American International University- Bangladesh
Department of Computer Engineering
COE 3201: Data Communication Laboratory

Title: Study of Digital to Digital Conversion (Line Coding) Using MATLAB

Abstract:

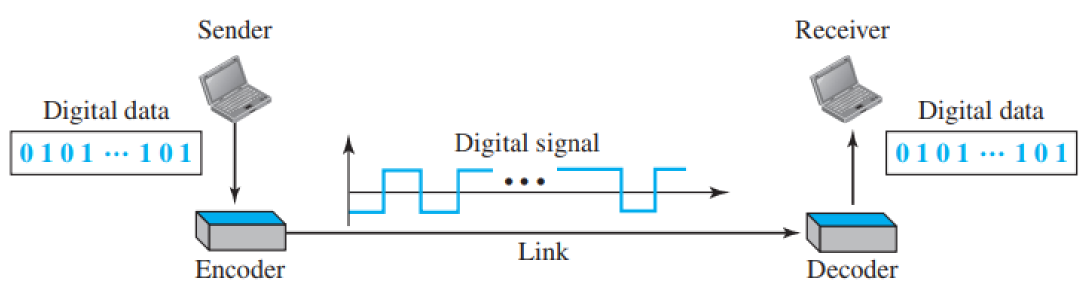
This experiment is designed to-

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:

I. Line Coding: Line coding is the process of converting digital data to digital signals. We assume that data, in the form of text, numbers, graphical images, audio, or video, are stored in computer memory as sequences of bits. Line coding converts a sequence of bits to a digital signal. At the sender, digital data are encoded into a digital signal; at the receiver, the digital data are recreated by decoding the digital signal. Figure 1 shows the process. [Data Communications and Networking, 5^{th} Edition, Behrouz A. Forouzan, Pages: 96 to 109]

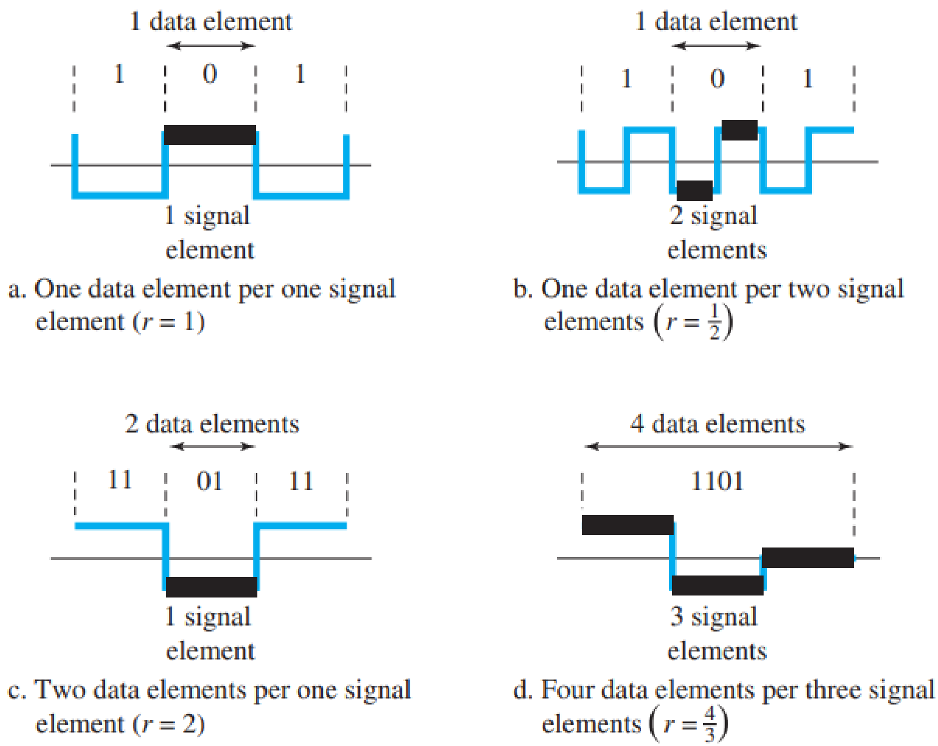
Figure 1 *Line coding and decoding*



II. Signal Elements and Data Elements:
Let us distinguish between a data element and a signal element. In data communications, our goal is to send data elements. A data element is the smallest entity that can represent a piece of information: this is the bit. In digital data communications, a signal element carries data elements. A signal element is the shortest unit (timewise) of a digital signal. In other words, data elements are what we need to send; signal elements are what we can send.

