EXP NO.1	DESIGN AND IMPLEMENTATION OF CYCLIC CODER AND DECODER

CYCLIC ENCODER

```
n = input('Enter the value of n: ');
k = input('Enter the value of k: ');
g = input('Enter the co-efficients of g(x): ');
G = zeros(k, n);
for i = 1:k
    % generate x^n-i
    polyx = [zeros(1, i-1) 1 zeros(1, n-i)];
    % divide x^n-i by g(x)
    [q, r] = deconv(polyx, g);
    r = mod(r, 2);

    G(i, :) = r;
    G(i, i) = 1;
end

m = input('Enter the message bits: ');
codeword = m * G;
codeword = mod(codeword, 2);

disp('Generated Codeword:');
disp(codeword);
```

```
Enter the value of n: 7
Enter the value of k: 4
Enter the co-efficients of g(x): [1 0 1 1]
Enter the message bits: [1 1 0 1]
Generated Codeword:
     1     1     0     1     0     0     1
```

CYCLIC DECODER

```
n = input('Enter the value of n: ');
k = input('Enter the value of k: ');
g = input('Enter the co-efficients of g(x): ');
rc = input('Enter the received codeword: ');
e = zeros(n, n+3);
for i = 1:n
```

```matlab
    % generate x^n
    polyx = [zeros(1,i-1) 1 zeros(1,n-i)];
    % divide x^n-i by g(x)
    [q, r] = deconv(polyx, g);
    r = mod(r, 2);
    e(i, 4:n+3) = r;
    e(i, i) = 1;
end

[q1, r1] = deconv(rc, g);
syndrome = r1(5:7);
error = -1;
for i = 1:n
    if syndrome == e(i, n+1:n+3)
        error = i;
        break;
    end
end

if error == -1
    disp('No error');
else
    disp('Error at position:');
    disp(error);
    rc(error) = mod(rc(error) + 1, 2);
    disp('Corrected code:');
    disp(rc);
end
```

```
Enter the value of n: 7
Enter the value of k: 4
Enter the co-efficients of g(x): [1 0 1 1]
Enter the received codeword: [1 0 0 1 0 0 0]
Error at position:
     3

Corrected code:
     1     0     1     1     0     0     0
```

EXP NO.2	DESIGN AND IMPLEMENTATION LINEAR BLOCK CODER AND DECODER

LINEAR BLOCK ENCODER

```
n = input('Enter the value of n: ');
k = input('Enter the value of k: ');
P = input('Enter the parity matrix: ');
m = input('Enter the message bits: ');

% Create the generator matrix G = [I | P]
G = [eye(k), P];  % eye(k) generates a kxk identity matrix, and P is
the parity matrix

% Calculate the codeword
codeword = m * G;

% Apply modulo 2 to the result to ensure binary values
codeword = mod(codeword, 2);

disp('The codeword is:');
disp(codeword);
```

```
Enter the value of n: 6
Enter the value of k: 3
Enter the parity matrix: [1 1 0; 0 1 1 ; 1 1 1]
Enter the message bits: [1 1 1]
The codeword is:
     1     1     1     0     1     0
```

LINEAR BLOCK DECODER

```
n = input('Enter the value of n: ');
k = input('Enter the value of k: ');
P = input('Enter the parity matrix: ');
r = input('Enter the received codeword: ');

% Construct the parity check matrix H = [P' | I]
H = [P', eye(n-k)];
Ht = H';  % Transpose of H

% Syndrome calculation
syndrome = r * Ht;
syndrome = mod(syndrome, 2);  % Modulo 2 to get binary values
```

```matlab
error = -1;  % Initialize error position as -1 (no error by default)

% Loop through to find the position of the error
for i = 1:n
    if syndrome == Ht(i, :)  % Check if syndrome matches a row of H'
        error = i;
        break;
    end
end

% Check if error detected
if error == -1
    disp('No error');
else
    disp('Error at position:');
    disp(error);

    % Correct the error by flipping the bit
    r(error) = mod(r(error) + 1, 2);

    disp('Corrected code:');
    disp(r);
end
```

```
Enter the value of n: 6
Enter the value of k: 3
Enter the parity matrix: [1 1 0 ; 0 1 1 ; 1 1 1]
Enter the received codeword: [0 1 0 1 1 1]
Error at position:
     4

Corrected code:
     0     1     0     0     1     1
```

EXP NO.3	DESIGN AND IMPLEMENTATION OF CONVOLUTIONAL CODER AND DECODER

CONVOLUTIONAL ENCODER

```
n = input('Enter the value of n: ');
k = input('Enter the value of k: ');
L = input('Enter the value of L: ');
m = input('Enter the message bits: ');
g1 = input('Enter the generator polynomial g1: ');
g2 = input('Enter the generator polynomial g2: ');

shiftreg = zeros(1, L);
codeword = [];

% Append zeros to complete the cycle
m = [m zeros(1, L-1)];

for i = 1:length(m)
    shiftreg = [m(i) shiftreg(1:end-1)];
    c1 = mod(sum(shiftreg .* g1), 2);
    c2 = mod(sum(shiftreg .* g2), 2);
    codeword = [codeword c1 c2];
end

disp('Codeword:');
disp(codeword);
```

```
Enter the value of n: 2
Enter the value of k: 1
Enter the value of L: 3
Enter the message bits: [1 0 0 1 1]
Enter the generator polynomial g1: [1 1 1]
Enter the generator polynomial g2: [1 0 1]
Codeword:
     1     1     1     0     1     1     1     1     0     1     0     1     1     1
```

CONVOLUTIONAL DECODER

```
clc;
close all;
clear vars;
m=input('Enter the message bits');
m1=[m,0,0];
s1=0;
s2=0;
s3=0;
```

```
u=[];
l=4;
k=6;
for i=m1
s3=s2;
s2=s1;
s1=i;
u(end+1)=bitxor(bitxor(s1,s2),s3);
u(end+1)=bitxor(s1,s3);
end
disp('The Encoded Code Word is: ');
disp(u)
%creating ttrellis diagram
trellis=poly2trellis(3,[6,7]);
%Using Viterbi Decoder
decoded_msg=vitdec(u,trellis,4,'trunc','hard');
disp('Decoded using inbuilt functions');
disp(decoded_msg(1:4));
for i=1:k
if(i==4)
u(i)=~u(i);
end
end
disp('The received code word with one bit error is : ');
disp(u);
%Path metric and branch metric calculation
path_metric_1(1)=0;
path_metric_2(1)=1000;
path_metric_3(1)=1000;
path_metric_4(1)=1000;
u=[u,0,0,0,0]
for n=1:l
bm11=sum(abs([u(2*n-1),u(2*n)]-[0,0]));
bm13=sum(abs([u(2*n-1),u(2*n)]-[1,1]));
bm21=sum(abs([u(2*n-1),u(2*n)]-[1,1]));
bm23=sum(abs([u(2*n-1),u(2*n)]-[0,0]));
bm32=sum(abs([u(2*n-1),u(2*n)]-[1,0]));
bm34=sum(abs([u(2*n-1),u(2*n)]-[0,1]));
bm42=sum(abs([u(2*n-1),u(2*n)]-[0,1]));
bm44=sum(abs([u(2*n-1),u(2*n)]-[1,0]));
pm1_1=path_metric_1(n)+bm11;
pm1_2=path_metric_2(n)+bm21;
pm2_1=path_metric_3(n)+bm32;
pm2_2=path_metric_4(n)+bm42;
pm3_1=path_metric_1(n)+bm13;
pm3_2=path_metric_2(n)+bm23;
pm4_1=path_metric_3(n)+bm34;
pm4_2=path_metric_4(n)+bm44;
if pm1_1<=pm1_2
```

```
path_metric_1(n+1)=pm1_1;
tb_path(1,n)=0;
else
path_metric_1(n+1)=pm1_2;
tb_path(1,n)=1;
end
if pm2_1<=pm2_2
path_metric_2(n+1)=pm2_1;
tb_path(2,n)=0;
else
path_metric_2(n+1)=pm2_2;
tb_path(2,n)=1;
end
if pm3_1<=pm3_2
path_metric_3(n+1)=pm3_1;
tb_path(3,n)=0;
else
path_metric_3(n+1)=pm3_2;
tb_path(3,n)=1;
end
if pm4_1<=pm4_2
path_metric_4(n+1)=pm4_1;
tb_path(4,n)=0;
else
path_metric_4(n+1)=pm4_2;
tb_path(4,n)=1;
end
end
[last_pm,last_state]=min([path_metric_1(n+1),path_metric_2(n+1),path
_metric_3(n+1),path_metric_4(n+1)]);
m=last_state;
for n=l:-1:1
if(m==1)
if tb_path(m,n)==0
decoded(n)=0;
m=1;
elseif(tb_path(m,n)==1)
decoded(n)=0;
m=2;
end
elseif(m==2)
if tb_path(m,n)==0
decoded(n)=0;
m=3;
elseif(tb_path(m,n)==1)
decoded(n)=0;
m=4;
end
elseif(m==3)
```

```
if tb_path(m,n)==0
decoded(n)=1;
m=1;
elseif(tb_path(m,n)==1)
decoded(n)=1;
m=2;
end
elseif(m==4)
if tb_path(m,n)==0
decoded(n)=1;
m=3;
elseif(tb_path(m,n)==1)
decoded(n)=1;
m=4;
end
end
end
disp('Decoded without using built in functions ');
disp('The corrected dataword is: ');
disp(decoded);
```

```
Enter the message bits[1 0 1 0]
The Encoded Code Word is:
     1     1     1     0     0     0     1     0     1     1     0     0

Decoded using inbuilt functions
     1     0     0     0

The received code word with one bit error is :
     1     1     1     1     0     0     1     0     1     1     0     0


u =

     1     1     1     1     0     0     1     0     1     1     0     0     0     0     0     0

Decoded without using built in functions
The corrected dataword is:
     1     0     1     0
```

EXP NO.4	POWER SPECTRAL DENSITY OF DIFFERENT TYPES OF LINE CODES

```
% Input Bit Sequence
bits = input('Enter bit sequence : ', 's');
bitrate = 1;

% Unipolar NRZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));

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

figure(2)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Unipolar NRZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Manchester Encoding
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));

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
```

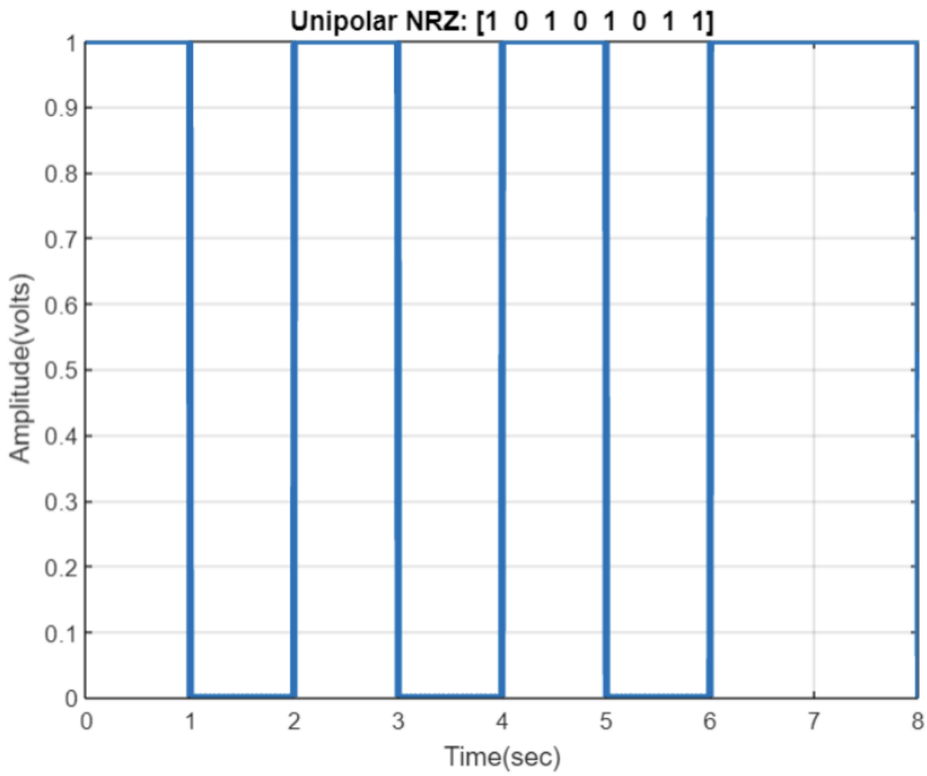
```
figure(3)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Manchester: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Unipolar RZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));

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+1)*n) = 0;
    end
end

figure(4)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Unipolar RZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Polar RZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));

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
```

```
figure(5)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Polar RZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Polar NRZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));

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

figure(6)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Polar NRZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Bipolar NRZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));
p = 1;

for i = 0:length(bits)-1
    if bits(i+1) == '1'
        x(i*n+1:(i+1)*n) = p;
        p = -1*p;
    else
        x(i*n+1:(i+1)*n) = 0;
    end
end

figure(7)
```

