ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 337E

Principles of Computer Communications Assignment 1: Encoding and Modulation Techniques Simulator

Report

Group Members:

150210926	:	Mohamed Ahmed Abdelsattar Mahmoud
150210928	:	Racha Badreddine

The web app is deployed and can be accessed via:
Transmission Simulator [1]

FALL 2025 / 2026

Contents

1 Introduction

Data communication systems rely on a sequence of structured signal transformations to enable reliable information exchange. Depending on the source data and transmission medium, information undergoes operations such as encoding, modulation, and sampling. The objective of this project is to design and implement an **interactive transmission simulator** that models the end-to-end communication process between two computers, encompassing the techniques covered in the *BLG 337E Principles of Computer Communications* course.

The simulator supports four fundamental transmission modes: [2, 1]

- **Digital-to-Digital:** Represents digital data using line coding techniques for reliable detection and synchronization.

  Techniques included:

  - NRZ-L and NRZI.
  - Manchester and Differential Manchester.
  - Bipolar-AMI and Pseudoternary.
  - Scrambling schemes with the AMI family (B8ZS, HDB3).

- **Digital-to-Analog:** Conveys digital data over analog carriers using modulation, suitable for bandpass channels.

  Techniques included:

  - Amplitude Shift Keying (ASK).
  - Frequency Shift Keying (BFSK, MFSK).
  - Phase Shift Keying (BPSK, DPSK).
  - Quadrature PSK (QPSK).
  - Quadrature Amplitude Modulation (QAM).

- **Analog-to-Digital:** Converts continuous signals into digital formats via sampling and quantization.

  Techniques included:

  - Pulse Code Modulation (PCM).
  - Delta Modulation (DM).

- **Analog-to-Analog:** Modulates analog carriers with analog messages, traditionally used in broadcasting.

  Techniques included:

  - Amplitude Modulation (AM).

- Frequency Modulation (FM).
- Phase Modulation (PM).

In addition to functional simulation, the project emphasizes **software quality and performance**. Artificial intelligence–based tools were used to optimize the initial implementation, and the original and optimized versions were compared through systematic benchmarking to highlight the intersection of communication principles and modern software engineering practices. [1]

2 Implementation, Setup, and Methods

2.1 System Architecture and Software Stack

The simulator is implemented using a modular architecture that separates user interaction, signal processing logic, and result visualization. The core logic is developed in **Python**, utilizing **NumPy** for vectorized numerical operations and signal generation, and **SciPy** for advanced signal processing tasks such as Hilbert transforms and filtering.

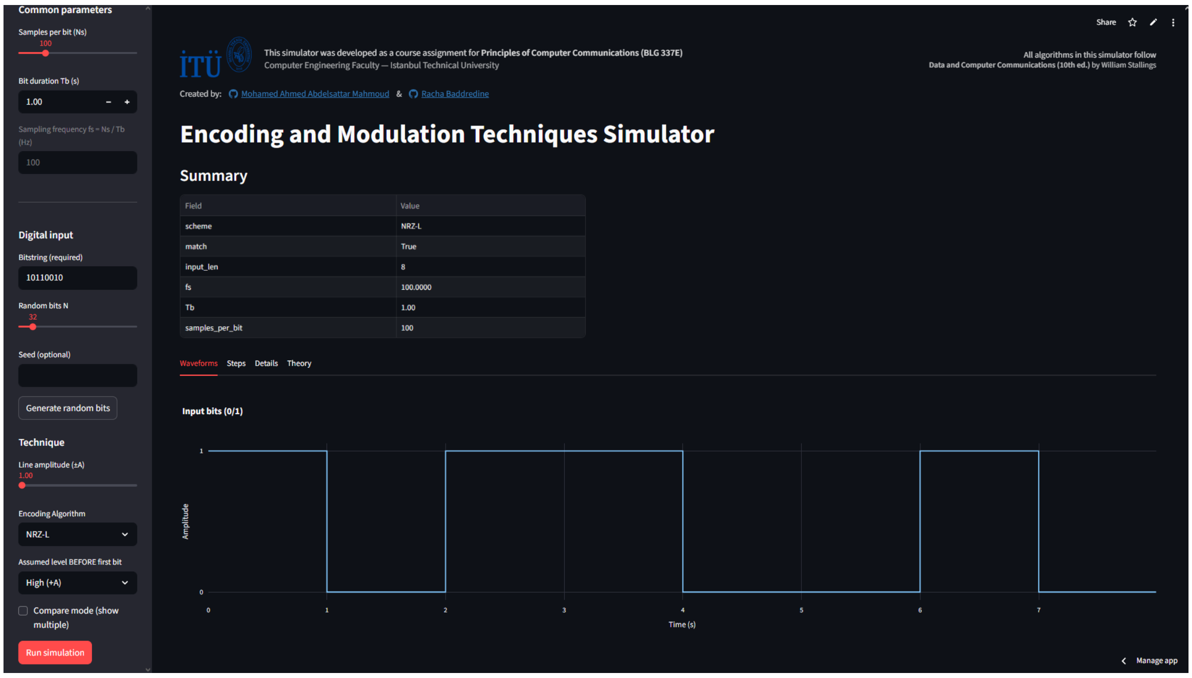
The system is deployed as a web-based application using **Streamlit**, which manages the frontend state and user inputs. Visualization is handled by **Plotly**, generating interactive waveforms that allow users to inspect signal details through zooming and panning. This separation of concerns ensures that the simulation engine remains independent of the presentation layer.

2.2 User Interaction and Configuration

The user interface adopts a centralized dashboard approach. The sidebar controls the simulation context, allowing users to: [1]

1. **Select Transmission Mode:** Choose between Digital $\rightarrow$ Digital, Digital $\rightarrow$ Analog, Analog $\rightarrow$ Digital, or Analog $\rightarrow$ Analog.
2. **Configure Techniques:** Select specific algorithms (e.g., QPSK, Delta Modulation).
3. **Tune Parameters:** Adjust critical constraints such as carrier frequency (f_c), bit duration (T_b), or quantization levels (L).
4. **Compare Schemes:** An optional "Compare Mode" overlays multiple signaling techniques (e.g., NRZ vs. Manchester) on the same plot to visually demonstrate bandwidth and synchronization differences.

Adjustments in the UI trigger a re-simulation of the signal pipeline. To maintain responsiveness, the system employs caching strategies that prevent redundant re-computations when visualization settings (like grid toggles) change but simulation parameters do not.



Common parameters

Samples per bit (Ns): 100

Bit duration Tb (s): 1.00

Sampling frequency fs = Ns / Tb (Hz): 100

Digital input

Bitstring (required): 10110010

Random bits N: 32

Seed (optional):

Technique

Line amplitude (±A): 1.00

Encoding Algorithm: NRZ-L

Assumed level BEFORE first bit: High (+A)

Compare mode (show multiple)

This simulator was developed as a course assignment for **Principles of Computer Communications (BLG 337E)**
Computer Engineering Faculty — Istanbul Technical University

All algorithms in this simulator follow
Data and Computer Communications (10th ed.) by William Stallings

Created by: Mohamed Ahmed Abdelsattar Mahmoud & Racha Baddredine

Encoding and Modulation Techniques Simulator

Summary

Field	Value
scheme	NRZ-L
match	True
input_len	8
fs	100.0000
Tb	1.00
samples_per_bit	100

Figure 1: Main user interface of the transmission simulator showing mode selection, parameter controls, and signal visualizations. [1]

2.3 Data Flow and Processing Pipeline

Each transmission mode implements a distinct data flow pipeline, transforming inputs into transmitted signals and attempting reconstruction at the receiver.

Figure 2 summarizes the four fundamental encoding and modulation approaches implemented in the simulator, illustrating the relationship between data types (digital/analog) and signal types (digital/analog).