S = c * N * (1/r); [S = Signal Rate, c = case factor, N = Data Rate, r = (Number of Data Elements)/(Number of Signal Elements)]

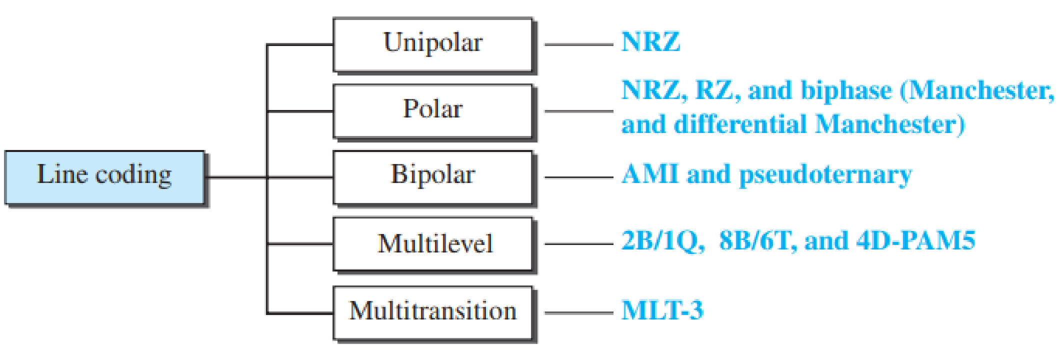
Figure 2 *Signal element versus data element*



III. Bandwidth: We know that a digital signal that carries information is nonperiodic. We also know that the bandwidth of a nonperiodic signal is continuous with an infinite range. However, most digital signals we encounter in real life have a bandwidth with finite values. In other words, the bandwidth is theoretically infinite, but many of the components have such a small amplitude that they can be ignored. The effective bandwidth is finite. From now on, when we talk about the bandwidth of a digital signal, we need to remember that we are talking about this effective bandwidth.

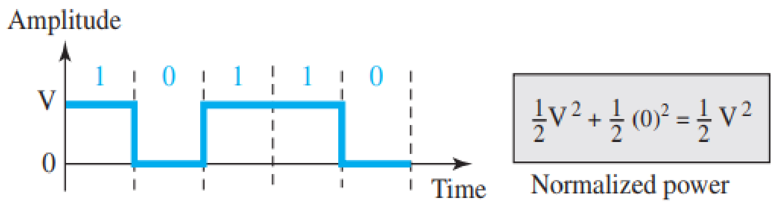
IV. Different Line Coding Schemes: Figure 3 shows different types of line coding schemes.

Figure 3 *Line coding schemes*



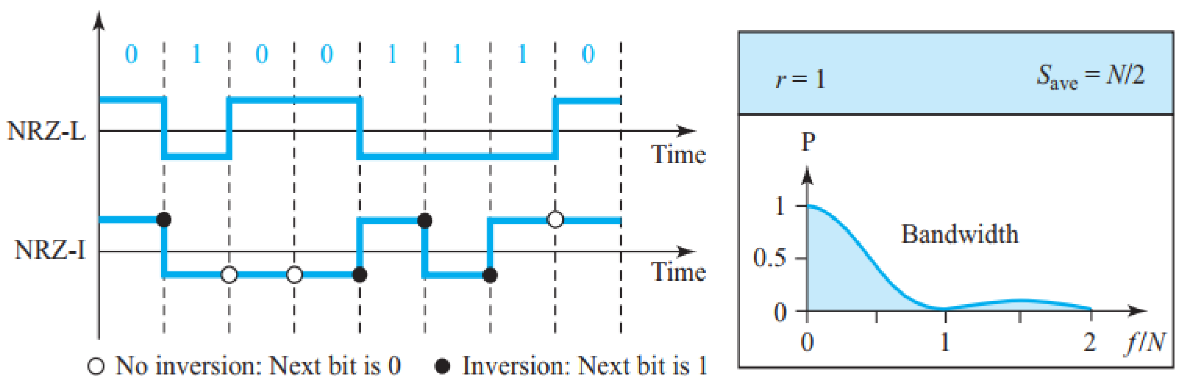
V. Unipolar:

