Signal Encoding and Modulation Techniques

Technical Report - BLG 337E Principles of Computer Communication

Authors:

Mustafa Bozdoğan 150210007
Enes Saçak 150210014

Instructor:
Prof. Dr. Abdül Halim Zaim

Teaching Assistants:
Gülizar Kondel   Büşra Bayram

January 2026
Istanbul Technical University

Abstract

In this report, we simulated data transmission between two computers. We used encoding, modulation, and demodulation techniques. We worked on four modes: Digital-to-Digital, Digital-to-Analog, Analog-to-Digital, and Analog-to-Analog. We coded 16 algorithms from William Stallings' book. We also made a website using Streamlit to show how it works. Finally, we used AI tools to make our code better and faster.

Contents

1 Introduction

Computers need to change data into signals to send them to other computers. In this project, we built a simulator program. This program shows four types of transmission:

1. **Digital-to-Digital (Line Coding)**: We change digital bits to digital signals.
2. **Digital-to-Analog (Modulation)**: We change digital bits to analog waves.
3. **Analog-to-Digital (Digitization)**: We change analog waves to digital bits.
4. **Analog-to-Analog (Modulation)**: We change analog signals to different analog signals.

We used the algorithms from the textbook "Data and Computer Communications". The user can see the signals on the screen.

2 Theoretical Background

2.1 Digital-to-Digital Encoding (Line Coding)

Line coding makes digital signals from digital data. It is important for synchronization and error checking.

2.1.1 NRZ (Non-Return to Zero)

NRZ-L (Level): The voltage level shows the bit value.

- Binary 0 $\rightarrow$ Positive voltage (+V)
- Binary 1 $\rightarrow$ Negative voltage (-V)

NRZI (Invert on ones): It changes voltage when the bit is 1.

- Binary 1 $\rightarrow$ Voltage changes at the start.
- Binary 0 $\rightarrow$ No change.

Good things: It is simple and easy.
Bad things: Synchronization is hard if there are many 0s or 1s in a row.

2.1.2 Bipolar-AMI (Alternate Mark Inversion)

It uses three levels:

- Binary 0 $\rightarrow$ Zero voltage
- Binary 1 $\rightarrow$ Positive and Negative (it changes every time)

Alternating 1s helps to find errors. But if we have many 0s, we can lose synchronization.

2.1.3 Manchester Encoding (IEEE 802.3)

This is used in Ethernet. The signal changes in the middle of the bit.

- Binary 0 $\rightarrow$ High to Low
- Binary 1 $\rightarrow$ Low to High

Good things: Good synchronization. No DC component.
Bad things: It needs more bandwidth.

2.1.4 Differential Manchester

It is like Manchester but:

- Always changes in the middle.
- Binary 0 $\rightarrow$ Change at the start.
- Binary 1 $\rightarrow$ No change at the start.

2.2 Scrambling Techniques: B8ZS and HDB3

AMI has a problem with long zero sequences. Scrambling fixes this.

2.2.1 B8ZS (Bipolar with 8-Zero Substitution)

This is used in North America. If there are 8 zeros, we change them to a special pattern.
Rule:

$$8 \text{ zeros} \rightarrow 000VB0VB$$

Here, V is Violation and B is Bipolar. The receiver sees this pattern and knows they are zeros.

2.2.2 HDB3 (High Density Bipolar 3)

This is used in Europe. It replaces 4 zeros. It looks at the number of 1s to decide the pattern.
Rules:

- Odd number of 1s: $0000 \rightarrow 000V$
- Even number of 1s: $0000 \rightarrow B00V$

2.3 Digital-to-Analog Modulation

We use this to send digital data over analog lines (like phones).

2.3.1 ASK (Amplitude Shift Keying)

We change the amplitude (height) of the signal.

$$s(t) = A(t) \cdot \cos(2\pi f_c t) \tag{1}$$

For 1, amplitude is high. For 0, amplitude is zero. ASK is easy but noise affects it a lot.

2.3.2 FSK (Frequency Shift Keying)

We change the frequency.

$$s(t) = \cos(2\pi f_i t) \tag{2}$$

We use f_1 for 0 and f_2 for 1. It is better than ASK for noise.

2.3.3 PSK (Phase Shift Keying)

We change the phase (angle) of the signal.
BPSK:

$$s(t) = \cos(2\pi f_c t + \phi) \tag{3}$$

Phase is 0 for binary 0 and 180 for binary 1.

2.3.4 QAM (Quadrature Amplitude Modulation)

QAM uses both Amplitude and Phase. It can send more bits at the same time. For example, 16-QAM sends 4 bits together.