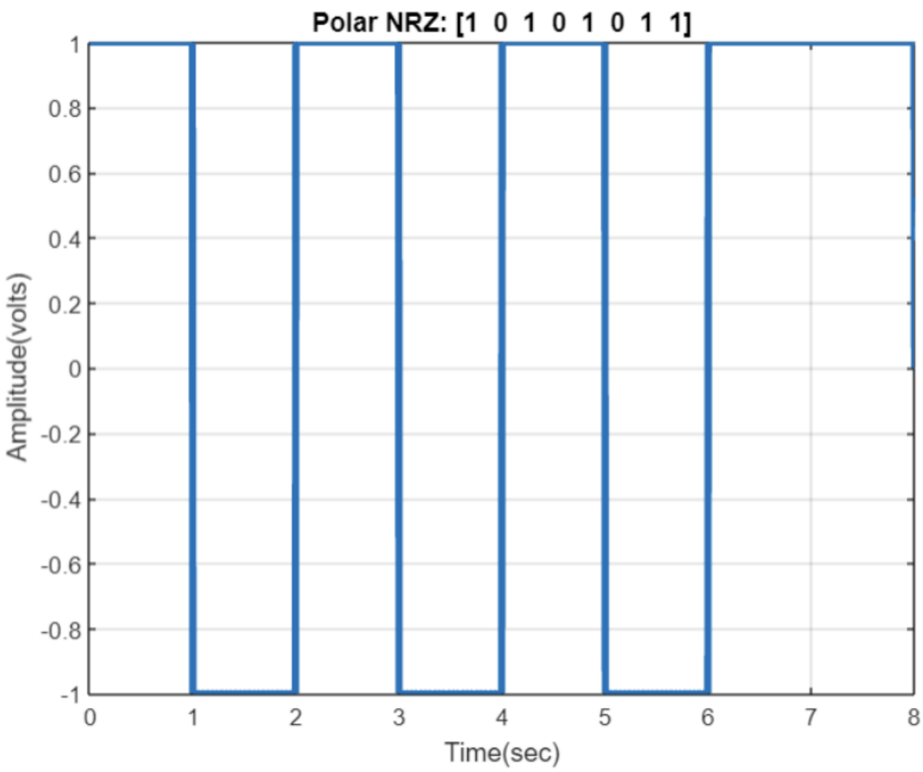
```matlab
plot(t, x, 'LineWidth', 3);
grid on;
title(['Bipolar NRZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Bipolar RZ
T = length(bits)/bitrate;
n = 200;
N = n*length(bits);
dt = T/N;
t = 0:dt:T;
x = zeros(1, length(t));
p = 1;

for i = 0:length(bits)-1
    if bits(i+1) == '1'
        x(i*n+1:(i+0.5)*n) = p;
        x((i+0.5)*n+1:(i+1)*n) = 0;
        p = -1*p;
    else
        x(i*n+1:(i+1)*n) = 0;
    end
end

figure(8)
plot(t, x, 'LineWidth', 3);
grid on;
title(['Bipolar RZ: [' num2str(bits) ']']);
xlabel('Time (sec)')
ylabel('Amplitude (volts)')

% Power Spectral Density (PSD)
Rb = 1;
Tb = 1/Rb;
f = 0:0.05*Rb:2*Rb;
x = f*Tb;

P = Tb*(sinc(x).*sinc(x)); % Polar
P1 = 0.5*Tb*(sinc(x).*sinc(x)) + 0.5 * dirac(f); % Unipolar
P2 = Tb*(sinc(x/2)).^2 .* (sin(pi*x/2)).^2; % Manchester
P3 = Tb*(sinc(x/2)).^2 .* (sin(pi*x)).^2; % Bipolar

figure(9)
plot(f, P, 'r')
hold on
plot(f, P1, 'g')
plot(f, P2, 'b')
plot(f, P3, 'm')
```

```
grid on;
box on;
xlabel('f ---->')
ylabel('Power Spectral Density ---->')
title('PSD for Various Binary Line Codes')
legend('PSD for Polar Signal', 'PSD for Unipolar Signal', 'PSD for
Manchester Signal', 'PSD for Bipolar Signal')
```

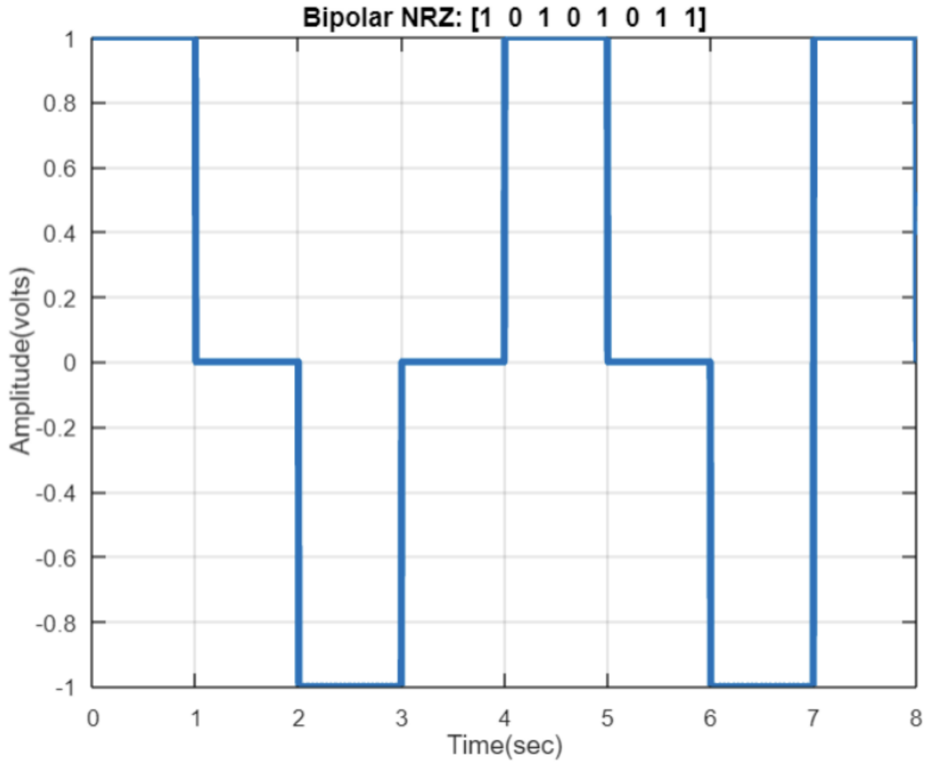
Input =[1 0 1 0 1 0 1 1]

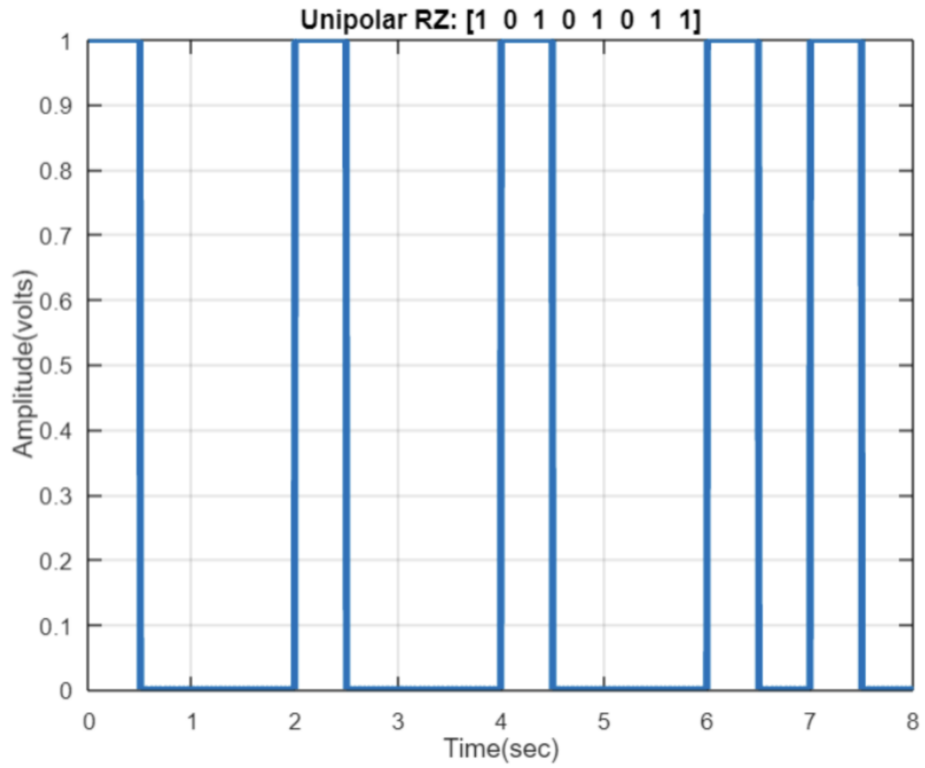


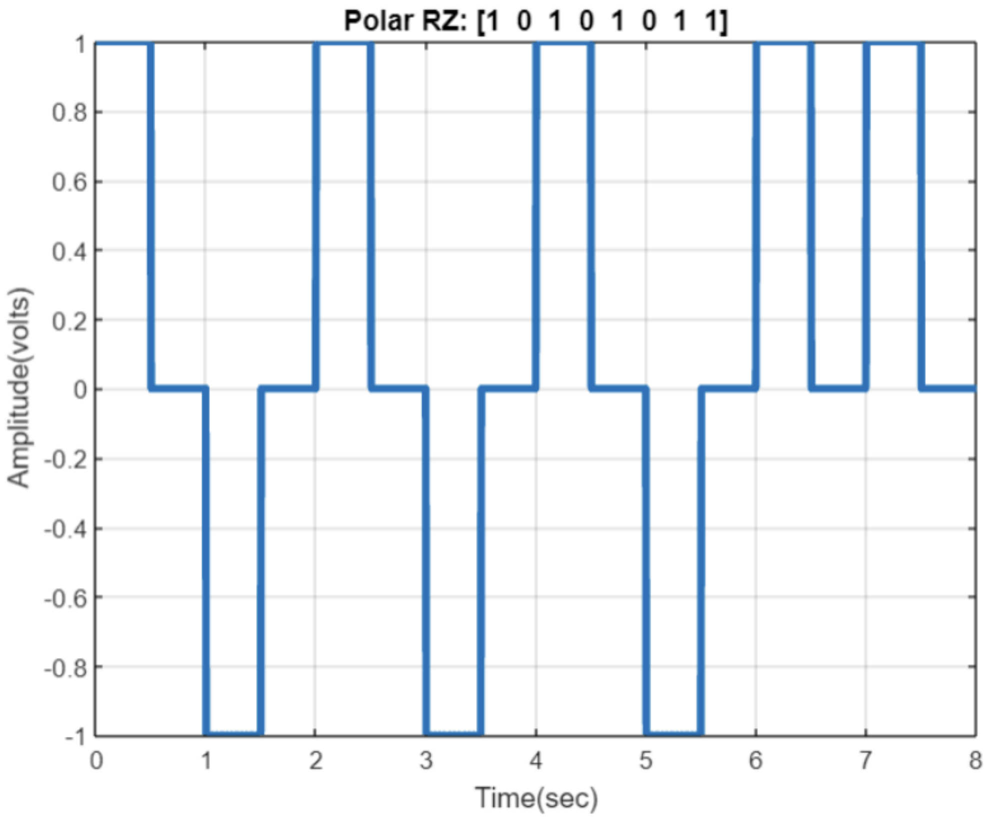
Unipolar NRZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
Time(sec)