Figure 4 *Unipolar NRZ scheme*



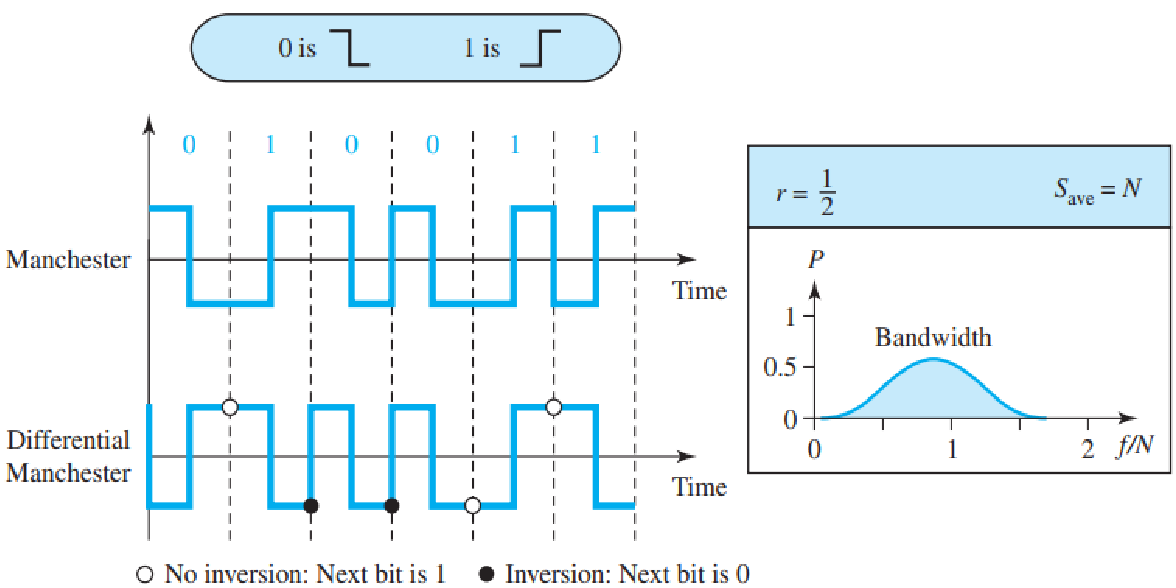
VI. Polar:

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



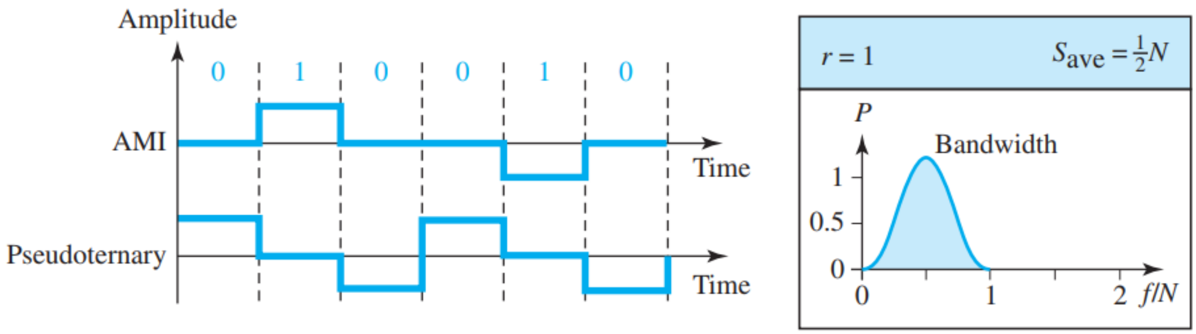
VII. Polar Biphase:

