**Title**

Study of Digital-to-Digital Conversion (Line Coding) Using MATLAB

**Abstract**

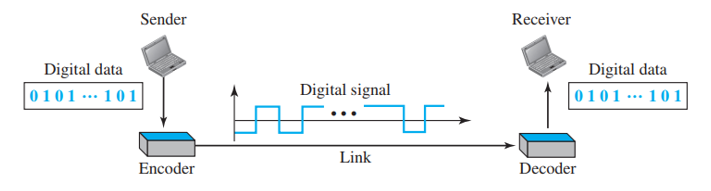
This experiment is designed to-

1.To understand the use of MATLAB for solving communication engineering problems.

2.To develop understanding of Digital to Digital Conversion (Line Coding) using MATLAB.

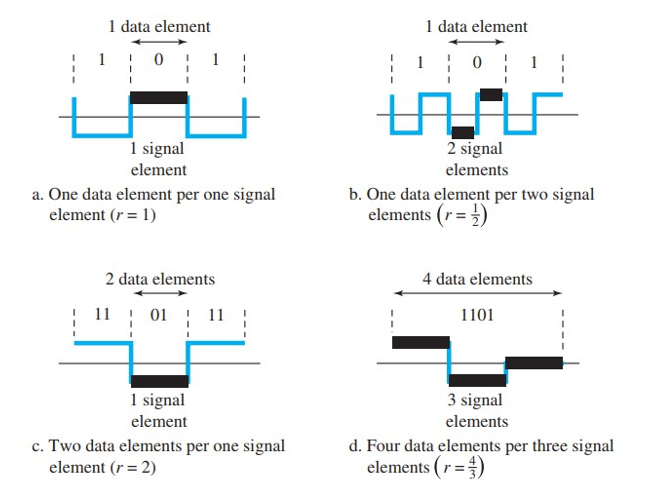
**Introduction**

**Line Coding:** The conversion of digital data into digital signals is known as line coding. We presume that information is stored in computer memory as a series of bits, whether it be text, numbers, graphical pictures, audio, or video. A digital signal is created from a series of bits via line coding. Digital data is encoded into a digital signal at the transmitter, and the digital signal is decoded at the receiver to reproduce the digital data. The procedure is depicted in Figure 1.



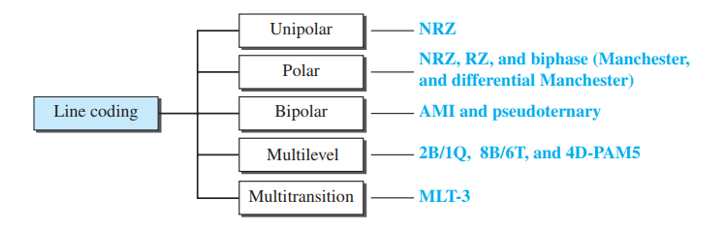
**Figure 1: Line Coding and Decoding**

**Signal and Data Elements:** Let's examine the differences between signal and data elements. Sending data components is the aim of data communications. The smallest thing that can represent a bit of information is called a data element. Data elements are carried by signal elements in digital data transfers. The smallest unit of a digital signal in terms of time is called a signal element. To put it another way, signal elements are what we may communicate, and data elements are what we must send.

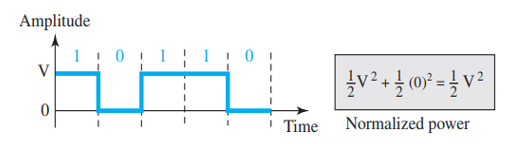


**Figure 2: Signal vs Data elements**

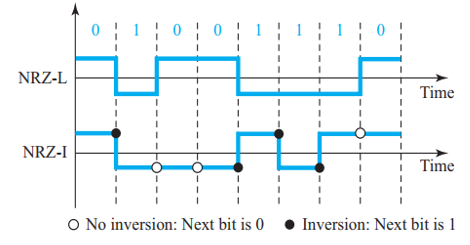
**Bandwidth:** A digital signal carrying information is known to be nonperiodic. We also know that a nonperiodic signal has an unlimited range and a constant bandwidth. The majority of digital signals that we come across in real life, however, have a limited bandwidth. To put it another way, even if the bandwidth is theoretically unlimited, many of the components are so microscopic that they may be disregarded. It has a limited effective bandwidth. Going forward, it is important to keep in mind that this effective bandwidth is being discussed whenever the bandwidth of a digital transmission is discussed.

