TASK 1

EXPERIMENT 1 – SAMPLING AND RECONSTRUCTION (SOFTWARE)

Date: 25/7/2024

Aim:

To generate sampled and reconstructed signals of different types(under, critical, over) using MATLAB.

Algorithm:

Initialize and Clear Workspace:

- Clear all variables and close all figures.

Define Time Vector:

- Define a time vector t ranging from 0 to 1 with a step of 0.01.

Input Frequencies:

- Prompt the user to input the following frequencies:
  - fm: Input signal frequency
  - fu: Undersampling frequency
  - fc: Critical sampling frequency
  - fo: Oversampling frequency

Generate Input Signal:

- Compute the input analog signal x using the sine function with frequency fm.

Plot Input Signal:

- Plot the input analog signal x against time t.

Undersampling:

- Define the undersampling time vector tu with step 1/fu.
- Compute the undersampled signal xu using the sine function with frequency fm and time vector tu.

Plot Undersampled Signal:

- Plot the undersampled signal xu against time tu.

Critical Sampling:

- Define the critical sampling time vector tc with step 1/fc.
- Compute the critically sampled signal xc using the sine function with frequency fm and time vector tc.

Plot Critically Sampled Signal:

- Plot the critically sampled signal xc against time tc.

Oversampling:

- Define the oversampling time vector to with step 1/fo.
- Compute the oversampled signal xo using the sine function with frequency fm and time vector to.

Plot Oversampled Signal:

- Plot the oversampled signal xo against time to.

Reconstruct Signals:

- Define reconstruction time vectors for each sampling method:
  - trec for critical sampling with step 1/fc.
  - treu for undersampling with step 1/fu.
  - treo for oversampling with step 1/fo.
- Use the interp1 function to interpolate and reconstruct signals:
  - xrec for critically sampled signal.
  - xreu for undersampled signal.
  - xreo for oversampled signal.

Plot Reconstructed Signals:

- Plot the reconstructed signal from critical sampling against trec.
- Plot the reconstructed signal from undersampling against treu.
- Plot the reconstructed signal from oversampling against treo.

Program:

```
clear all

close all

t=0:0.01:1;

fm=input('Input signal freq fm= ');

fu=input('Under sample freq fu= ');

fc=input('Crtical sample freq fc= ');

fo=input('Over sample freq fo= ');

%IP signal

x=sin(2*pi*fm*t);

subplot(3,3,2);

plot(t,x);

grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Input analog signal');

%Undersampling

tu=0:1/fu:1;

xu=sin(2*pi*fm*tu);

subplot(3,3,4);

stem(tu,xu);

grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Undersampled signal');

%Critical sampling

tc=0:1/fc:1;

xc=sin(2*pi*fm*tc);
```

```
subplot(3,3,5);
stem(tc,xc);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Critically sampled signal');
```

%Oversampling

```
to=0:1/fo:1;
xo=sin(2*pi*fm*to);
subplot(3,3,6);
stem(to,xo);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Oversampled signal');
```

%Reconstruction Critical

```
trec=0.1:1/fc:1;
xrec=interp1(tc,xc,trec);
subplot(3,3,8);
plot(trec,xrec);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Reconstructed Signal(Critical)');
```

%Reconstruction Undersampling

```
treu=0.1:1/fu:1;
xreu=interp1(tu,xu,treu);
subplot(3,3,7);
plot(treu,xreu);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Reconstructed Signal(Under)');
```

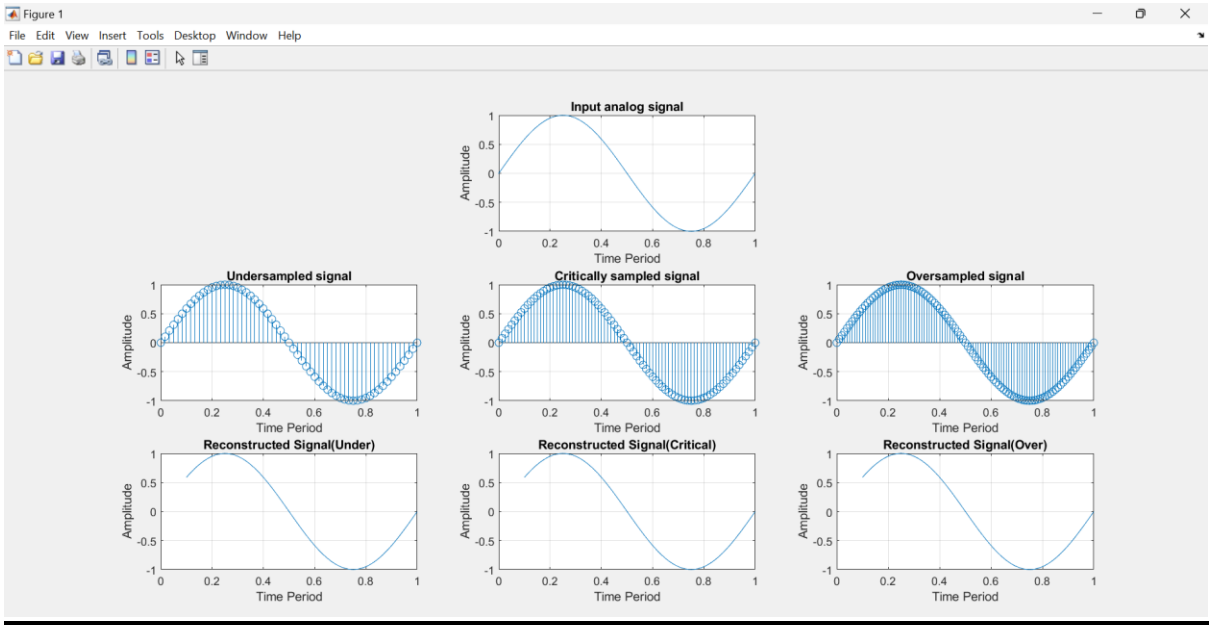
%Reconstruction Oversampling

```
treo=0.1:1/fo:1;

xreo=interp1(to,xo,treo);

subplot(3,3,9);

plot(treo,xreo);

grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Reconstructed Signal(Over)');
```

Observation/Snapshots of the Output taken from MATLAB-Simulink:

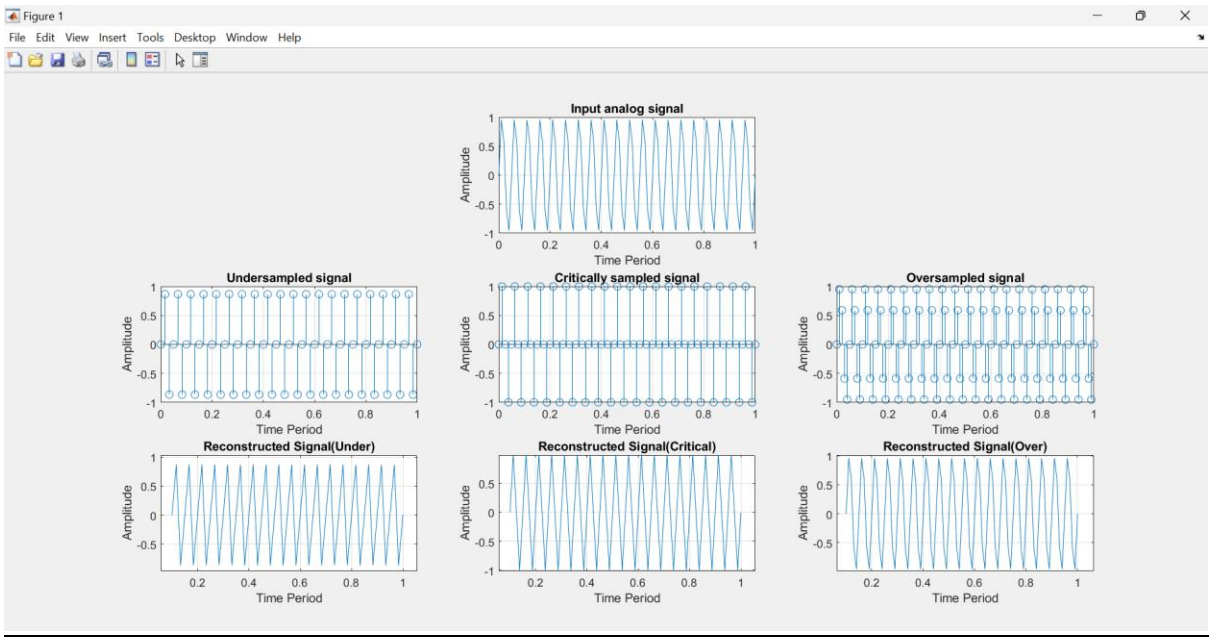
For:

```
Input signal freq fm= 1
Under sample freq fu= 60
Crtical sample freq fc= 80
Over sample freq fo= 100
>>
```

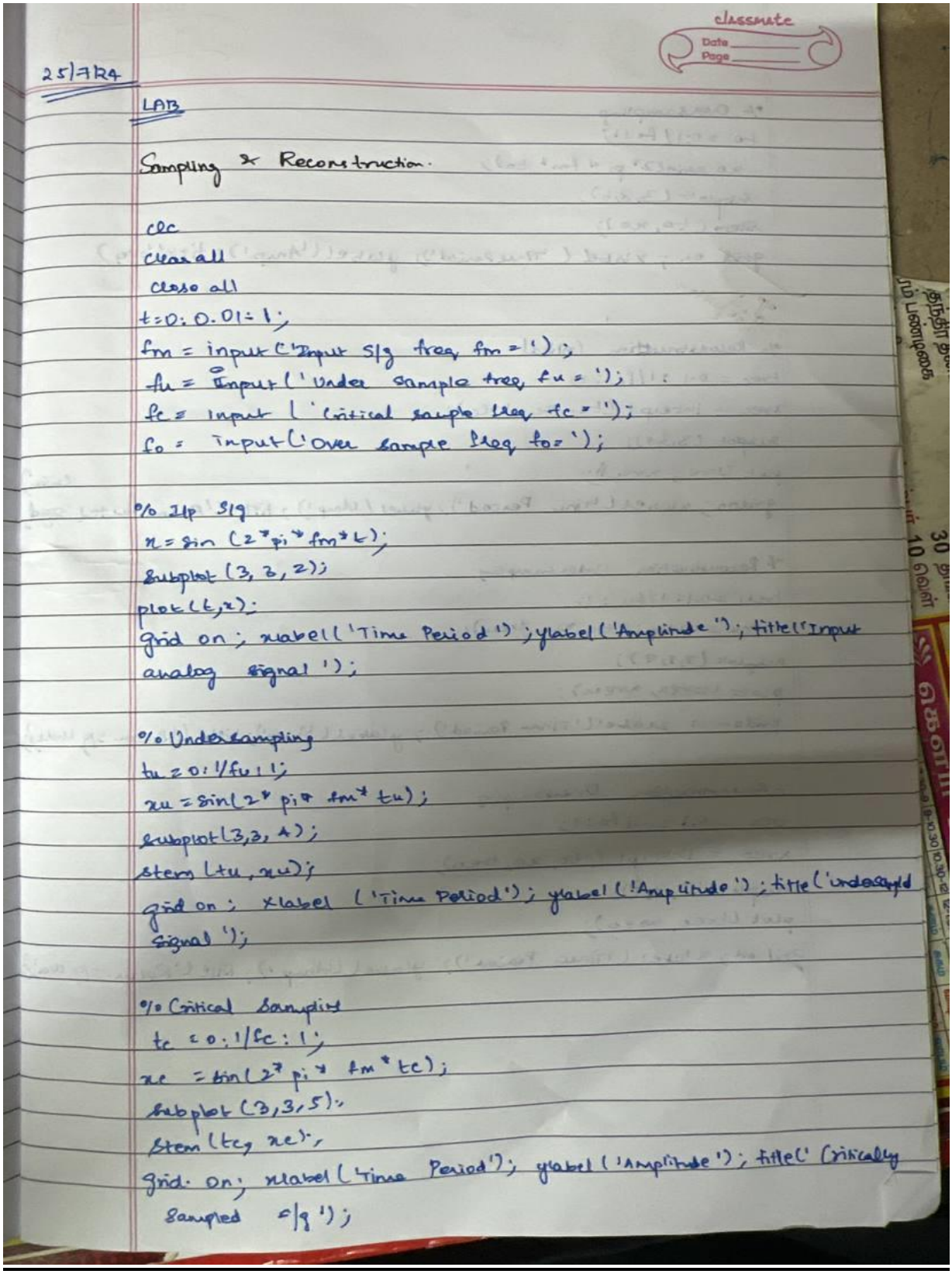


For:

```
Input signal freq fm= 20
Under sample freq fu= 60
Crtical sample freq fc= 80
Over sample freq fo= 100
>>
```



Proof of Output Verification:

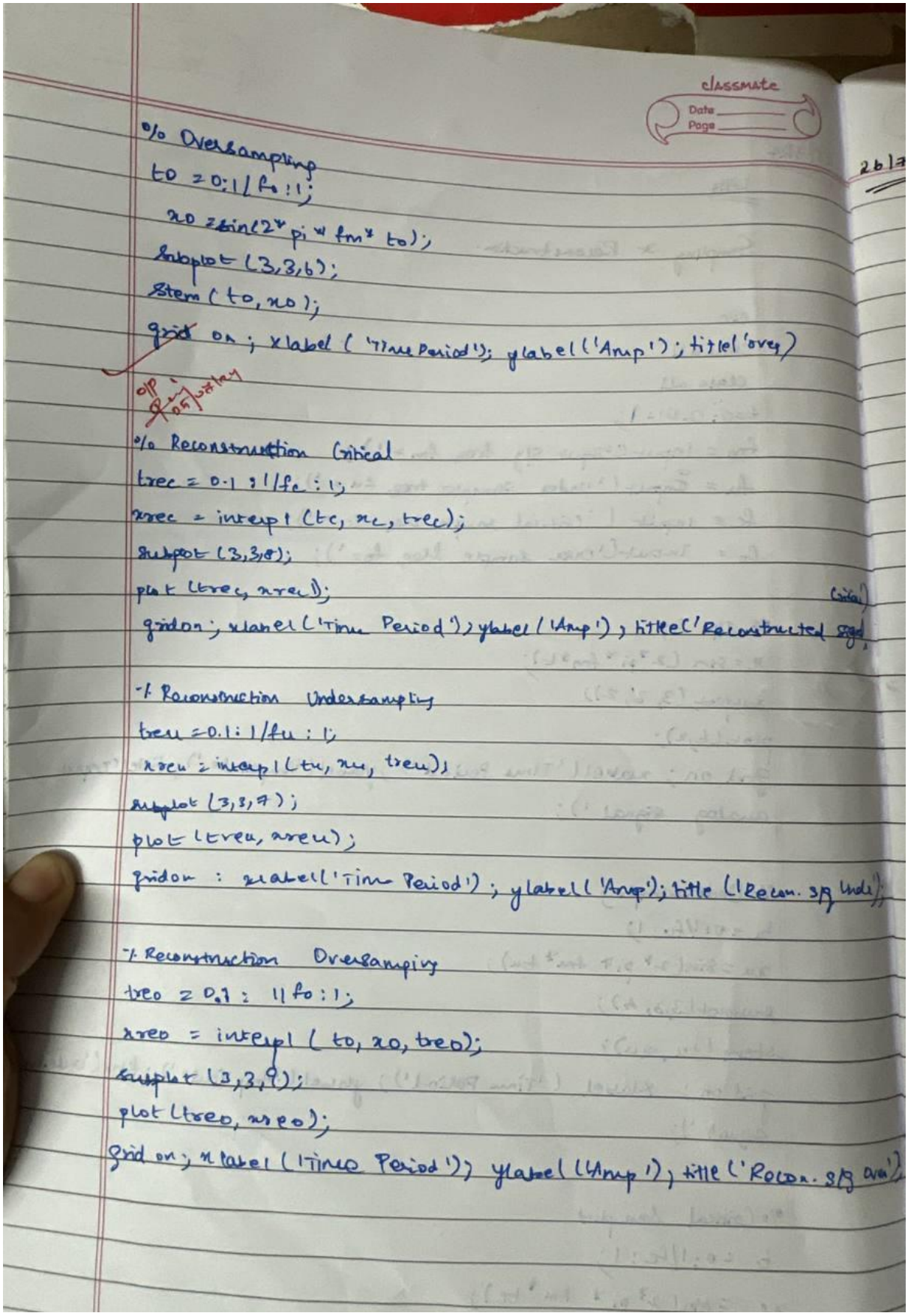


25/7/24

LAB

Sampling & Reconstruction.

```
clc
clear all
close all
t=0:0.01:1;
fm = input('Input S/g freq, fm = ');
fu = input('Under sample freq, fu = ');
fc = input('Critical sample freq, fc = ');
fo = input('Over sample freq, fo = ');

% Ip S/g
x = sin(2*pi*fm*t);
subplot(3,3,2);
plot(t,x);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Input
analog signal');

% Undersampling
tu = 0:1/fu:1;
xu = sin(2*pi*fm*tu);
subplot(3,3,4);
stem(tu,xu);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Undersampled
signal');

% Critical Sampling
tc = 0:1/fc:1;
xc = sin(2*pi*fm*tc);
subplot(3,3,5);
stem(tc,xc);
grid on; xlabel('Time Period'); ylabel('Amplitude'); title('Critically
sampled o/p');
```



```
% Oversampling
to = 0:1/fo:1;
xo = sin(2*pi*fm*to);
subplot(3,3,6);
stem(to,xo);
grid on; xlabel('Time Period'); ylabel('Amp'); title('Over
% Reconstruction Critical
trec = 0:1:1/fc:1;
xrec = interp1(tc, xc, trec);
subplot(3,3,8);
plot(trec, xrec);
grid on; xlabel('Time Period'); ylabel('Amp'); title('Reconstructed Sig
% Reconstruction Undersampling
treu = 0:1:1/fu:1;
xreu = interp1(tu, xu, treu);
subplot(3,3,7);
plot(treu, xreu);
grid on; xlabel('Time Period'); ylabel('Amp'); title('Recon. Sig Und
% Reconstruction Oversampling
treo = 0:1:1/fo:1;
xreo = interp1(to, xo, treo);
subplot(3,3,9);
plot(treo, xreo);
grid on; xlabel('Time Period'); ylabel('Amp'); title('Recon. Sig Ove
```

Result:

Thus the input signal, oversampled signal, critical sampled signal, under sampled signal, and corresponding reconstructed signals were plotted.

Inference:

Hence, the concept of “Sampling and Reconstruction” is understood.

EXPERIMENT 2 – PULSE CODE MODULATION AND DEMODULATION (HARDWARE)

Date: 18/7/2024

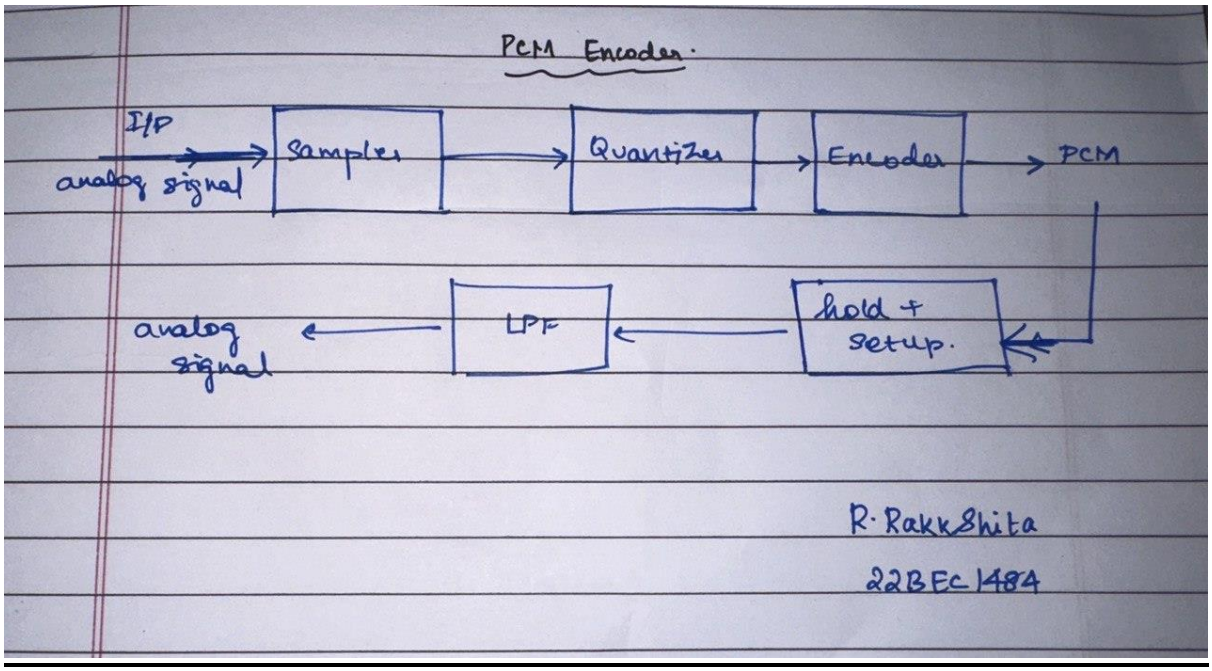
Aim:

To perform Pulse Code Modulation and Demodulation using necessary hardware components.

