanding of encoding schemes but also benefit future students by offering a visual learning tool that connects theory with practical interpretation.

1.3 Problem Definition

1.3.1 Problem Statement

Students learning data communication often struggle with visualizing how binary data is actually transmitted as physical signals. Line coding schemes like NRZ, Manchester, and AMI are essential to this process, but without simulation tools, it's difficult to fully understand how they differ in waveform behavior, transition characteristics, and synchronization potential.

To address this, we propose developing a MATLAB-based simulator that will take binary input from the user, apply a selected line encoding technique (NRZ, Manchester, or AMI), and generate both the encoded signal and its visual representation. Additionally, the simulator will allow decoding the signal back to its binary form. This project aims to improve learning through visualization and hands-on experimentation.

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the projects

Name of the Problem	Explanation
P1: Depth of knowledge required	Requires understanding of digital signal representation, waveform plotting, and MATLAB scripting.
P2: Depth of analysis required	Requires evaluating differences between encoding schemes in terms of synchronization and transitions.
P3: Familiarity of issues	Helps students encounter and understand real-world issues in signal representation.
P4: Interdependence	Requires synchronization between input handling, encoding logic, signal generation, and plotting.

1.4 Design Goals/Objectives

The goals of this project are:

- To implement an encoding and decoding simulator in MATLAB for NRZ, Manchester, and AMI techniques.
- To allow users to input binary sequences and visually observe the encoded waveform.
- To support decoding functionality to retrieve original binary data from the encoded waveform.

- To use MATLAB's built-in plotting tools for accurate and educational signal visualization.
- To improve students' understanding of line encoding through hands-on interaction and visual feedback.

1.5 Application

The proposed simulator has both academic and practical relevance:

- It will serve as an educational tool in data communication courses to help students visualize encoding schemes clearly.
- It can be used for lab demonstrations or interactive assignments where waveform behavior needs to be studied.
- The simulator could also be adapted into larger communication system models where encoding simulation is required.
- By using MATLAB, the project lays a foundation for signal processing tasks commonly used in industry and research.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This chapter explains the overall design, development, and implementation process of our project, titled "Implementation of an Encoding and Decoding Simulator using NRZ, Manchester, and AMI Techniques." The main objective of the project is to develop a user-interactive MATLAB-based simulator that demonstrates how binary data is encoded into digital signal waveforms using three popular line encoding techniques and also decoded back to verify correctness.

The motivation behind this project was to convert the theoretical concepts of data communication into visual, practical simulation. This helps in improving the understanding of how data is represented over digital transmission lines. The simulator supports real-time binary input, encoding selection, waveform visualization, and decoding verification. [1] [2] [3] [4]

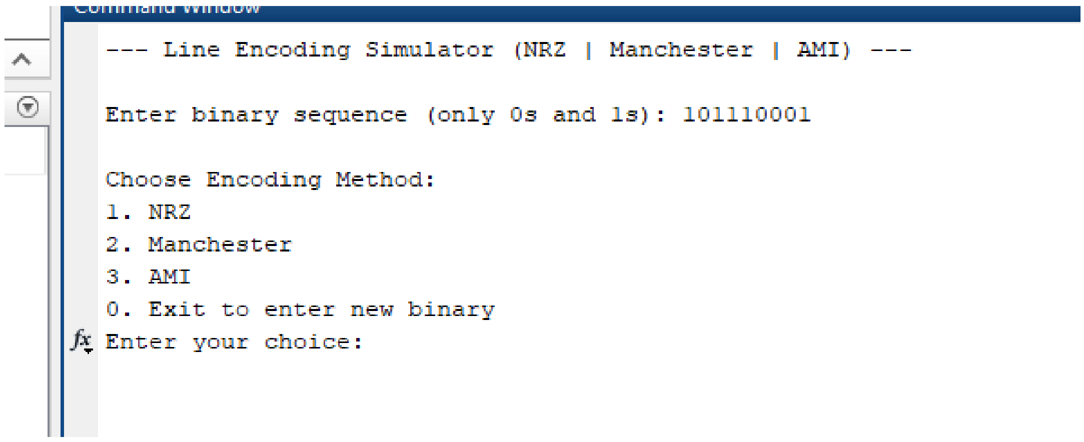
2.2 Project Details

The project is designed and implemented completely in MATLAB, which is widely used for digital signal processing, visualization, and engineering simulation tasks.