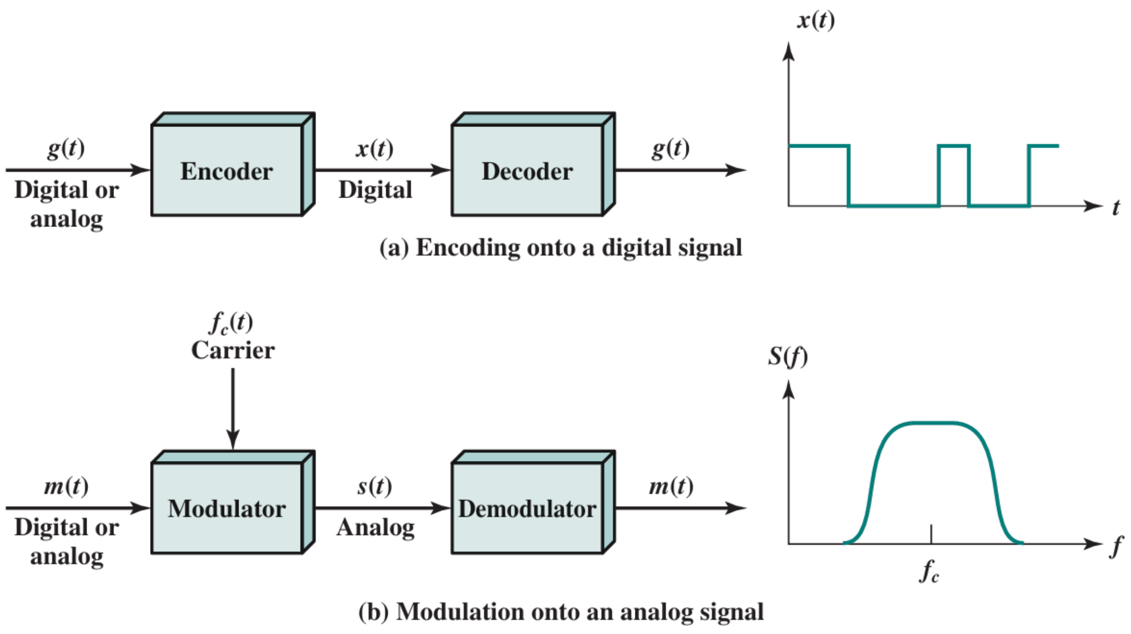


Figure 2: Encoding and Modulation Techniques. [2]

2.3.1 Digital-to-Digital Processing

The processing pipeline converts the input bitstream into a discrete-time voltage signal using two primary stages:

- **Line Coding:** Bits are mapped to voltage levels over a time axis defined by the sampling rate (f_s).
  - Schemes like **NRZ** and **AMI** map bits directly to constant amplitude levels (e.g., $+V, -V, 0$).
  - **Manchester** variants introduce mid-bit transitions to embed clock information directly into the signal.
- **Scrambling:** For AMI-based schemes, post-processing modules (**B8ZS, HDB3**) analyze the encoded buffer. They identify specific patterns of consecutive zeros and replace them with deliberate violation sequences (pulses with invalid polarity) to maintain synchronization without altering the effective data rate. [2]

2.3.2 Digital-to-Analog Modulation

The simulator generates a continuous time axis and a carrier wave to map digital data onto an analog passband channel. The process involves:

- **Binary Modulation (ASK, BFSK, BPSK):** Each bit is mapped directly to a modification of the carrier's amplitude, frequency, or phase.
- **Multilevel Modulation (QPSK, QAM):** Input bits are grouped into symbols (e.g., 2 bits for QPSK, 4 bits for 16-QAM) to increase spectral efficiency.
  - **QPSK Mapping:** Bits are grouped as pairs (b_0, b_1) and mapped to Gray-coded quadrants: $11 \to (+1, +1)$, $01 \to (-1, +1)$, $00 \to (-1, -1)$, and $10 \to (+1, -1)$.
  - **QAM Splitting:** The bitstream is split into two substreams: odd bits define the In-phase (I) component and even bits define the Quadrature (Q) component. These are modulated onto orthogonal carriers (cos and sin) and summed. [2]

2.3.3 Analog-to-Digital Conversion

The simulator implements two distinct digitization strategies:

- **Pulse Code Modulation (PCM):** The analog message undergoes a three-step process:
  1. **Sampling:** The signal is discretized at rate f_s.
  2. **Quantization:** Samples are mapped to the nearest level using a uniform mid-rise quantizer with step size $\Delta = (v_{\max} - v_{\min})/L$. The reconstruction level is defined as:

$$\hat{x} = v_{\min} + \left(k + \frac{1}{2}\right)\Delta$$

3. **Encoding:** The resulting indices k are converted into binary words.

- **Delta Modulation (DM):** The system uses a feedback loop to track signal changes. It outputs a single bit per sample (1 if the signal rose, 0 if it fell) and updates an internal staircase approximation by a fixed step size $\pm\Delta$.

2.3.4 Analog-to-Analog Modulation

The simulator generates baseband messages (Sine/Triangle) and modulates them onto a high-frequency carrier $c(t) = A_c \cos(2\pi f_c t)$.[1]

- **Amplitude Modulation (AM):** The carrier amplitude is varied in proportion to the normalized message $m(t)$ with modulation index μ:

$$s_{\mathrm{AM}}(t) = A_c \left[1 + \mu\, m(t)\right] \cos(2\pi f_c t)$$

- **Frequency Modulation (FM):** The instantaneous frequency is modified based on the message integral and frequency sensitivity k_f:

$$s_{\mathrm{FM}}(t) = A_c \cos\!\Big(2\pi f_c t + 2\pi k_f \int_0^t m(\tau)\, d\tau\Big)$$

- **Phase Modulation (PM):** The instantaneous phase is modified directly by the message and phase sensitivity k_p:

$$s_{\mathrm{PM}}(t) = A_c \cos(2\pi f_c t + k_p\, m(t))$$

At the receiver, the simulator applies an analytic signal transformation (via the Hilbert transform) to extract the instantaneous envelope (for AM) or instantaneous phase (for FM/PM). [2]

2.4 Assumptions and Timebase Conventions

To ensure consistent visualization, the simulator assumes a noise-free, ideal channel. Time is discretized based on a unified sampling convention:

- **Digital Modes:** A bit duration T_b is defined, with N_s samples per bit, yielding a sampling rate defined as:

$$f_s = \frac{N_s}{T_b}$$

- **Carrier Constraints:** The carrier frequency f_c is constrained relative to the sampling rate ($f_c < f_s/2$) to prevent aliasing, and relative to the message bandwidth to ensure the modulation is physically meaningful.

[1] In the implementation, the carrier is generated using a sine reference (e.g., $\sin(2\pi f_c t)$). This is equivalent to the cosine form in the theory up to a constant phase shift, since $\sin(\omega t) = \cos(\omega t - \pi/2)$. Therefore, bandwidth, modulation indices, and recovered baseband signals are unaffected by this choice of phase reference.

2.5 Verification and Testing Framework

The implementation is validated using an automated test suite built with `pytest`. The testing strategy focuses on:

- **Invariant Checks:** Ensuring outputs have correct array shapes and finite values.
- **Round-trip Correctness:** Verifying that, in a noiseless channel, the decoded bits match the input bits (for digital modes) and recovered signals correlate with the message (for analog modes).
- **Deterministic Fuzzing:** Running simulations with seeded random inputs to detect edge-case failures.

This framework was also used to benchmark the performance of the original implementation against AI-optimized refactoring candidates. [1]

Test Execution and Regression Testing:
The test suite supports cross-version regression testing via environment variables, allowing identical tests to run against different implementation backends:

```
# Run all tests against the original implementation (default)
pytest -q

# Run against GPT-optimized implementation
TEST_TARGET=GPT_optimized pytest -q -s

# Run against Gemini-optimized implementation
TEST_TARGET=gemini_optimized pytest -q -s
```

Test Coverage Summary:
Table 1 summarizes the validation scope for each transmission mode.

Mode	**Test file**	**What it validates (high-level)**
Digital → Digital	`test_d2d.py`	Line coding rules, scrambling substitutions (B8ZS/HDB3), violation-pattern decoding, and exact bit recovery.
Digital → Analog	`test_d2a.py`	Modulation/demodulation correctness (ASK/FSK/PSK/QAM) and symbol grouping sanity.
Analog → Digital	`test_a2d.py`	PCM quantization/encoding consistency and Delta Modulation step tracking.
Analog → Analog	`test_a2a.py`	AM/FM/PM modulation identities and Hilbert-based demodulation quality.