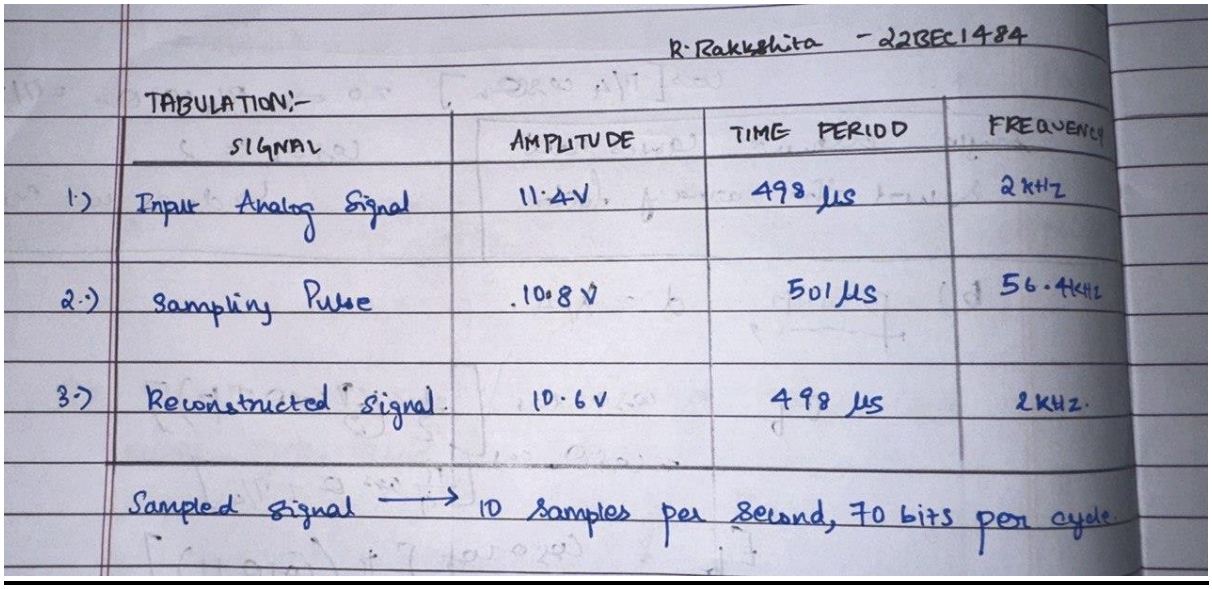
Procedure:

- Make the necessary connections as in the block diagram
- Give appropriate frequencies in the function generator.
- Turn on the oscilloscope for viewing the required waveforms.
- Note down the following values of required waveforms:
  - Amplitude
  - Time Period
  - Frequency
- Make a tabular column of the above mentioned required values.

Block Diagram:



Tabulation:



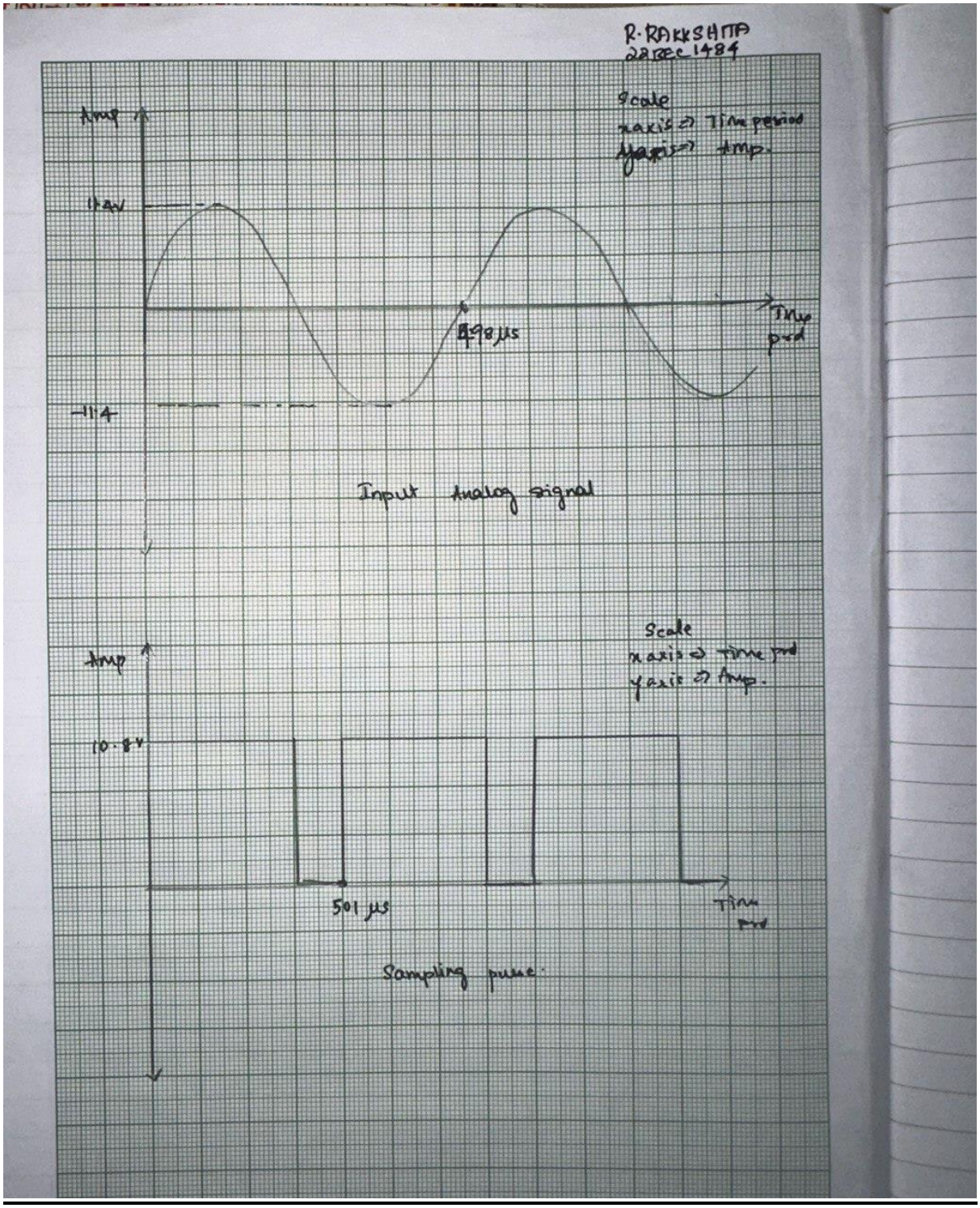
R. Rakkshita - 22BEC1484

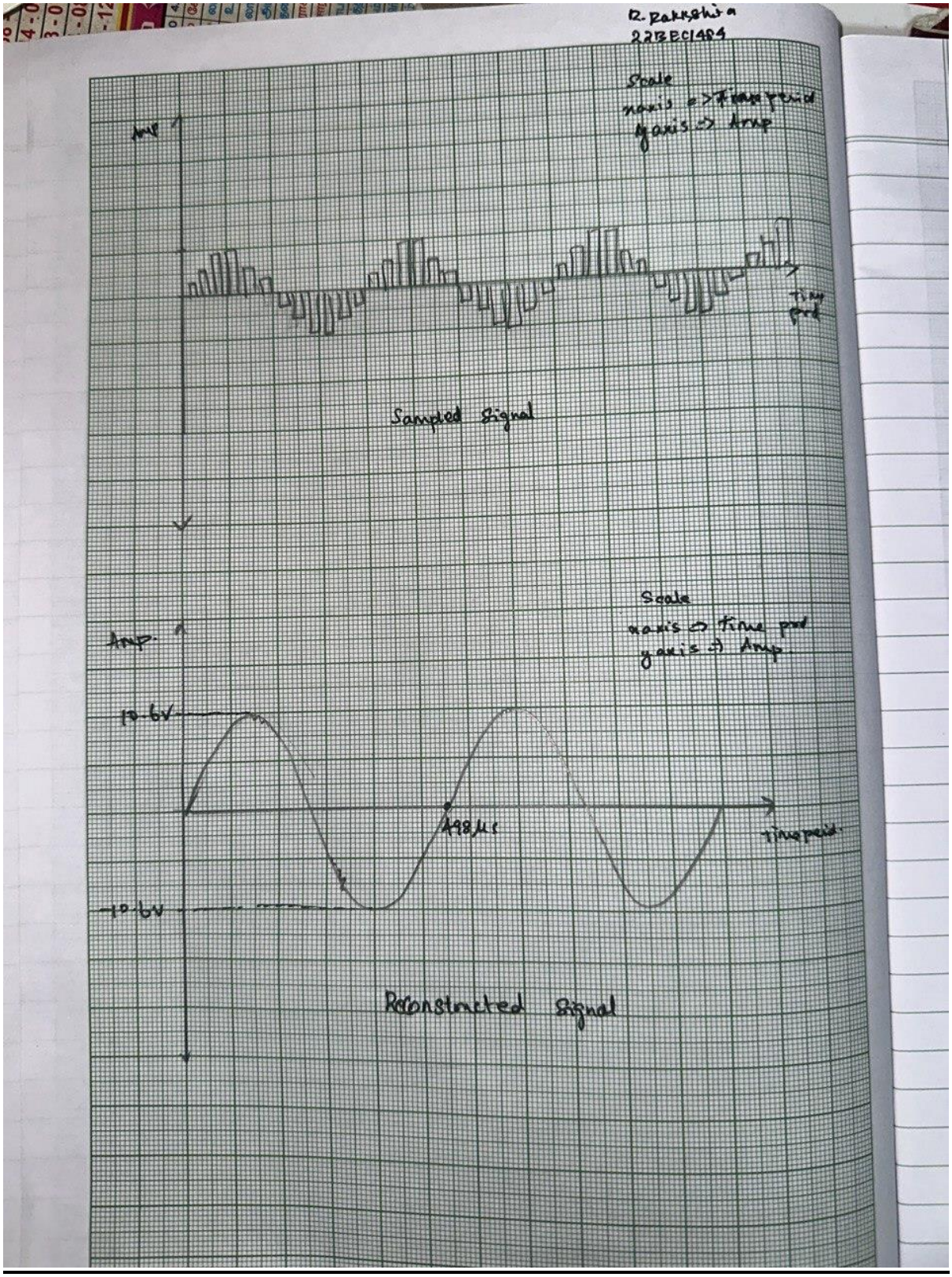
TABULATION:-

	SIGNAL	AMPLITUDE	TIME PERIOD	FREQUENCY
1.)	Input Analog Signal	11.4V	498 μs	2 kHz
2.)	Sampling Pulse	10.8 V	501 μs	56.4kHz
3.)	Reconstructed signal	10.6 V	498 μs	2KHz.

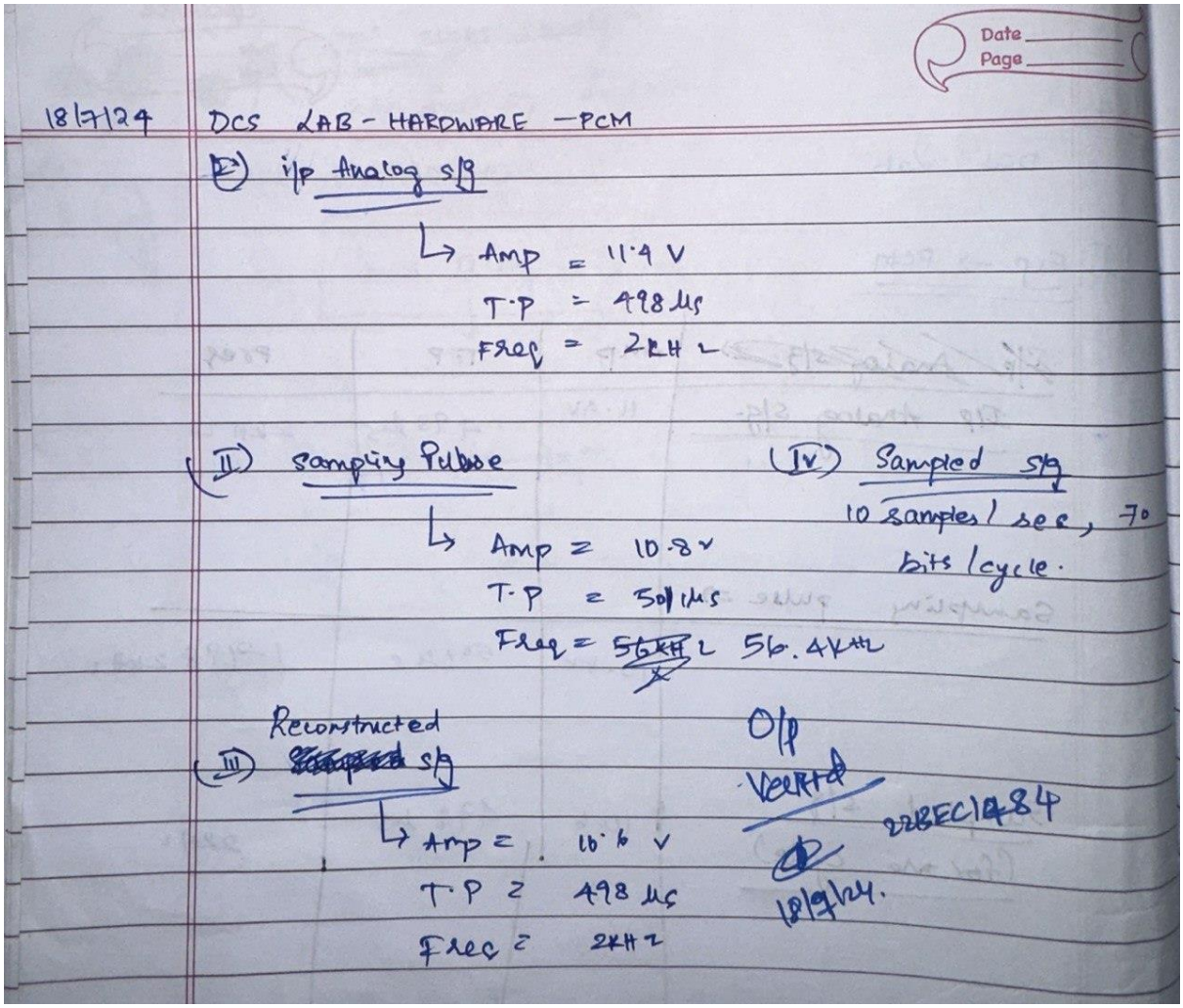
Sampled signal ⟶ 10 samples per second, 70 bits per cycle.

Graph:


R. RAKKSHITA
22BEC1484
Scale
x axis ⇒ Time period
y axis ⇒ Amp.
Amp
1.4V
-1.1V
498 µs
Time period
Input analog signal
Amp
Scale
x axis ⇒ Time period
y axis ⇒ Amp.
10.8V
501 µs
Time period
Sampling pulse.


R. Rakkshita
22BEC1484
Scale
x axis ⇒ Time period
y axis ⇒ Amp
Amp
Time period
Sampled Signal
Scale
x axis ⇒ Time period
y axis ⇒ Amp.
Amp.
10.6V
-10.6V
498μs
Time period
Reconstructed Signal

Proof of Output Verification:



Result:

Thus the input analog signal, sampling pulse, sampled signal, &reconstructed signals are plotted and viewed in oscilloscope.

Inference:

Thus the concept of “Pulse Code Modulation and Demodulation” is understood.

DIGITAL COMMUNICATIONS AND SYSTEM LAB

TASK 2

Exp 3: Pulse Code Modulation & Demodulation:

(software)

Date: 08/08/2024

Aim:

To perform Pulse Code Modulation and Demodulation of a system using Matlab Simulink and answer the following questions.

Experimentation

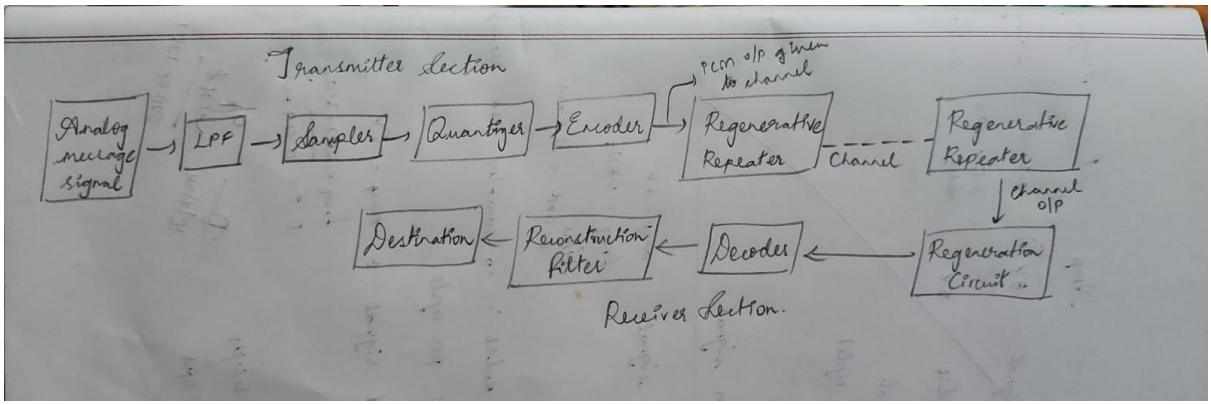
- With initial frequency setup for Analog Signal and Pulse Stream generator as discussed perform following experiments

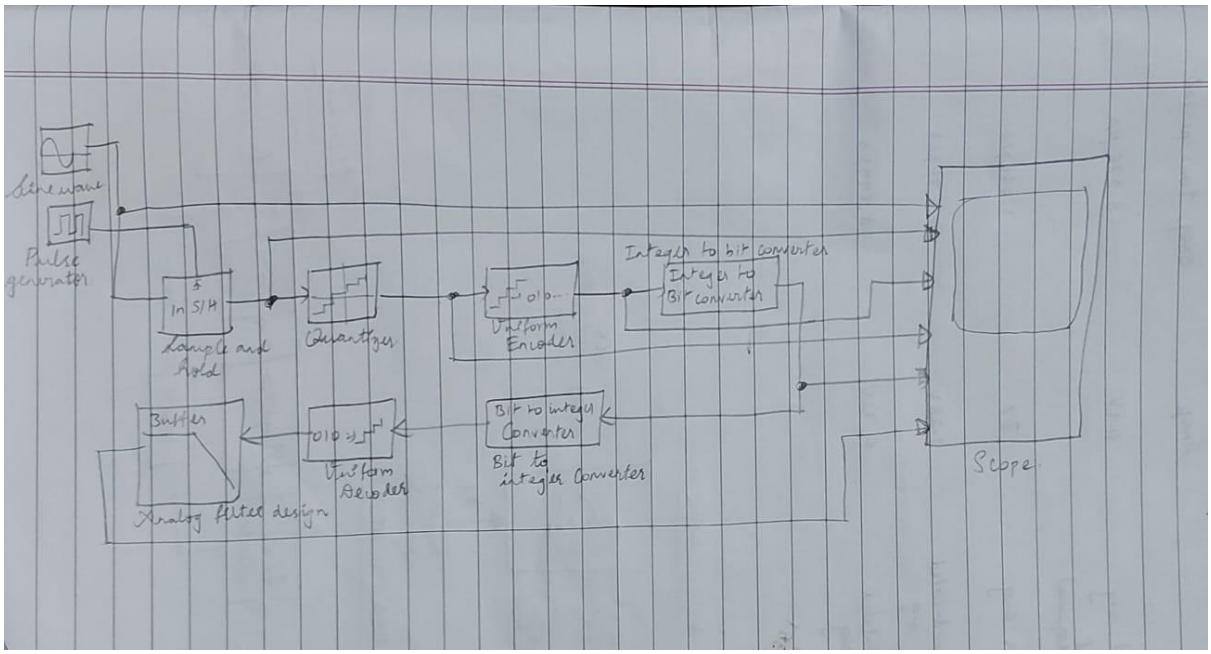
1. Generate the outputs (all 6 waveforms) for Quantizer Levels of (4,8,16) and amplitude of signals V_p=10V.
2. Generate the outputs (all 6 waveforms) for Various Quantization Intervals of (0.1, 5, 20) with L=8.
3. Generate the outputs (all 6 waveforms) for Various Amplitudes of (5V, 15V, 20V) with L=16.