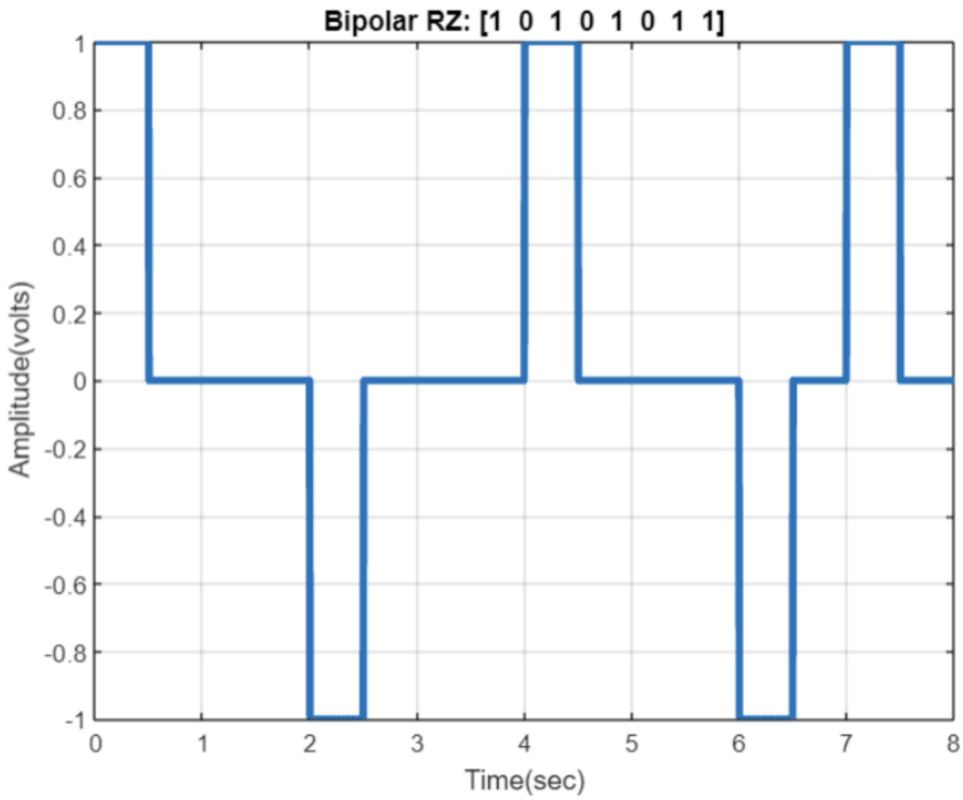


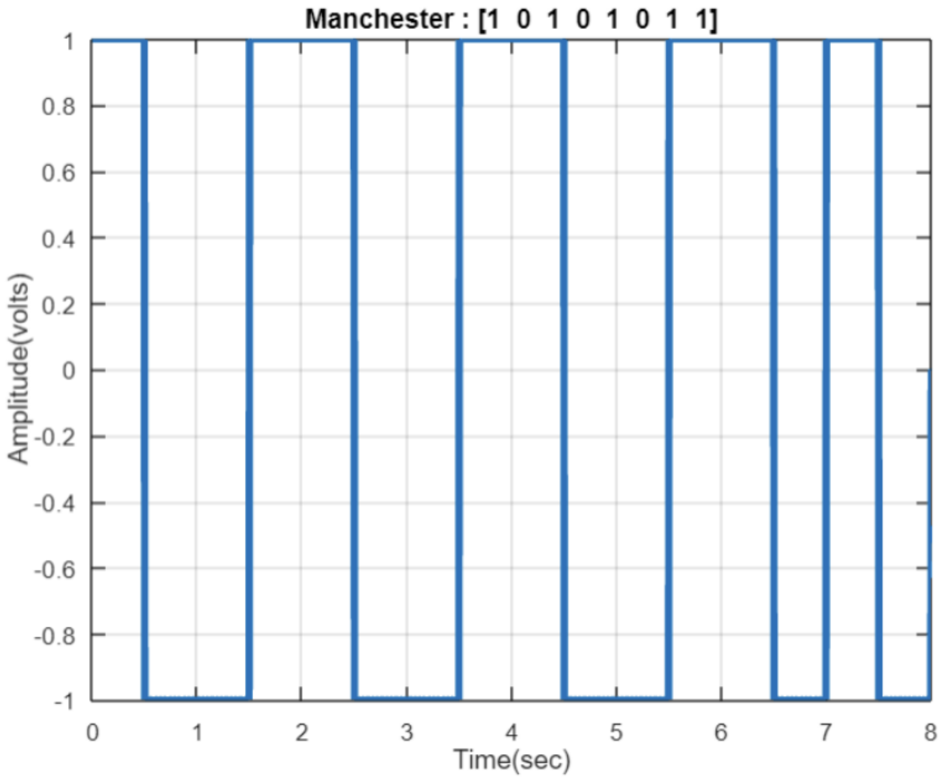
Polar NRZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
Time(sec)

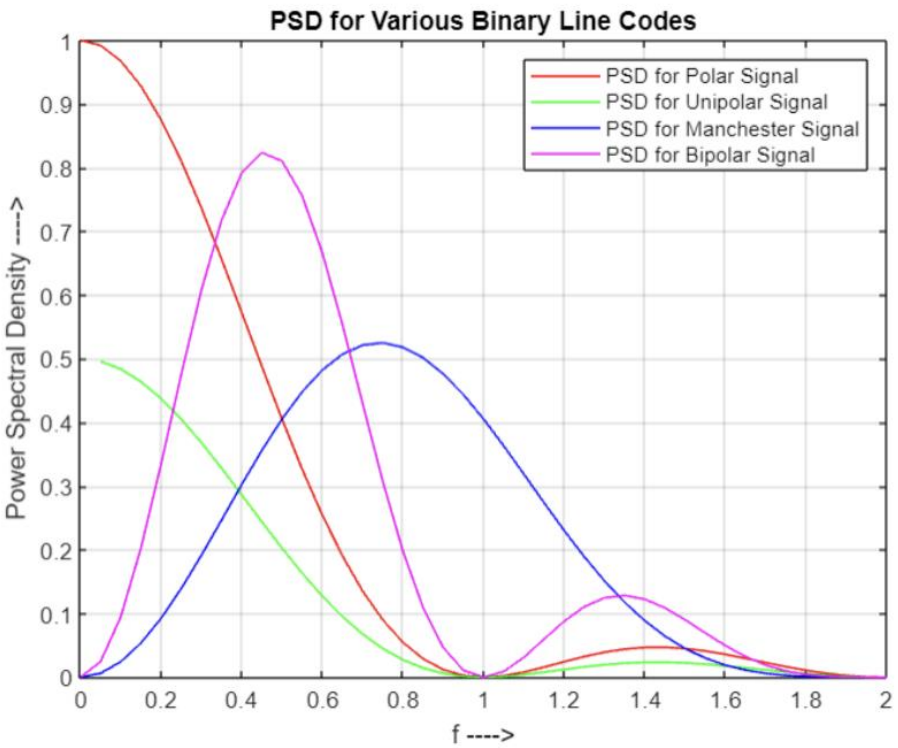

Bipolar NRZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
8
Time(sec)


Unipolar RZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7
8
Time(sec)


Polar RZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
Time(sec)


Bipolar RZ: [1 0 1 0 1 0 1 1]
Amplitude(volts)
Time(sec)


Manchester : [1 0 1 0 1 0 1 1]
Amplitude(volts)
Time(sec)

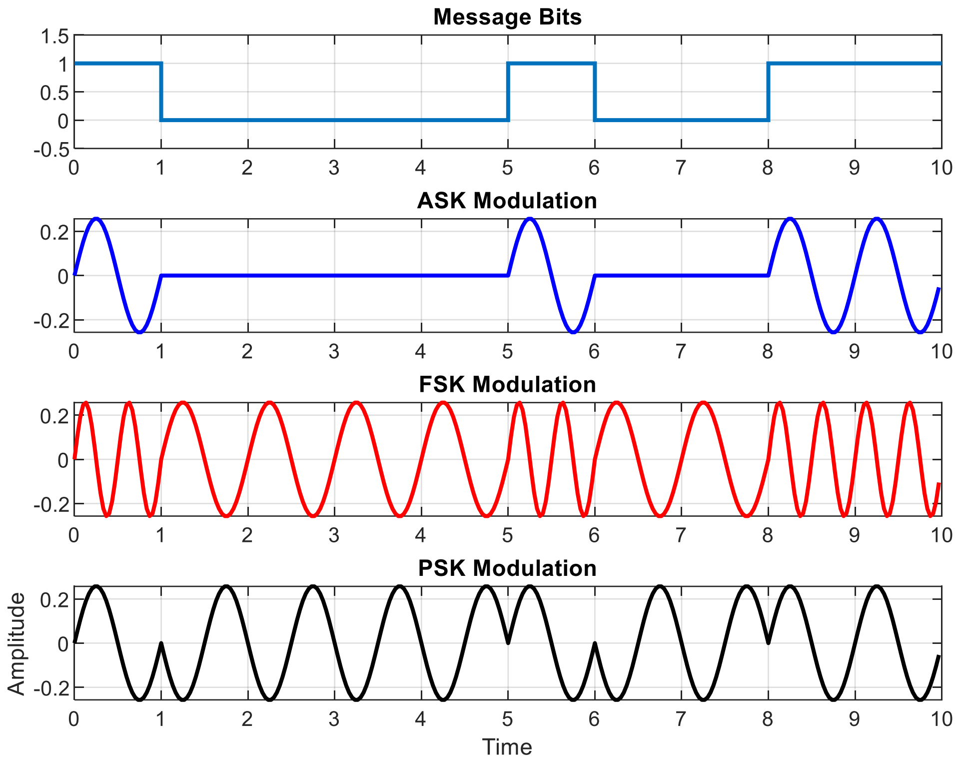

PSD for Various Binary Line Codes
PSD for Polar Signal
PSD for Unipolar Signal
PSD for Manchester Signal
PSD for Bipolar Signal
Power Spectral Density ---->
f ---->>

