University of Dhaka

Department of Computer Science and Engineering

CSE 2213 – Data and Telecommunication Laboratory   Credits: 0.75

Batch: 29/2nd Year 2nd Sem 2025

Instructors: Prof. Dr. Md. Mustafizur Rahman (MMR), Mr. Palash Roy (PR)

<u>Lab Experiment # 2</u>

Name of the Experiment:

Implementing encoding and decoding scheme using Alternate Mark Inversion (AMI), Pseudoternary and Multi-Level Line

Objectives

- Understand the working principles of AMI, Pseudoternary, and Multi-Level Line encoding and decoding.
- Implement the encoding and decoding processes.
- Observe polarity alternation and reconstruct the original binary message.
- Analyze the benefits and drawbacks of multi-level schemes over binary encoding.

AMI Encoding

AMI (Alternate Mark Inversion) is a synchronous clock encoding technique that uses bipolar pulses to represent a logical 1. The next logic 1 is represented by a pulse of the opposite polarity. Hence, a sequence of logical 1s is represented by a sequence of pulses of alternating polarity. That means----

- Binary 0 → 0V
- Binary 1 → Alternates between +V and -V
- Used to maintain synchronization and eliminate DC bias.

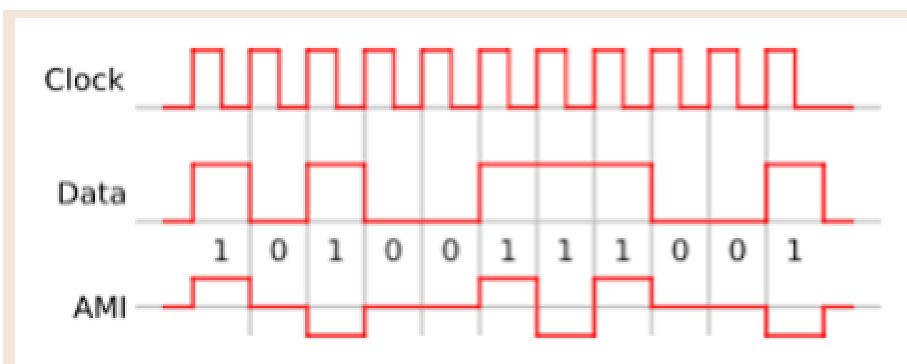

Figure 1: An example of AMI line coding

Example of AMI encoding

A logical 1 value is represented by a high or low level, and a zero by no signal. The logical 1 by pulses use alternating polarity.

Input Binary String: `"10110010"`

AMI Encoding

- 1 → +1 → `+1`
- 0 → 0 → `0`
- 1 → -1 → `-1`
- 1 → +1 → `+1`
- 0 → 0 → `0`
- 0 → 0 → `0`
- 1 → -1 → `-1`
- 0 → 0 → `0`

Algorithm

AMI Encoding Algorithm

Input: Binary string (e.g., "10110010")
Output: Encoded list of voltage levels

1. Initialize `last = -1` (this tracks polarity of last '1')
2. For each bit in the binary string:
   - If bit is '0', append `0` (neutral)
   - If bit is '1':
     - Flip `last` to its opposite sign
     - Append `last`

AMI Decoding Algorithm

Input: Encoded voltage list
Output: Binary string

1. For each voltage in the list:
   - If value is `0`, append `'0'`
   - If value is `+V` or `-V`, append `'1'`

Pseudoternary encoding and decoding.

Pseudoternary line coding is a line coding technique that uses three voltage levels (+V, 0, -V) to represent binary data, where logic 0 is represented by alternating positive and negative voltages, and logic 1 is represented by zero volts. That means----

- Binary 1 → 0V
- Binary 0 → Alternates between +V and -V
- Opposite of AMI. Same benefits: no DC component and help synchronization.

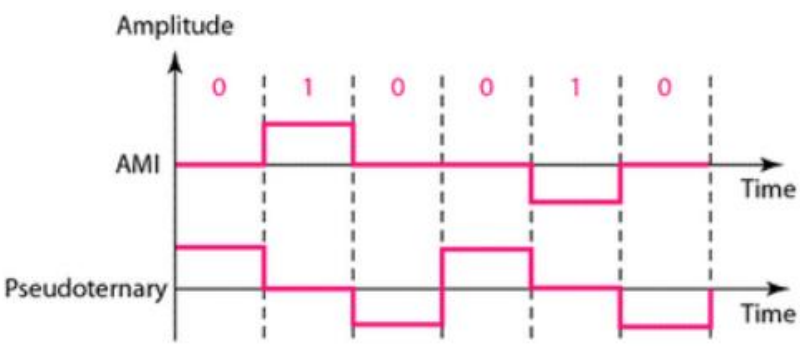

Figure 2: An example of AMI and Pseudoternary line coding

Example of Pseudoternary Encoding

A logical 0 value is represented by a high or low level, and a 1 by no signal. The logical 0 by pulses use alternating polarity.