Algorithm:

Make the circuit for Pulse Code Modulation and Demodulation in Simulink (with appropriate parameters for Sine wave signal, Pulse Generator, Sample and Hold, Quantizer, Uniform Encoder, Bit to Integer converter, Integer to Bit converter, Uniform Decoder, Analog filter design) and run it.

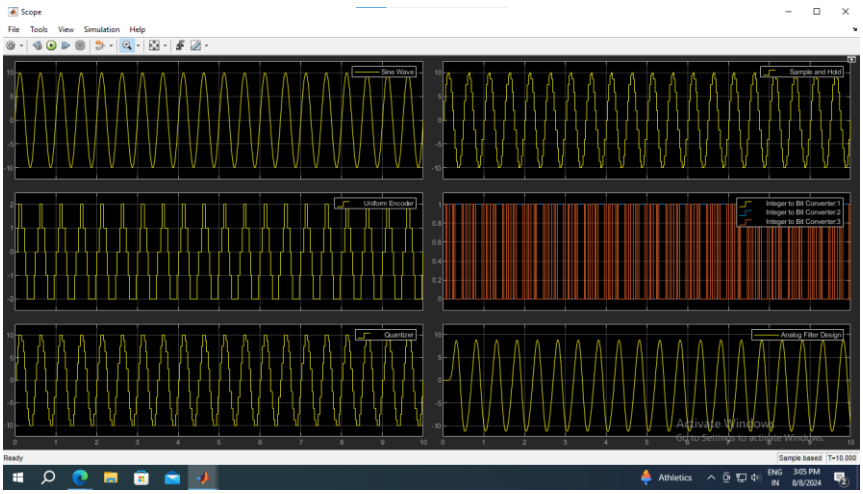
Block Diagram:





Outputs taken from the Matlab-Simulink:

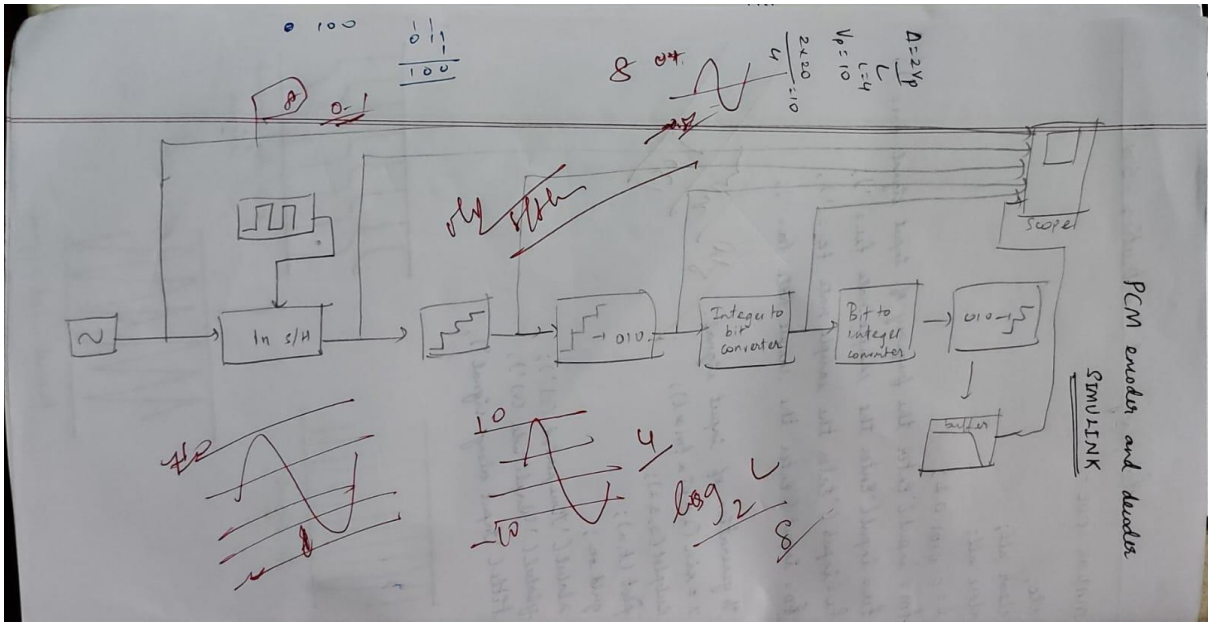
1. Quantizer Levels (4,8,16) with Vp=10V:



2. Quantization Intervals (0.1,5,20) with L=8:

3. Amplitude (5V,15V,20V) with L=16:

Proof of Output Verification:



Result/Inference:

Hence, pulse code modulation and demodulation of a signal is performed, and the waveforms for different quantizer levels, quantizer intervals and amplitudes are generated.

Exp 4: ASK Modulation & Demodulation

(hardware)

Date: 01/08/2024

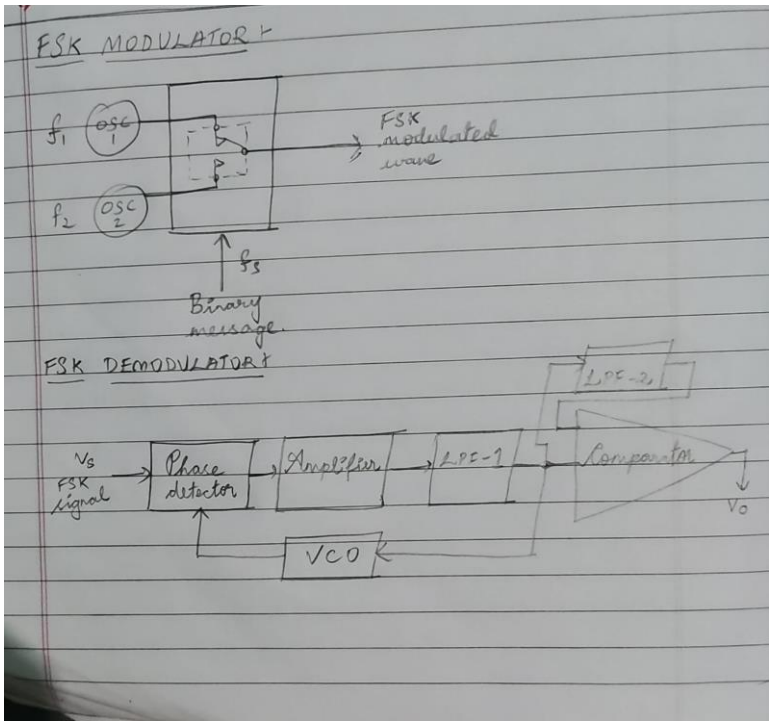
Aim:

To perform ASK modulation & demodulation of a signal using hardware trainer kit.

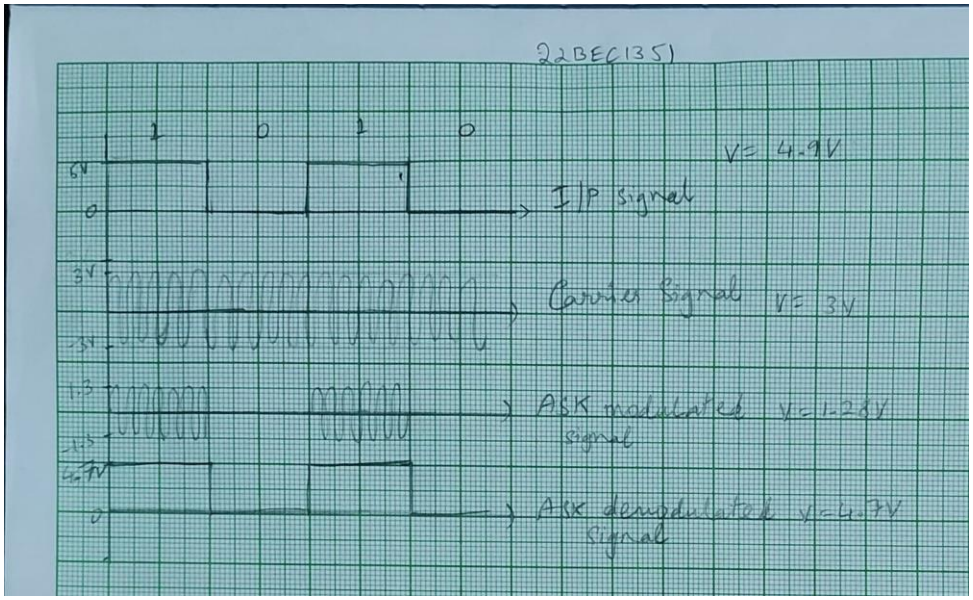
Procedure:

- Connect the circuit as per the kit connection diagram in the hardware trainer kit.
- Take Amplitude and Time period readings of input signal, carrier signal, ASK modulated signal and Demodulated signal using Oscilloscope (Connect the positive and negative probes accordingly while taking respective readings).

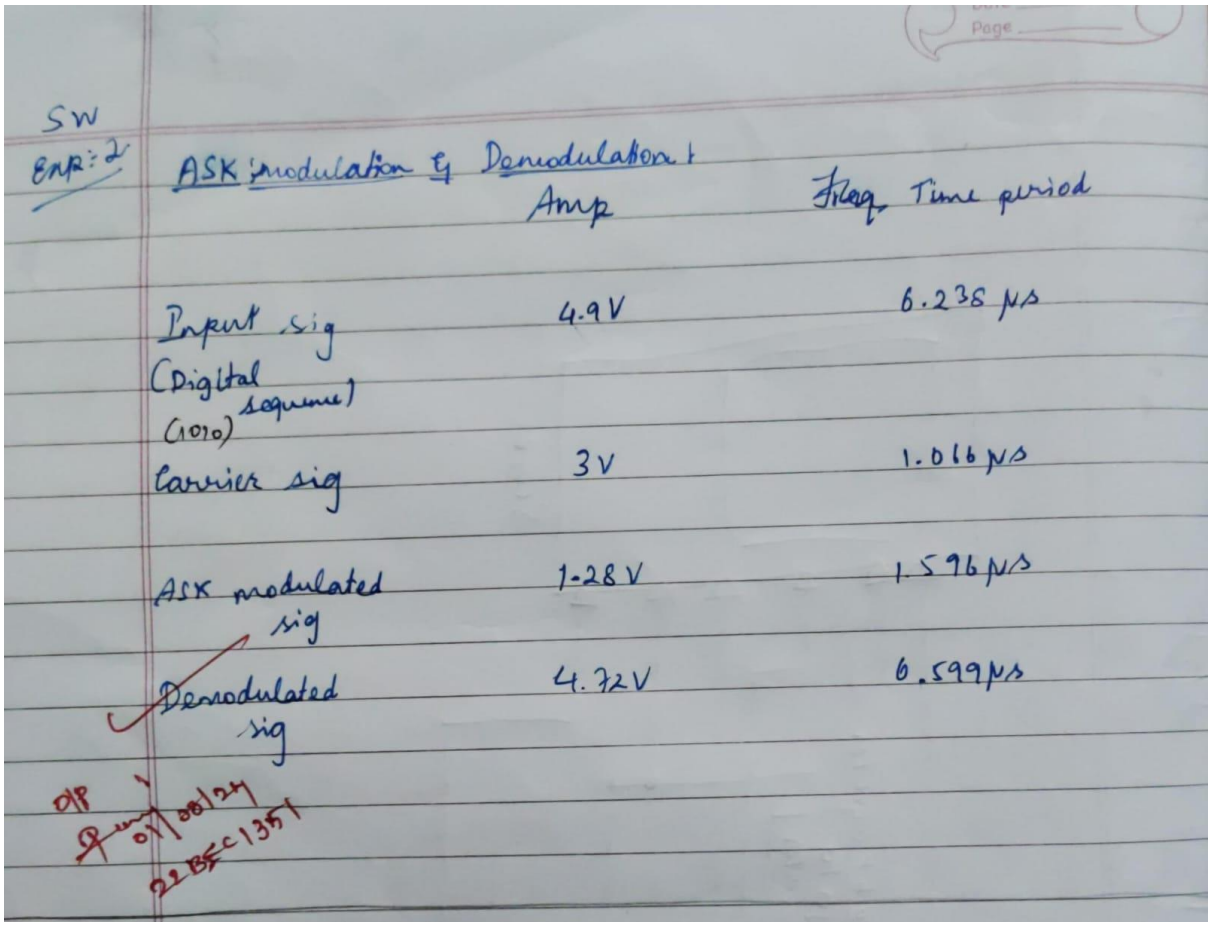
Block Diagram:



Graph:



Tablulation with proof of Output Verification:



Result/Inference:

Hence, ASK modulation and demodulation of a signal is performed using hardware trainer kit.

K Kishore Raghav
22BEC1480

EXPERIMENT 5 – DIFFERENTIAL PULSE CODE MODULATION ENCODER AND DECODER (SOFTWARE)

Date: 5/9/2024

Aim:

To perform Pulse Code Modulation and Demodulation of a system using Matlab Simulink and answer the following questions.

Experiment

1. Implement DPCM encoding and compare it with PCM (using same Setup in previous slide) and show how many bits are reduced per sample in DPCM.

2. Implement DPCM encoding (using same Setup in previous slide) but change the sample period in Sine wave block from zero(0) to (1/1000; 1/10000; 1/100000) and write the inference from them.

3. Implement DPCM decoder

Algorithm:

 Start Simulink:

- Open MATLAB.
- Launch Simulink by typing simulink in the MATLAB command window.

 Create a New Model:

- Open a blank Simulink model.

 Generate Analog Signal (Input Signal):

- From the Simulink library, choose a **Sine Wave** block to generate an analog signal. Set its frequency, amplitude, and sampling time as required.

 Sampling:

- Add a **Zero-Order Hold** block to simulate the sampling process.
- Connect the **Sine Wave** block output to the **Zero-Order Hold** input.

- Configure the sample time based on the desired sampling rate (Nyquist criteria: sampling rate should be at least twice the highest frequency of the signal).

 Quantization:

- Add a **Quantizer** block to quantize the sampled signal.
- Set the quantization interval (quantization step size) based on the number of bits you intend to use for encoding.
- Connect the output of the **Zero-Order Hold** block to the **Quantizer** block.

 Encoding (DPCM):

- Insert a **Bit Conversion** or a custom block to convert the quantized samples into binary form.
- Simulink does not have a direct DPCM block, but you can use MATLAB functions or other Simulink blocks to simulate the encoding process.

 Reconstruction (Optional):

- Use a **Zero-Order Hold** block again to hold the digital values and recreate the sampled signal.
- Add a **Lowpass Filter** block to filter out high-frequency components and reconstruct the original analog signal.

 Output Visualization:

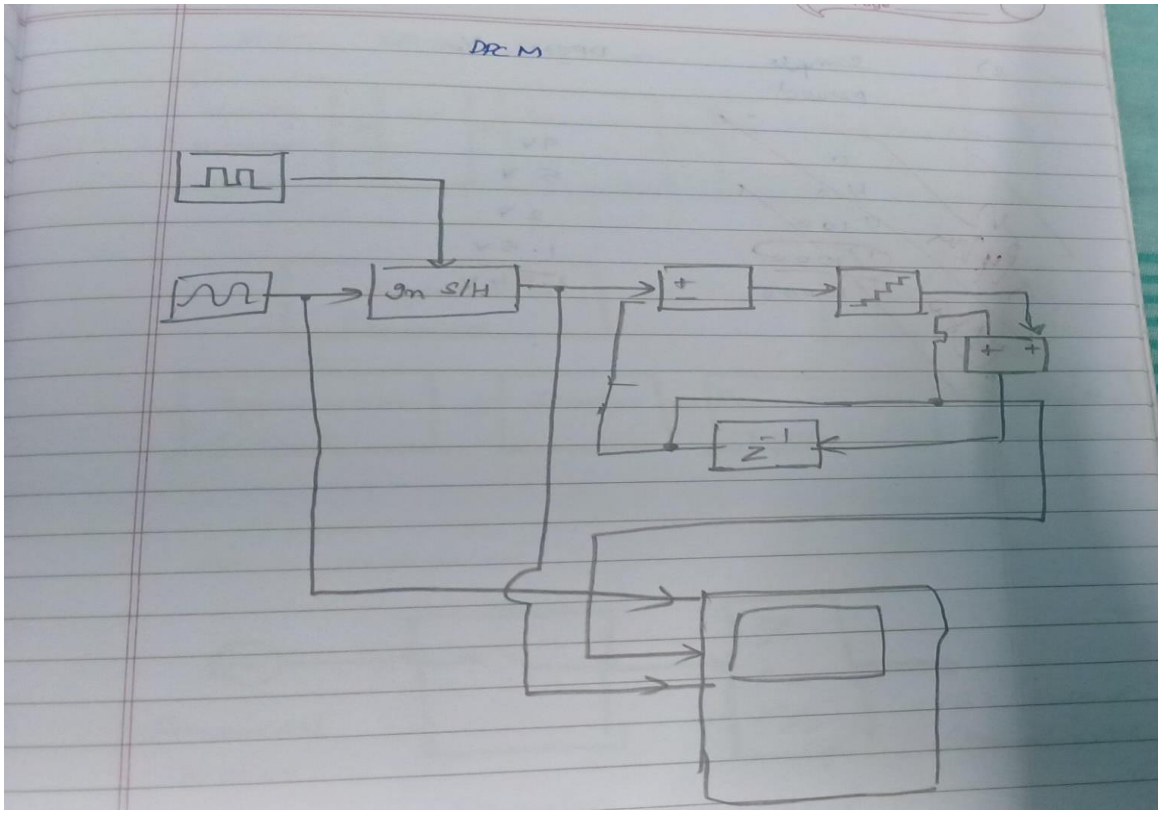
- Use a **Scope** block to visualize the original analog signal and the reconstructed signal (if you perform reconstruction).
- Compare the original and reconstructed signals in terms of Signal-to-Noise Ratio (SNR) or other performance metrics.

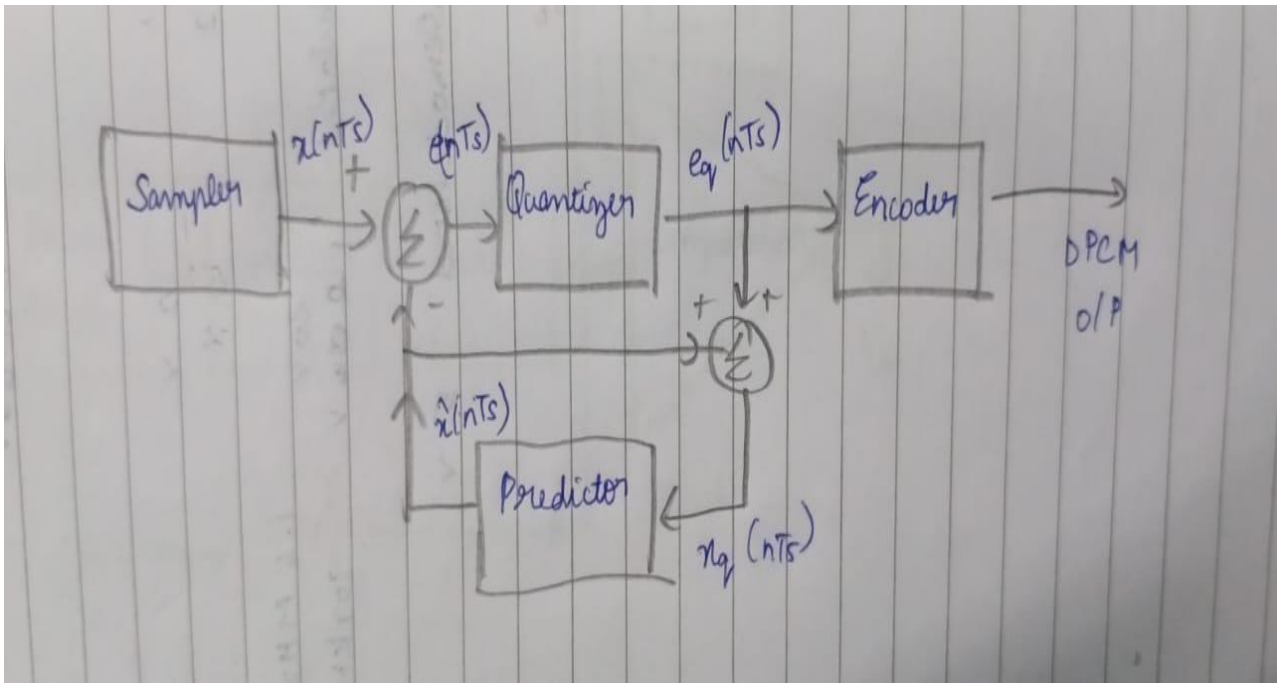
 Run the Simulation:

- Set appropriate simulation parameters (start time, stop time).
- Click on the **Run** button in the Simulink interface to execute the simulation and observe the results.