Table 1: Mode-wise test coverage summary.

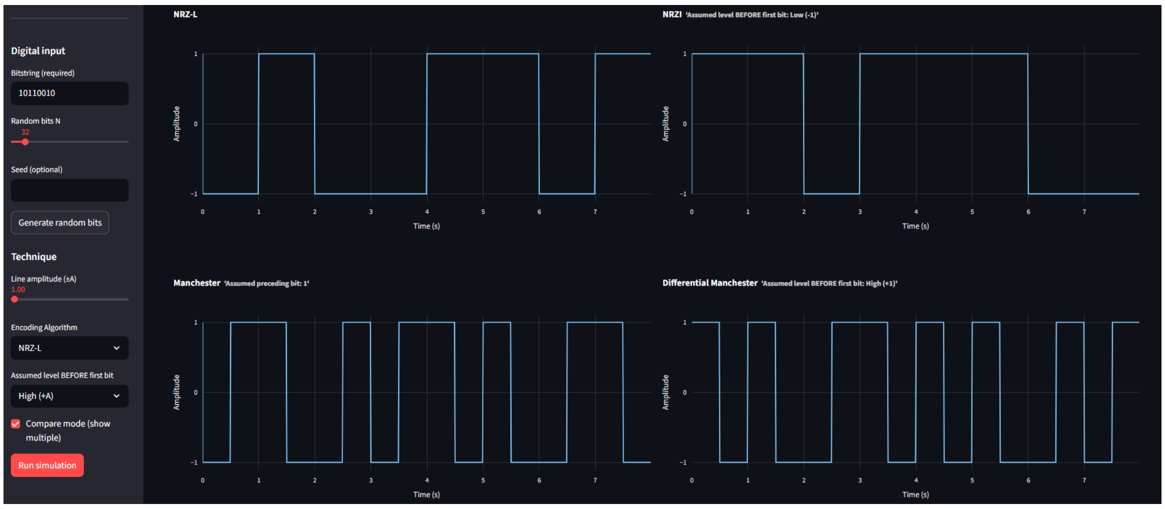
3 Results

This section presents the output waveforms generated by the simulator across the four transmission modes, as well as the performance benchmarking results comparing the original and AI-optimized implementations.

3.1 Transmission Waveforms

3.1.1 Digital-to-Digital Outputs

The waveforms below illustrate the behavior of various line coding schemes for the bit sequence `10110010`. Figure 3 contrasts simple non-return-to-zero schemes (NRZ-L, NRZI) with self-synchronizing schemes (Manchester). Figure 4 demonstrates how scrambling techniques (B8ZS, HDB3) introduce intentional violations to break long sequences of zeros. **See Section 4.1.1 for a detailed discussion on synchronization trade-offs and spectral efficiency.**


Digital input

Bitstring (required): 10110010

Random bits N: 32

Seed (optional):

Technique

Line amplitude (±A): 1.00

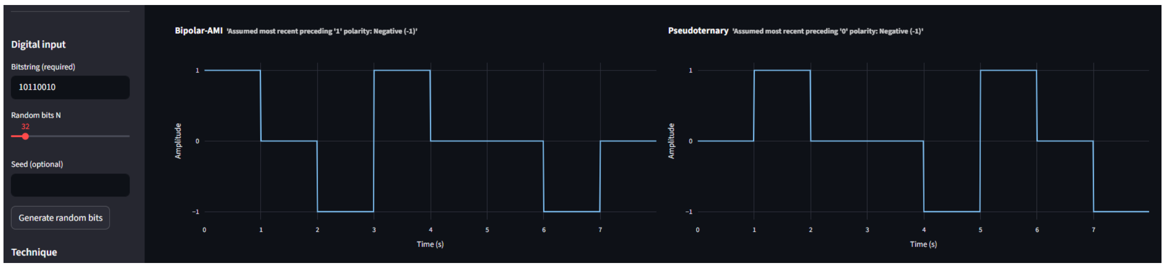
Encoding Algorithm: NRZ-L

Assumed level BEFORE first bit: High (+A)

[✓] Compare mode (show multiple)

Generate random bits
Run simulation
NRZ-L
NRZI 'Assumed level BEFORE first bit: Low (-1)'
Manchester 'Assumed preceding bit: 1'
Differential Manchester 'Assumed level BEFORE first bit: High (+1)'
Amplitude
Time (s)

(a) **NRZ-L, NRZI, Manchester, and Differential Manchester** line coding schemes applied to the input bit sequence.


Digital input

Bitstring (required): 10110010

Random bits N: 32

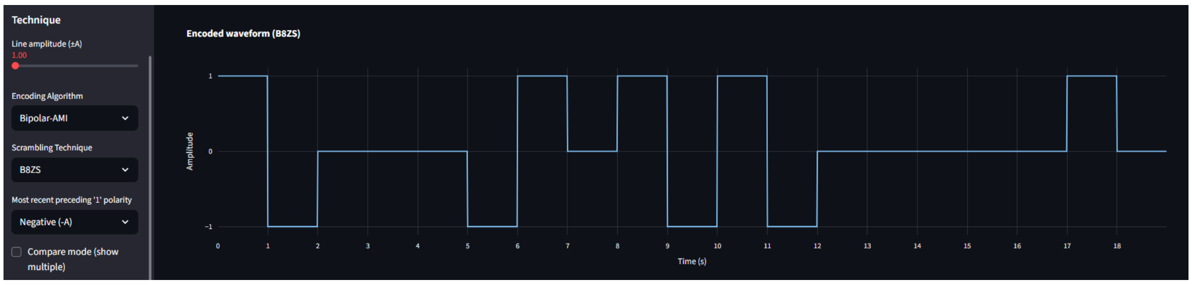
Seed (optional):

Technique

Generate random bits
Bipolar-AMI 'Assumed most recent preceding '1' polarity: Negative (-1)'
Pseudoternary 'Assumed most recent preceding '0' polarity: Negative (-1)'
Amplitude
Time (s)

(b) **Bipolar AMI** and **Pseudoternary** line coding schemes applied to the input bit sequence.

Figure 3: Digital-to-digital line coding waveforms generated by the simulator for the input bit sequence (10110010). [1]


Technique

Line amplitude (±A)
1.00

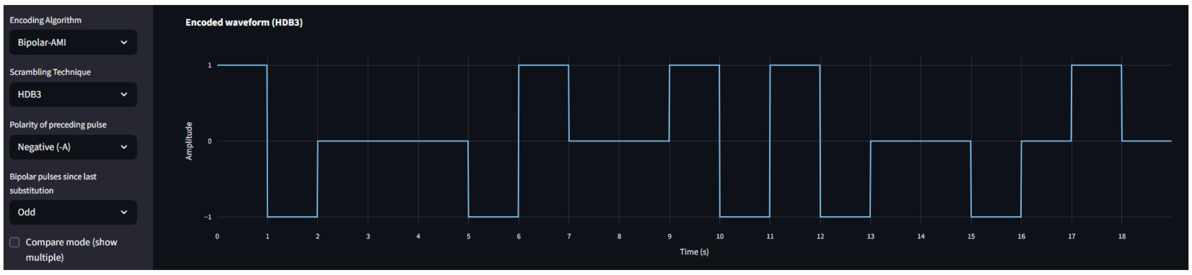
Encoding Algorithm
Bipolar-AMI

Scrambling Technique
B8ZS

Most recent preceding '1' polarity
Negative (-A)

Compare mode (show multiple)

(a) AMI with **B8ZS** scrambling showing substitution patterns.