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



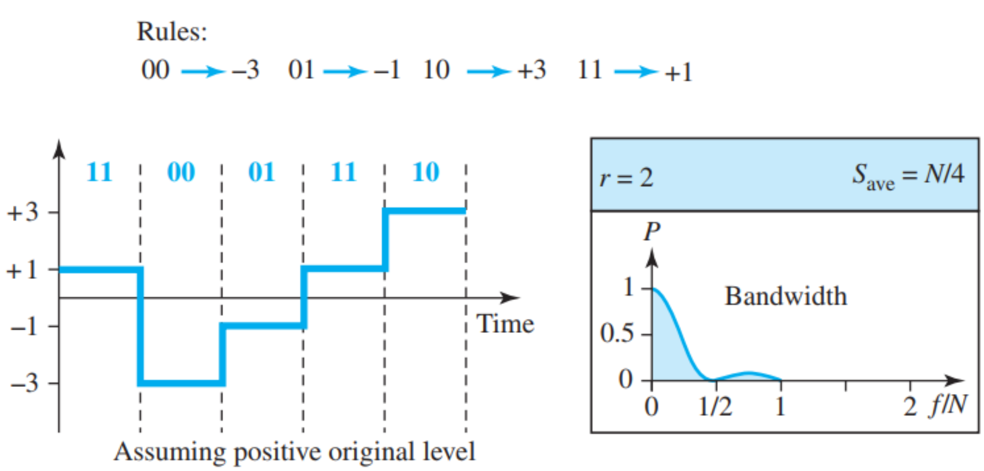
VIII. Bipolar:

Figure 7 *Bipolar schemes: AMI and pseudoternary*



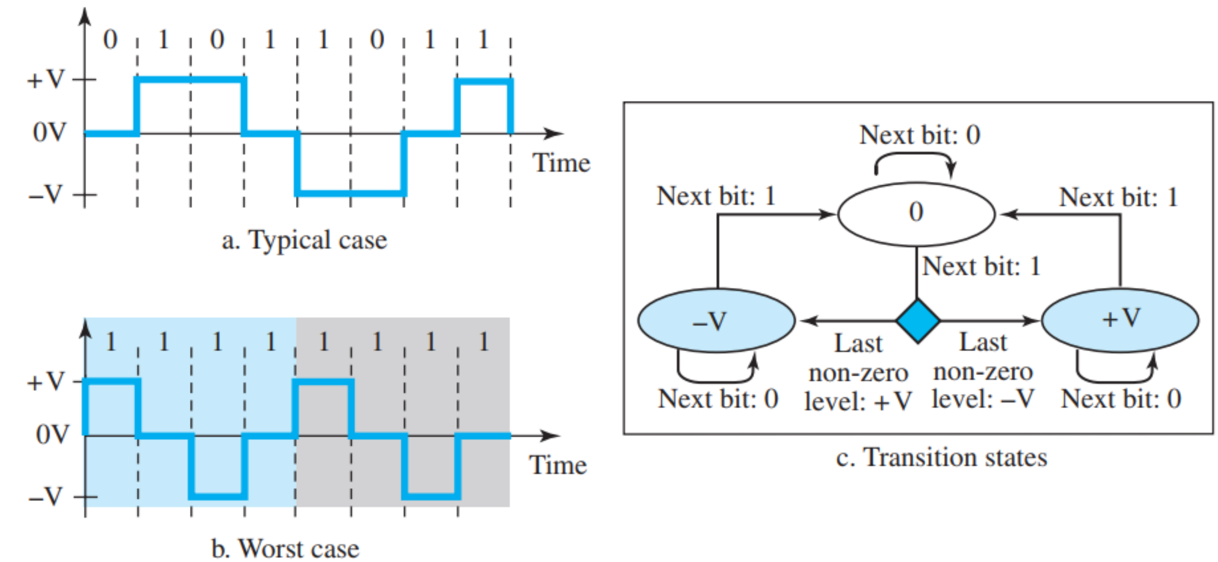
IX. Multilevel: The desire to increase the data rate or decrease the required bandwidth has resulted in the creation of many schemes. The goal is to increase the number of bits per baud by encoding a pattern of ***m*** data elements into a pattern of ***n*** signal elements. We only have two types of data elements (***0***s and ***1***s), which means that a group of ***m*** data elements can produce a combination of $\mathbf{2^m}$ data patterns. We can have different types of signal elements by allowing different signal levels. If we have ***L*** different levels, then we can produce $\mathbf{L^n}$ combinations of signal patterns. If $\mathbf{2^m = L^n}$, then each data pattern is encoded into one signal pattern. If $\mathbf{2^m < L^n}$, data patterns occupy only a subset of signal patterns. The subset can be carefully designed to prevent baseline wandering, to provide synchronization, and to detect errors that occurred during data transmission. Data encoding is not possible if $\mathbf{2^m > L^n}$ because some of the data patterns cannot be encoded.