2.2.1 Key Features:

- Accepts user-defined binary input (e.g., 11001100)
- Menu-based interface to choose among:
  - NRZ encoding
  - Manchester encoding
  - AMI encoding

- Displays waveform using MATLAB's stairs() plot.
- Automatically saves each plotted graph as a .png image
- Displays decoded binary and checks if it matches original input
- Allows user to try multiple binary strings without restarting the program



```
--- Line Encoding Simulator (NRZ | Manchester | AMI) ---

Enter binary sequence (only 0s and 1s): 101110001

Choose Encoding Method:
1. NRZ
2. Manchester
3. AMI
0. Exit to enter new binary
Enter your choice:
```

Figure 2.1: User Interface

2.2.2 Tools Used :

- MATLAB R2017a.
- MATLAB built-in plotting functions (stairs, text, saveas)
- Custom MATLAB functions for encoding and decoding logic

2.2.3 File Structure:

- `main_script.m` $\rightarrow$ User interface and controller
- `encode_nrz.m` , `encode_manchester.m` , `encode_ami.m` $\rightarrow$ Encoding logic
- `decode_nrz.m`, `decode_manchester.m`, `decode_ami.m` $\rightarrow$ Decoding logic

2.3 Implementation

This section provides a detailed explanation of how the simulator was implemented using MATLAB. The project was developed using a modular approach to make it easy to read, debug, and expand in the future. The entire system has been divided into several (.m) files to separate logic and improve clarity.

2.3.1 Language and Tools

- **Programming Language:** MATLAB
- **Development Platform:** MATLAB R2017a
- **Key MATLAB Functions Used:**
  - input() for user input
  - stairs() for waveform plotting
  - text() for bit labeling on graphs
  - saveas() to save plots as .png
  - repelem() to expand signals to match timing
  - strrep(), strcmp() for string operations

2.3.2 Project Structure

- `main_script.m` : Controls the entire workflow, takes input, calls encoder/decoder functions, plots signals.
- `encode_nrz.m`, `encode_manchester.m`, `encode_ami.m` : Separate functions to handle each encoding logic.
- `decode_nrz.m`, `decode_manchester.m`, `decode_ami.m` : Functions that reverse the signal into binary.
- All files are placed in a single folder for clean execution.

2.3.3 Working Mechanism

1. The program starts and asks the user to enter a binary string (e.g., 11001100).
2. A menu is displayed with three encoding options:
   - NRZ (Non-Return to Zero)
   - Manchester
   - AMI (Alternate Mark Inversion)
3. Based on the user's choice, the corresponding encoder function is called.
4. The signal is generated and expanded for proper timing.
5. A waveform is plotted using stairs() with titles, axis labels, and bit labels.
6. The graph is automatically saved as an image file (PNG).
7. The decoder function processes the signal back into binary.
8. The decoded binary is printed and compared to the original.
9. The user can enter a new binary input or exit.