Encoding Algorithm
Bipolar-AMI

Scrambling Technique
HDB3

Polarity of preceding pulse
Negative (-A)

Bipolar pulses since last substitution
Odd

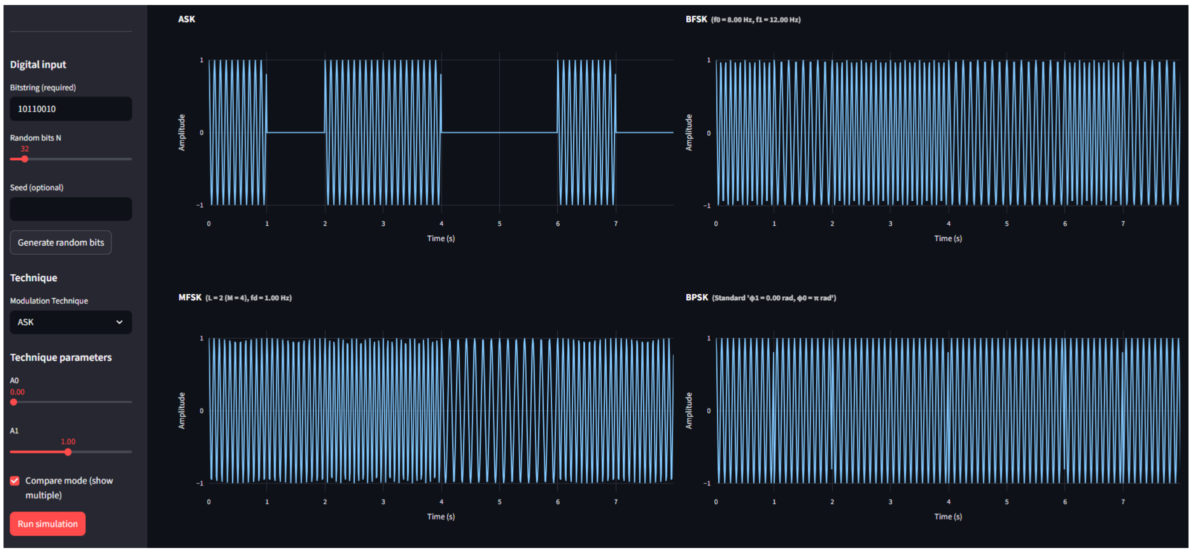
Compare mode (show multiple)

(b) AMI with **HDB3** scrambling showing violation patterns.

Figure 4: Scrambling simulation results showing the replacement of long zero-sequences (Input: 1100000000110000010). [1]

3.1.2 Digital-to-Analog Outputs

Figure 5 compares binary modulation schemes (ASK, BPSK) against multilevel schemes (QAM, 16-QAM). The plots highlight how multilevel schemes encode multiple bits per symbol change, affecting the signal complexity. **The impact of these schemes on bandwidth efficiency and noise susceptibility is analyzed in Section 4.1.2.**


Digital input

Bitstring (required): 10110010

Random bits N: 32

Seed (optional):

Technique

Modulation Technique: ASK

Technique parameters

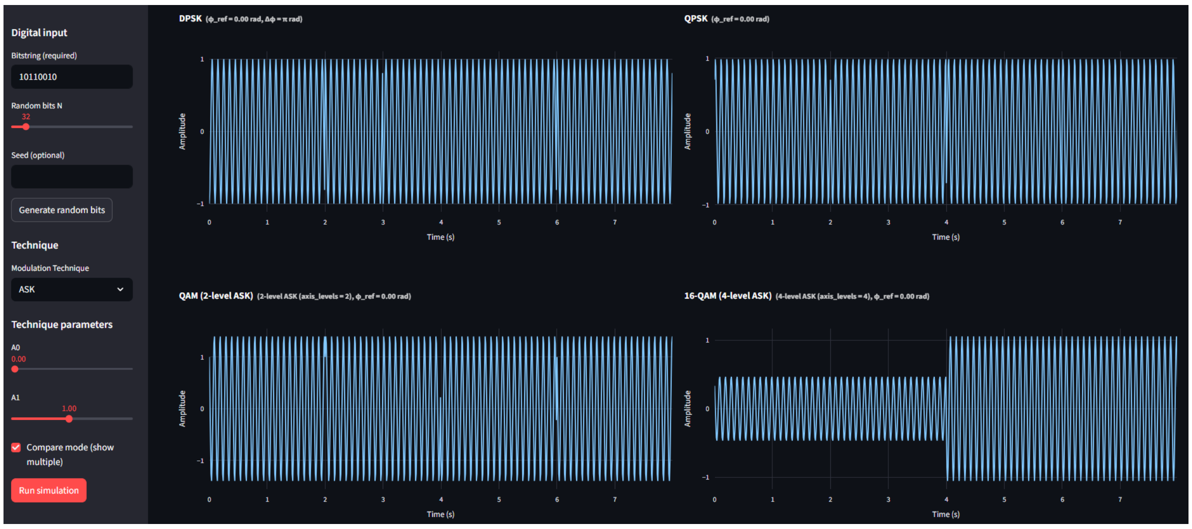
A0: 0.00

A1: 1.00

[✓] Compare mode (show multiple)

Generate random bits
Run simulation
ASK
BFSK (f0 = 8.00 Hz, f1 = 12.00 Hz)
MFSK (L = 2 (M = 4), fd = 1.00 Hz)
BPSK (Standard 'ϕ1 = 0.00 rad, ϕ0 = π rad')
Amplitude
Time (s)

(a) **ASK, BFSK, MFSK, and BPSK** modulation waveforms.


Digital input

Bitstring (required): 10110010

Random bits N: 32

Seed (optional):

Technique

Modulation Technique: ASK

Technique parameters

A0: 0.00

A1: 1.00

[✓] Compare mode (show multiple)

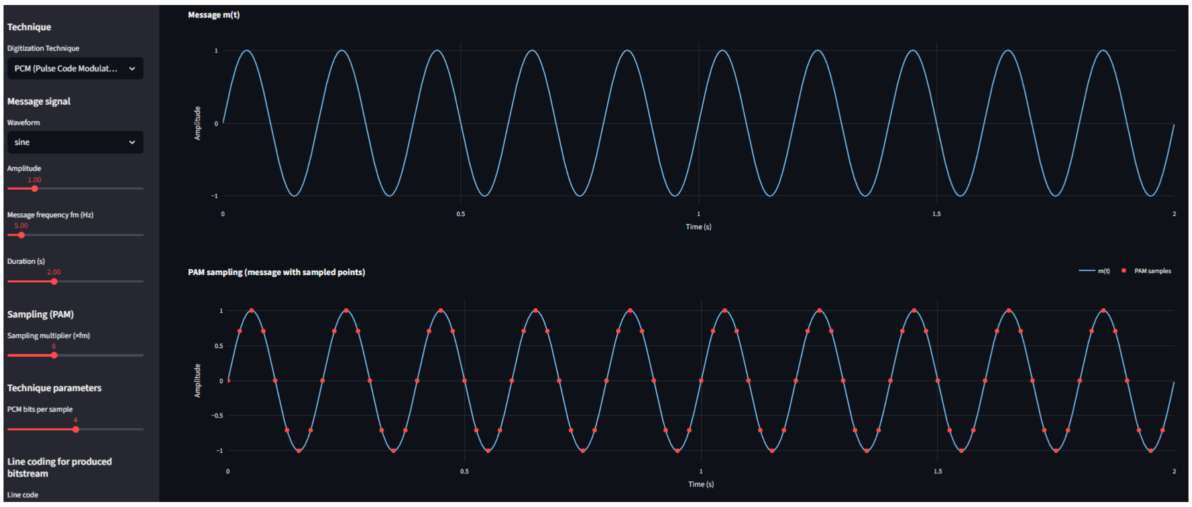
Generate random bits
Run simulation
DPSK (ϕ_ref = 0.00 rad, Δϕ = π rad)
QPSK (ϕ_ref = 0.00 rad)
QAM (2-level ASK) (2-level ASK (axis_levels = 2), ϕ_ref = 0.00 rad)
16-QAM (4-level ASK) (4-level ASK (axis_levels = 4), ϕ_ref = 0.00 rad)
Amplitude
Time (s)

(b) **DPSK, QPSK, QAM, and 16-QAM** modulation waveforms.