Figure 8 *Multilevel: 2B1Q scheme*



X. MLT-3:

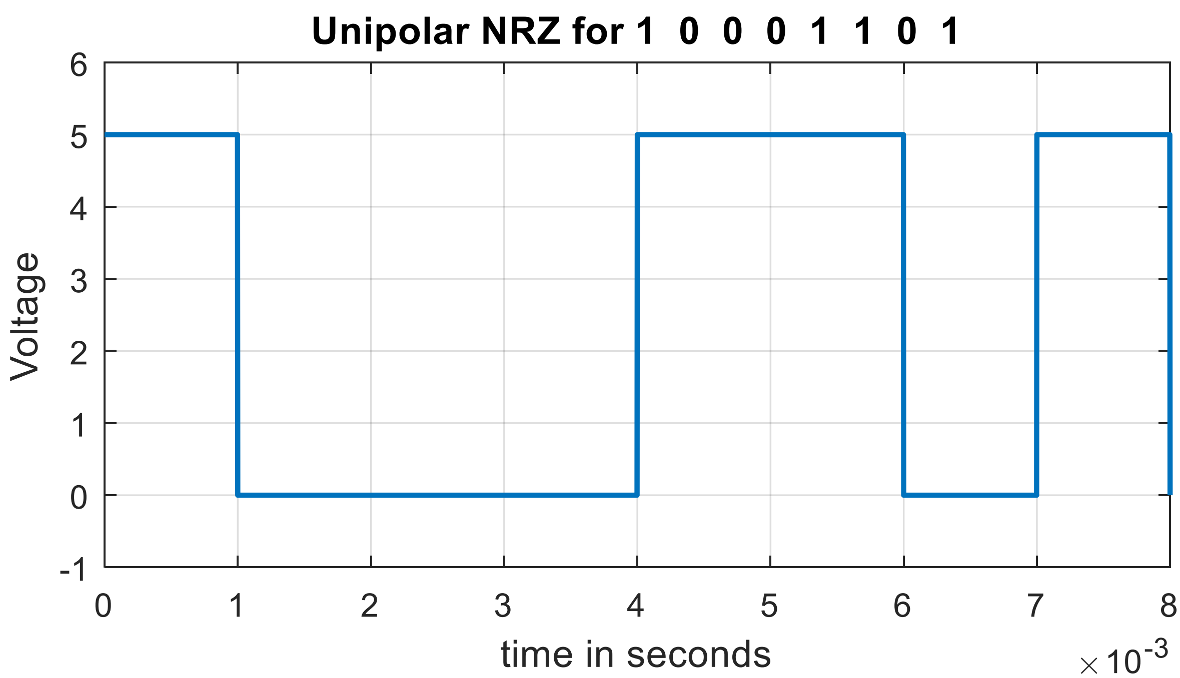
Figure 9 *Multitransition: MLT-3 scheme*



1. MATLAB code for Unipolar NRZ:

```
clc
clear all
close all
bit_stream = [1 0 0 0 1 1 0 1];
no_bits = length(bit_stream);
bit_rate = 1000; % 1 kbps
pulse_per_bit = 1; % for unipolar nrz
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
% including pulse duration in sampling frequency
% ensures having enough samples in each pulse
t = 0:1/fs:(no_pulses)*(pulse_duration); % sampling
interval
% total duration = (no_pulse)*(pulse_duration)
no_samples = length(t); % total number of samples
dig_sig = zeros(1,no_samples);
max_voltage = 5;
min_voltage = 0;
for i = 1:no_bits
    if bit_stream(i) == 1
```