**Different Line Coding Schemes:-**

**Figure 3 Line coding schemes**

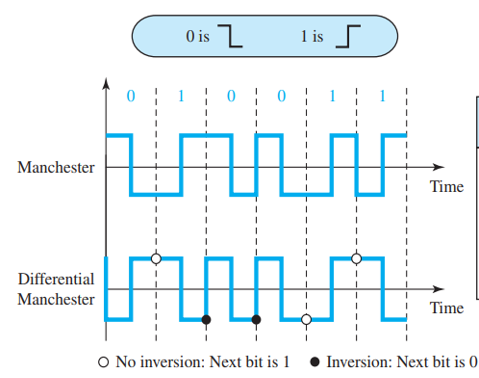
**Unipolar: -**

**Figure 4: Unipolar NRZ scheme**

**Polar: -**

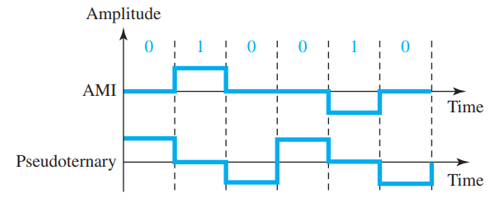
**Figure 5: Polar NRZ-L and NRZ-I schemes**

**Polar Biphase:-**

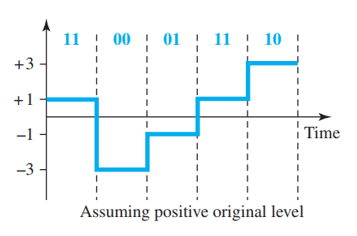
****

**Figure 6: Manchester and differential Manchester schemes**

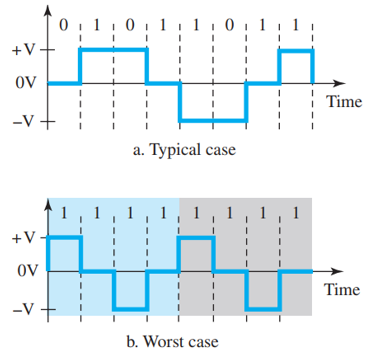
**Bipolar: -**

****

**Figure 7: AMI and pseudoternary**

**Multilevel: -** To enhance data rates or reduce bandwidth usage, various encoding schemes have been developed. The main idea is to increase the number of bits transmitted per baud by mapping a sequence of **m** data elements into **n** signal elements. Since data elements consist of only **0**s and **1**s, there are **2m** possible data combinations. By utilizing multiple signal levels (**L**), we can generate **Ln** different signal patterns. When **2m = Ln** each data pattern directly corresponds to a unique signal pattern. If **2m < Ln**, only a selected subset of signal patterns is used, allowing for better synchronization, baseline wandering prevention, and error detection. However, if **2m > Ln** some data patterns cannot be encoded, making transmission infeasible.

**Figure 8: 2B1Q scheme**

**MLT-3:-**

**Figure 9: Multitransitional scheme**

**Results and Discussion**

**AB-CDEFG-H**

**ID: 23-50953-1**

**Bit Stream**

𝐸 = 9 = 1001

𝐹 = 5 = 0101

𝐺 = 3 = 0011

**(a)** Polar NRZ-L assuming bit rate is 4 kbps.

|  |
| --- |
| **Code:** |
| clc;  clear all;  close all;  % Input Parameters  bit\_stream = [1 0 0 1 0 1 0 1 0 0 1 1];  no\_bits = length(bit\_stream);  bit\_rate = 4000; % 4 kbps  pulse\_per\_bit = 1; % Polar NRZ uses 1 pulse per bit  pulse\_duration = 1 / (pulse\_per\_bit \* bit\_rate);  no\_pulses = no\_bits \* pulse\_per\_bit;  samples\_per\_pulse = 500;  fs = samples\_per\_pulse / pulse\_duration; % Sampling frequency  t = 0:1/fs:(no\_pulses \* pulse\_duration); % Time vector  no\_samples = length(t); % Total number of samples  % Voltage levels for Polar NRZ  max\_voltage = 5; % Logic '1' → +5 V  min\_voltage = -5; % Logic '0' → -5 V  % Digital Signal Construction  dig\_sig = zeros(1, no\_samples);  for i = 1:no\_bits  if bit\_stream(i) == 1  dig\_sig(((i-1) \* samples\_per\_pulse + 1):i \* samples\_per\_pulse) = max\_voltage;  else  dig\_sig(((i-1) \* samples\_per\_pulse + 1):i \* samples\_per\_pulse) = min\_voltage;  end  end  % Plot the Polar NRZ-L Signal  figure;  plot(t, dig\_sig, 'LineWidth', 1.5);  grid on;  xlabel('Time (seconds)');  ylabel('Voltage (V)');  ylim([min\_voltage - abs(max\_voltage)\*0.2, max\_voltage + abs(max\_voltage)\*0.2]);  title(['Polar NRZ-L for Bitstream: ', num2str(bit\_stream)]); |