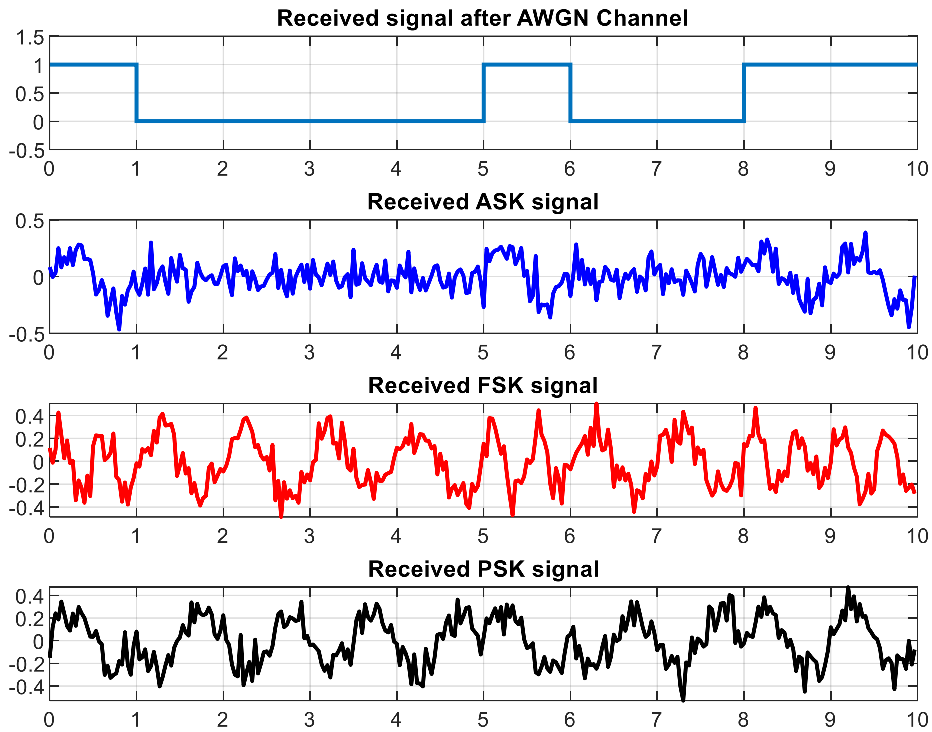
EXP NO.5	ERROR PERFORMANCE OF ASK, FSK, PSK MODULATION SCHEMES

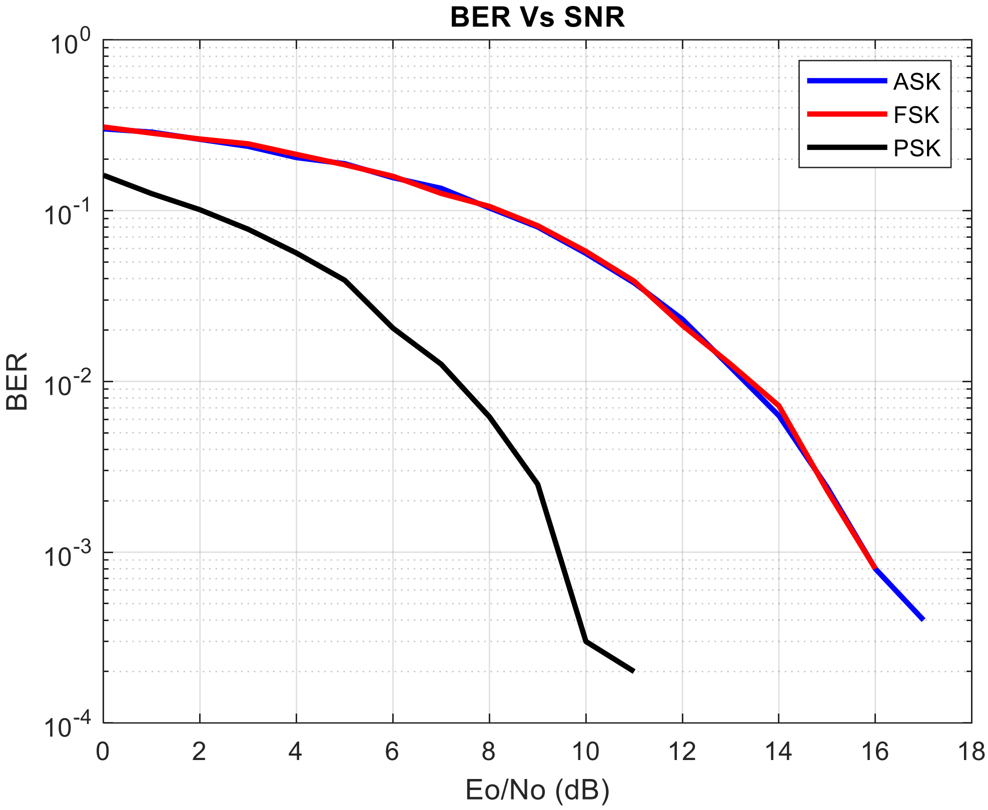
```
clc;
clear;
close all;
n = 10000;
b = randi([0,1], 1, n);
f1 = 1;
f2 = 2;
t = 0:1/30:1-1/30;
% ASK
sa1 = sin(2*pi*f1*t);
E1 = sum(sa1.^2);
sa1 = sa1/sqrt(E1); % unit energy
sa0 = 0*sin(2*pi*f1*t);
% FSK
sf0 = sin(2*pi*f1*t);
E = sum(sf0.^2);
sf0 = sf0/sqrt(E);
sf1 = sin(2*pi*f2*t);
E = sum(sf1.^2);
sf1 = sf1/sqrt(E);
% PSK
sp0 = -sin(2*pi*f1*t)/sqrt(E1);
sp1 = sin(2*pi*f1*t)/sqrt(E1);
% MODULATION
ask = []; psk = []; fsk = [];
for i = 1:n
 if b(i) == 1
 ask = [ask sa1];
 psk = [psk sp1];
 fsk = [fsk sf1];
 else
 ask = [ask sa0];
 psk = [psk sp0];
 fsk = [fsk sf0];
 end
end
figure(1)
subplot(411)
stairs(0:10, [b(1:10) b(10)], 'linewidth', 1.5)
axis([0 10 -0.5 1.5])
title('Message Bits'); grid on
subplot(412)
tb = 0:1/30:10-1/30;
plot(tb, ask(1:10*30), 'b', 'linewidth', 1.5)
title('ASK Modulation'); grid on
subplot(413)
```

```
plot(tb, fsk(1:10*30), 'r', 'linewidth', 1.5)
title('FSK Modulation'); grid on
subplot(414)
plot(tb, psk(1:10*30), 'k', 'linewidth', 1.5)
title('PSK Modulation'); grid on
xlabel('Time'); ylabel('Amplitude')
% AWGN
for snr = 0:20
 askn = awgn(ask, snr);
 pskn = awgn(psk, snr);
 fskn = awgn(fsk, snr);
 % DETECTION
 A = []; F = []; P = [];
 for i = 1:n
 % ASK Detection
 if sum(sa1 .* askn(1+30*(i-1):30*i)) > 0.5
 A = [A 1];
 else
 A = [A 0];
 end
 % FSK Detection
 if sum(sf1 .* fskn(1+30*(i-1):30*i)) > 0.5
 F = [F 1];
 else
 F = [F 0];
 end
 % PSK Detection
 if sum(sp1 .* pskn(1+30*(i-1):30*i)) > 0
 P = [P 1];
 else
 P = [P 0];
 end
 end
 % BER
 errA = 0; errF = 0; errP = 0;
 for i = 1:n
 if A(i) == b(i)
 errA = errA;
 else
 errA = errA + 1;
 end
if F(i) == b(i)
 errF = errF;
 else
 errF = errF + 1;
 end
 if P(i) == b(i)
 errP = errP;
 else
```

```
 errP = errP + 1;
 end
 end
 BER_A(snr+1) = errA/n;
 BER_F(snr+1) = errF/n;
 BER_P(snr+1) = errP/n;
end
figure(2)
subplot(411)
stairs(0:10, [b(1:10) b(10)], 'linewidth', 1.5)
axis([0 10 -0.5 1.5]); grid on
title('Received signal after AWGN Channel')
subplot(412)
tb = 0:1/30:10-1/30;
plot(tb, askn(1:10*30), 'b', 'linewidth', 1.5)
title('Received ASK signal'); grid on
subplot(413)
plot(tb, fskn(1:10*30), 'r', 'linewidth', 1.5)
title('Received FSK signal'); grid on
subplot(414)
plot(tb, pskn(1:10*30), 'k', 'linewidth', 1.5)
title('Received PSK signal'); grid on
figure(3)
semilogy(0:20, BER_A, 'b', 'linewidth', 2)
title('BER Vs SNR')
grid on; hold on
semilogy(0:20, BER_F, 'r', 'linewidth', 2)
semilogy(0:20, BER_P, 'k', 'linewidth', 2)
xlabel('Eo/No (dB)')
ylabel('BER')
hold off
legend('ASK', 'FSK', 'PSK');
```


Message Bits
ASK Modulation
FSK Modulation
PSK Modulation
Amplitude
Time

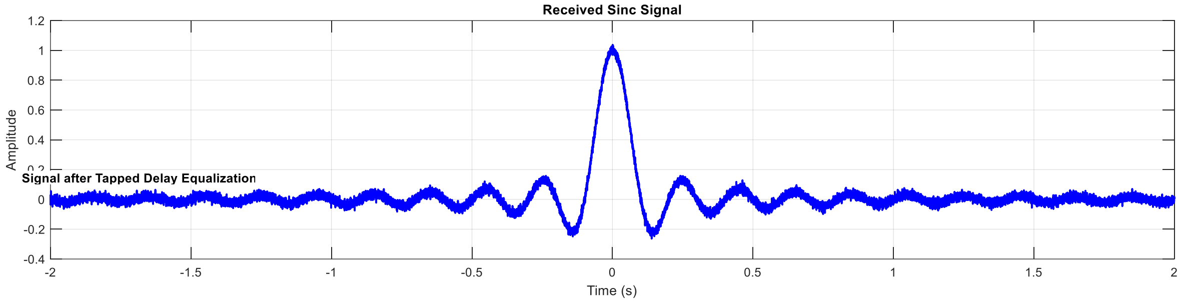

Received signal after AWGN Channel
Received ASK signal
Received FSK signal
Received PSK signal


BER Vs SNR
BER
Eo/No (dB)
ASK
FSK
PSK
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
2
4
6
8
10
12
14
16
18