2.4 Analog-to-Digital Conversion

2.4.1 The Nyquist Sampling Theorem

To change analog to digital correctly, we must sample fast enough. The theorem says:

Sampling frequency f_s must be at least two times the max frequency f_{max}.

$$\boxed{f_s \geq 2 \cdot f_{max}} \tag{4}$$

If f_s is low, **aliasing** happens. This destroys the signal.

2.4.2 PCM (Pulse Code Modulation)

PCM has three steps:

1. **Sampling**: Taking samples.
2. **Quantization**: Giving values to samples.

$$L = 2^n \tag{5}$$

3. **Encoding**: Changing values to binary 0 and 1.

2.4.3 Delta Modulation

This is simpler. It uses 1 bit. It checks if signal goes up or down.

- If signal > approximation: output 1 (go up)
- If signal <= approximation: output 0 (go down)

2.5 Analog-to-Analog Modulation

2.5.1 AM (Amplitude Modulation)

The carrier amplitude changes with the message signal.

$$s(t) = [1 + m \cdot x(t)] \cdot \cos(2\pi f_c t) \tag{6}$$

2.5.2 FM (Frequency Modulation)

The carrier frequency changes with the message signal. FM sounds better than AM because noise is less effective.

3 Implementation

3.1 System Architecture

We used Python classes for the code. It is modular.

Listing 1: Class Structure

```
# Encoder Classes
class DigitalToDigitalEncoder:
    ALGORITHMS = ['NRZ-L', 'NRZI', 'Bipolar-AMI',
                  'Manchester', 'Differential Manchester',
                  'B8ZS', 'HDB3']
    def encode(data, algorithm) -> (time, signal)

class DigitalToAnalogModulator:
    ALGORITHMS = ['ASK', 'BFSK', 'BPSK', 'DPSK', 'QAM']
    def modulate(data, algorithm) -> (time, signal)
```

3.2 User Interface

We used **Streamlit**. It is a library for web apps. Our app has:

- A menu to choose transmission mode.
- Settings for frequency and sample rate.
- Graphs to see the waves.
- Match indicator to check if input and output are the same.

4 AI Optimization and Benchmarking

The project assignment wanted us to use **two AI tools** to make the code better. We used:

- **Google Gemini Pro** - To make it run faster (Version B).
- **OpenAI ChatGPT** - To use less memory (Version C).

4.1 Optimization Versions

4.1.1 Version A: Original Code

- We wrote this code first.
- It uses normal Python loops.
- It is easy to read but maybe slow.

4.1.2 Version B: Gemini Pro (Speed)

We asked Gemini: *"Make this code faster using NumPy."* Gemini changed loops to vectors. It used ‘np.where‘ and ‘np.repeat‘.

4.1.3 Version C: ChatGPT (Memory)

We asked ChatGPT: *"Make this code use less memory."* ChatGPT told us to use ‘float32‘ instead of ‘float64‘. It also added good comments.

4.2 Results

We tested the codes to see time and memory usage.

Table 1: Time Comparison (milliseconds)

Data Size	**Version A** (Original)	**Version B** (Gemini)	**Version C** (ChatGPT)	**C vs A**
100 samples	0.745	0.632	0.319	Fast
500 samples	1.882	2.203	1.318	Fast
1000 samples	3.614	4.413	2.556	Fast
5000 samples	18.645	22.168	18.185	Similar

Table 2: Memory Usage (kilobytes)

Data Size	**Version A** (Original)	**Version B** (Gemini)	**Version C** (ChatGPT)	**Improvement**
100 samples	1,891.90	3,504.87	1,329.34	29.7%
1000 samples	18,790.66	34,449.40	13,282.47	29.3%

4.3 Analysis

1. **Version B (Gemini):** It was sometimes slower. Creating new arrays takes time. Vectorization is not always good for small things.
2. **Version C (ChatGPT):** It saved **29% memory**. Using ‘float32‘ is a good idea.
3. **Conclusion:** For our project, saving memory is better.

5 Conclusions

In this project, we learned how signals work. We did these things:

1. We coded **16 Algorithms** for line coding and modulation.
2. We made a website so people can use it easily.
3. We used AI to fix our code. We saw that ‘float32‘ is better for memory.

You can see our project here:
https://signal-encoding-simulator-yuqjysbpdtyaloro8tp4bb.streamlit.app/

References

1. Stallings, W. (2014). *Data and Computer Communications.* Pearson.
2. Forouzan, B. A. (2013). *Data Communications and Networking.* McGraw-Hill.
3. IEEE 802.3 Standard.

A How to Run

Listing 2: Commands

```
# Download code
git clone https://github.com/nonamexishere/signal-encoding-
   simulator.git
cd signal-encoding-simulator

# Install libraries
pip install -r requirements.txt

# Run the app
streamlit run app.py
```