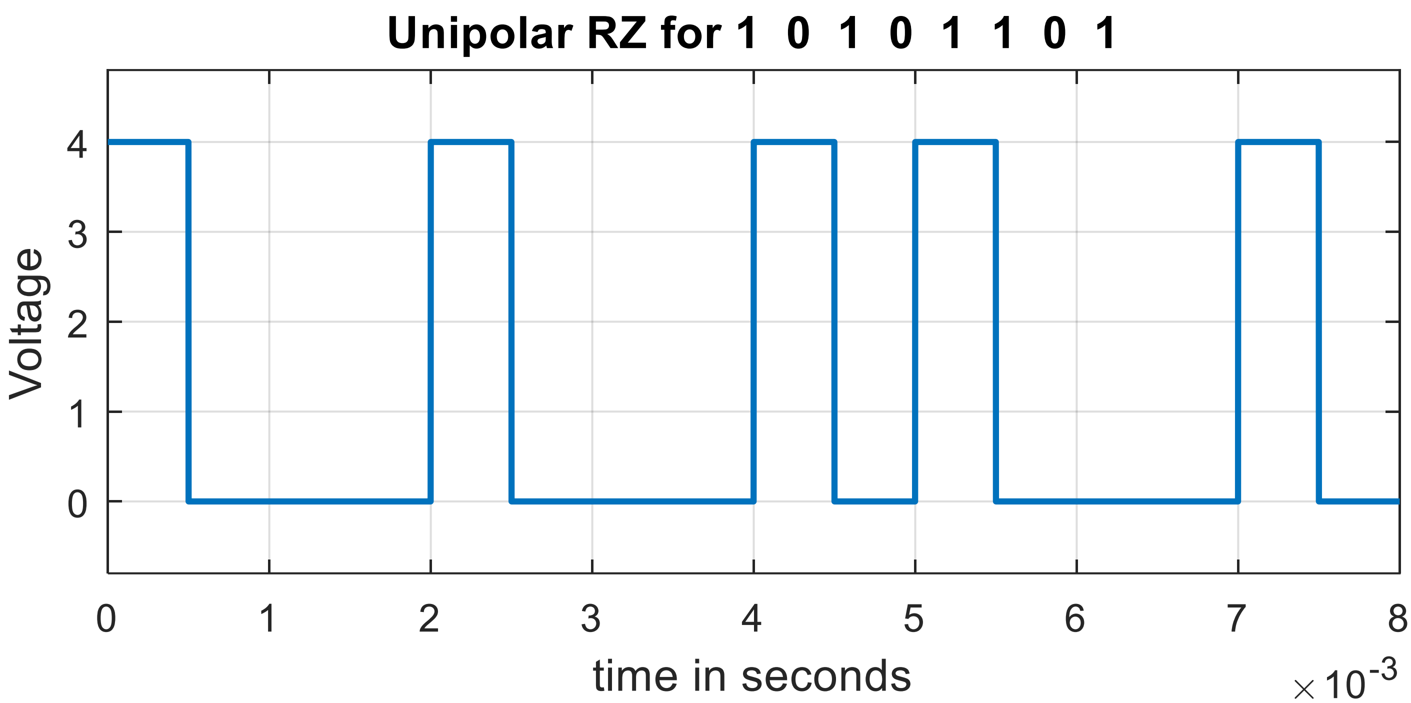
```
        dig_sig(((i-
1)*(samples_per_pulse)+1):i*(samples_per_pulse)) =
max_voltage*ones(1,samples_per_pulse);
    else
        dig_sig(((i-
1)*(samples_per_pulse)+1):i*(samples_per_pulse)) =
min_voltage*ones(1,samples_per_pulse);
    end
end
plot(t,dig_sig,'linewidth',1.5)
grid on
xlabel('time in seconds')
ylabel('Voltage')
ylim([(min_voltage - (max_voltage)*0.2)
(max_voltage+max_voltage*0.2)])
title(['Unipolar NRZ for ',num2str(bit_stream),''])
```