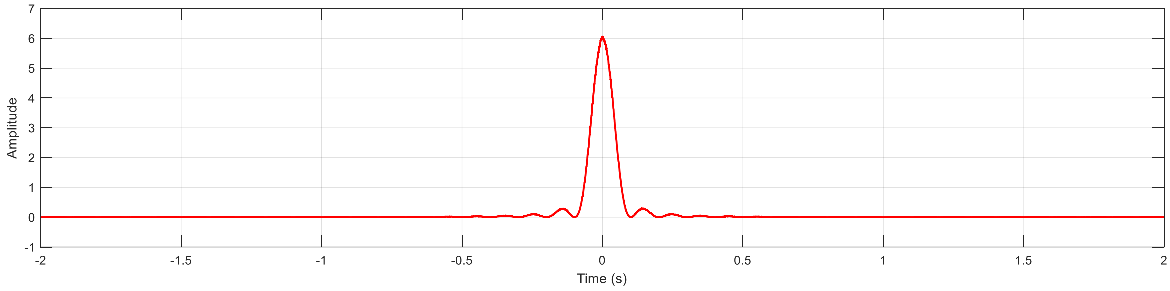
EXP NO.7	DESIGN AND IMPLEMENTATION OF TAPPED DELAY EQUALIZER

```
% Parameters
numTaps = 5; % Number of taps in the equalizer
channelDelay = 5; % Delay of the channel (for simulation purposes)
SNR = 20; % Signal-to-noise ratio (dB)
symbolRate = 10e3; % Symbol rate (symbols per second)
sincDuration = 2; % Duration of the sinc signal in seconds
numSamples = symbolRate * sincDuration; % Total number of samples
t = linspace(-2, sincDuration, numSamples); % Time vector
% Generate a sinc signal
sincSignal = sinc(10 * t);
% Create a channel with delay and noise
channel = zeros(1, channelDelay + 1);
channel(channelDelay + 1) = 1; % Impulse response with a delay
receivedSignal = filter(channel, 1, sincSignal); % Apply the channel
% Add noise to the received signal
receivedSignal = awgn(receivedSignal, SNR, 'measured');
% Tapped Delay Equalizer
equalizerOutput = zeros(1, numSamples);
for i = numTaps + 1:numSamples
% Use the past numTaps received samples to estimate the current
sample
equalizerOutput(i) = receivedSignal(i - numTaps:i) * sincSignal(i -
numTaps:i).';
end
% Plot the received sinc signal and the equalized signal
figure;
subplot(2, 1, 1);
plot(t, receivedSignal, 'b', 'LineWidth', 1.5);
title('Received Sinc Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
subplot(2, 1, 2);
plot(t, equalizerOutput, 'r', 'LineWidth', 1.5);
title('Signal after Tapped Delay Equalization');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

Received Sinc Signal



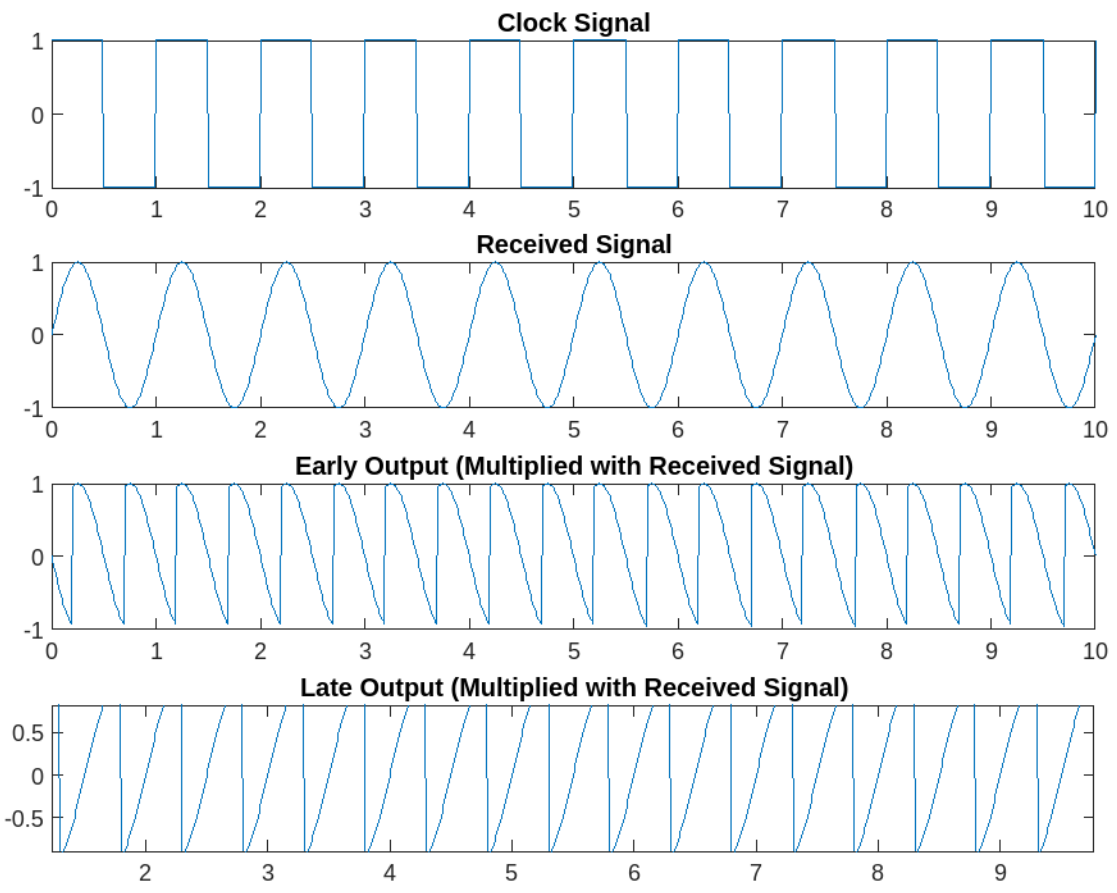
Signal after Tapped Delay Equalization

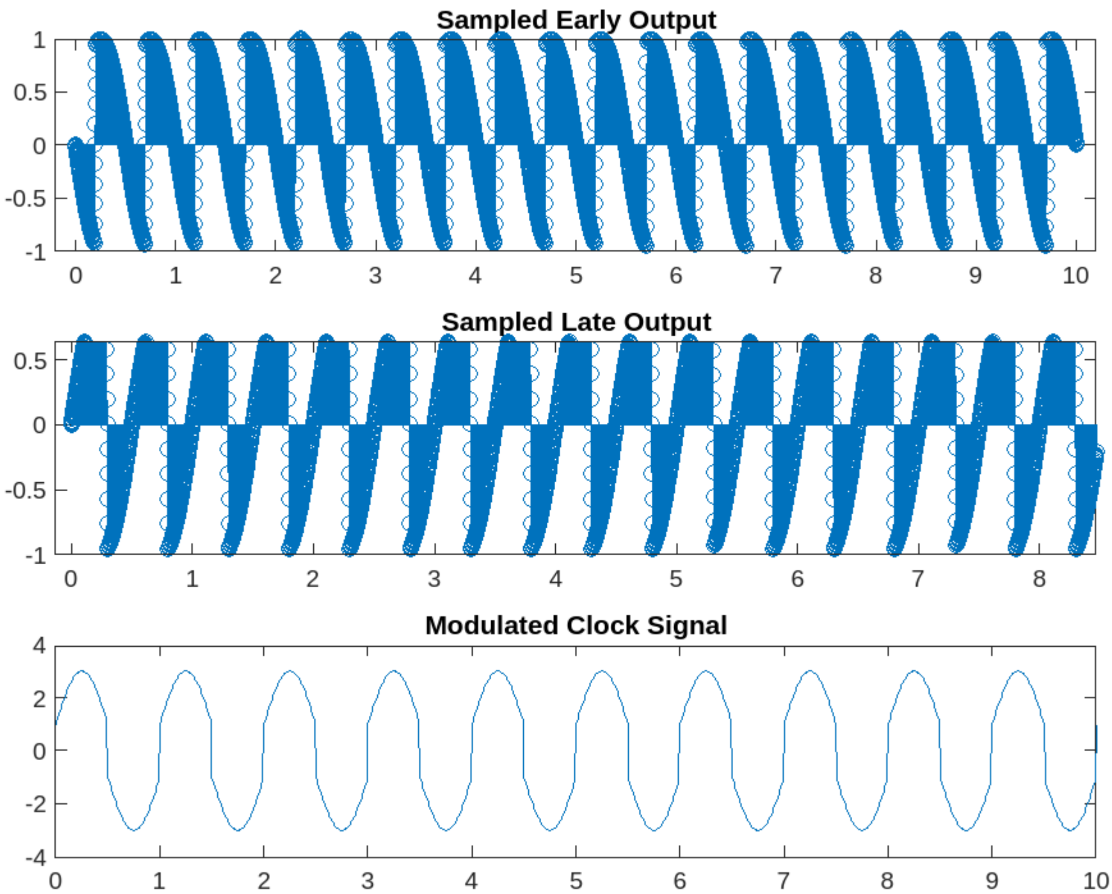


EXP NO.8	DESIGN AND IMPLEMENTATION OF EARLY LATE GATE SYCHRONIZATION

```
% Parameters
clock_period = 1; % Clock period in seconds
sample_delay = 0.2 * clock_period; % Delay for early and late
signals
sampling_frequency = 1000; % Sampling frequency in Hz
% Generate the clock signal
t = 0:0.01:10*clock_period; % Simulation time
clock_signal = square(2*pi*t/clock_period, 50);
% Generate a synthetic received signal (you can replace this with
your own signal)
% For demonstration purposes, let's use a sinusoidal signal
received_signal =sin(2*pi*t/clock_period);
% Apply the early and late gating
early_clock = square(2*pi*(t -sample_delay)/clock_period, 50);
late_clock = square(2*pi*(t +sample_delay)/clock_period, 50);
early_output = early_clock .*received_signal;
late_output = late_clock .*received_signal;
% Integrate the multiplied outputs over time period t
early_integral = trapz(t,early_output);
late_integral = trapz(t,late_output);
% Display the integrated results
disp(['Early Integral: ',num2str(early_integral)]);
disp(['Late Integral: ',num2str(late_integral)]);
% Determine the sampling instants
sampling_instants =0:1/sampling_frequency:t(end);
% Sample the integrated outputs at the determined instants
early_samples = interp1(t,early_output, sampling_instants,'linear',
0);
late_samples = interp1(t,late_output, sampling_instants,'linear',
0);
% Take the magnitude of the sampled outputs
early_magnitude = abs(early_samples);
late_magnitude = abs(late_samples);
% Add the magnitudes of early and late outputs
combined_magnitude = early_magnitude+ late_magnitude;
% Resample combined_magnitude to match the length of clock_signal
combined_magnitude =interp1(sampling_instants,combined_magnitude, t,
'linear', 0);
% Modulate the clock signal using the combined magnitude
modulated_clock_signal = clock_signal.* (1 + combined_magnitude);
% Plot the received signal, early output, late output, and modulated
clock signal
figure;
subplot(4,1,1);
plot(t, clock_signal);
```

```
title('Clock Signal');
subplot(4,1,2);
plot(t, received_signal);
title('Received Signal');
subplot(4,1,3);
plot(t, early_output);
title('Early Output (Multiplied with Received Signal)');
subplot(4,1,4);
plot(t, late_output);
title('Late Output (Multiplied with Received Signal)');
% Plot the sampled outputs
figure;
subplot(3,1,1);
stem(sampling_instants,early_samples);
title('Sampled Early Output');
subplot(3,1,2);
stem(sampling_instants,late_samples);
title('Sampled Late Output');
subplot(3,1,3);
plot(t, modulated_clock_signal);
title('Modulated Clock Signal');
```



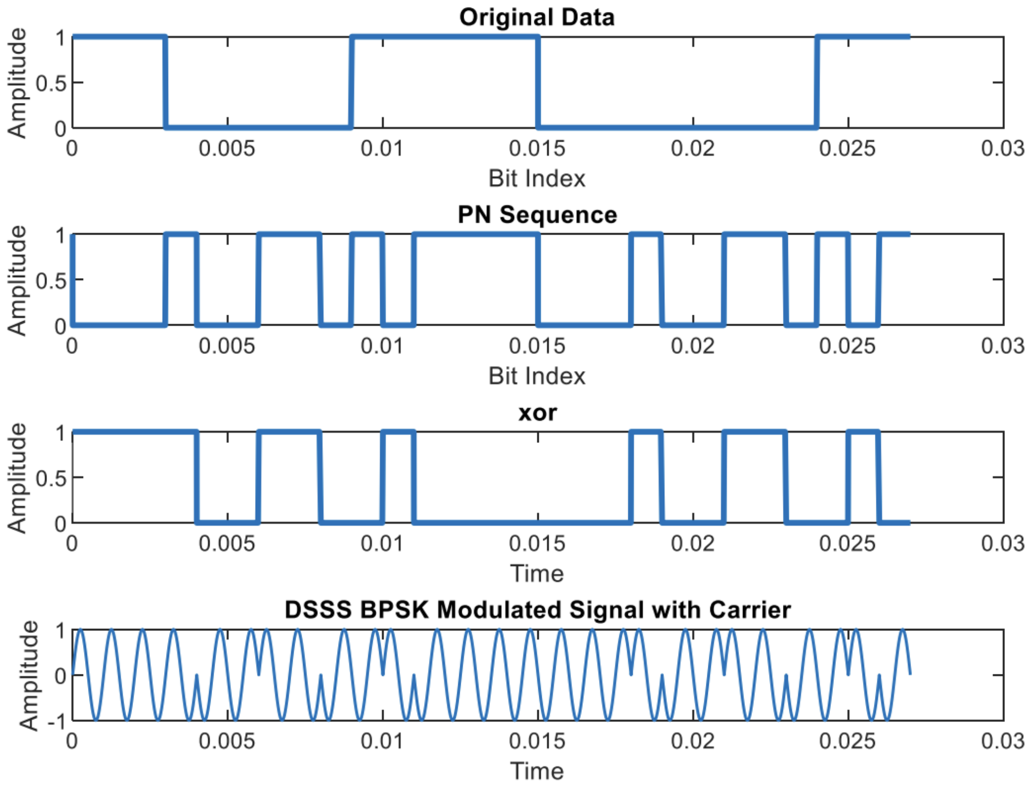

Sampled Early Output
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
Sampled Late Output
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8
Modulated Clock Signal
4
2
0
-2
-4
0 1 2 3 4 5 6 7 8 9 10

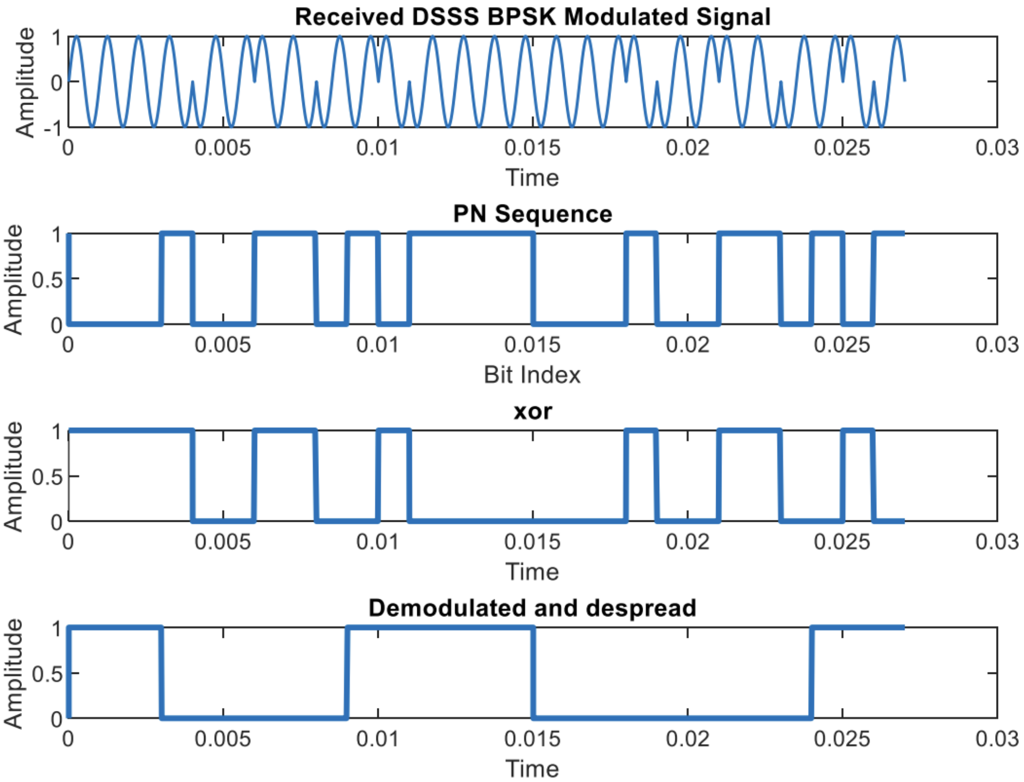
EXP NO.10	IMPLEMENTATION OF SPREAD SPECTRUM SYSTEMS