K Kishore Raghav
22BEC1480

Block Diagram:

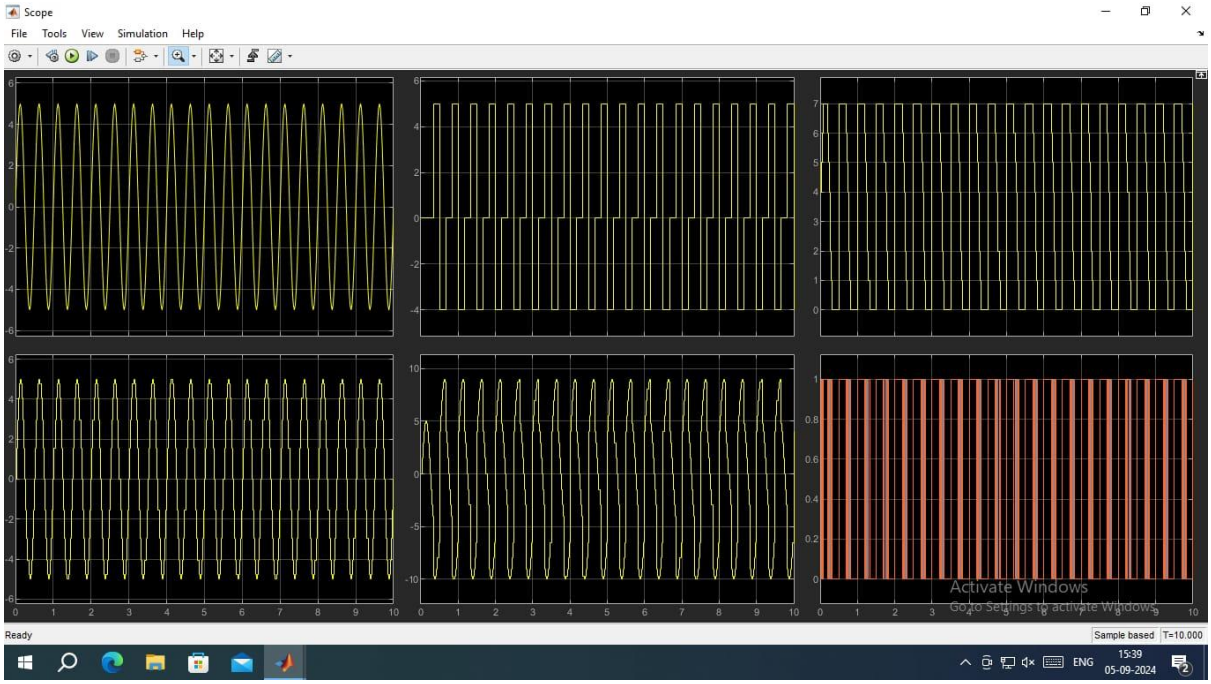




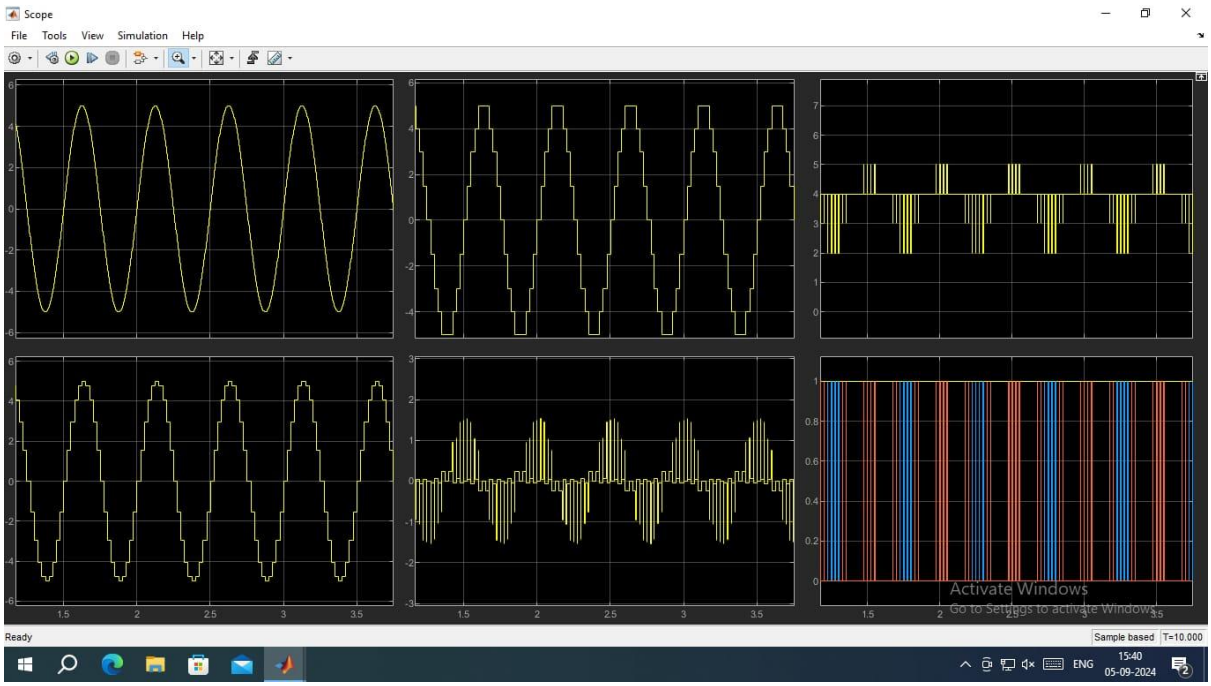
K Kishore Raghav
22BEC1480

Observation/Snapshots of the Output taken from MATLAB-Simulink:

1.Worst case scenario(Sample period of sine wave is 0):

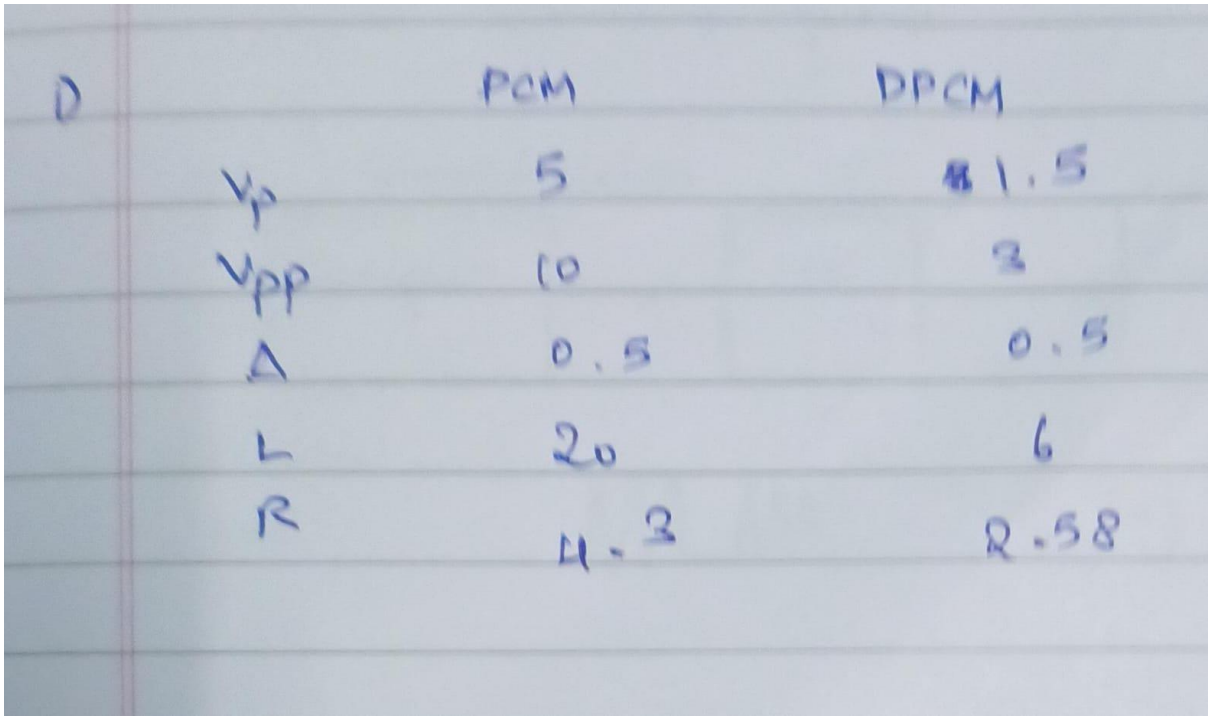


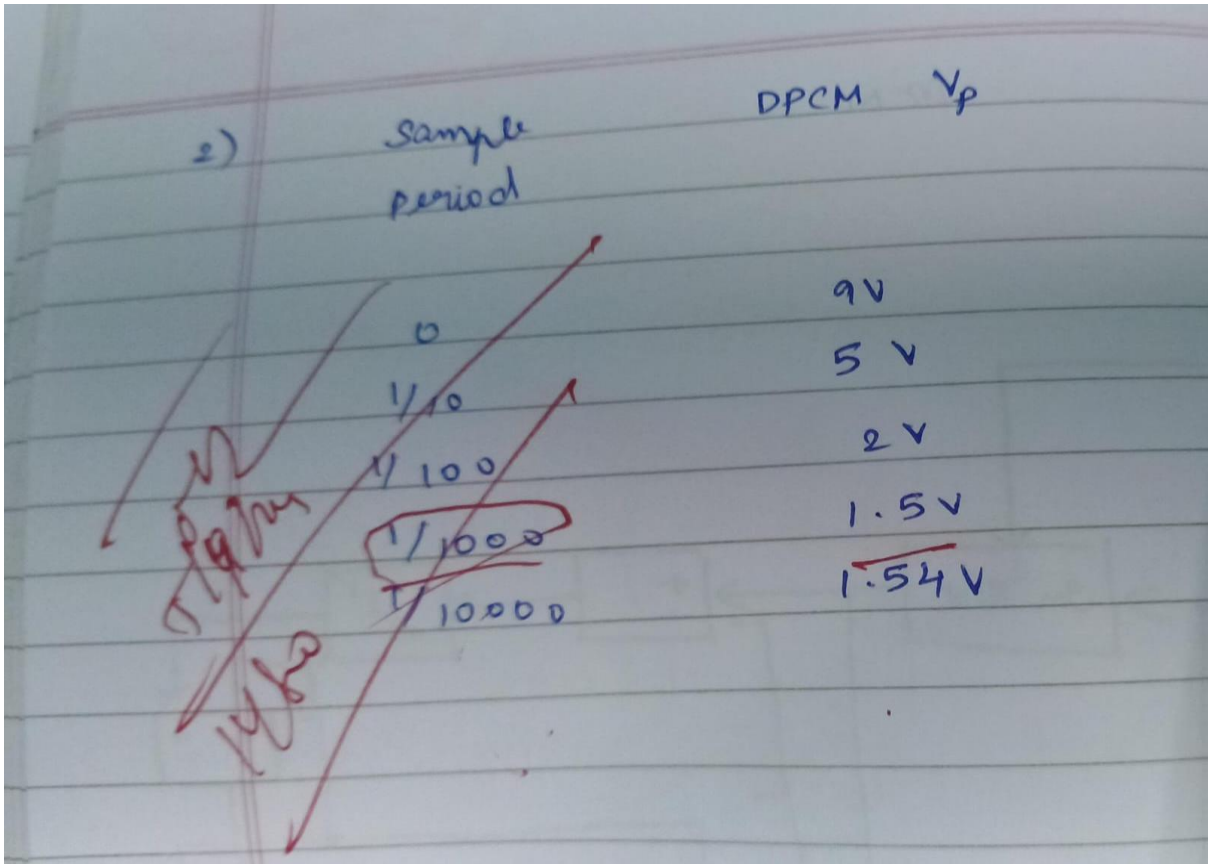
2. Best case scenario(Sample period of sine wave is 1/1000):



K Kishore Raghav
22BEC1480

Proof of Output Verification:





K Kishore Raghav
22BEC1480

Result:

Hence, pulse code modulation and demodulation of a signal is performed, and the waveforms for different quantizer levels, quantizer intervals and amplitudes are generated.

K Kishore Raghav
22BEC1480

EXPERIMENT 6 – FSK MODULATION AND DEMODULATION(HARDWARE)

Date: 22/8/2024

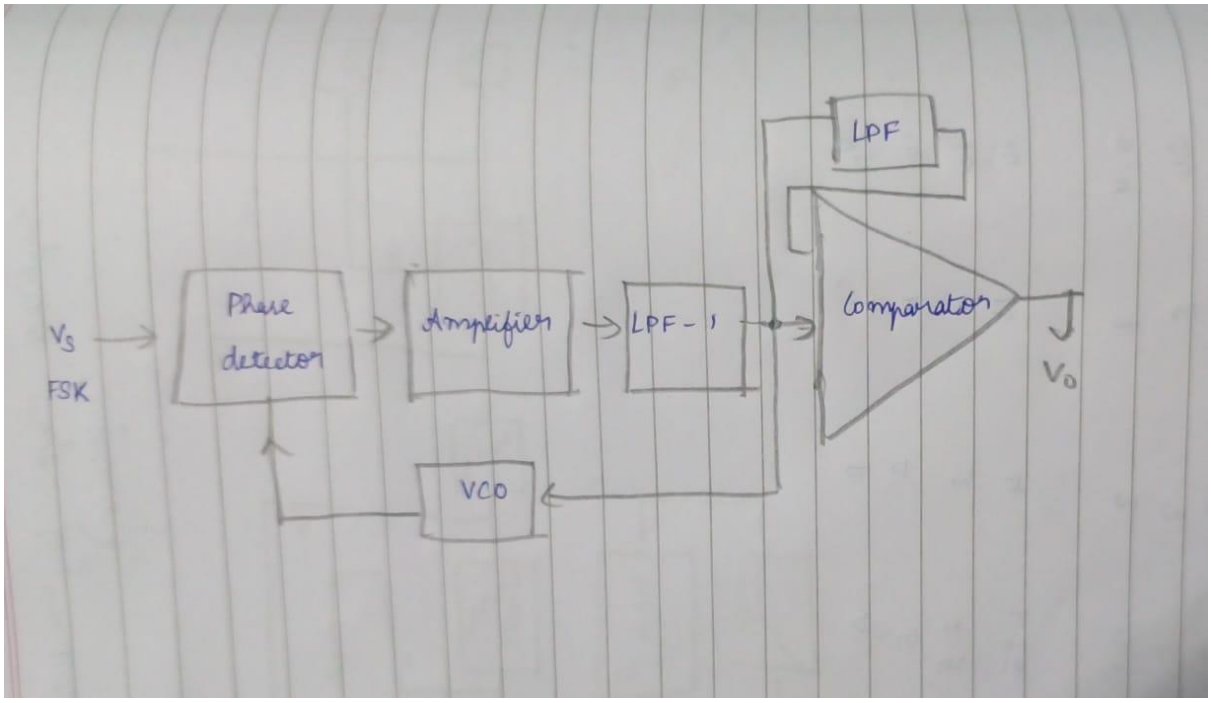
Aim:

To perform FSK Modulation and Demodulation using necessary hardware components.

Procedure:

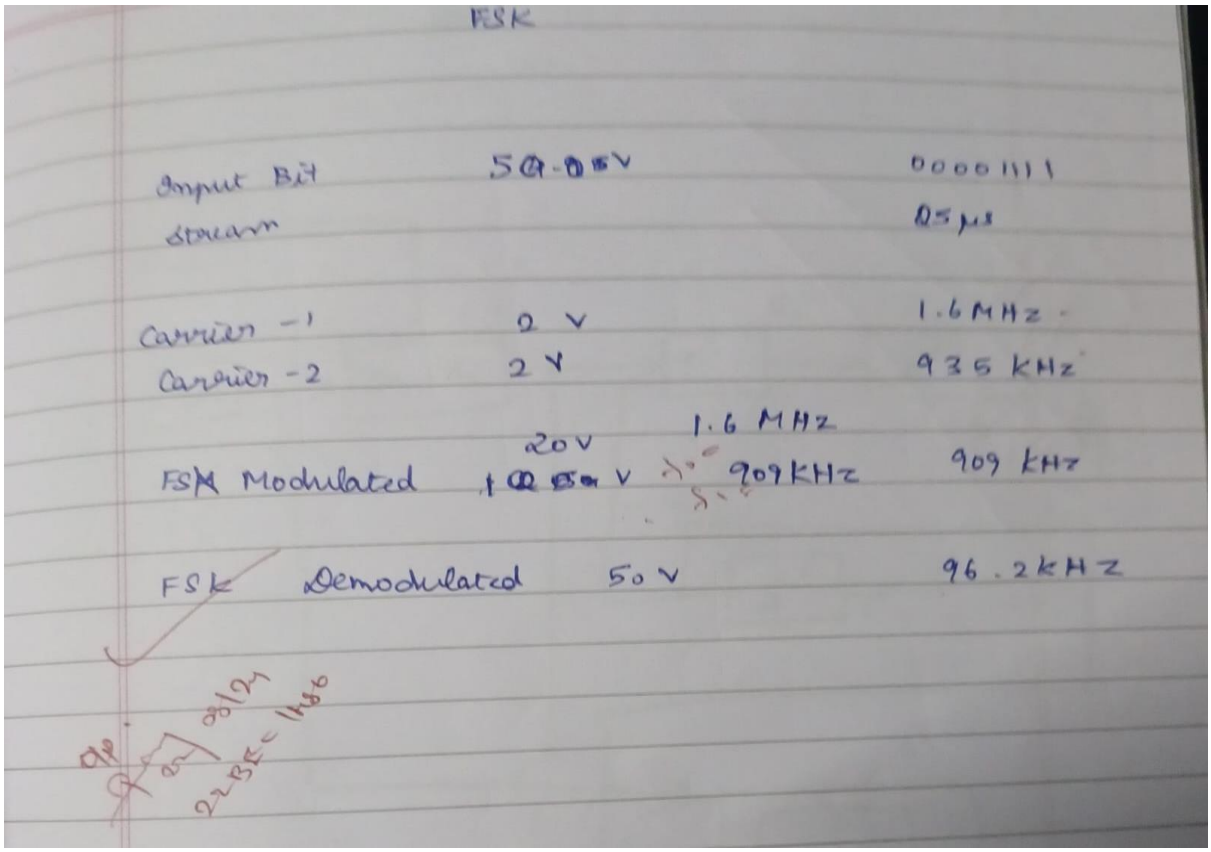
- Connect the necessary connections as in the block diagram
- Setup the suitable frequencies in the function generator.
- Connect to the oscilloscope for obtaining the required waveforms.
- Note down the Amplitude, Frequency and Time Period of required waveforms.
- Note down the appropriate values of the necessary conditions.