2. MATLAB code for Unipolar RZ:

```
clc
clear all
close all
bit_stream = [1 0 1 0 1 1 0 1];
no_bits = length(bit_stream);
bit_rate = 1000; % 1 kbps
pulse_per_bit = 2; % for unipolar rz
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
% including pulse duration in sampling frequency
% ensures having enough samples in each pulse
t = 0:1/fs:(no_pulses)*(pulse_duration); % sampling
interval
% total duration = (no_pulse)*(pulse_duration)
no_samples = length(t); % total number of samples
dig_sig = zeros(1,no_samples);
max_voltage = 4;
min_voltage = 0;
for i = 1:no_bits
    j = (i-1)*2;
    if bit_stream(i) == 1

dig_sig((j*(samples_per_pulse)+1):(j+1)*(samples_per_pu
lse)) = max_voltage*ones(1,samples_per_pulse);

dig_sig(((j+1)*(samples_per_pulse)+1):(j+2)*(samples_pe
r_pulse)) = zeros(1,samples_per_pulse);
    else

dig_sig((j*(samples_per_pulse)+1):(j+1)*(samples_per_pu
lse)) = min_voltage*ones(1,samples_per_pulse);

dig_sig(((j+1)*(samples_per_pulse)+1):(j+2)*(samples_pe
r_pulse)) = zeros(1,samples_per_pulse);
    end
end
plot(t,dig_sig,'linewidth',1.5)
grid on
xlabel('time in seconds')
```

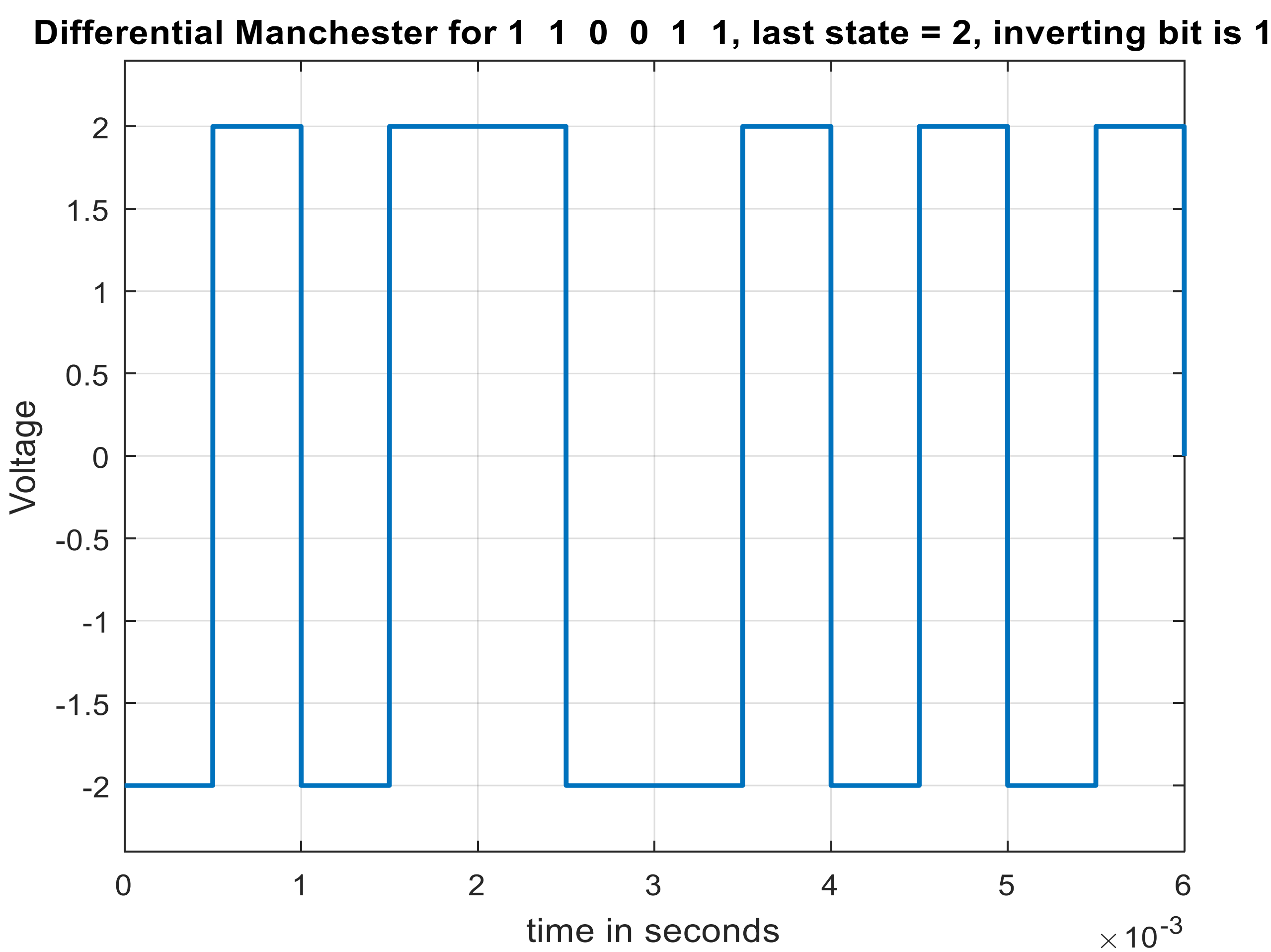
```
ylabel('Voltage')
ylim([(min_voltage - (max_voltage)*0.2)
(max_voltage+max_voltage*0.2)])
title(['Unipolar RZ for ',num2str(bit_stream),''])
```