Figure 5: Digital-to-analog modulation outputs for the input bit sequence (10110010). [1]

3.1.3 Analog-to-Digital Outputs

The digitization process results are shown below. Figure 6 visualizes the quantization error inherent in PCM as a "staircase" approximation. Figure 7 illustrates Delta Modulation, where the reconstruction tracks the slope of the input signal. **Specific artifacts such as quantization noise and slope overload are interpreted in Section 4.1.3.**



Technique

Digitization Technique: PCM (Pulse Code Modulat…

Message signal

Waveform: sine

Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

Sampling (PAM)

Sampling multiplier (×fm): 8

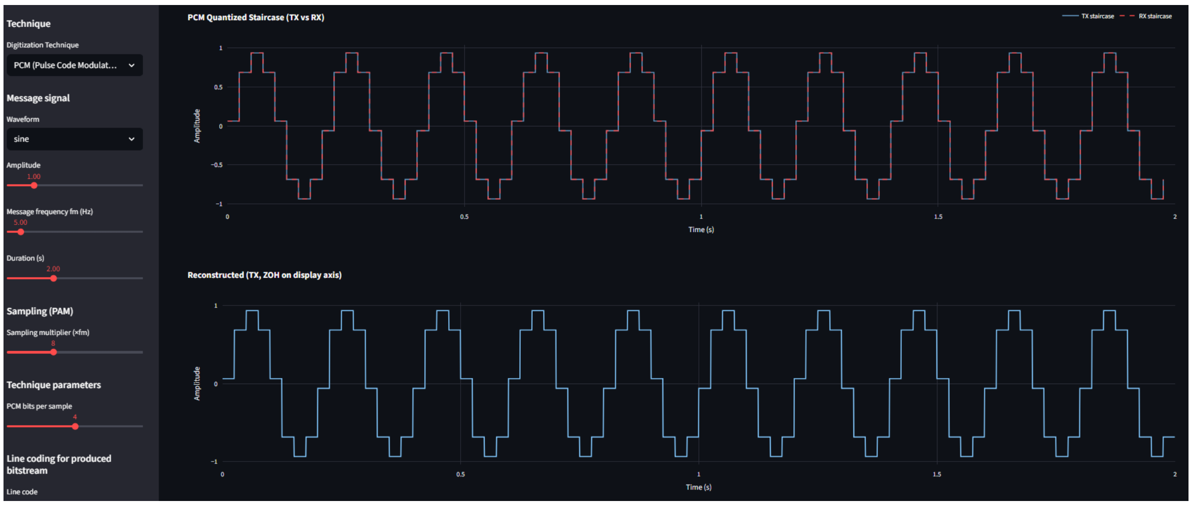
Technique parameters

PCM bits per sample: 4

Line coding for produced bitstream

Line code

(a) Original analog message and PAM samples.


PCM Quantized Staircase (TX vs RX)
TX staircase
RX staircase
Amplitude
Time (s)
Reconstructed (TX, ZOH on display axis)

Technique

Digitization Technique: PCM (Pulse Code Modulat…

Message signal

Waveform: sine

Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

Sampling (PAM)

Sampling multiplier (×fm): 8

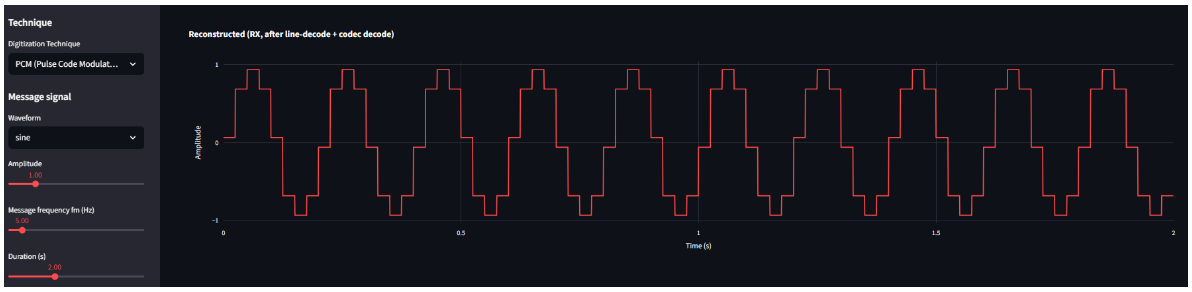
Technique parameters

PCM bits per sample: 4

Line coding for produced bitstream

Line code

(b) PCM quantized staircase signal (Transmitter).



Technique

Digitization Technique: PCM (Pulse Code Modulat…

Message signal

Waveform: sine

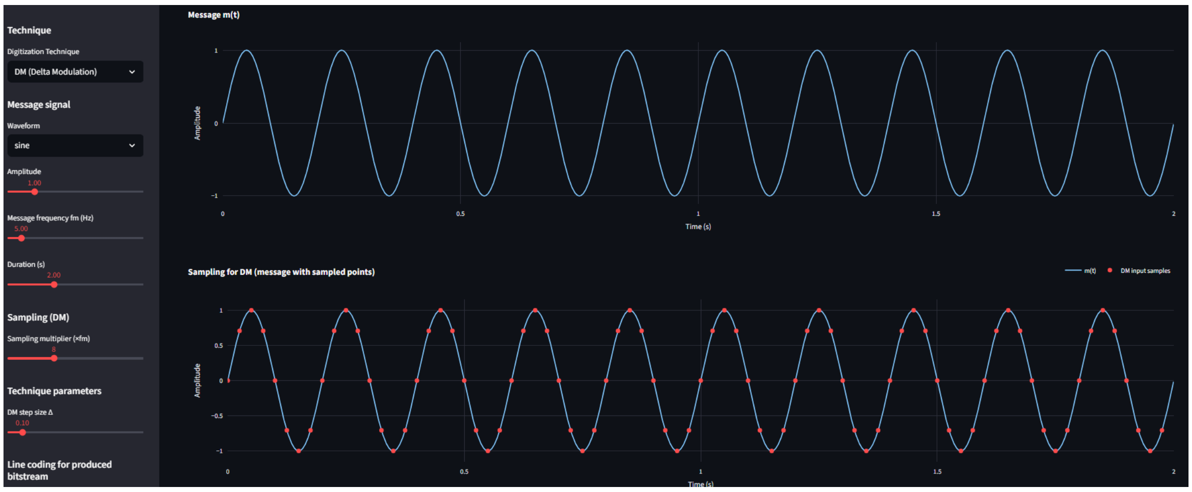
Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

(c) Reconstructed signal at the receiver.

Figure 6: Pulse Code Modulation (PCM) simulation results. [1]


Technique

Digitization Technique

DM (Delta Modulation)

Message signal

Waveform

sine

Amplitude

1.00

Message frequency fm (Hz)

5.00

Duration (s)

2.00

Sampling (DM)

Sampling multiplier (×fm)

8

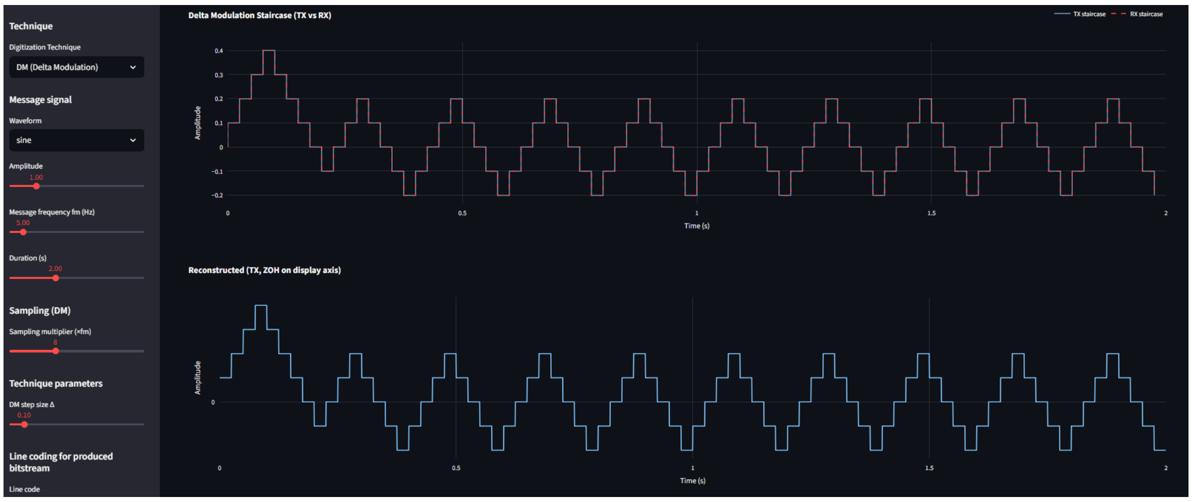
Technique parameters

DM step size Δ

0.10

Line coding for produced bitstream

(a) Original message and sampling points.