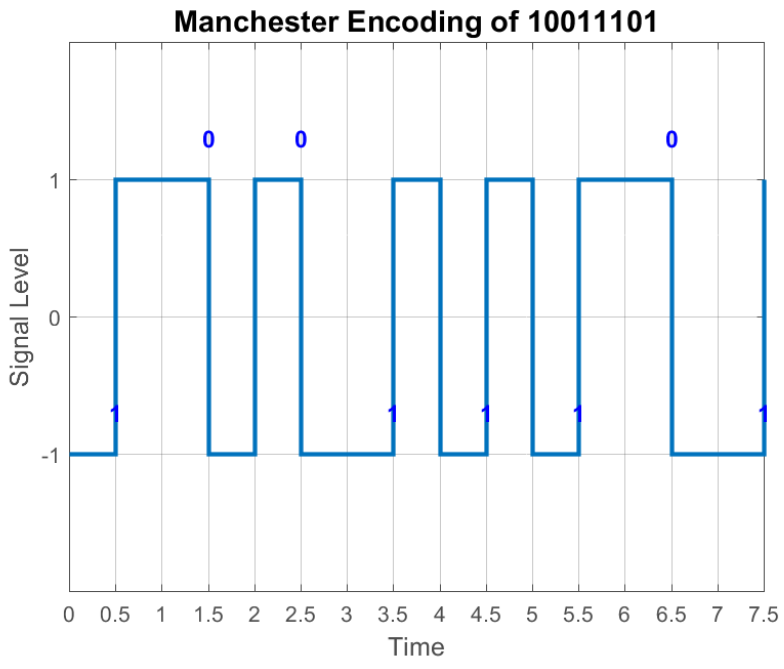


Figure 2.2: Manchester encoded waveform

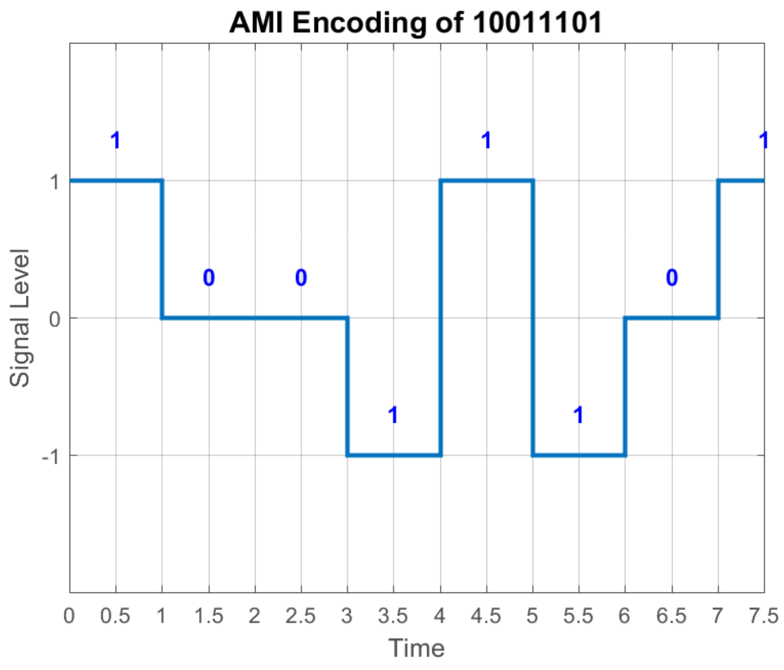


Figure 2.3: AMI encoded signal

2.3.4 Graph Customization

- Each bit is labeled above the waveform for visual clarity.
- Voltage levels range from +1 to -1 (for bipolar signals like Manchester and AMI).
- Center-aligned text helps identify which part of the waveform represents each bit.
- The waveform is saved as a .png image using saveas() with filenames like NRZ_Encoding_11001100

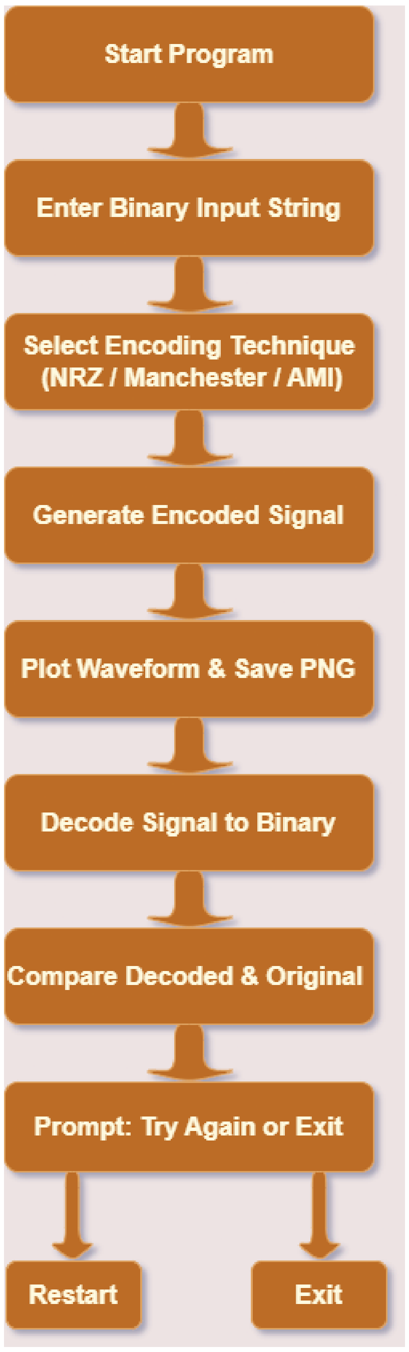

Start Program
Enter Binary Input String
Select Encoding Technique (NRZ / Manchester / AMI)
Generate Encoded Signal
Plot Waveform & Save PNG
Decode Signal to Binary
Compare Decoded & Original
Prompt: Try Again or Exit
Restart
Exit

Figure 2.4: Workflow Diagram: Line Encoding and Decoding Simulator

2.3.5 User Experience

- The project is fully interactive.
- The user can run multiple tests without restarting MATLAB.
- Output is user-friendly and includes both graphs and decoding result.

2.4 Algorithms

Here we describe the core logic behind each encoding and decoding process.