3. MATLAB code for Differential Manchester:

```
Clc
clear all
close all
bit_stream = [1 1 0 0 1 1];
no_bits = length(bit_stream);
bit_rate = 1000; % 1 kbps
pulse_per_bit = 2; % for differential manchester
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
% including pulse duration in sampling frequency
% ensures having enough samples in each pulse
t = 0:1/fs:(no_pulses)*(pulse_duration); % sampling
interval
% total duration = (no_pulse)*(pulse_duration)
no_samples = length(t); % total number of samples
dig_sig = zeros(1,no_samples);
max_voltage = +2;
min_voltage = -2;
inv_bit = 1; % inverting bit
last_state = max_voltage;
inv_last_state = min_voltage; % inverse of last state
for i = 1:no_bits
    j = (i-1)*2;
    if bit_stream(i) == inv_bit

dig_sig((j*(samples_per_pulse)+1):(j+1)*(samples_per_pu
lse)) = inv_last_state*ones(1,samples_per_pulse);

dig_sig(((j+1)*(samples_per_pulse)+1):(j+2)*(samples_pe
r_pulse)) = last_state*ones(1,samples_per_pulse);
    else

dig_sig((j*(samples_per_pulse)+1):(j+1)*(samples_per_pu
lse)) = last_state*ones(1,samples_per_pulse);

dig_sig(((j+1)*(samples_per_pulse)+1):(j+2)*(samples_pe
r_pulse)) = inv_last_state*ones(1,samples_per_pulse);
        temp_cons = last_state; % temporary constant
        last_state = inv_last_state;
```

```
        inv_last_state = temp_cons;
    end
end
figure
plot(t,dig_sig,'linewidth',1.5)
grid on
xlabel('time in seconds')
ylabel('Voltage')
ylim([(min_voltage - (max_voltage)*0.2)
(max_voltage+max_voltage*0.2)])
title(['Differential Manchester for
',num2str(bit_stream),', last state =
',num2str(last_state),', inverting bit is
',num2str(inv_bit),''])
```



Performance Task:

Assume your ID is **AB-CDEFG-H**, and then convert 'E', 'F' and 'G' to 4-bit binary to have a bit stream of 12 bits. Convert this bit stream to digital signal using the following methods:

1. Polar NRZ-L assuming bit rate is 4 kbps.
2. Manchester assuming bit rate is 2 kbps.
3. AMI assuming bit rate is 5 kbps.
4. MLT-3 assuming bit rate is 10 kbps.