|  |
| --- |
| **Result and Discussion:** |
| **Fig10: Polar NRZ-L Signal** |
|  In this scheme, a logical '1' is represented by a positive voltage and a logical '0' by a negative voltage.   The waveform clearly shows that there is no transition unless the bit changes.   It provides a simple and efficient way of line coding but suffers from DC component issues and baseline wandering. |

**(b)** Manchester assuming bit rate is 2 kbps.

|  |
| --- |
| **Code:** |
| clc;  clear all;  close all;  % Input Parameters  bit\_stream = [1 0 0 1 0 1 0 0 0 1 0 1];  no\_bits = length(bit\_stream);  bit\_rate = 2000; % 2 kbps  pulse\_per\_bit = 2; % For Manchester encoding (2 transitions per bit)  pulse\_duration = 1 / (pulse\_per\_bit \* bit\_rate);  no\_pulses = no\_bits \* pulse\_per\_bit;  samples\_per\_pulse = 500;  fs = samples\_per\_pulse / pulse\_duration; % Sampling frequency  t = 0:1/fs:(no\_pulses \* pulse\_duration); % Time vector  no\_samples = length(t);  % Voltage levels for Manchester encoding  max\_voltage = +2; % High level  min\_voltage = -2; % Low level  % Digital Signal Construction  dig\_sig = zeros(1, no\_samples);  for i = 1:no\_bits  j = (i-1) \* 2; % Two pulses per bit  if bit\_stream(i) == 1  % Logic '1': Low to High transition  dig\_sig((j \* samples\_per\_pulse + 1):(j + 1) \* samples\_per\_pulse) = min\_voltage;  dig\_sig(((j + 1) \* samples\_per\_pulse + 1):(j + 2) \* samples\_per\_pulse) = max\_voltage;  else  % Logic '0': High to Low transition  dig\_sig((j \* samples\_per\_pulse + 1):(j + 1) \* samples\_per\_pulse) = max\_voltage;  dig\_sig(((j + 1) \* samples\_per\_pulse + 1):(j + 2) \* samples\_per\_pulse) = min\_voltage;  end  end  % Plot the Manchester Signal  figure;  plot(t, dig\_sig, 'LineWidth', 1.5);  grid on;  xlabel('Time (seconds)');  ylabel('Voltage (V)');  ylim([min\_voltage - abs(max\_voltage) \* 0.2, max\_voltage + abs(max\_voltage) \* 0.2]);  title(['Manchester Encoding for Bitstream: ', num2str(bit\_stream)]); |
| **Result and Discussion:** |
| **Fig11: Manchester Encoded Signal** |
|  Each bit has a transition in the middle, making clock synchronization easier.   '0' is represented by a high-to-low transition, while '1' is represented by a low-to-high transition.   The self-clocking nature helps in reducing synchronization issues, but it requires twice the bandwidth compared to NRZ. |

**Conclusion**

In this experiment, we successfully implemented and analyzed various digital-to-digital line coding schemes using MATLAB. Each coding scheme has unique characteristics and applications; for example, Manchester encoding ensures synchronization through mid-bit transitions, while AMI eliminates the DC component by alternating the polarity of '1's. MLT-3 efficiently reduces bandwidth requirements by cycling through three voltage levels. These techniques are essential in digital communication systems, where factors such as bandwidth efficiency, synchronization, and baseline wandering need to be considered. Understanding these coding methods provides valuable insights into designing reliable and efficient data transmission systems.