```
% DSSS with symbol period(symbol_per) and chip period (chip_per)
including Carrier
clear; close all;
state = [1 0 0 0];
feedback_taps = [3 4];
pn_seq = zeros(1, 15);
for i = 1:15
 pn_seq(i) = state(end);
 feedback = mod(state(feedback_taps(1)) + state(feedback_taps(2)),
2);
 state = [feedback state(1:end-1)];
end
% Inputs
data = input('Enter binary data (as a vector of 1s and 0s): ');
%pn_seq = input('Enter PN sequence (as a vector of 1s and 0s): ');
symbol_per = input('Enter the symbol period (duration each data bit
is held): ');
% Symbol period in arbitrary units
chip_per = input('Enter the chip period (duration each PN bit is
held): '); %
Chip period in arbitrary units
% Carrier frequency
carrier_freq = input('Enter carrier frequency (in Hz): ');
cons=symbol_per/chip_per;
% Step 1: Ensure PN sequence is long enough, extend if needed
data_len = length(data)*cons;
pn_len = length(pn_seq);
if pn_len < data_len
 % Extend the PN sequence if needed by repeating it
 pn_seq = repmat(pn_seq, 1, ceil(data_len / pn_len));
 pn_seq = pn_seq(1:data_len); % Match the length exactly
end
% Step 2: XOR data with the PN sequence (for spreading)
spreaded_signal = xor(repelem(data, cons), pn_seq(1:data_len));
% Step 3: BPSK Modulation (convert 0 to -1 and 1 to 1)
bpsk_signal = 2*spreaded_signal - 1;
reference = (1:numel(bpsk_signal))*chip_per;
x=[]; y=[]; ydata=[]; ypn_seq=[]; yspread=[];
timee=linspace(0,numel(pn_seq)*chip_per,100000);
z=repelem(data, cons);
for i=1:length(timee)
 temp=find(reference>timee(i),1);
 ydata=[ydata z(temp)];
 ypn_seq=[ypn_seq pn_seq(temp)];
 yspread=[yspread spreaded_signal(temp)];
```

```
 if bpsk_signal(temp)==1
 y=[y sin(2*pi*carrier_freq*timee(i))];
 else
 y=[y -sin(2*pi*carrier_freq*timee(i))];
 end
end
% Plot the results
figure(1);
subplot(4,1,1);
plot(timee,[1 ydata], 'LineWidth', 2);
title('Original Data');
xlabel('Bit Index');
ylabel('Amplitude');
subplot(4,1,2);
plot(timee,[1 ypn_seq], 'LineWidth', 2);
title('PN Sequence');
xlabel('Bit Index');
ylabel('Amplitude');
subplot(4,1,3);
plot(timee,[1 yspread], 'LineWidth', 2);
title('xor');
xlabel('Time');
ylabel('Amplitude');
subplot(4,1,4);
plot(timee,y, 'LineWidth', 1);
title('DSSS BPSK Modulated Signal with Carrier');
xlabel('Time');
ylabel('Amplitude');
%RECEIVER
rx_spread=[1 yspread];
rx_pn_seq=[1 ypn_seq];
rx_data=xor(rx_pn_seq, rx_spread);
% Plot the results
figure(2);
subplot(4,1,1);
plot(timee,y, 'LineWidth', 1);
title('Received DSSS BPSK Modulated Signal');
xlabel('Time');
ylabel('Amplitude');
subplot(4,1,2);
plot(timee,rx_pn_seq, 'LineWidth', 2);
title('PN Sequence');
xlabel('Bit Index');
ylabel('Amplitude');
subplot(4,1,3);
plot(timee,rx_spread, 'LineWidth', 2);
title('xor');
xlabel('Time');
ylabel('Amplitude');
```

```
subplot(4,1,4);
plot(timee,rx_data, 'LineWidth', 2);
title('Demodulated and despread');
xlabel('Time');
ylabel('Amplitude');
```

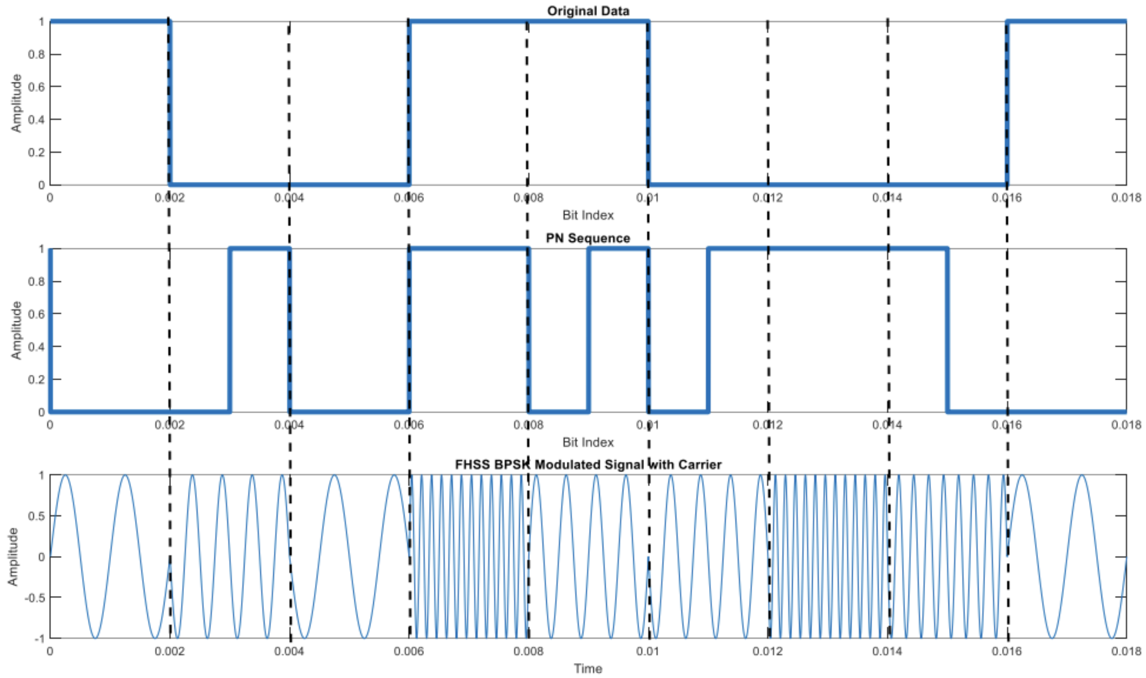


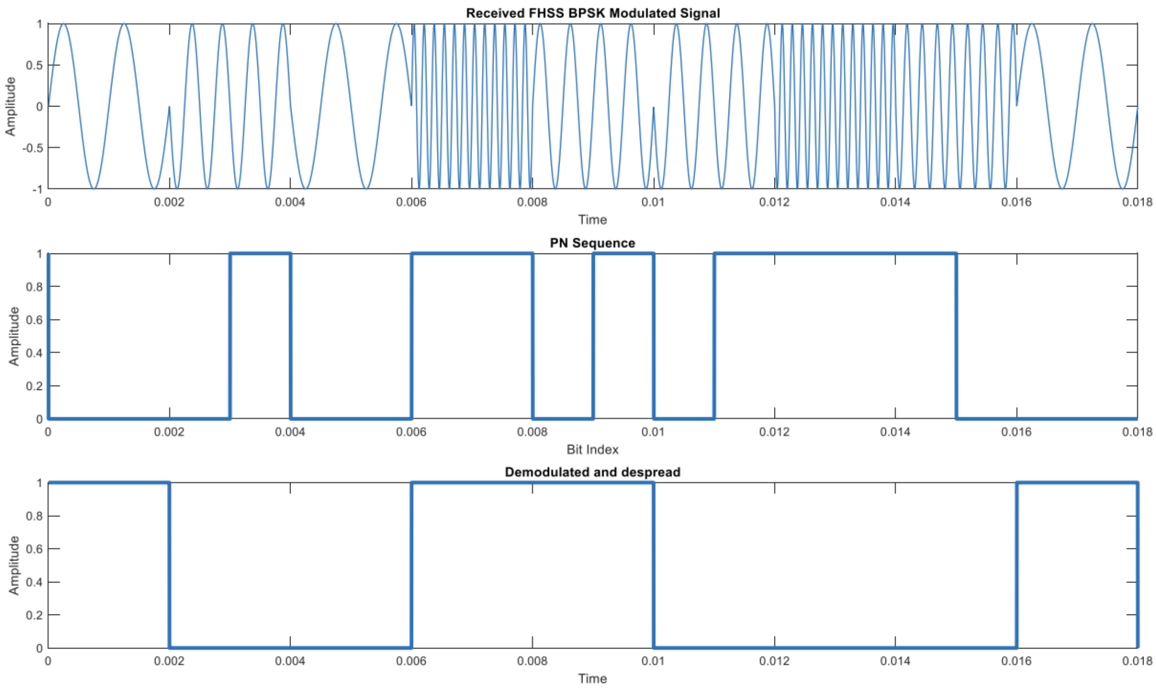


```
% FSSS with symbol period(symbol_per) and chip period(chip_per)
including carrier
clear; close all;
state = [1 0 0 0];
feedback_taps = [3 4];
pn_seq = zeros(1, 15);
for i = 1:15
 pn_seq(i) = state(end);
 feedback = mod(state(feedback_taps(1)) + state(feedback_taps(2)),
2);
 state = [feedback state(1:end-1)];
end
% Inputs
data = input('Enter binary data (as a vector of 1s and 0s): ');
%pn_seq = input('Enter PN sequence (as a vector of 1s and 0s): ');
symbol_per = input('Enter the symbol period (duration each data bit
is held): ');
% Symbol period in arbitrary units
chip_per = 0.5*symbol_per; % Chip period in arbitrary units
% Carrier frequency
carrier_freq1 = input('Enter lowest carrier frequency (in Hz): ');
carrier_freq2 = 2*carrier_freq1;
carrier_freq3 = 4*carrier_freq1;
carrier_freq4 = 6*carrier_freq1;
carrier = [carrier_freq1 carrier_freq2 carrier_freq3 carrier_freq4];
cons=symbol_per/chip_per;
% Step 1: Ensure PN sequence is long enough, extend if needed
data_len = length(data)*cons;
pn_len = length(pn_seq);
if pn_len < data_len
 % Extend the PN sequence if needed by repeating it
 pn_seq = repmat(pn_seq, 1, ceil(data_len / pn_len));
 pn_seq = pn_seq(1:data_len); % Match the length exactly
end
% Reshape into pairs of two elements each
new_pn = reshape(pn_seq, 2, [])';
% Convert binary pairs to decimal
new_pn = new_pn(:,1) * 2 + new_pn(:,2);
new_pn = reshape(new_pn,1,[]);
timee=linspace(0,symbol_per*numel(data),1000*numel(data));
tx = zeros(1,1000*numel(data));
for i=1:length(data)
 for j=1:length(timee)
 if timee(j)<(i*symbol_per) && timee(j)>=((i-1)*symbol_per)
 tx(j)=(2*data(i)-1)*sin(2*pi*carrier(1+new_pn(i))*timee(j));
```