Technique

Digitization Technique

DM (Delta Modulation)

Message signal

Waveform

sine

Amplitude

1.00

Message frequency fm (Hz)

5.00

Duration (s)

2.00

Sampling (DM)

Sampling multiplier (×fm)

8

Technique parameters

DM step size Δ

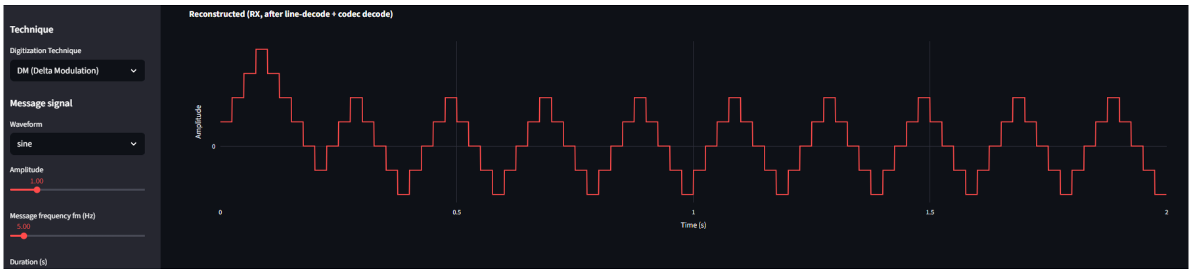
0.10

Line coding for produced bitstream

Line code

Delta Modulation Staircase (TX vs RX)
TX staircase
RX staircase
Amplitude
Time (s)
Reconstructed (TX, ZOH on display axis)
Amplitude
Time (s)

(b) Delta modulation staircase approximation (Transmitter).


Technique

Digitization Technique

DM (Delta Modulation)

Message signal

Waveform

sine

Amplitude

1.00

Message frequency fm (Hz)

5.00

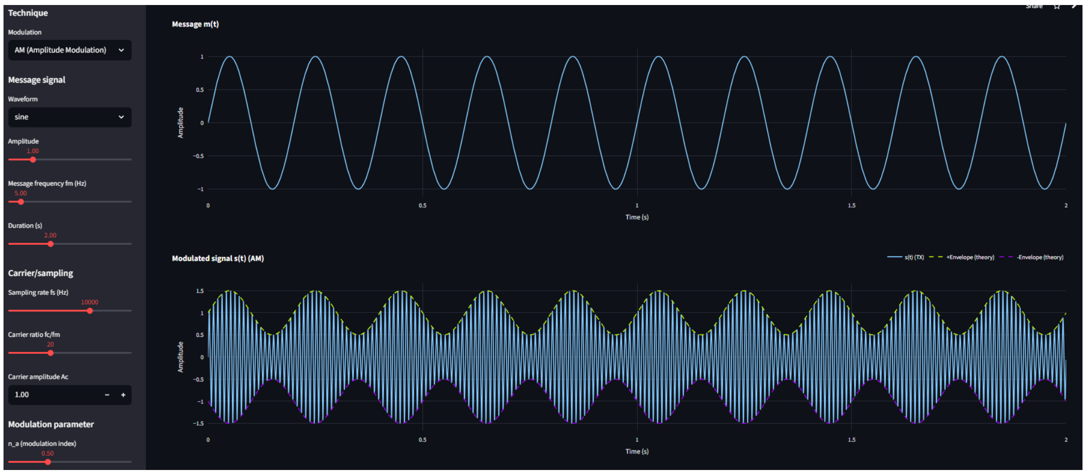
Duration (s)

(c) Reconstructed signal at the receiver.

Figure 7: Delta Modulation (DM) simulation results. [1]

3.1.4 Analog-to-Analog Outputs

These figures compare Amplitude Modulation (AM) with Angle Modulation (FM/PM). Note that the envelope of the AM signal (Figure 8) varies with the message, while the FM and PM signals (Figures 9, 10) maintain a constant amplitude. **The theoretical implications of these differences regarding power efficiency and bandwidth are discussed in Section 4.1.4.**


Technique

Modulation: AM (Amplitude Modulation)

Message signal

Waveform: sine

Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

Carrier/sampling

Sampling rate fs (Hz): 10000

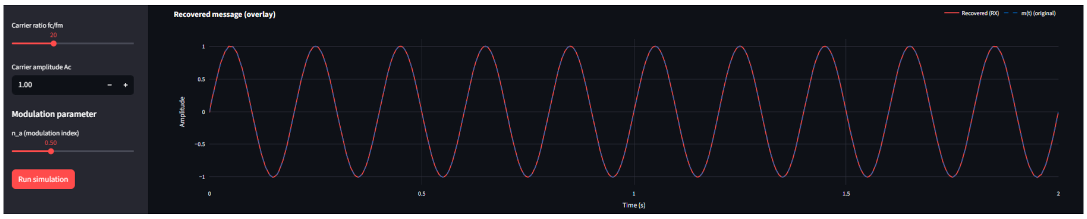
Carrier ratio fc/fm: 20

Carrier amplitude Ac: 1.00

Modulation parameter

n_a (modulation index): 0.50

(a) Transmitted AM signal.


Carrier ratio fc/fm: 20

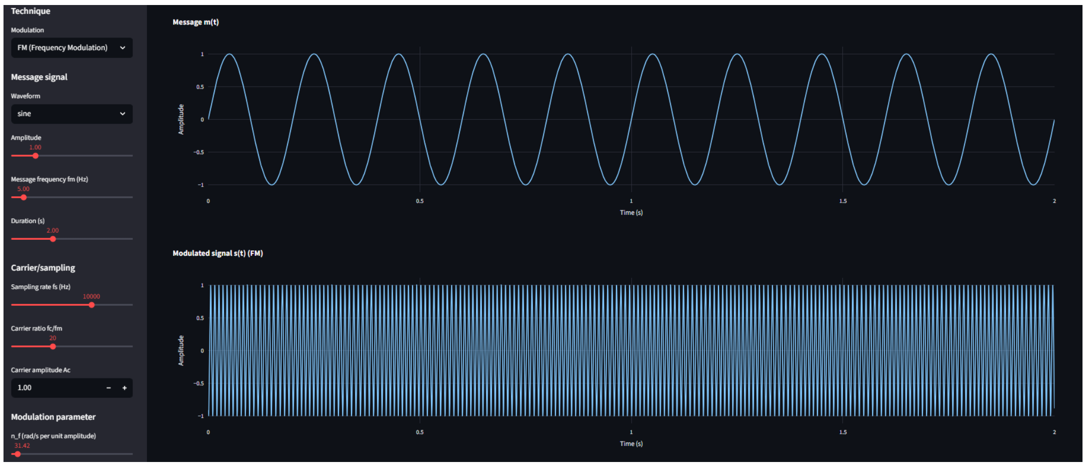
Carrier amplitude Ac: 1.00

Modulation parameter

n_a (modulation index): 0.50

(b) Recovered message signal.

Figure 8: Amplitude Modulation (AM) results using a sine message.