Block Diagram



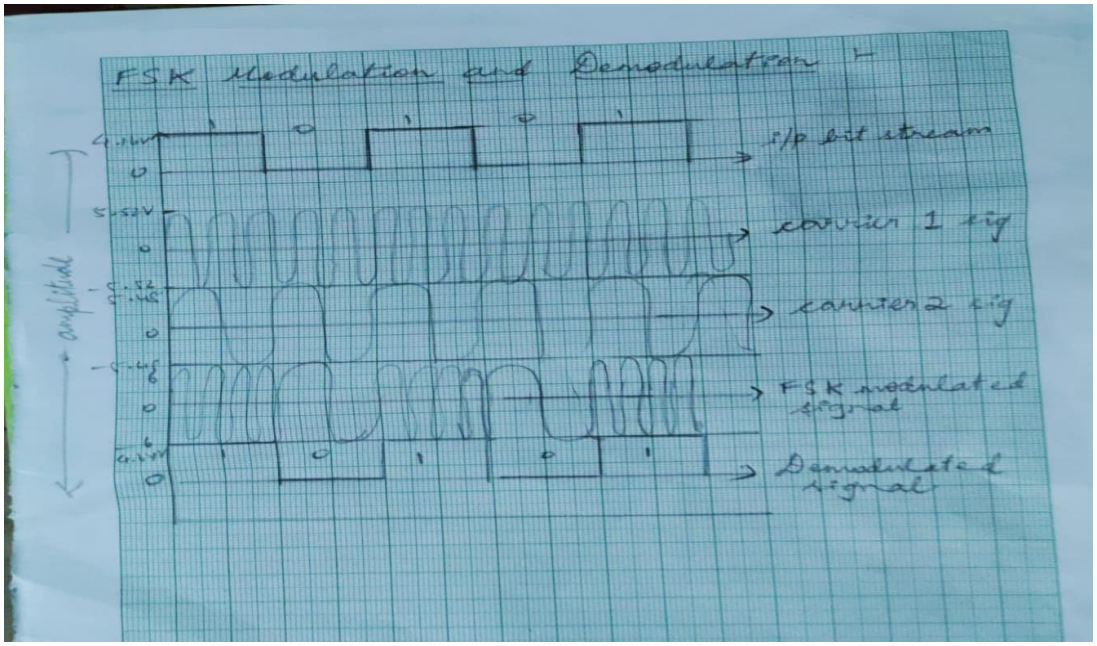
K Kishore Raghav
22BEC1480

Tabulation and Verification:



K Kishore Raghav
22BEC1480

Graph:



Result:

Hence, FSK modulation and demodulation of a signal is performed using hardware trainer kit

TASK-4

DATE : 26/09/2024

EXP 5: DELTA MODULATION AND DEMODULATION (SOFTWARE)

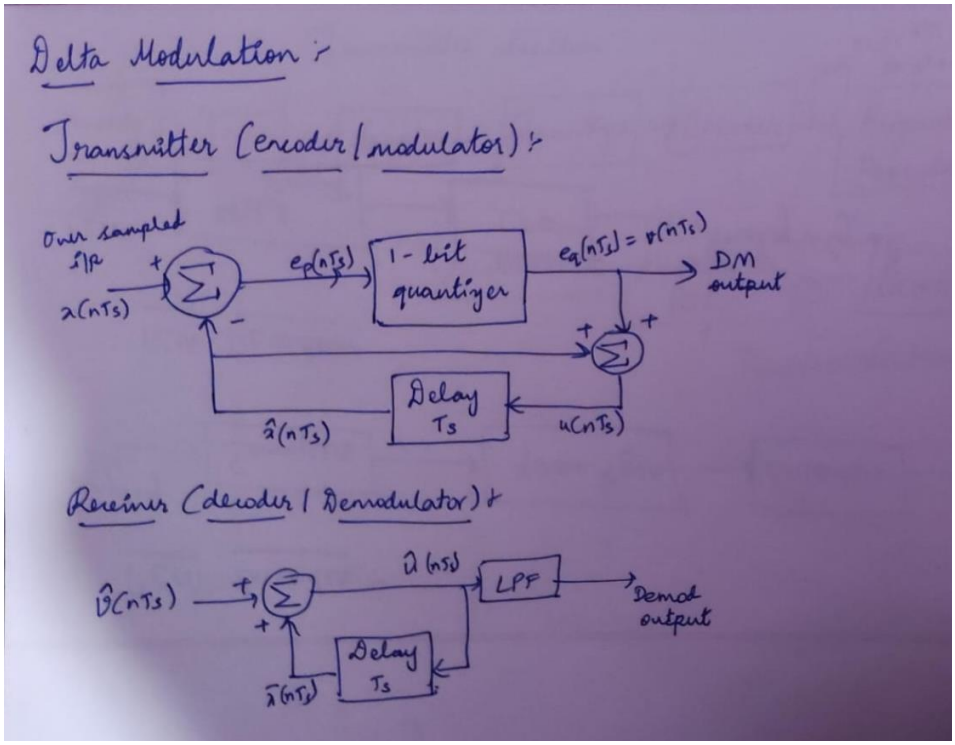
AIM:

To generate Delta Modulated and Demodulated signal for a sinosoidal input. Compare it with the PCM and DPCM modulated signals and write the inference from them.

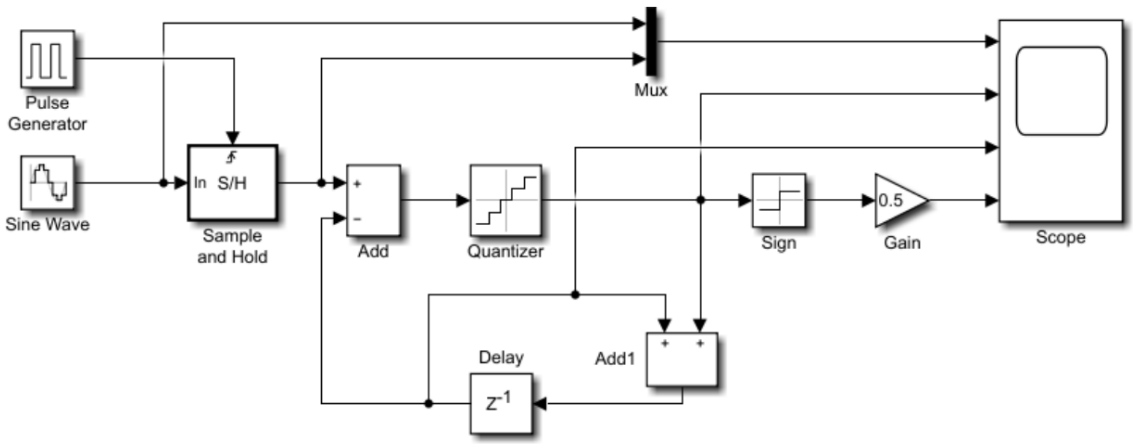
Experiment

1. Implement DM encoding and compare it with PCM, DPCM (using same Setup in previous slide) and portray the significance of every method.
2. Implement DM encoding (using same Setup in previous slide) but change the sample period in Sine wave block to zero and write the inference from them.
3. Implement DM decoder

BLOCK DIAGRAM:

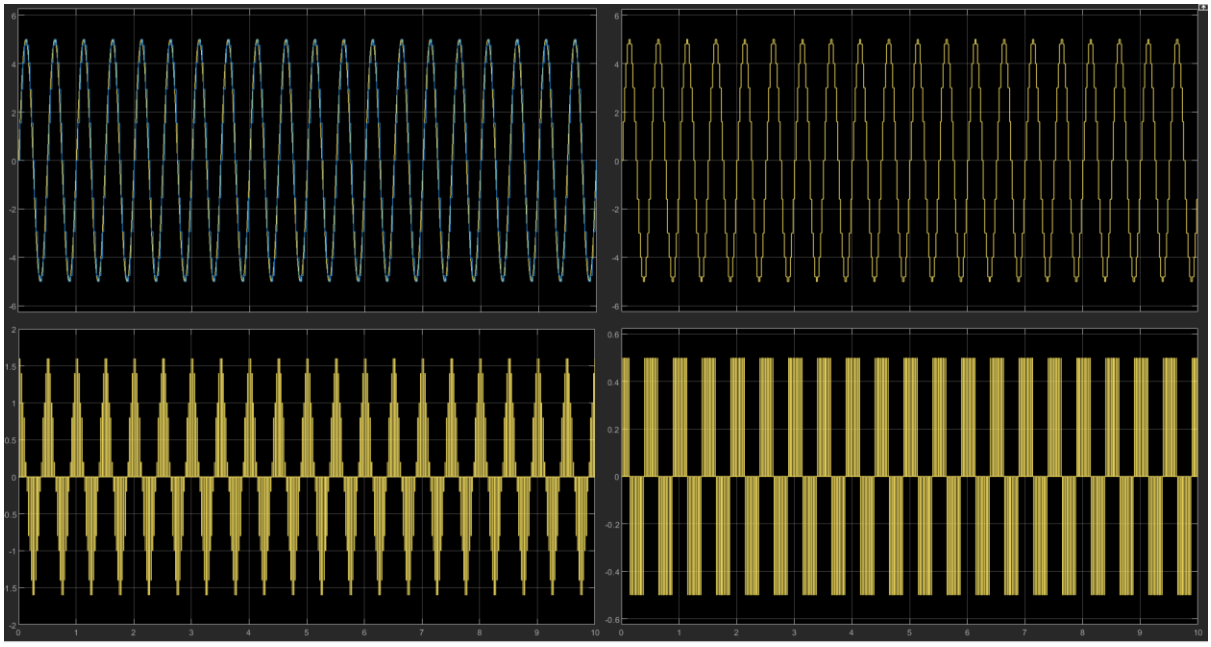


SIMULINK BLOCK DIAGRAM:

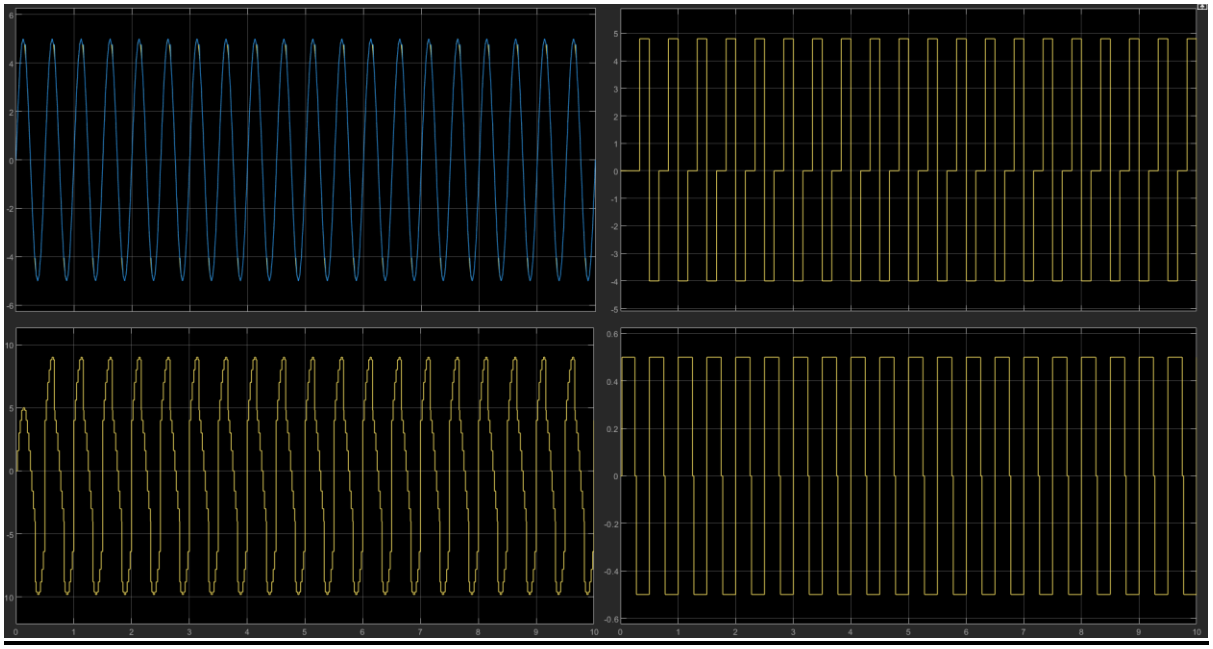


OBSERVATION/SNAPSHOTS OF THE OUTPUT:

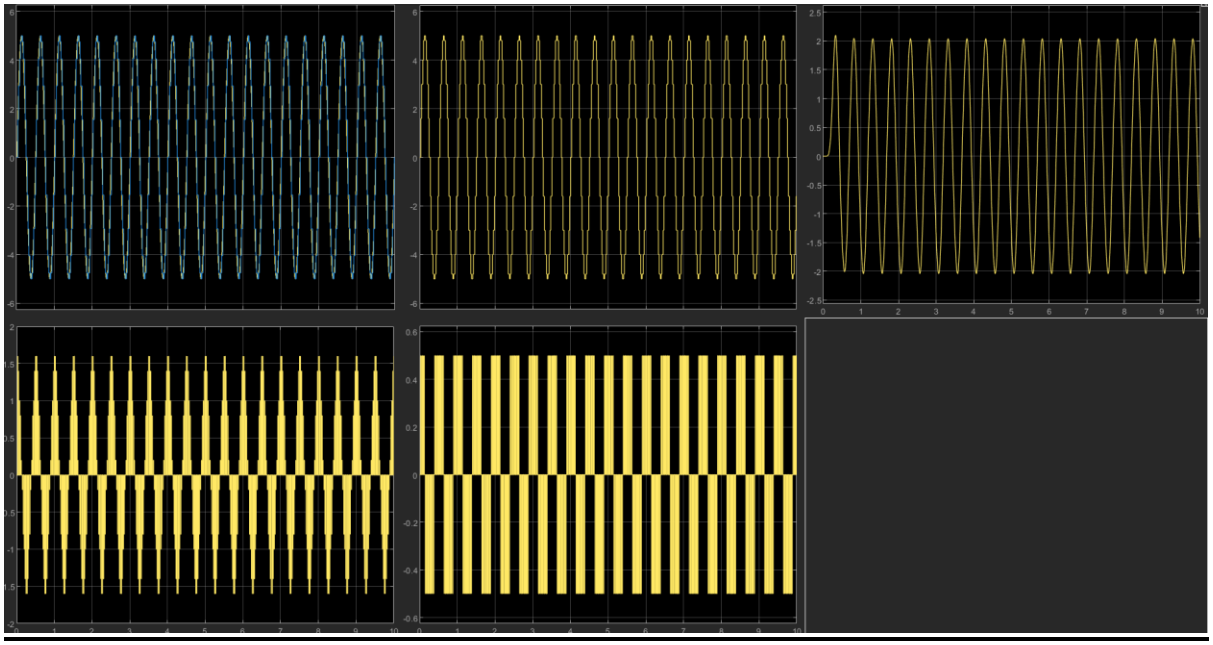
DM ENCODER:



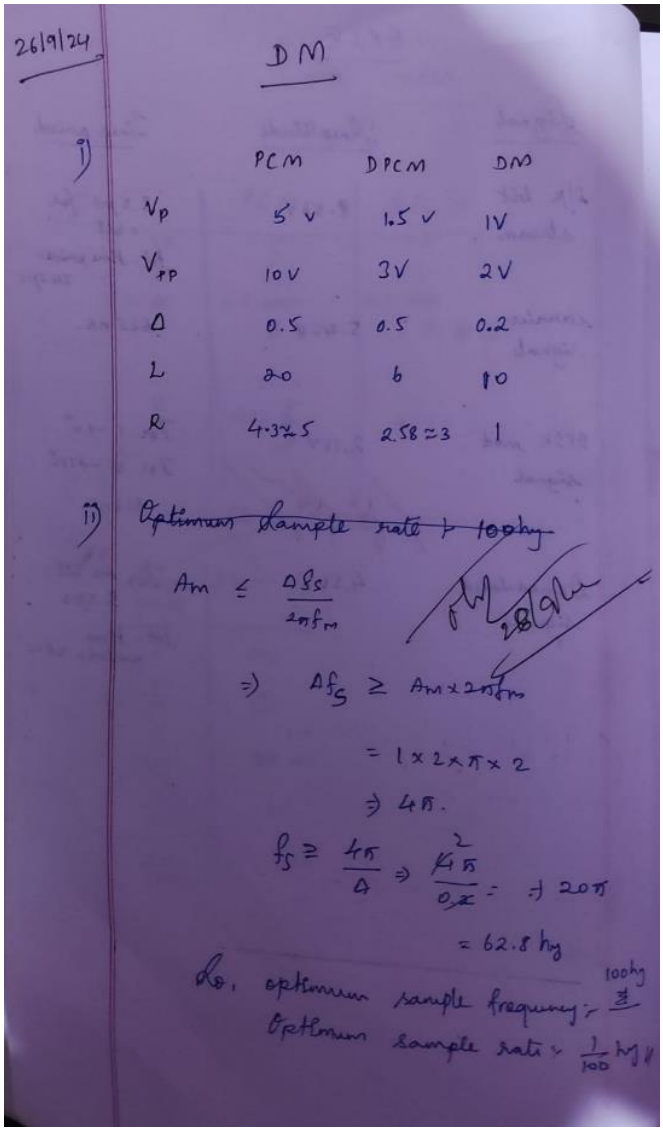
DM ENCODER (WHEN SAMPLE TIME=0):



DM DECODER:



OUTPUT VERIFICATION PROOF:



26/9/24

DM

i)

	PCM	DPCM	DM
V_p	5 V	1.5 V	1V
V_{pp}	10V	3V	2V
Δ	0.5	0.5	0.2
L	20	6	10
R	4.325	2.58 ≈ 3	1

ii) ~~Optimum sample rate → 100hz~~

$A_m \leq \frac{\Delta f_s}{2\pi f_m}$

$\Rightarrow \Delta f_s \geq A_m \times 2\pi f_m$

$= 1 \times 2 \times \pi \times 2$

$\Rightarrow 4\pi.$

$f_s \geq \frac{4\pi}{\Delta} \Rightarrow \frac{4\pi^2}{0.2} = 20\pi$

$= 62.8$ hz

Ans. optimum sample frequency = $\frac{100hz}{\pi}$

Optimum sample rate = $\frac{1}{100}$ hz

INFERENCE:

Thus, the Outputs signal for Delta Modulation and Demodulation were generated. Also, the optimum sample rate was inferred using MATLAB – Simulink with the given Block diagram.

EXP 8: BPSK MODULATION AND DEMODULATION (HARDWARE)

DATE : 12/09/2024

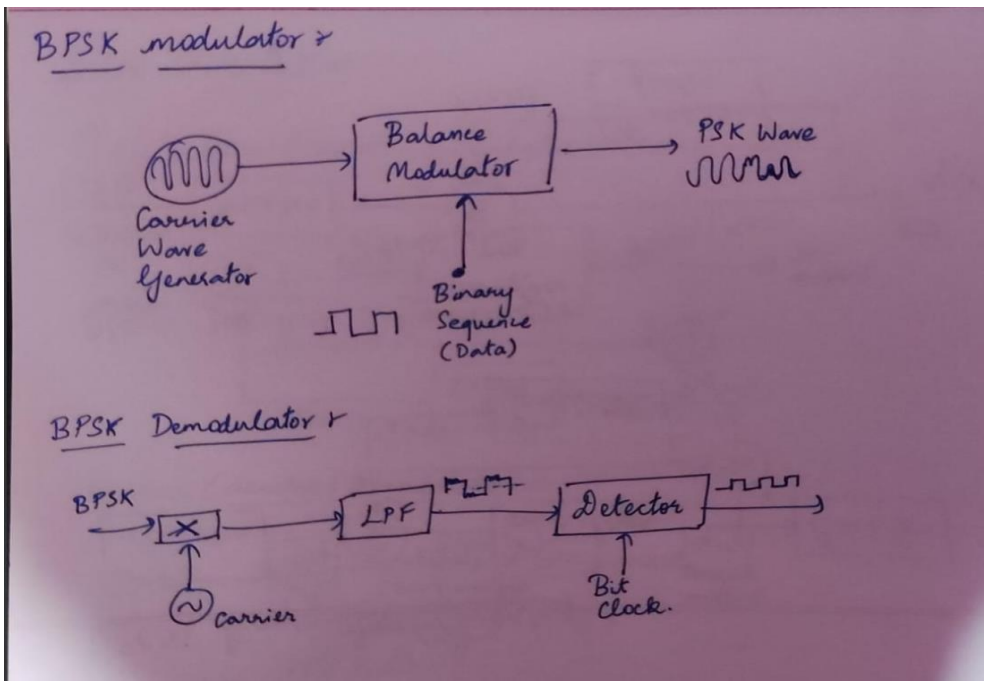
AIM:

To perform BPSKModulation and Demodulation using Data transmitting and Carrier Modulation Transmitter and Receiver Trainer Kit.