```
 end
 end
end
data1 = repelem(data, cons);
reference = (1:numel(data1))*chip_per;
ydata=[]; ypn=[];
for i=1:length(timee)
 temp1=find(reference>timee(i),1);
 ydata=[ydata data1(temp1)];
 ypn=[ypn pn_seq(temp1)];
end
% Plot the results
figure(1);
subplot(3,1,1);
plot(timee,[1 ydata], 'LineWidth', 4);
title('Original Data');
xlabel('Bit Index');
ylabel('Amplitude');
subplot(3,1,2);
plot(timee,[1 ypn], 'LineWidth', 4);
title('PN Sequence');
xlabel('Bit Index');
ylabel('Amplitude');
subplot(3,1,3);
plot(timee,tx, 'LineWidth', 1);
title('FHSS BPSK Modulated Signal with Carrier');
xlabel('Time');
ylabel('Amplitude');
%RECEIVER
rx=zeros(1,1000*numel(data));
for i=1:length(data)
 for j=1:length(tx)
 if timee(j)<(i*symbol_per) && timee(j)>=((i-1)*symbol_per)
 rx(j)=(1+tx(j)/sin(2*pi*carrier(1+new_pn(i))*timee(j)))/2;
 end
 end
end
rx2=rx(2:1000:end);
disp(rx2);
% Plot the results
figure(2);
subplot(3,1,1);
plot(timee,tx, 'LineWidth', 1);
title('Received FHSS BPSK Modulated Signal');
xlabel('Time');
ylabel('Amplitude');
subplot(3,1,2);
plot(timee,[1 ypn], 'LineWidth', 2);
title('PN Sequence');
```

```
xlabel('Bit Index');
ylabel('Amplitude');
subplot(3,1,3);
plot(timee,rx, 'LineWidth', 2);
title('Demodulated and despread');
xlabel('Time');
ylabel('Amplitude');
```





EXP NO.11	WIRELESS CHANNEL SIMULATION INCLUDING FADING AND DOPPLER EFFECTS

```
% Parameters for the simulation
N = 1000;
% Number of waves (adjust as needed)
E0 = 1;
% Amplitude of local average E-field (adjust as needed)
fc = 2e9;
% Carrier frequency in Hz (adjust as needed)
% Generate random amplitudes for Cn (Equation 4.58)
Cn = rand(1, N);
% Normalize amplitudes (Equation 4.62)
Cn = Cn / sum(Cn);
% Initialize time parameters
t = linspace(0, 1, 1000);  % Time vector (adjust as needed)
% Initialize arrays to store Tc(t) and Ts(t)
Tc = zeros(size(t));
Ts = zeros(size(t));
% Calculate Tc(t) and Ts(t) over time (Equations 4.64 and 4.65)
for n = 1:N
% Random phase for the nth component (Equation 4.61)
phase_n = rand * 2 * pi;
% Calculate Tc(t) and Ts(t) components
Tc = Tc + E0 * Cn(n) * cos(2 * pi * fc * t + phase_n);
Ts = Ts + E0 * Cn(n) * sin(2 * pi * fc * t + phase_n);
end
% Calculate E_z field component (Equation 4.63)
Ez_field = Tc .* cos(2 * pi * fc * t) - Ts .* sin(2 * pi * fc * t);
% Calculate the Doppler shift
v = 30; % Velocity of the mobile receiver in m/s (adjust as needed)
angle_of_arrival_deg = 30; % Angle of arrival in degrees (adjust as
needed)
% Convert angle of arrival from degrees to radians
angle_of_arrival_rad = deg2rad(angle_of_arrival_deg);
% Calculate the Doppler shift in Hertz
c = 3e8; % Speed of light in m/s
doppler_shift = (v / c) * fc * cos(angle_of_arrival_rad);
% Complex Gaussian random variables
N = 10^6;
x = randn(1, N); % Gaussian random variable, mean 0, variance 1
y = randn(1, N); % Gaussian random variable, mean 0, variance 1
z = (x + 1i * y); % Complex random variable
% Probability density function of abs(z)
zBin = [0:0.01:7];
sigma2 = 1;
pzTheory = (zBin / sigma2) .* exp((-zBin.^2) / (2 * sigma2)); %
Theory
[nzSim, zBinSim] = hist(abs(z), zBin); % Simulation
```

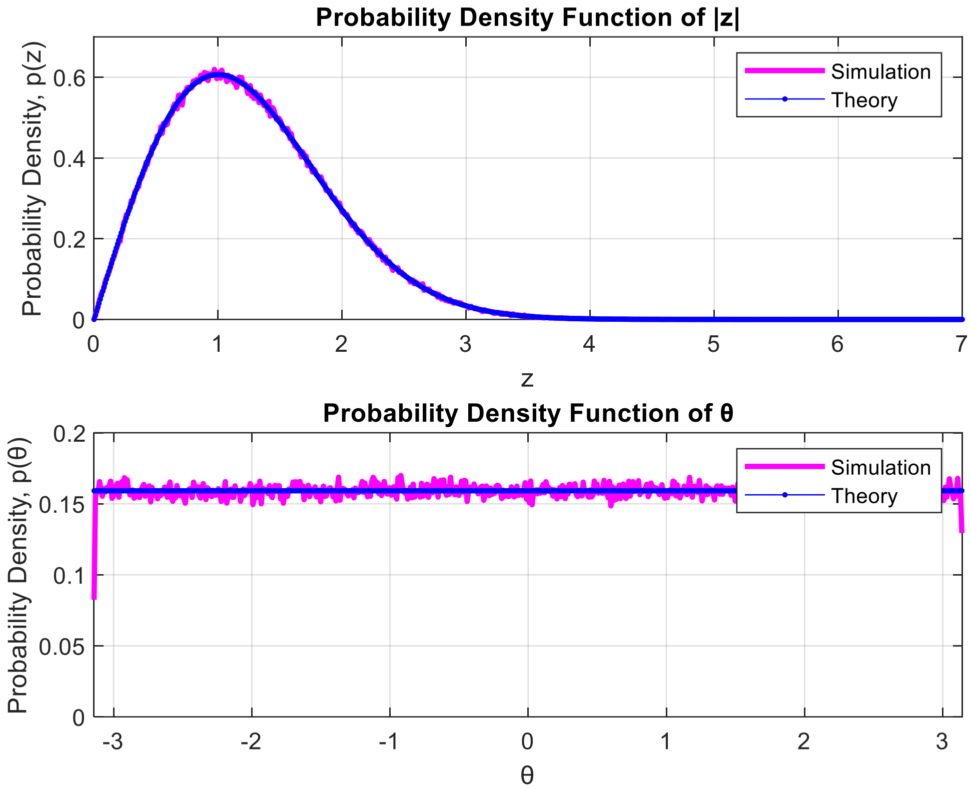
```
% Probability density of theta
thetaBin = [-pi:0.01:pi];
pThetaTheory = 1 / (2 * pi) * ones(size(thetaBin));
[nThetaSim, thetaBinSim] = hist(angle(z), thetaBin); % Simulation
% Plot the PDFs
figure;
subplot(2, 1, 1);
plot(zBinSim, nzSim / (N * 0.01), 'm', 'LineWidth', 2);
hold on;
plot(zBin, pzTheory, 'b.-');
xlabel('z');
ylabel('Probability Density, p(z)');
legend('Simulation', 'Theory');
title('Probability Density Function of |z|');
axis([0 7 0 0.7]);
grid on;
subplot(2, 1, 2);
plot(thetaBinSim, nThetaSim / (N * 0.01), 'm', 'LineWidth', 2);
hold on;
plot(thetaBin, pThetaTheory, 'b.-');
xlabel('θ');
ylabel('Probability Density, p(θ)');
legend('Simulation', 'Theory');
title('Probability Density Function of θ');
axis([-pi pi 0 0.2]);
grid on;
% Apply the filter to the received signal (Ez_field)
Fc = 0; % Center frequency (adjust as needed)
Fm = doppler_shift; % Maximum Doppler shift (adjust as needed)
% Define the frequency axis
fs = 1 / (t(2) - t(1)); % Sampling frequency
f_axis = linspace(-fs / 2, fs / 2, length(t));
% Calculate the filter response
frequency_response = (1.5 / (pi * Fm)) * sqrt(1 - ((f_axis - Fc) /
Fm).^2);
% Apply the filter to the received signal (Ez_field)
filtered_signal = ifft(fft(Ez_field) .*
fftshift(frequency_response));
% Plot the magnitude of Ez_field as the received signal (r)
figure;
plot(t, abs(filtered_signal));
xlabel('Time (s)');
ylabel('Received Signal Amplitude (r)');
title('Received Signal (Magnitude of Ez\_field)');
% Plot the Doppler shift as before
figure;
plot(t, doppler_shift * ones(size(t)), 'r--');
xlabel('Time (s)');
ylabel('Doppler Shift (Hz)');
```

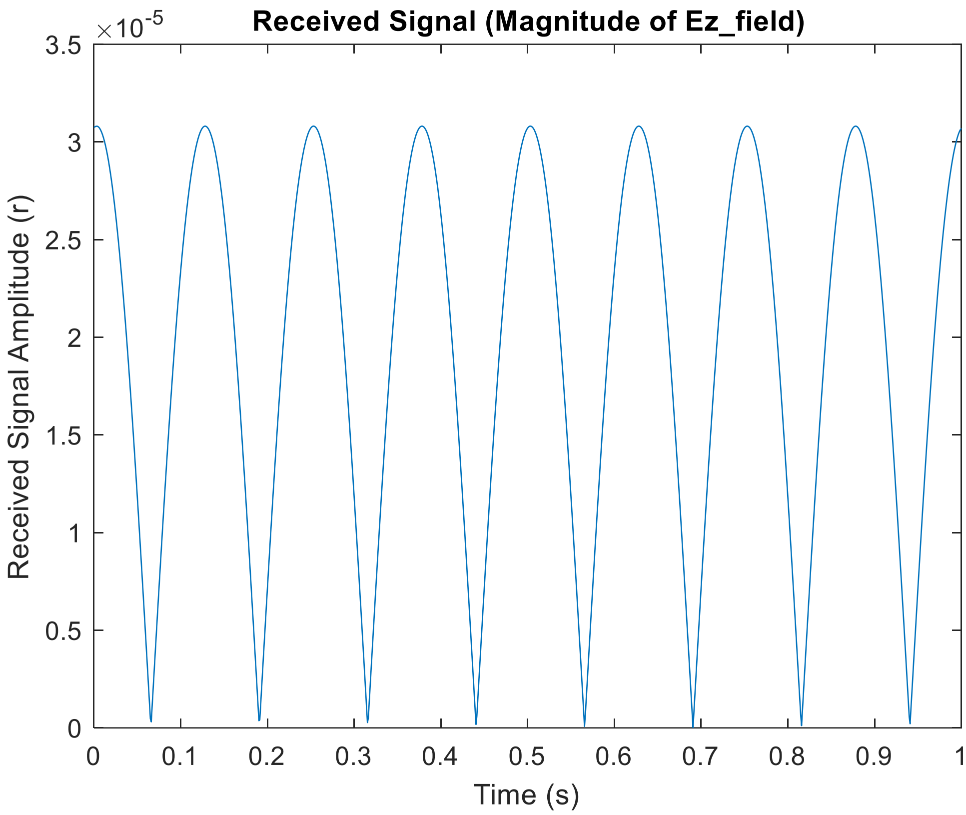
```
title('Doppler Shift Over Time');
% Plot the power spectrum of the filtered signal
psd_filtered = (1 / (fs * length(filtered_signal))) *
abs(fft(filtered_signal)).^2;
f_axis_filtered = linspace(-fs / 2, fs / 2, length(psd_filtered));
figure;
plot(f_axis_filtered, psd_filtered);
% Power spectrum
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
title('Power Spectrum of Filtered Received Signal');% Parameters for
the simulation
N = 1000;                    % Number of waves (adjust as needed)
E0 = 1;
% Amplitudef local average E-field (adjust as needed)
fc = 2e9;
% Carrier frequency in Hz (adjust as needed)
fs = 1000;
% Sampling frequency in Hz (adjust as needed)
% Initialize time parameters
t = linspace(0, 1, fs);
% Time vector (adjust as needed)
% Number of waveforms
num_waveforms = 3;
% Initialize cross-correlation matrix
cross_corr_matrix = zeros(num_waveforms, num_waveforms);
% Generate three r(t) waveforms
rt_waveforms = cell(num_waveforms, 1);
for waveform_idx = 1:num_waveforms
% Initialize arrays to store Tc(t) and Ts(t)
Tc = zeros(size(t));
Ts = zeros(size(t));
% Calculate Tc(t) and Ts(t) over time (Equations 4.64 and 4.65)
for n = 1:N
% Random phase for the nth component (Equation 4.61)
phase_n = rand * 2 * pi;
% Calculate Tc(t) and Ts(t) components
Tc = Tc + E0 * rand * cos(2 * pi * fc * t + phase_n);
Ts = Ts + E0 * rand * sin(2 * pi * fc * t + phase_n);
end
% Calculate E_z field component (Equation 4.63)
Ez_field = Tc .* cos(2 * pi * fc * t) - Ts .* sin(2 * pi * fc * t);
% Apply the filter to the received signal (Ez_field)
Fc = 0; % Center frequency (adjust as needed)
Fm = doppler_shift; % Maximum Doppler shift (adjust as needed)
% Define the frequency axis
f_axis = linspace(-fs / 2, fs / 2, length(t));
% Calculate the filter response
```

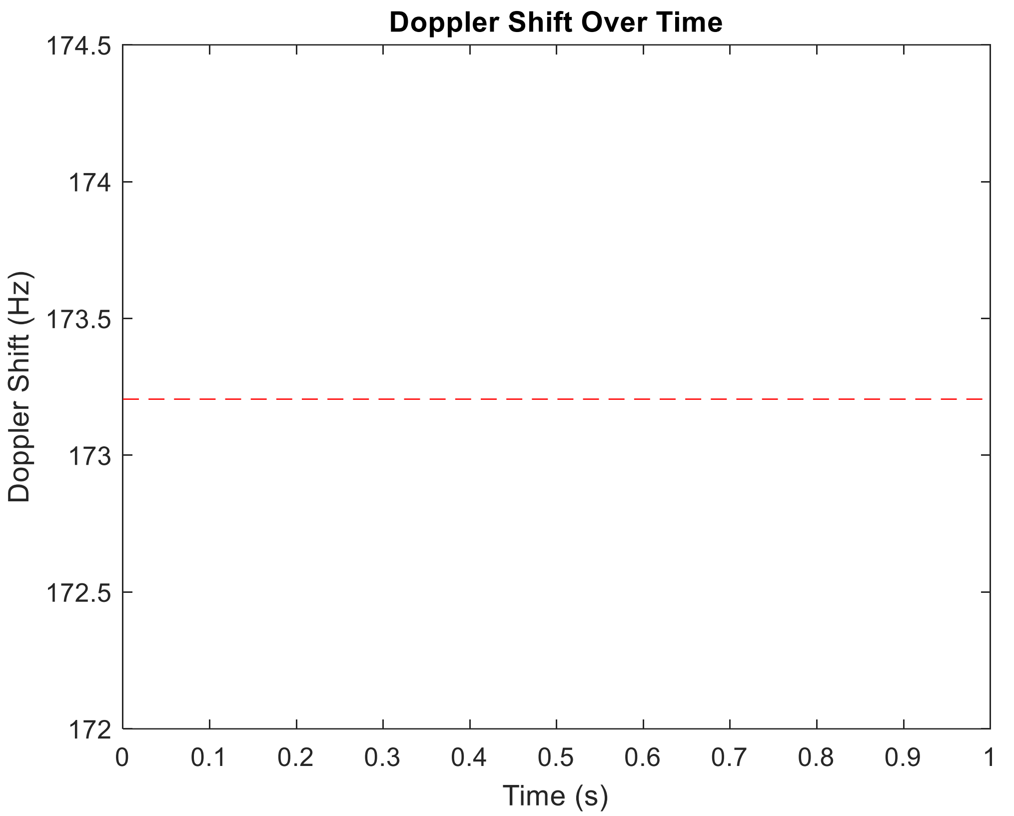
```
frequency_response = (1.5 / (pi * Fm)) * sqrt(1 - ((f_axis - Fc) /
Fm).^2);
% Apply the filter to the received signal (Ez_field)
filtered_signal = ifft(fft(Ez_field) .*
fftshift(frequency_response));
% Store the generated r(t) waveform
rt_waveforms{waveform_idx} = abs(filtered_signal);
end
% Calculate cross-correlation values among r(t) waveforms
for i = 1:num_waveforms
for j = 1:num_waveforms
cross_corr = xcorr(rt_waveforms{i}, rt_waveforms{j});
cross_corr_matrix(i, j) = max(cross_corr);
end
end
% Display the cross-correlation matrix
disp('Cross-Correlation Matrix:');
disp(cross_corr_matrix);
```

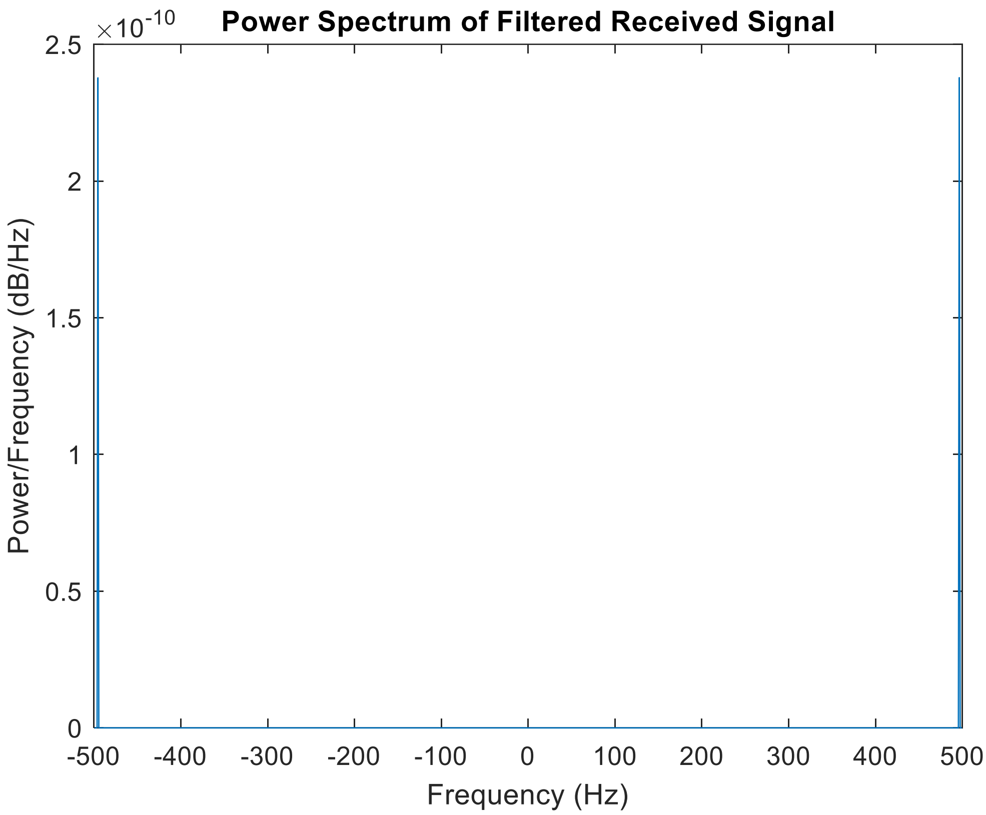
OUTPUT:

```
>> wirelessfading
Cross-Correlation Matrix:
    3.6077    1.5098    2.3971
    1.5098    0.7767    1.1097
    2.3971    1.1097    1.7596
```


Probability Density Function of |z|
Probability Density, p(z)
z
Simulation
Theory
Probability Density Function of θ
Probability Density, p(θ)
θ
Simulation
Theory


Received Signal (Magnitude of Ez_field)
×10^-5
Received Signal Amplitude (r)
Time (s)


Doppler Shift Over Time
174.5
174
173.5
173
172.5
172
Doppler Shift (Hz)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Time (s)


×10^-10
Power Spectrum of Filtered Received Signal
2.5
2
1.5
1
0.5
0
Power/Frequency (dB/Hz)
-500
-400
-300
-200
-100
0
100
200
300
400
500
Frequency (Hz)

EXP NO.12	ERROR PERFORMANCE OF OFDM

```
clear all
close all
clc
nbits = 208000;
modlevel = 2;
nbitpersym = 52; % number of bits per QAM OFDM symbol (same as the
number of subcarriers for 16-QAM)
nsym = 10^4; % number of symbols
len_fft = 64; % FFT size
sub_car = 52; % number of data subcarriers
EbNo = 0:2:15;
EsNo = EbNo + 10*log10(52/64) + 10*log10(64/80) + 10*log10(4);
snr = EsNo - 10*log10((64/80));
M = 16; % modulation order for 16-QAM
% Generating data
t_data = randi([0 1], nbitpersym*nsym*4, 1); % Generate random bits
qamdata = bi2de(reshape(t_data, 4, 520000).', 'left-msb');
% Modulating data
mod_data = 1/sqrt(10) * qammod(qamdata, M);
% Serial to parallel conversion
par_data = reshape(mod_data, nbitpersym, nsym).';
% Pilot insertion
pilot_ins_data = [zeros(nsym, 6), par_data(:, 1:nbitpersym/2),
zeros(nsym, 1), par_data(:,nbitpersym/2+1:nbitpersym), zeros(nsym,
5)];
% Fourier transform (time domain data)
IFFT_data = ifft(fftshift(pilot_ins_data.')).';
a = max(max(abs(IFFT_data)));
IFFT_data = IFFT_data / a; % Normalization
% Addition of cyclic prefix
cyclic_add_data = [IFFT_data(:, 49:64), IFFT_data].';
% Parallel to serial conversion
ser_data = reshape(cyclic_add_data, 80*nsym, 1);
% Passing through channel
no_of_error = [];
ratio = [];
for ii = 1:length(snr)
 chan_awgn = awgn(ser_data, snr(ii), 'measured'); % AWGN addition
 ser_to_para = reshape(chan_awgn, 80, nsym).'; % Serial to parallel
conversion
 cyclic_pre_rem = ser_to_para(:, 17:80); % Cyclic prefix removal
 FFT_recdata = a * fftshift(fft(cyclic_pre_rem.')).'; % Frequency
domain transform
 rem_pilot = FFT_recdata(:, [6 + (1:nbitpersym/2), 7 + (nbitpersym/2
+ 1:nbitpersym)]); % Pilot removal
```

```
 ser_data_1 = sqrt(10) * reshape(rem_pilot.', nbitpersym*nsym, 1); %
Serial conversion
 demod_Data = qamdemod(ser_data_1, M); % Demodulating the data
 data1 = de2bi(demod_Data, 'left-msb');
 data2 = reshape(data1.', nbitpersym*nsym*4, 1);
 [no_of_error(ii), ratio(ii)] = biterr(t_data, data2); % Error rate
calculation
end
% Plotting the result
semilogy(EbNo, ratio, 'b', 'linewidth', 2);
hold on;
theoryBer = (1/4) * 3/2 * erfc(sqrt(4*0.1*(10.^(EbNo/10))));
semilogy(EbNo, theoryBer, '--r', 'linewidth', 2);
axis([0 15 10^-5 1])
legend('Simulated', 'Theoretical')
grid on
xlabel('EbNo');
ylabel('BER');
title('Bit error probability curve for QAM using OFDM');
```