Technique

Modulation: FM (Frequency Modulation)

Message signal

Waveform: sine

Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

Carrier/sampling

Sampling rate fs (Hz): 10000

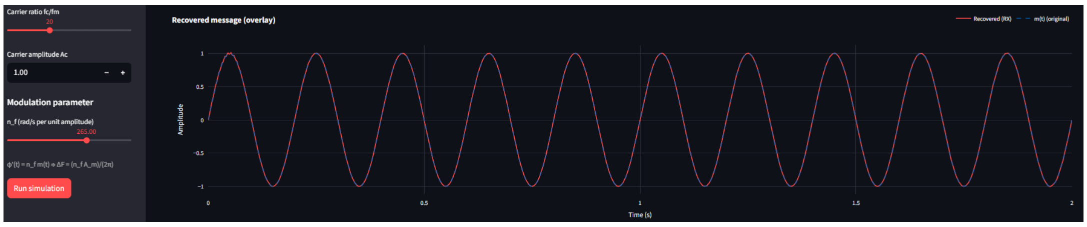
Carrier ratio fc/fm: 20

Carrier amplitude Ac: 1.00

Modulation parameter

n_f (rad/s per unit amplitude): 31.42

(a) Transmitted FM signal.


Carrier ratio fc/fm: 20

Carrier amplitude Ac: 1.00

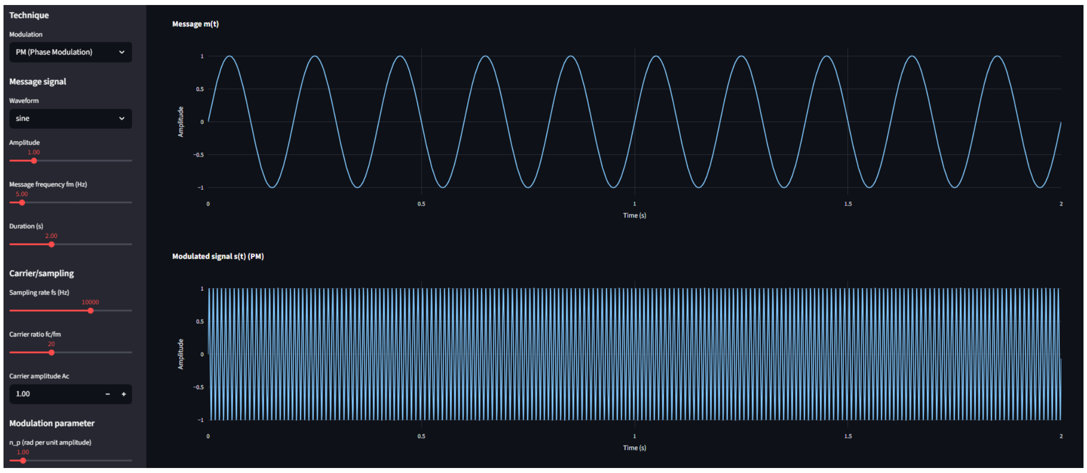
Modulation parameter

n_f (rad/s per unit amplitude): 265.00

φ'(t) = n_f m(t) ⇒ ΔF = (n_f A_m)/(2π)

(b) Recovered message signal.

Figure 9: Frequency Modulation (FM) results using a sine message. [1]


Technique

Modulation: PM (Phase Modulation)

Message signal

Waveform: sine

Amplitude: 1.00

Message frequency fm (Hz): 5.00

Duration (s): 2.00

Carrier/sampling

Sampling rate fs (Hz): 10000

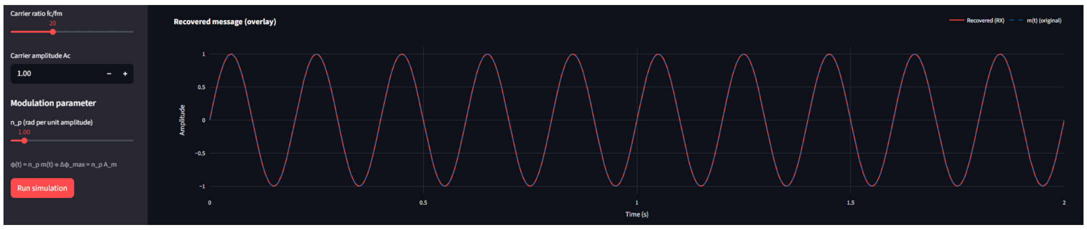
Carrier ratio fc/fm: 20

Carrier amplitude Ac: 1.00

Modulation parameter

n_p (rad per unit amplitude): 1.00

(a) Transmitted PM signal.


Carrier ratio fc/fm: 20

Carrier amplitude Ac: 1.00

Modulation parameter

n_p (rad per unit amplitude): 1.00

ϕ(t) = n_p m(t) ⇒ Δϕ_max = n_p A_m

(b) Recovered message signal.

Figure 10: Phase Modulation (PM) results using a sine message. [1]

3.2 Performance Measurements

The results of the automated regression tests are summarized below. Table 2 details the raw runtime consistency across five distinct executions, while Table 3 aggregates these results to calculate the relative speedup of the AI-optimized code. **A detailed analysis of why specific modes (like D2D) achieved higher speedups than others (like D2A) is provided in Section 4.2.**

Test file	Target	Run 1	Run 2	Run 3	Run 4	Run 5	μ	δ
test_d2d.py (1398 tests)	Original	13.51s	12.22s	12.73s	12.19s	13.12s	12.754s	0.572s
	GPT_opt	4.83s	4.08s	4.19s	3.90s	3.93s	4.186s	0.379s
	Gemini_opt	4.76s	4.55s	4.62s	5.36s	4.45s	4.748s	0.360s
test_d2a.py (854 tests)	Original	17.11s	18.70s	18.34s	18.25s	18.64s	18.208s	0.643s
	GPT_opt	17.85s	17.56s	17.67s	17.12s	16.98s	17.436s	0.371s
	Gemini_opt	16.94s	16.03s	15.87s	16.64s	15.92s	16.280s	0.481s
test_a2d.py (250 tests)	Original	3.87s	3.79s	4.33s	4.20s	4.24s	4.086s	0.240s
	GPT_opt	2.22s	2.34s	2.72s	2.43s	2.35s	2.412s	0.188s
	Gemini_opt	3.61s	3.65s	3.94s	4.00s	3.96s	3.832s	0.186s
test_a2a.py (68 tests)	Original	1.31s	1.44s	1.52s	1.78s	1.65s	1.540s	0.182s
	GPT_opt	1.22s	1.19s	1.29s	1.33s	1.30s	1.266s	0.059s
	Gemini_opt	1.21s	1.25s	1.24s	1.29s	1.19s	1.236s	0.038s

Table 2: Runtime measurements for test suites across 5 executions.

Test file	Target	Avg. (s)	Std. dev. (s)	Speedup	Notes
test_d2d.py	Original	12.754	0.572	1.00×	Baseline runtime.
	GPT_opt	4.186	0.379	3.05×	Optimized vectorization.
	Gemini_opt	4.748	0.360	2.69×	Logic improvements.
test_d2a.py	Original	18.208	0.643	1.00×	Baseline runtime.
	GPT_opt	17.436	0.371	1.04×	Minor gains (compute-bound).
	Gemini_opt	16.280	0.481	1.12×	Optimized modulation loops.
test_a2d.py	Original	4.086	0.240	1.00×	Baseline runtime.
	GPT_opt	2.412	0.188	1.69×	Optimized PCM pipeline.
	Gemini_opt	3.832	0.186	1.07×	Conservative refactoring.
test_a2a.py	Original	1.540	0.182	1.00×	Baseline runtime.
	GPT_opt	1.266	0.059	1.22×	Improved unwrapping logic.
	Gemini_opt	1.236	0.038	1.25×	Optimized demodulation.

Table 3: Summary of average runtime and speedup improvements.

4 Discussion

This section interprets the simulation results presented in Section 3, analyzing the behavior of the implemented transmission techniques and evaluating the impact of AI-based optimizations on software performance.

4.1 Analysis of Transmission Techniques

4.1.1 Digital-to-Digital: Synchronization and Spectral Efficiency

The waveform results (Fig. 3) clearly illustrate the fundamental trade-offs in line coding. While **NRZ-L** provides the simplest representation, it lacks inherent clock synchronization information during long sequences of unchanging bits. This limitation is addressed by **Manchester** encoding, which forces a transition in the middle of every bit period. However, our observations confirm that this advantage comes at the cost of doubling the signal bandwidth, as the modulation rate becomes twice the data rate. [2]

The effectiveness of scrambling is demonstrated in Fig. 4. In the standard **AMI** scheme, a long string of zeros results in a flat signal, potentially causing loss of synchronization at the receiver. The **B8ZS** and **HDB3** simulations show how deliberate bipolar violations are inserted to maintain signal activity. The perfect recovery of the original bitstream confirms that the receiver correctly identifies and removes these violation patterns, trading a slight increase in decoder logic complexity for robust synchronization without increasing the transmission bandwidth.