Pseudoternary Encoding

- 1 → 0 → `0`
- 0 → -1 → `-1`
- 1 → 0 → `0`
- 1 → 0 → `0`
- 0 → +1 → `+1`
- 0 → -1 → `-1`
- 1 → 0 → `0`
- 0 → +1 → `+1`

Encoded Pseudoternary Signal: `[0, -1, 0, 0, +1, -1, 0, +1]`

Pseudoternary Encoding Algorithm

Input: Binary string
Output: Encoded voltage list

1. Initialize last = -1 (polarity tracker)
2. For each bit:
    - If bit is '1', append 0

- If bit is '0':
  - Flip polarity (last *= -1)
  - Append last

Pseudoternary Decoding Algorithm

Input: Encoded voltage list
Output: Binary string

1. For each voltage in the list:
   - If value is 0, append '1'
   - If value is +V or -V, append '0'

Multi-Level Line Encoding

- Multi-level encoding uses more than two voltage levels to encode binary data.
- This helps reduce the frequency spectrum and makes signals more bandwidth-efficient.
- Common examples: **MLT-3, 2B1Q, 4B3T**, etc.

MLT-3 Encoding (Multi-Level Transmit - 3 levels)

- Voltage levels: +1, 0, -1
- Rules:
  - **If the input bit is 0**: no change in signal level.
  - **If the input bit is 1**: transition to the **next** level in the sequence:
    → 0 → +1 → 0 → -1 → 0 → +1 and so on.

This pattern repeats cyclically:
... → 0 → +1 → 0 → -1 → 0 → +1 → ...

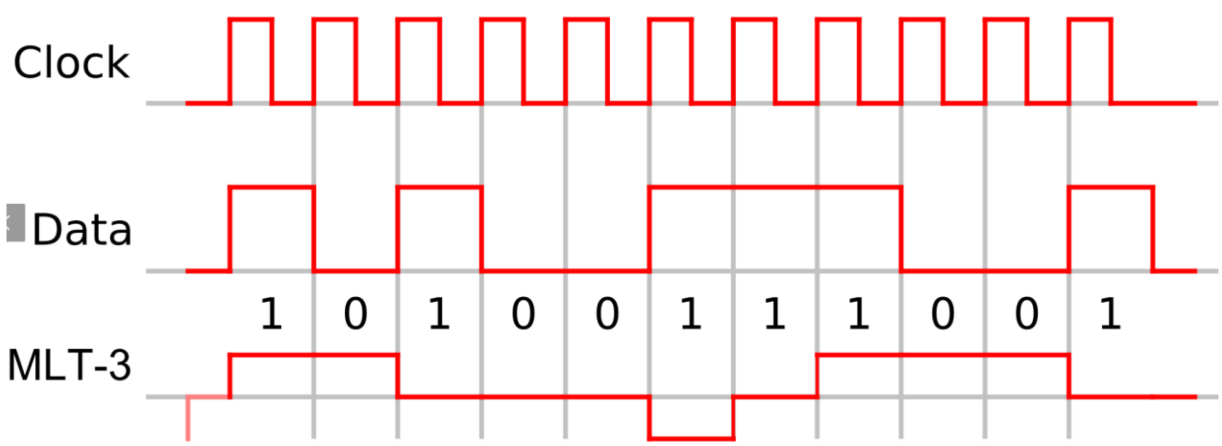

Figure 3: An example of MLT-3 line coding

Example of MLT-3 Encoding

Input Binary String: `"1100101"`

Encoding Process (MLT-3)

Bit	Signal Level	Reason
1	`+1`	Transition from `0` → `+1`
1	`0`	Transition from `+1` → `0`
0	`0`	No transition
0	`0`	No transition
1	`-1`	Transition from `0` → `-1`
0	`-1`	No transition
1	`0`	Transition from `-1` → `0`

Encoded MLT-3 Signal: `[+1, 0, 0, 0, -1, -1, 0]`

MLT-3 Encoding Algorithm

Input: Binary string (e.g., "1100101")
Output: Voltage level sequence

1. Set initial level to 0.
2. Define the transition sequence: [0, +1, 0, -1].
3. Initialize an index to track the current position in the transition cycle.
4. For each bit:
   - If bit is '0': repeat last level.
   - If bit is '1': move to the **next** level in the transition sequence.

Decoding

Input: Encoded signal (list of voltage levels)
Output: Binary string

1. Compare each signal level with the previous one.
   - If it **changes**: append '1'
   - If it **remains the same**: append '0'

Experimental Procedure:

1. Write a program in Python or MATLAB to implement Manchester encoding.
2. Input a sample binary sequence (e.g., ‘1011001’).
3. Compute the Manchester encoded signal.
4. Display and plot the encoded waveform ((Simulation software (e.g., MATLAB, Python with Matplotlib/Numpy libraries).
5. Implement the Manchester decoding algorithm.
6. Decode the encoded signal back to the original binary sequence.
7. Verify the correctness by comparing the decoded sequence with the original sequence.
8. Observe the effect of encoding and decoding and document the results.