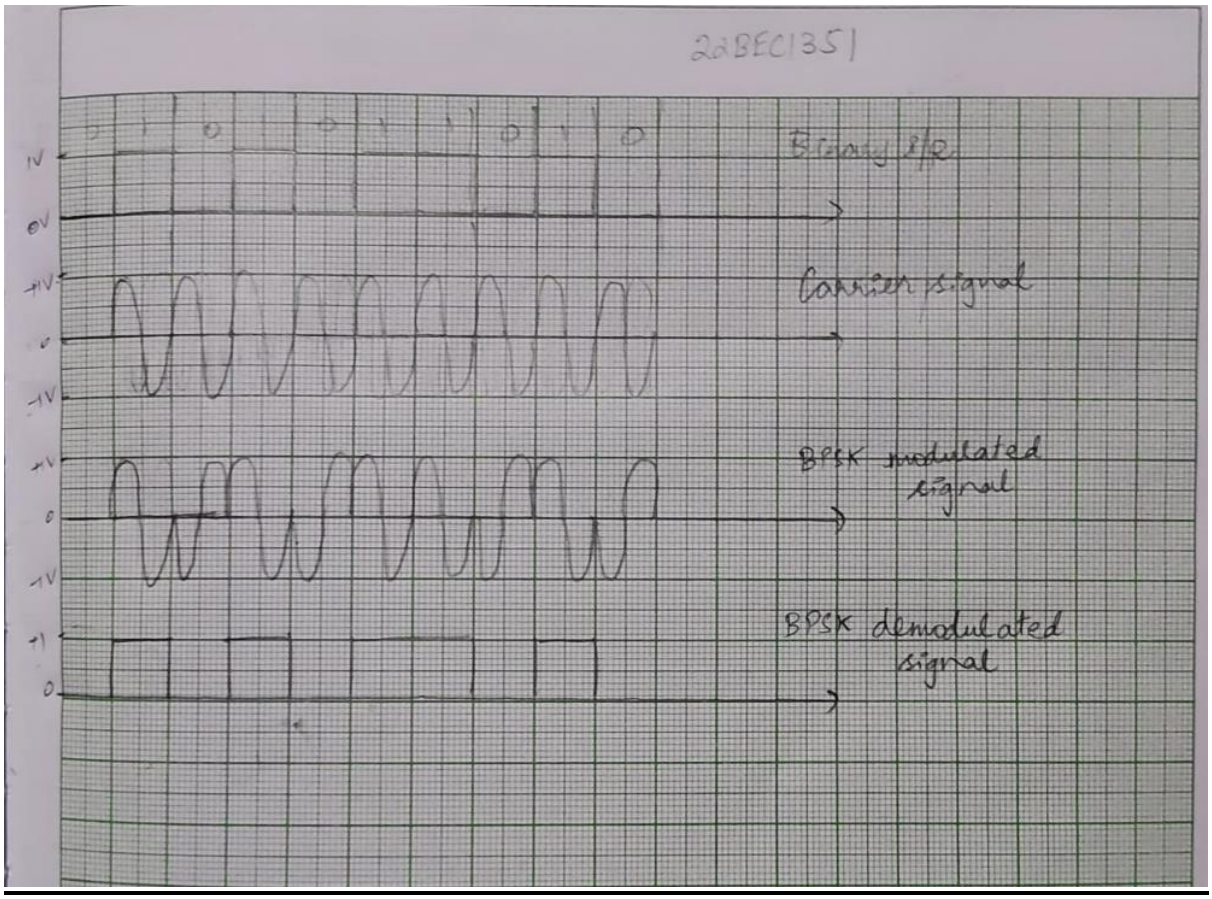
PROCEDURE:

- Make the necessary connections as in the block diagram.
- Give appropriate frequencies in the function generator.
- Turn on the oscilloscope for viewing the required waveforms.
- Note down the following values of required waveforms:
  - Amplitude
  - Time Period
  - Bit Pattern
  - Make a tabular column of the above-mentioned required values.

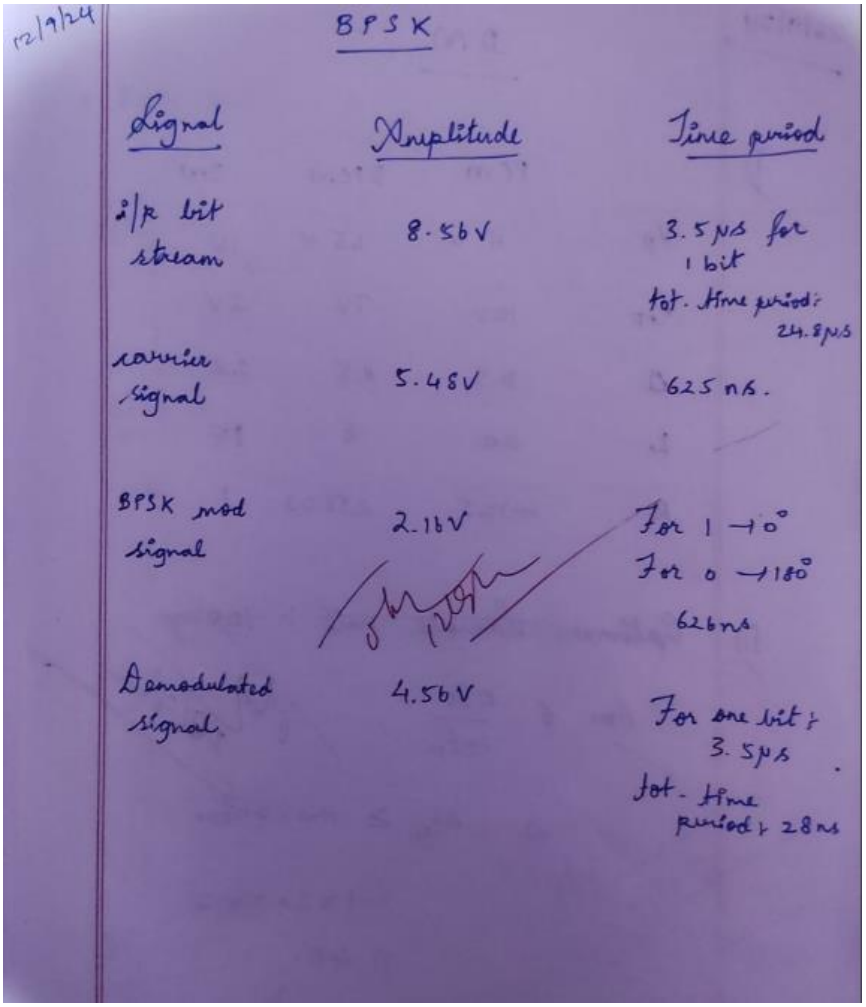
BLOCK DIAGRAM:



GRAPH:



TABULATION & OUTPUT VERFICATION PROOF:



12/9/24

BPSK

Signal	Amplitude	Time period
i/p bit stream	8.56V	3.5μs for 1 bit tot. time period: 24.8μs
carrier signal	5.48V	625 nS.
BPSK mod signal	2.16V	For 1 → 0° For 0 → 180° 626nS
Demodulated signal	4.56V	For one bit: 3.5μs tot. time period: 28ns

INFERENCE:

Hence, the input analog signal, carrier signal, BPSK Modulated signal and BPSK Demodulated signal are plotted and viewed in oscilloscope. Hence, the concept of "BPSK Modulation and Demodulation" is understood.

TASK 5

DATE : 10/10/2024

EXP 9: LINE CODING TECHNIQUES (SOFTWARE)

AIM:

To generate Line Coding Output for different techniques for a data pattern using given MATLAB code.

CODE:

1.Unipolar NRZ:

```
clc
clear all
bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x


%line coding :Uni-polar
for(i=0:length(bits)-1)
      if bits(i+1)==1
           x((i*n)+1:(i+1)*n)=1;
else
      x((i*n)+1:(i+1)*n)=0;
      end
end


subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('NRZ UNI POLAR-LINE CODING')
axis([0 2500 0 2])
```

```
sx=((a^2)/4)*TB*(sinc(x1).^2)+((a^2)/4)*dirac(f); %power spectra UNIPOLAR formula
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title('NRZ UNI POLAR-POWER SPECTRUM-LINE CODING');
```

2.Unipolar RZ:

```
clc
clear all
bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x
%line coding :Uni-polar
for(i=0:length(bits)-1)
     if bits(i+1)==1
          x((i*n)+1:(i+0.5)*n)=1;
          x(((i+0.5)*n)+1:(i+1)*n)=0;
else
    x((i*n)+1:(i+0.5)*n)=0;
    x(((i+0.5)*n)+1:(i+1)*n)=0;
    end
end


subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('RZ UNI POLAR-LINE CODING')
axis([0 2500 0 2])
sx=((a^2)/4)*TB*(sinc(x1).^2)+((a^2)/4)*dirac(f); %power spectra UNIPOLAR formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
```

```
ylabel('Power');
title('RZ UNI POLAR-POWER SPECTRUM-LINE CODING');
```

3. Polar NRZ:

```
clc
clear all

bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x

%line coding -polar
for(i=0:length(bits)-1)
if bits(i+1)==1
x((i*n)+1:(i+1)*n)=1;
else
x((i*n)+1:(i+1)*n)=-1;
end
end

subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('NRZ POLAR-LINE CODING')
axis([0 2500 -1 1])
sx=(a^2)*TB*(sinc(x1).^2); %power spectra POLAR formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title('NRZ POLAR-POWER SPECTRUM-LINE CODING');
```

4. Polar RZ:

```
clc
clear all

bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x
%line coding -polar
for(i=0:length(bits)-1)
if bits(i+1)==1
x((i*n)+1:(i+0.5)*n)=1;
x(((i+0.5)*n)+1:(i+1)*n)=0;
else
x((i*n)+1:(i+0.5)*n)=-1;
x(((i+0.5)*n)+1:(i+1)*n)=0;
end
end
subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('RZ POLAR-LINE CODING')
axis([0 2500 -1 1])
sx=(a^2)*TB*(sinc(x1).^2); %power spectra POLAR formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title('RZ POLAR-POWER SPECTRUM-LINE CODING');
```

5. Bi-Polar AMI:

```
clc
clear all
bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration – Full time of bit sequence
n=200; %bit duration
dt=1/n;
```

```
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x
count=0;
%line coding :BI-polar
for(i=0:length(bits)-1)
if bits(i+1)==1
if mod(count,2)==0
x((i*n)+1:(i+1)*n)=1;
else
x((i*n)+1:(i+1)*n)=-1;
end
count=count+1;
else
x((i*n)+1:(i+1)*n)=0;
end

end
subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('BI-POLAR-LINE CODING - AMI')
axis([0 2500 -2 2])
sx=(a^2)*TB*(sinc(x1)).*(sinc(x1)).*(sin(pi*x1)).*(sin(pi*x1)); %power spectra
BIPOLAR formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title(' BI-POLAR-POWER SPECTRUM-LINE CODING-AMI');
```

6. Bi-Polar Pseusoterenary:

```
clc
clear all
bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration – Full time of bit sequence
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
```

```
f=0:bitrate/100:2*bitrate;
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x
count=0;
%line coding :BI-polar
for(i=0:length(bits)-1)
if bits(i+1)==0
if mod(count,2)==0
x((i*n)+1:(i+1)*n)=1;
else
x((i*n)+1:(i+1)*n)=-1;
end
count=count+1;
else
x((i*n)+1:(i+1)*n)=0;
end

end
subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('BI-POLAR-LINE CODING-PSUEDOTERNARY')
axis([0 2500 -2 2])
sx=(a^2)*TB*(sinc(x1)).*(sinc(x1)).*(sin(pi*x1)).*(sin(pi*x1)); %power spectra
BIPOLAR formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title(' BI-POLAR-POWER SPECTRUM-LINE CODING-PSUEDOTERNARY');
```

7. Manchester:

```
clc
clear all

bitrate=1; %Rb-bitrate
bits=[1 0 0 1 0 1 1 1 0 1 0];
T= length(bits)/bitrate; %Symbol duration
n=200; %bit duration
dt=1/n;
TB=1/bitrate; %1/bitrate value
a=1;
f=0:bitrate/100:2*bitrate;
```

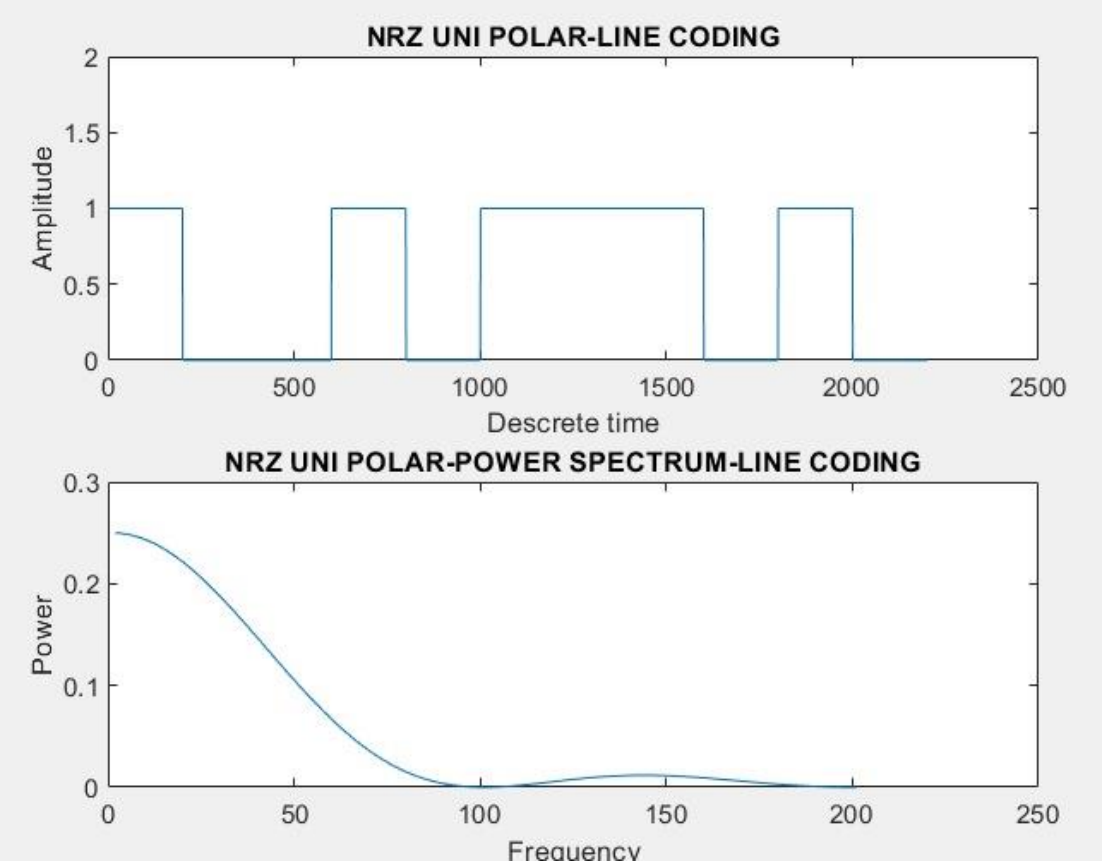
```
x1=f*TB; % power spectra frequency
t=0:dt:T; %t takes the valu 0 to T with innerspace equal to dt
x=zeros(1,length(t)); %initialisation of x
%line coding :MANCHESTER
for(i=0:length(bits)-1)
if bits(i+1)==1
x((i*n)+1:(i+0.5)*n)=1;
x(((i+0.5)*n)+1:(i+1)*n)=-1;
else
x((i*n)+1:(i+0.5)*n)=-1;
x(((i+0.5)*n)+1:(i+1)*n)=1;
end
end
subplot(2,1,1)
plot(x);
xlabel('Descrete time');
ylabel('Amplitude');
title('MANCHESTER-LINE CODING')
axis([0 2500 -2 2])
sx=(a^2)*TB*(sinc(x1/2)).*(sinc(x1/2)).*(sin(pi*x1/2)).*(sin(pi*x1/2));
%power spectra MANCHESTER formila
subplot(2,1,2)
plot(sx);
xlabel('Frequency');
ylabel('Power');
title('MANCHESTER-POWER SPECTRUM-LINE CODING');
```

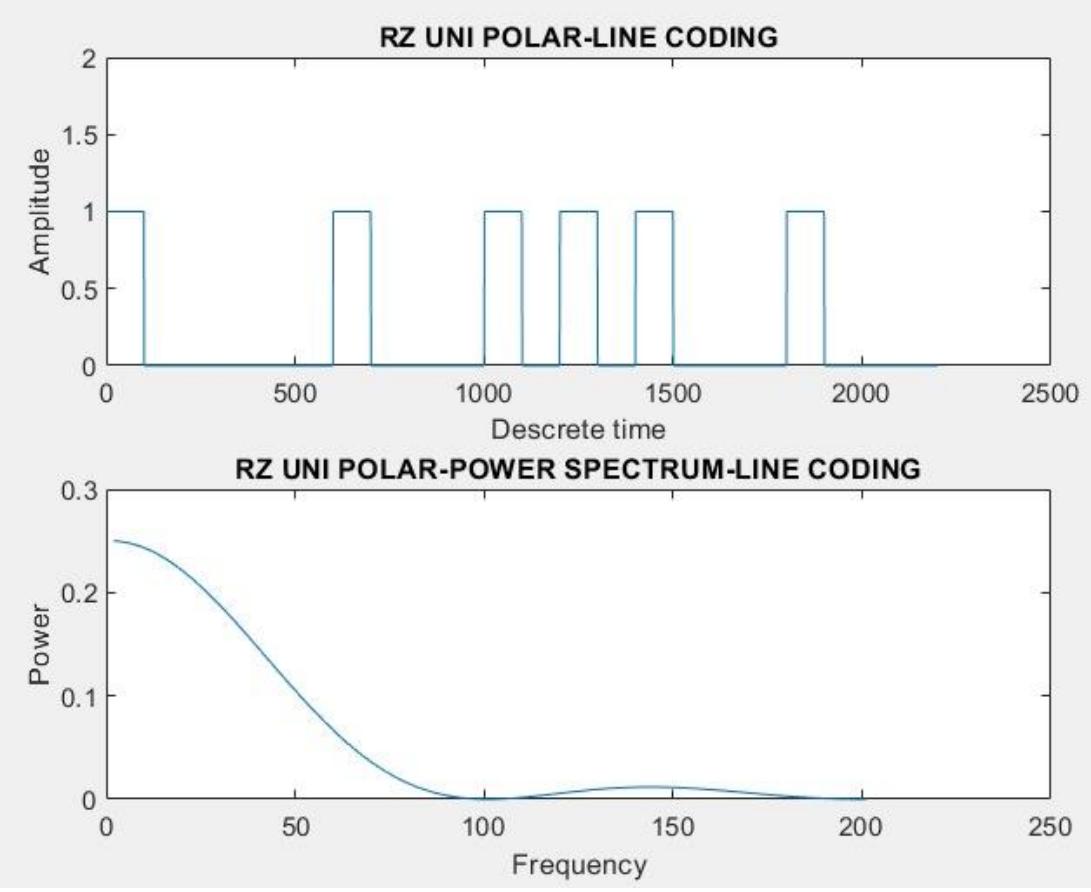
8.Differential Manchester:

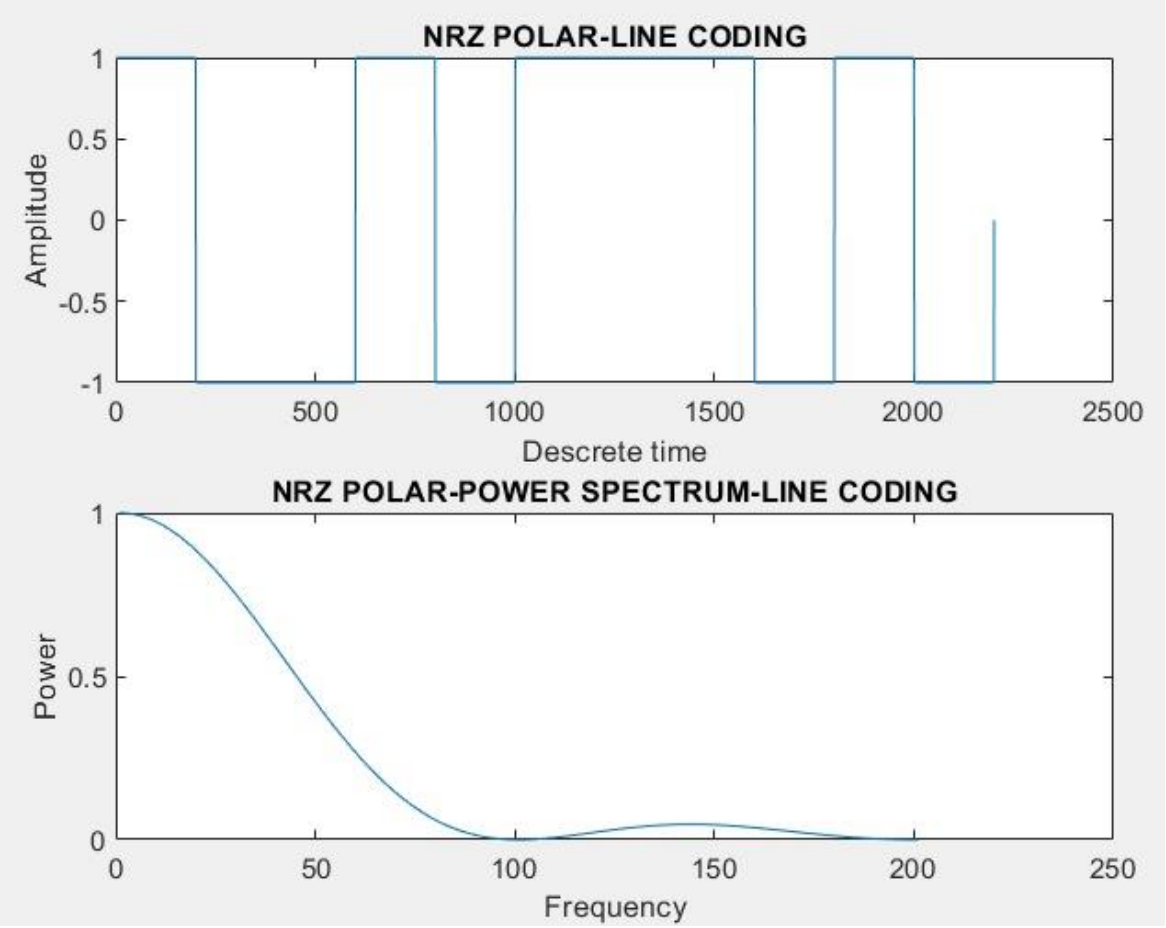
```
clc
clear all
bitrate = 1; % Rb-bitrate
bits = [1 0 0 1 0 1 1 1 0 1 0]; % Input bits
T = length(bits) / bitrate; % Total duration
n = 200; % Bit duration
dt = 1/n; % Time increment
TB = 1 / bitrate; % Bit duration
f = 0:bitrate/100:2*bitrate; % Frequency range for power spectrum
x1 = f * TB; % Power spectrum frequency
t = 0:dt:T; % Time vector
x = zeros(1, length(t)); % Initialize signal array
% Differential Manchester Encoding
prev_signal = 1; % Initial signal level (arbitrary choice, can start with +1 or -1)
```

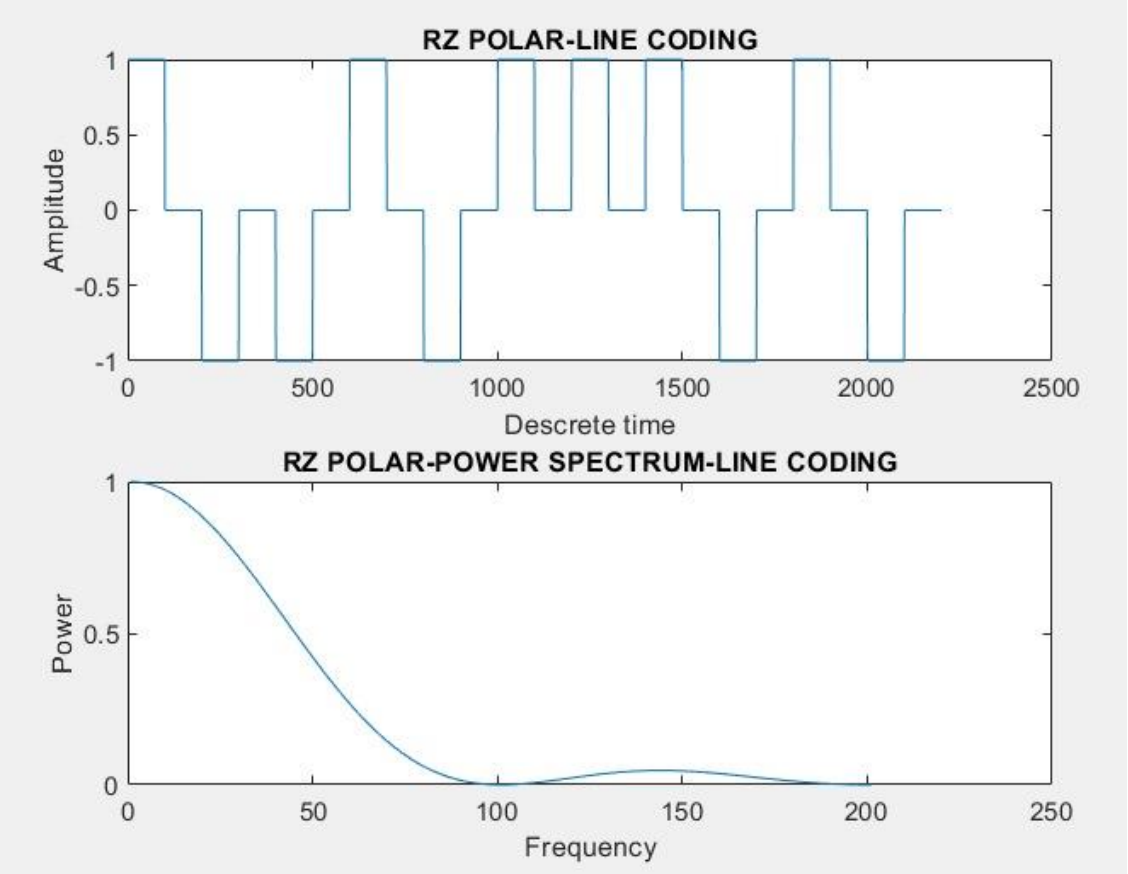
```
for i = 0:length(bits)-1
  mid_index = (i * n) + (0.5 * n);
  % Always transition at the middle of the bit duration
  x((i*n)+1:mid_index) = prev_signal; % First half
  prev_signal = -prev_signal; % Transition for the second half
  x(mid_index+1:(i+1)*n) = prev_signal; % Second half
  % Change the signal level based on the bit value
  if bits(i+1) == 0
    prev_signal = -prev_signal; % Change level if the bit is 0
  end
end
% Plotting the Differential Manchester signal
figure(8)
subplot(2,1,1)
plot(x);
xlabel('Discrete time');
ylabel('Amplitude');
title('DIFFERENTIAL MANCHESTER LINE CODING');
axis([0 2500 -2 2]);
% Power spectrum calculation
sx1 = (1^2) * TB * (sinc(x1 / 2)).^2 .* (sin(pi * x1 / 2)).^2; % Power spectrum
subplot(2,1,2)
plot(sx1);
xlabel('Frequency');
ylabel('Power');
title('DIFFERENTIAL MANCHESTER POWER SPECTRUM');
```

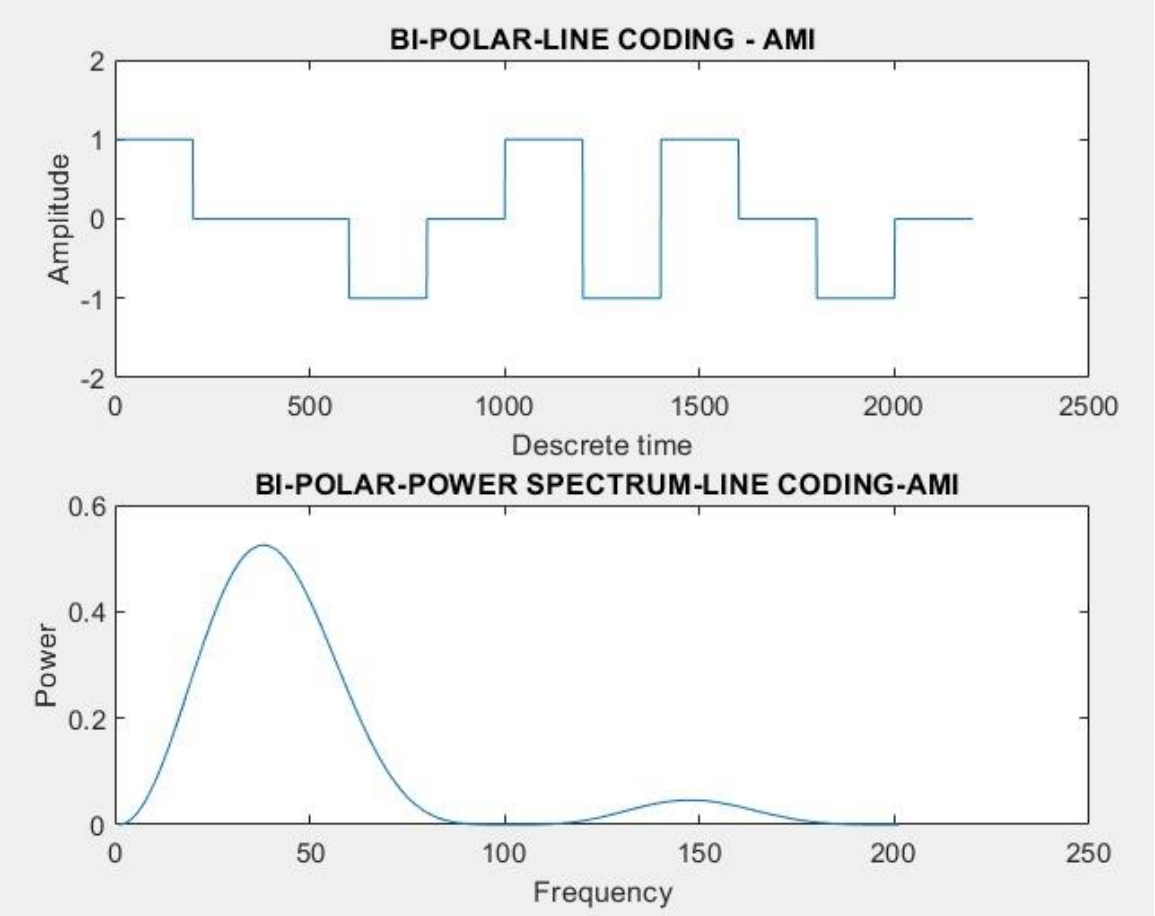
OBSERVATION/SNAPSHOTS OF THE OUTPUT:

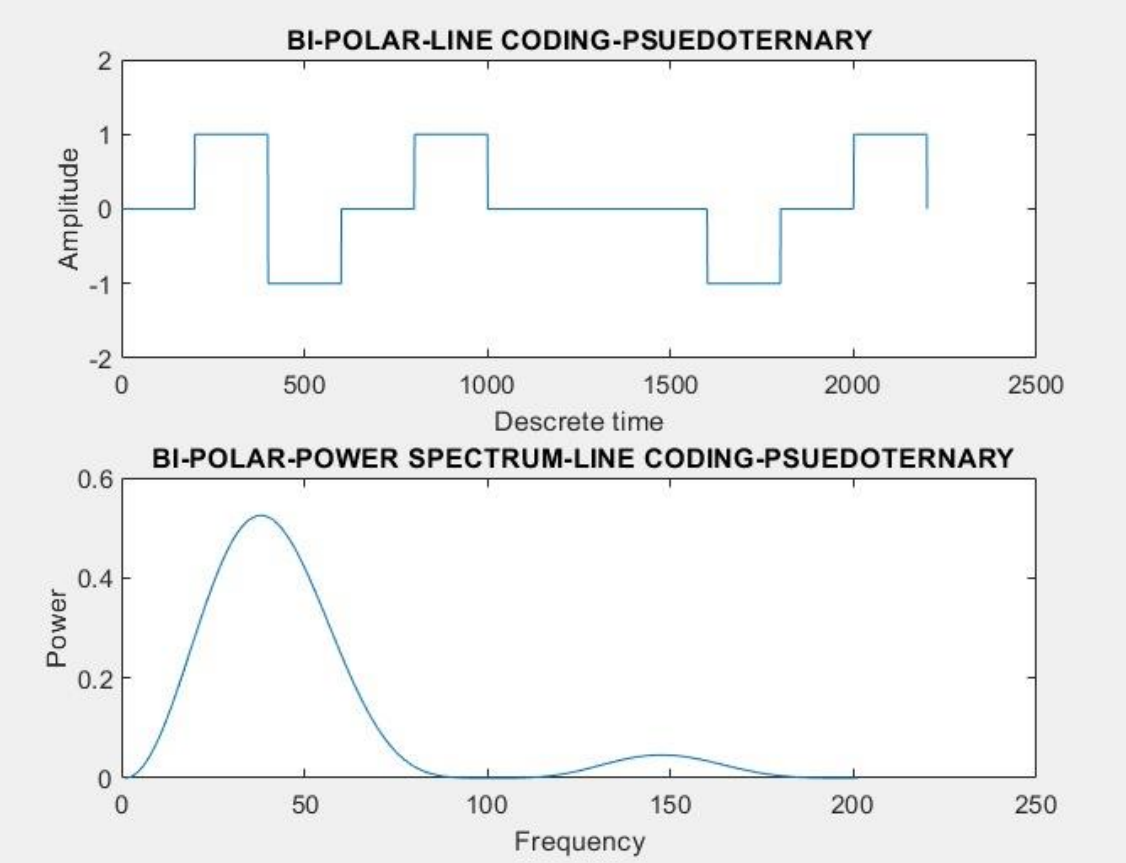


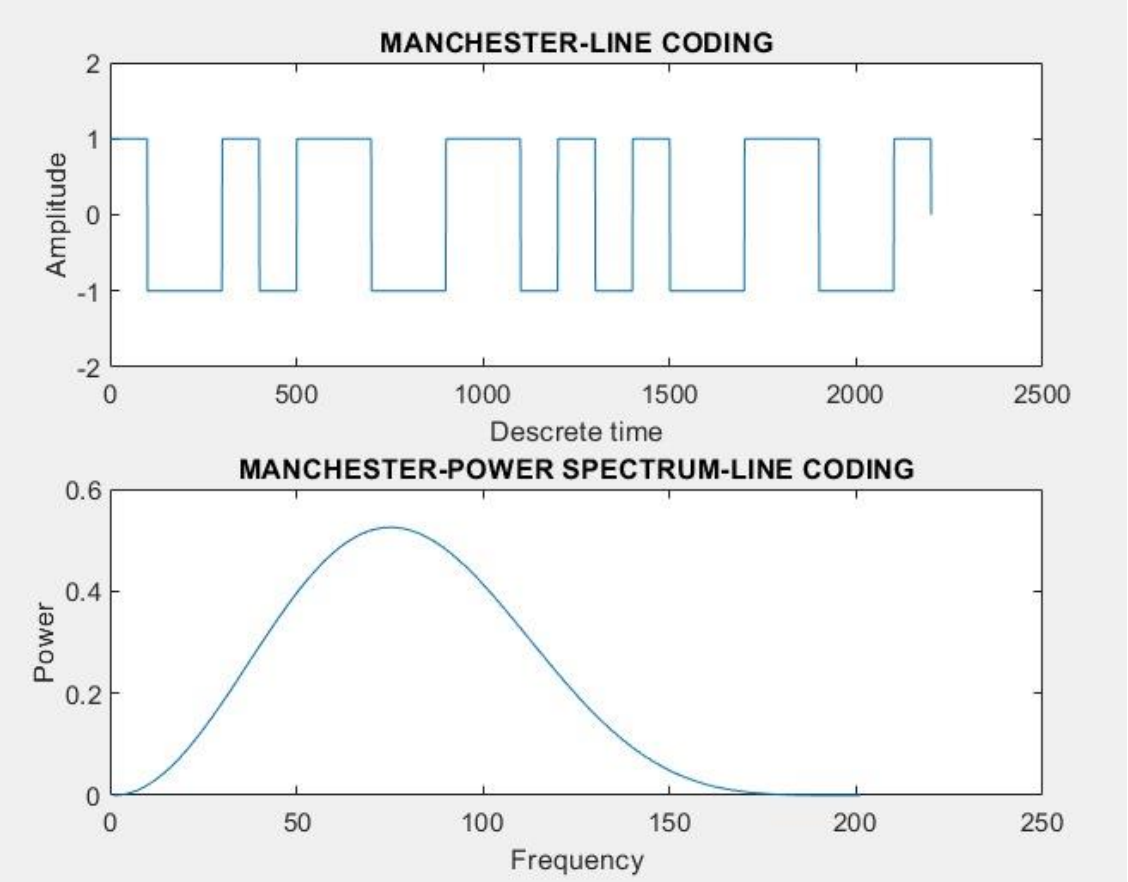


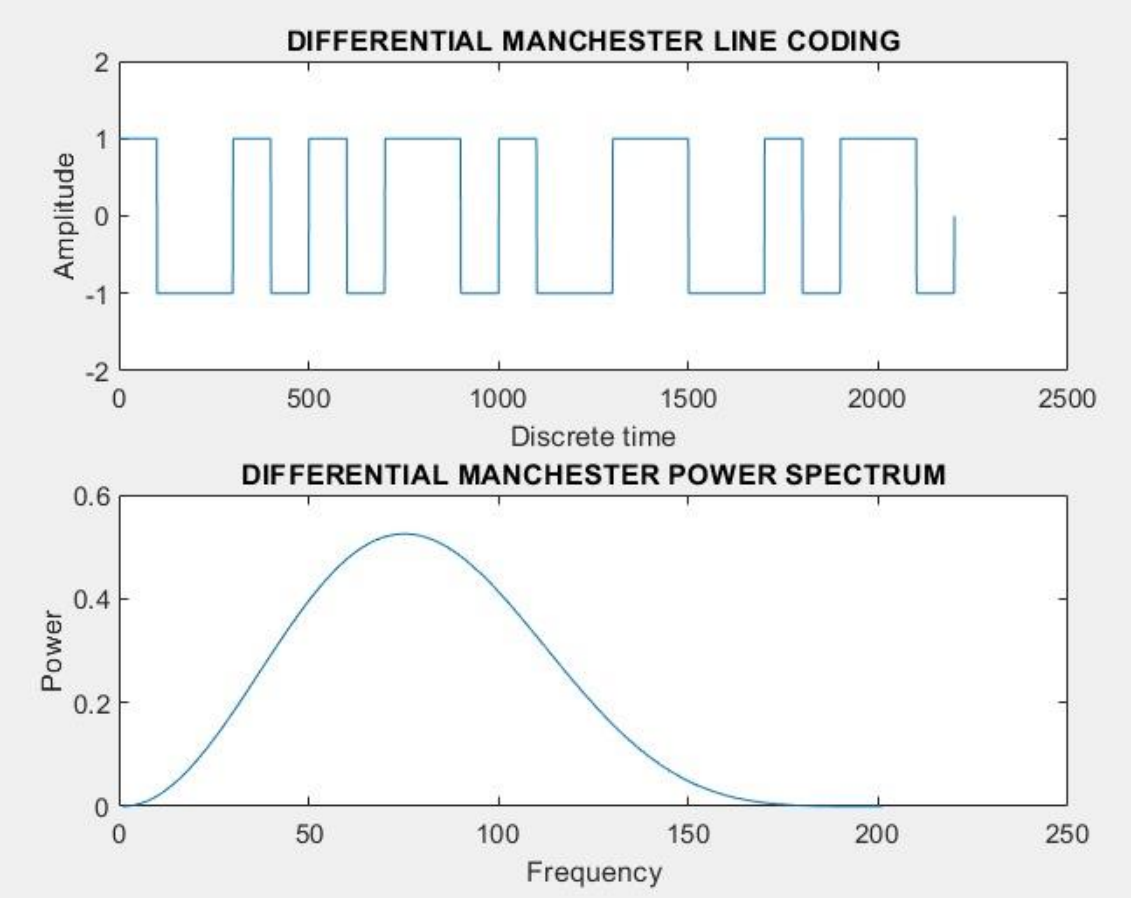

NRZ POLAR-LINE CODING
Amplitude
Descrete time
NRZ POLAR-POWER SPECTRUM-LINE CODING
Power
Frequency