4.1.2 Digital-to-Analog: Bandwidth vs. Complexity

The comparison between binary (ASK, BFSK, BPSK) and multilevel (QPSK, QAM) modulation in Fig. 5 highlights the impact of symbol mapping on spectral efficiency. Binary schemes transmit one bit per symbol, requiring a bandwidth proportional to the bit rate. In contrast, **16-QAM** groups four bits into a single symbol by varying both amplitude and phase. This allows the system to transmit data at higher rates within the same channel bandwidth, but it requires a more complex detection mechanism and is theoretically more susceptible to noise due to the reduced Euclidean distance between constellation points. [2]

4.1.3 Analog-to-Digital: Quantization and Tracking Artifacts

The digitization results reveal distinct reconstruction artifacts inherent to each technique.

- **PCM (Fig. 6):** The "staircase" effect is a direct visual representation of quantization noise. The fidelity of the reconstructed signal depends entirely on the number of levels L; fewer levels result in larger steps and higher distortion.

- **Delta Modulation (Fig. 7):** The results expose the "slope overload" phenomenon. When the input signal changes rapidly (high frequency or amplitude), the fixed step size Δ fails to keep up, causing a lag in the reconstructed waveform. Conversely, in flat regions, the signal oscillates around the true value ("granular noise"). This validates the theoretical trade-off: increasing Δ reduces slope overload but increases granular noise. [2]

4.1.4 Analog-to-Analog: Bandwidth Trade-offs

The spectral differences between AM and Angle Modulation are inferred from the time-domain behavior. **AM** (Fig. 8) modifies the envelope, preserving the message bandwidth linearly:

$$B_{\text{AM}} \approx 2B_m$$

In contrast, **FM** and **PM** (Fig. 9, 10) encode information in the zero-crossings and phase variations. While this consumes significantly more bandwidth, it follows Carson's rule:

$$B_{\text{FM}} \approx 2\,(\Delta f + B_m)$$

where Δf is the peak frequency deviation [2]. This bandwidth expansion offers superior immunity to amplitude noise—a crucial advantage for high-fidelity transmission which is visually represented by the constant envelope of the FM/PM signals.

4.2 Performance Analysis: Original vs. AI-Optimized

The regression testing results (Table 3) provide a quantitative basis for evaluating the AI-based refactoring. [1] The observed speedups vary significantly across transmission modes, driven by the specific nature of the computational bottlenecks (CPU-bound arithmetic vs. Python interpreter overhead) in each module.

4.2.1 Digital-to-Digital (D2D) Optimization

This mode yielded the highest performance gains, with speedups of **3.05**$\times$ (GPT) and **2.69**$\times$ (Gemini).

- **Original Implementation:** The baseline `d2d.py` relied heavily on Python list comprehensions and iterative concatenation. For example, `_nrzl_levels` iterated over the input bit list one by one: `[ -1 if b == 1 else +1 for b in bits]`. Similarly, Manchester encoding constructed small arrays for every bit and concatenated them at the end, incurring massive memory allocation overhead.
- **AI Optimizations:** Both AI models replaced these loops with vectorized **NumPy** operations.
  - **GPT_optimized:** Replaced the encoding loop with a single vector operation: `(1 - 2 * bits).astype(np.int8)`. It also pre-allocated the output array for scrambling, avoiding dynamic list growth.

  - **Gemini_optimized:** Utilized `np.where` and `np.repeat` to generate waveforms instantly without bit-wise iteration.

- **Impact:** Since D2D logic is simple (conditional assignments), the Python interpreter overhead was the dominant bottleneck. Vectorization removed this overhead entirely.

4.2.2 Digital-to-Analog (D2A) Optimization

Improvements in this mode were marginal ($\approx 1.04\times$ to $1.12\times$).

- **Original Implementation:** The `d2a.py` module used a loop over bits, but the inner operation involved generating cosine segments: `np.cos(2*pi*f*t)`.

- **AI Optimizations:** Both models successfully vectorized the modulation. Instead of looping, they generated a global frequency/amplitude array using `np.repeat` and applied `np.cos` once over the entire time axis.

- **Impact:** Despite vectorization, the runtime is dominated by the transcendental trigonometric calculations (cos, sin) inside NumPy's C-backend. Since the original loop already offloaded the heavy math to NumPy, removing the loop structure provided only diminishing returns.

4.2.3 Analog-to-Digital (A2D) Optimization

This mode showed a clear divergence between the two AI models (GPT: $1.69\times$ vs. Gemini: $1.07\times$).

- **Original Implementation:** The PCM encoding involved converting integers to binary strings using `format()`, then iterating over characters to build the bit list. Delta Modulation (DM) used a mandatory feedback loop using NumPy scalar indexing.

- **AI Optimizations:**

  - **GPT_optimized:** Achieved significant speedup by replacing string formatting with bitwise shifting operations `(idx >> shifts) & 1` to extract bits directly. Crucially, it optimized the DM loop by converting the NumPy array to a native Python list (`x.tolist()`) before iterating, as Python list indexing is faster than NumPy scalar access for sequential logic.
  - **Gemini_optimized:** Also vectorized the PCM bit extraction using broadcasting. However, in the DM loop, it retained NumPy array indexing inside the loop, which carries a higher per-element overhead, resulting in a lower speedup compared to GPT.

4.2.4 Analog-to-Analog (A2A) Optimization

Both models achieved consistent speedups of $\approx 1.25\times$.

- **Original Implementation:** The demodulation relied on `scipy.signal.hilbert` to compute the analytic signal.
- **AI Optimizations:** Both models introduced the `next_fast_len` optimization. The Fast Fourier Transform (FFT) used internally by the Hilbert transform is sensitive to input length; padding the signal length to the next power of 2 (or composite number) significantly reduces computation time.

4.3 Limitations and Constraints

The current implementation assumes an **ideal, noiseless channel**. While this clarifies the theoretical behavior of modulation schemes, it hides the practical robustness differences (e.g., QPSK vs. BPSK in noise). Additionally, the use of **Streamlit** introduces a frontend latency overhead that is separate from the signal processing time. While the backend logic is optimized, rendering large Plotly charts remains a memory-intensive operation that limits the maximum feasible simulation duration (N_s) for interactive use.

5 Conclusion

This project successfully implemented an interactive transmission simulator that models the four fundamental communication modes: digital-to-digital, digital-to-analog, analog-to-digital, and analog-to-analog. By integrating a modular Python backend with a Streamlit interface, the system provides a visual and practical environment for exploring signal processing concepts, directly aligning with the theoretical foundations of the *Principles of Computer Communications* course. [2]

The development process demonstrated that separating simulation logic from visualization allows for a robust and maintainable architecture. The comprehensive testing framework verified functional correctness across all techniques and facilitated a systematic performance comparison between the original and AI-optimized implementations. The benchmarking results conclusively showed that AI-assisted refactoring significantly improved execution speed in loop-dominated logic (such as line coding) but offered diminishing returns in heavily vectorized mathematical operations.

While the simulator effectively demonstrates modulation principles in an ideal environment, it currently relies on a noiseless channel model. Future extensions of this work should include **Gaussian noise modeling** to demonstrate the impact of signal-to-noise ratio (SNR) on error rates, as well as **error control coding** (e.g., CRC, Hamming) to visualize error detection and correction. Additionally, integrating support for **real-time audio input** would further bridge the gap between theoretical simulation and real-world application.

References

[1] Signal Transmission Simulator Project. Signal transmission simulator. https://signal-transmission-simulator.streamlit.app, 2026. Interactive web-based transmission simulator developed as part of the BLG 337E course project.

[2] William Stallings. *Data and Computer Communications*. Pearson, 10 edition, 2013.