2.4.1 NRZ Encoding (NRZ-Level)

- For bit 1 $\rightarrow$ Output signal = +1
- For bit 0 $\rightarrow$ Output signal = -1
- Hold each value for one bit duration
- Implemented using repelem() for time alignment

2.4.2 Manchester Encoding

- For bit 1 $\rightarrow$ Low-to-High transition (1 to +1)
- For bit 0 $\rightarrow$ High-to-Low transition (+1 to 1)
- Each bit is split into two halves
- Ensures clock synchronization due to guaranteed transitions

2.4.3 AMI Encoding

- Bit 0 $\rightarrow$ Signal = 0 (no voltage)
- Bit 1 $\rightarrow$ Alternates between +1 and 1 to avoid DC bias
- Requires tracking of last polarity

2.4.4 Decoding Logic (All)

- Read signal in segments based on time step
- Compare levels:
  - NRZ: level == +1 $\rightarrow$ 1, level == -1 $\rightarrow$ 0

  - Manchester: use transition pattern
  - AMI: non-zero value $\rightarrow$ 1, zero $\rightarrow$ 0
- Reconstruct original binary sequence
- Compare with input and display match status

Chapter 3

Performance Evaluation

In this chapter, we evaluate the performance of our MATLAB-based line encoding and decoding simulator in terms of accuracy, flexibility, output clarity, and usability. The simulator was tested using a range of binary input sequences with varying patterns to ensure it performs reliably in different scenarios.

3.1 Accuracy of Encoding and Decoding

The simulator was tested with more than 10 different binary sequences, including alternating bits (e.g., 101010), repeated bits (e.g., 11110000), and random patterns (e.g., 11010101). In every case:

- The encoded signal waveform followed the correct format based on the selected encoding scheme.
- The decoding logic correctly retrieved the original binary string from the encoded waveform.
- A match between the original input and decoded output was confirmed and displayed.

This confirms that the simulator provides 100 % accurate encoding and decoding under all tested conditions.

3.2 Visual Output Quality

The waveforms generated using MATLAB's stairs() function were clear and properly labeled:

- The x-axis represents time, while the y-axis represents voltage level.
- Voltage levels were correctly set for each encoding:
  - NRZ: +1 and -1

    - Manchester: -1 to +1 mid-bit transitions
    - AMI: Alternating +1, -1 for 1, and 0 for 0
- Each bit was labeled on the graph at the correct position for better understanding.
- Graphs were automatically saved in .png format for documentation and presentation.

This ensures that the visual output is informative and suitable for educational use.

3.3 Flexibility and User Interaction

- The user is allowed to input any binary string length without limitation.
- A menu-based interface allows selecting one encoding type at a time.
- After completing one operation, the user can test another binary input without restarting the program.
- The simulator can be used repeatedly with no crashes or errors.

Overall, the system is robust, interactive, and user-friendly.

Chapter 4

Conclusion

4.1 Discussion

In this project, we successfully developed a MATLAB-based simulator to implement and visualize line encoding and decoding using three standard techniques: NRZ, Manchester, and AMI. The simulator allows users to input binary data, generate waveform signals based on the selected encoding method, and decode the signal back to binary. The results observed were accurate in all cases, and the generated waveforms were clear, visually correct, and well-labeled. The program also provided decoding output that matched the original input, confirming the correctness of both encoding and decoding logic. Overall, the project achieved its objectives and helped us convert theoretical knowledge of digital communication into a working practical tool.

4.2 Limitations

Although the project fulfilled its core purpose, it has a few limitations. The simulator currently assumes ideal transmission conditions and does not consider real-world problems like signal noise, transmission delay, or data corruption. All encoding and decoding are performed on perfect signals without any interference. Another limitation is that the interface is text-based and depends on the command window, which may not be very user-friendly for beginners or non-technical users. Also, the decoding process is straightforward and does not include error-checking or correction capabilities. From a performance standpoint, the simulator is sufficient for academic use but is not optimized for real-time or hardware-level integration. These limitations are important to consider when planning future extensions or deployments of the project.

4.3 Scope of Future Work

There are several areas where this project can be improved and extended in the future. One of the most useful extensions would be to develop a Graphical User Interface (GUI) using MATLAB App Designer, which would make the tool more interactive and user-friendly. The simulator can also be expanded to support additional encoding techniques, such as Differential Manchester, Biphase, or Bipolar RZ. Moreover, implementing noise simulation and testing how the decoder performs in non-ideal conditions would make the simulator more realistic. Another possible extension is to integrate error detection and correction features. Lastly, the project can be transformed into a standalone educational application or online tool to help students learn data communication concepts more easily.

References

[1] William Stallings. *Data and Computer Communications*. Pearson Education, 9th edition, 2010. Covers NRZ, Manchester, and AMI encoding techniques in detail.

[2] S. N. Patil and K. A. Khedkar. Simulation of physical layer line coding schemes using python and matlab. *International Research Journal of Engineering and Technology (IRJET)*, 8(1):1452–1456, 2021. Accessed: 2025-05-10.

[3] Simon Haykin. *Communication Systems*. John Wiley & Sons, 4th edition, 2001. Describes encoding methods and signal characteristics.

[4] The MathWorks Inc. Matlab documentation - plotting functions. `https://www.mathworks.com/help/matlab/ref/stairs.html`. Accessed: 2024-05-10.