RZ POLAR-LINE CODING
Amplitude
Descrete time
RZ POLAR-POWER SPECTRUM-LINE CODING
Power
Frequency

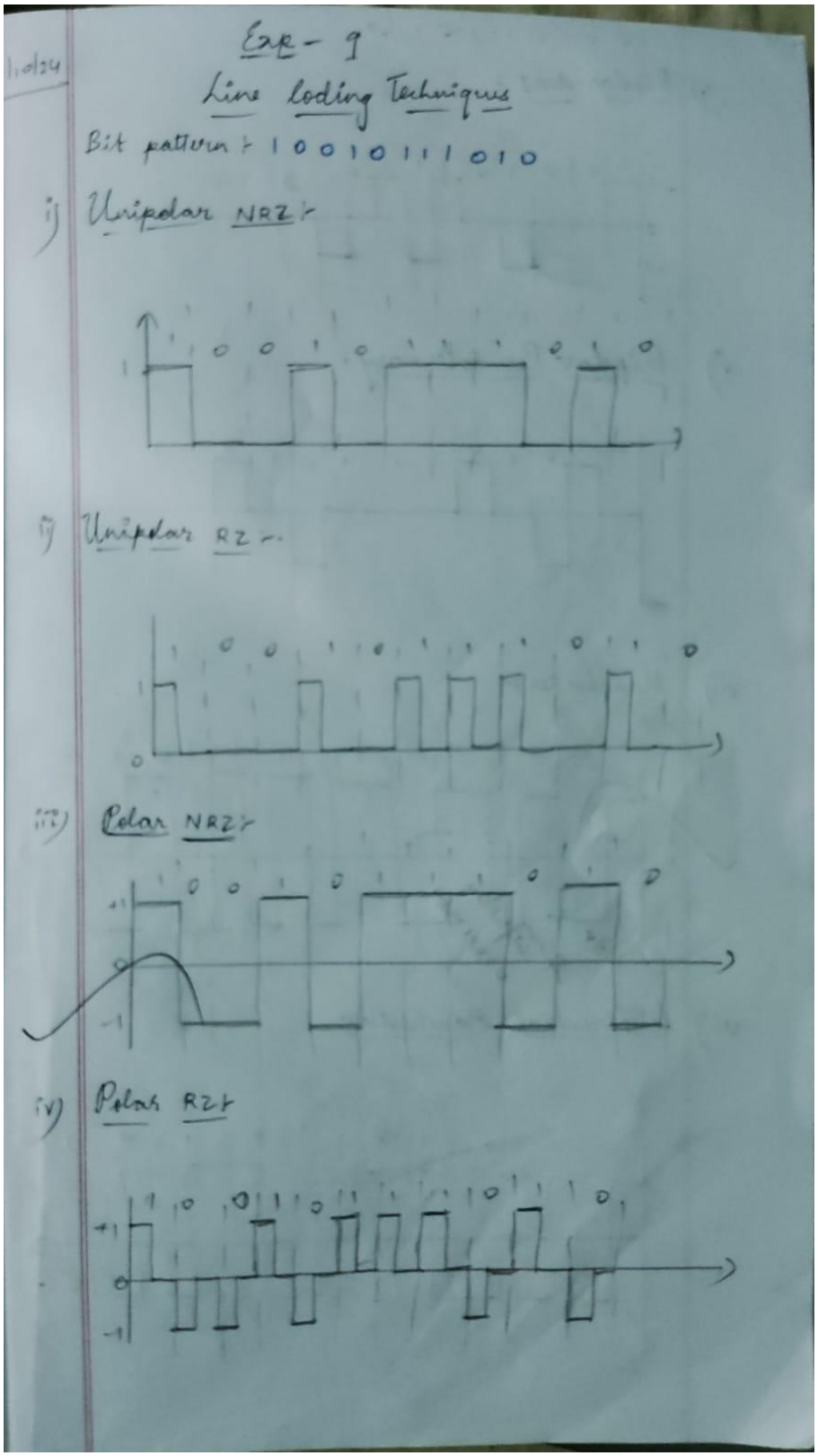

BI-POLAR-LINE CODING - AMI
Amplitude
Descrete time
BI-POLAR-POWER SPECTRUM-LINE CODING-AMI
Power
Frequency

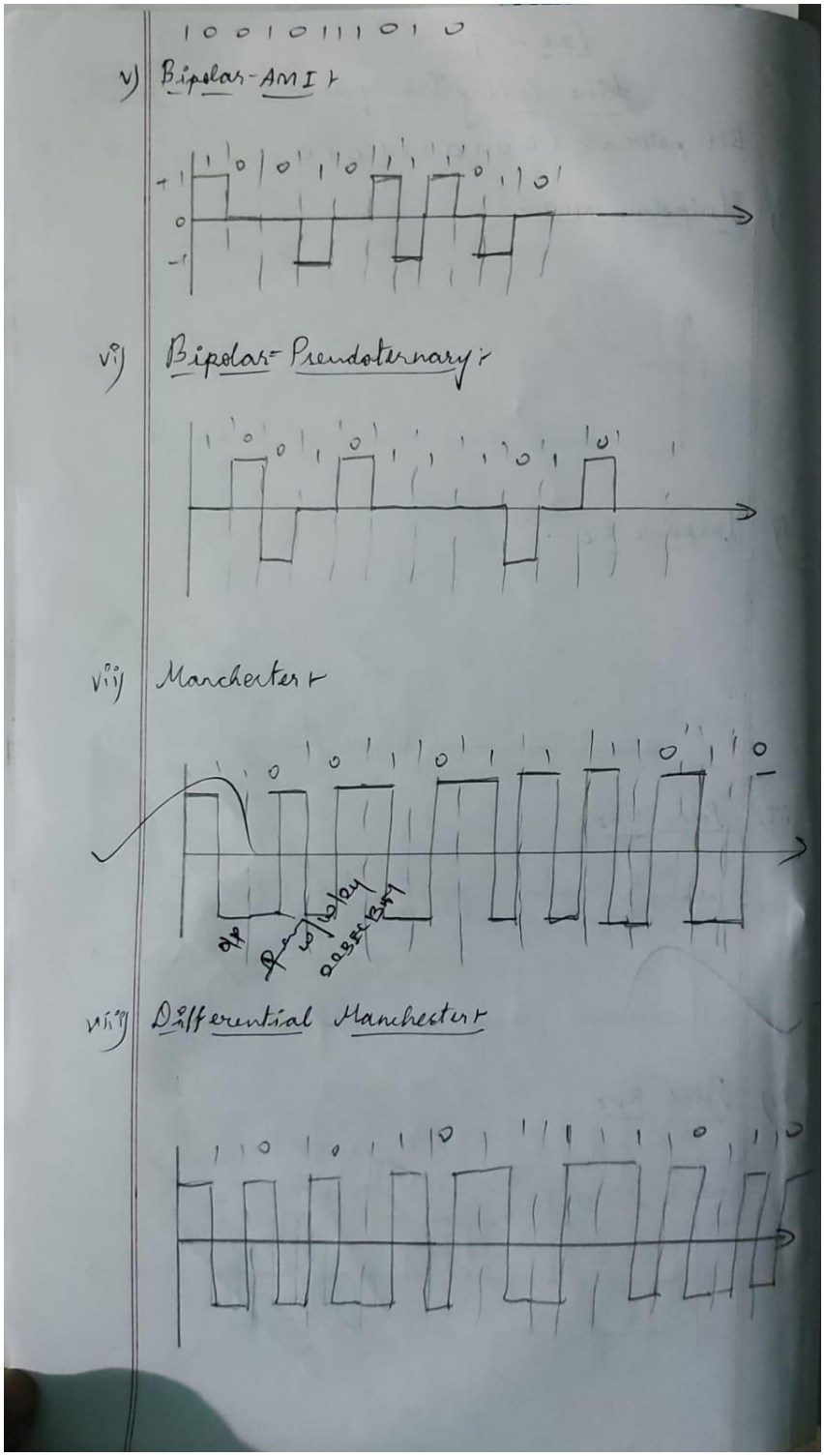

BI-POLAR-LINE CODING-PSUEDOTERNARY
Amplitude
Descrete time
BI-POLAR-POWER SPECTRUM-LINE CODING-PSUEDOTERNARY
Power
Frequency


MANCHESTER-LINE CODING
Amplitude
Descrete time
MANCHESTER-POWER SPECTRUM-LINE CODING
Power
Frequency


DIFFERENTIAL MANCHESTER LINE CODING
Amplitude
Discrete time
DIFFERENTIAL MANCHESTER POWER SPECTRUM
Power
Frequency

OUTPUT VERIFICATION PROOF:





INFERENCE:

Thus, the Outputs signal for different Line coding techniques for the given data pattern is observed and verified using MATLAB code.

EXP 10: LINE CODING TECHNIQUES (HARDWARE)

DATE : 03/10/2024

AIM:

To perform Different Line coding techniques for a data pattern and get the output waveform for respective line coding technique using Data transmitting and Carrier Modulation Transmitter and Receiver Trainer Kit.

PROCEDURE:

In the kit , give the input data pattern to the line coding converter block to get the output for each line coding technique.

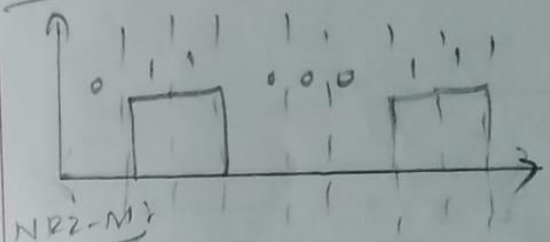
OBSERVATION/GRAPH/OUTPUT VERFICATION PROOF:

Line coding schemes (H.W)

- Data pattern:

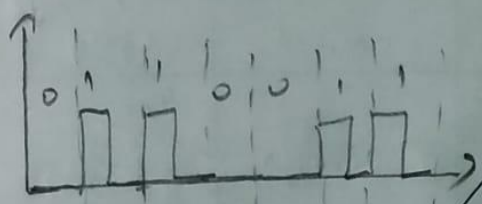
  0 1 1 0 0 0 1 1

NRZ-L:

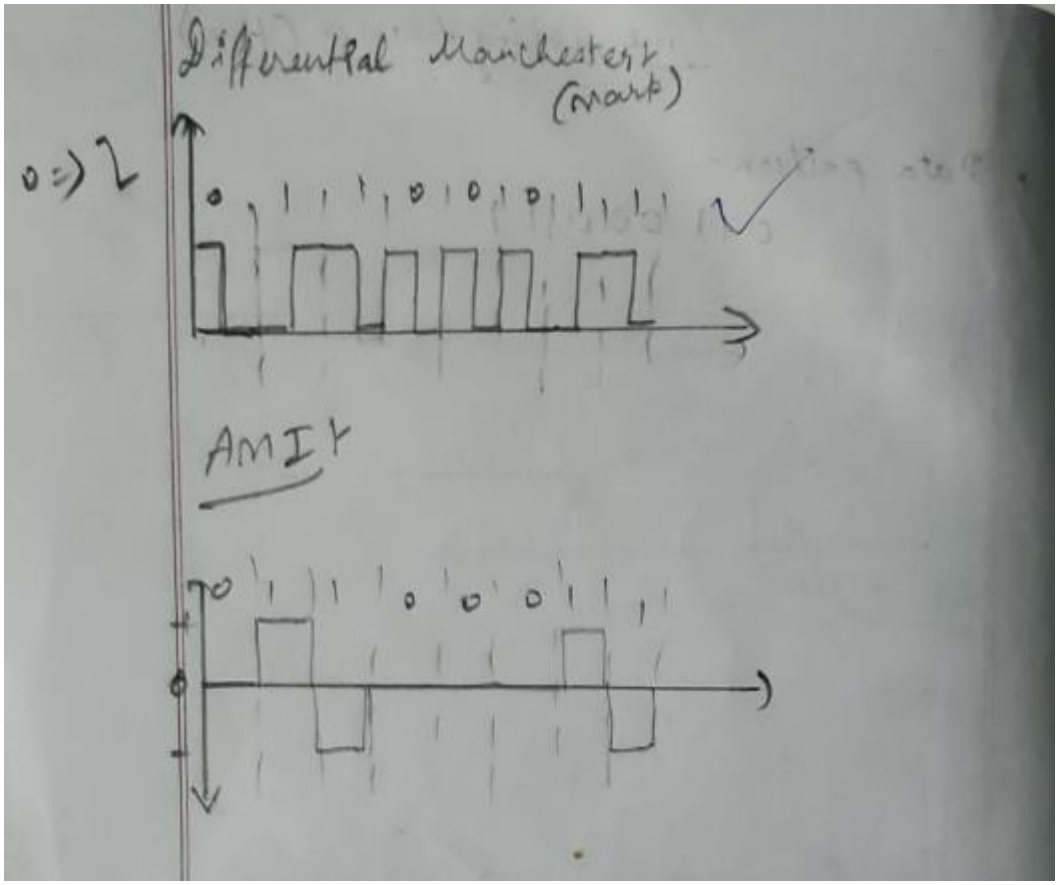


NRZ-M:

RZ:



Manchester:



INFERENCE:

Hence, the input data pattern is transmitted into data signals using different Line coding techniques. Hence, the concept of “Line Coding Techniques” is understood.

EXP 11: TIME DIVISION MULTIPLEXING & DEMULTIPLEXING (HARDWARE)

DATE : 03/10/2024

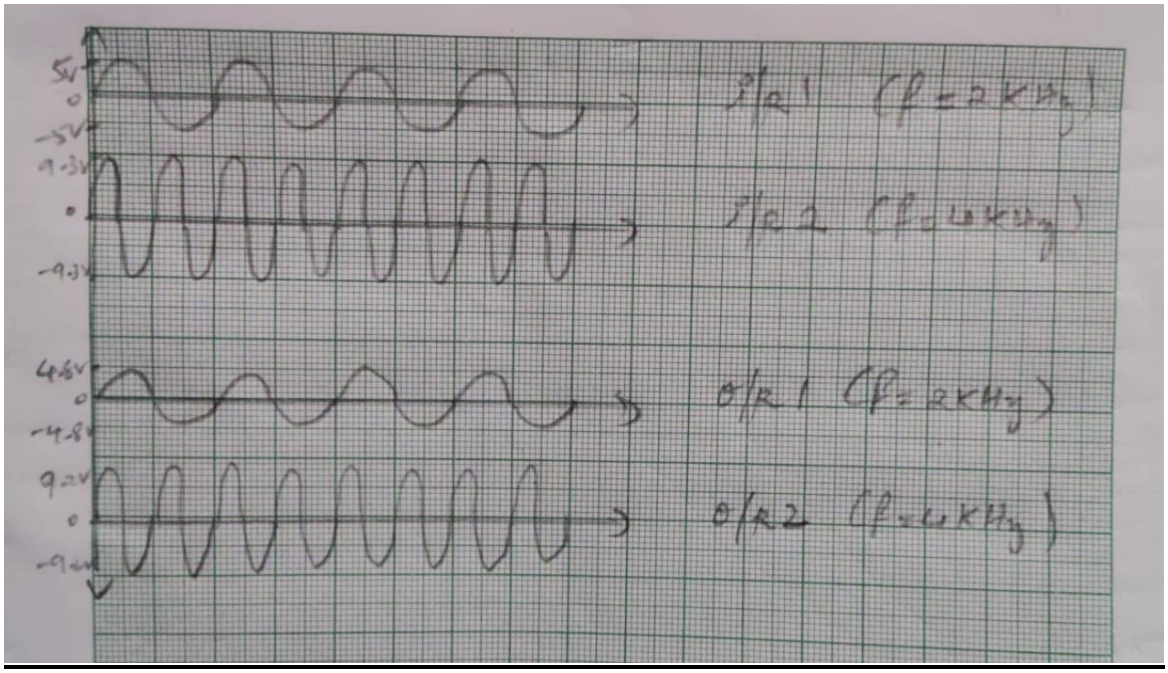
AIM:

To perform Time division multiplexing and demultiplexing for two different sinusoidal signals using Time Multiplexing kit.

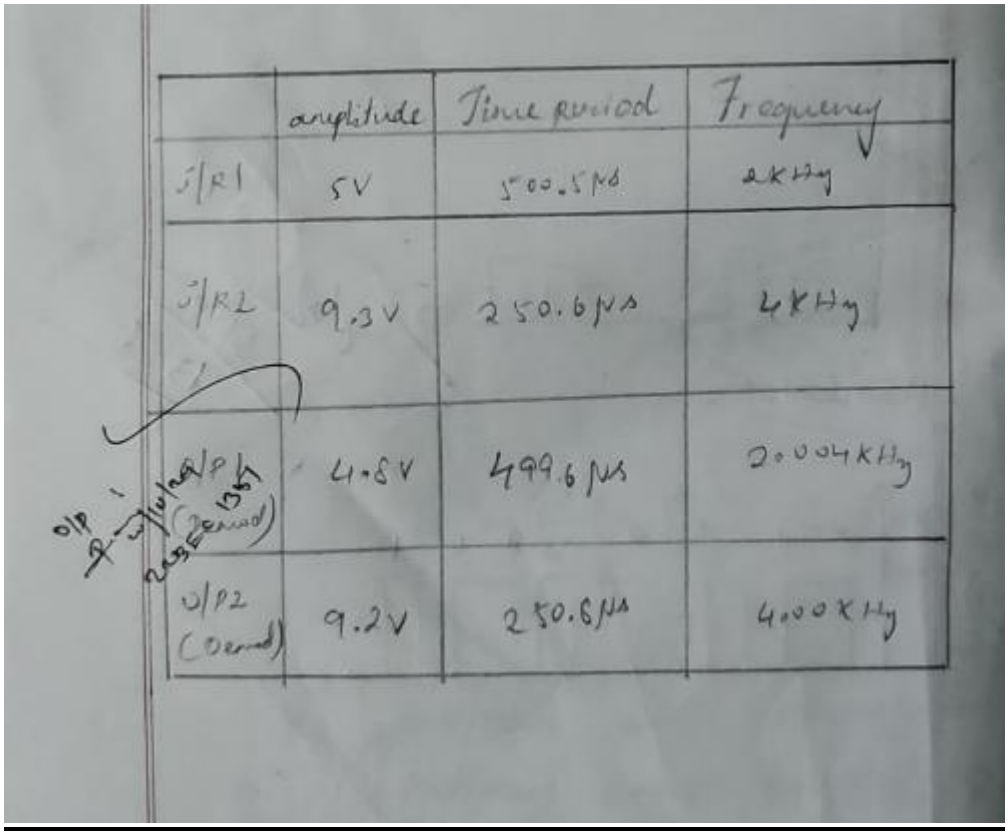
PROCEDURE:

1. Give the two different sinusoidal signal to the time multiplexing unit in the kit.
2. Note down the amplitude, time period and frequency for the two inputs.
3. Now get the output from the time multiplexing units and note down the respective values.
4. Give the output of time multiplexing to the input of time demultiplexing units and get the output for the two sinusoidal signals.

OBSERVATION, GRAPH:



TABULATION, OUTPUT VERFICATION PROOF:



	amplitude	Time period	Frequency
I/R1	5V	500.5μs	2KHz
I/R2	9.3V	250.6μs	4KHz
O/P1 (Demod)	4.8V	499.6μs	2.004KHz
O/P2 (Demod)	9.2V	250.6μs	4.00KHz

INFERENCE:

Hence, the input two sinusoidal signals are multiplexed and then demultiplexed and the out put waveform were observed. Hence, the concept of “Time Division Multiplexing and Demultiplexing” is understood.