DIGITAL COMMUNICATIONS LAB MANUAL

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Prasad V. Potluri Siddhartha Institute of Technology

(Autonomous)

Accredited with A+ grade by NAAC

ISO 9001:2015 Certified Institute, Permanently Affiliated to JNTUK, KAKINADA

All the UG Programmes are accredited by NBA Under Tier-1

Course Code: 20EC3551 **III Year I Semester**

INDEX

Experiment - 1

Pulse Code Modulation & Demodulation

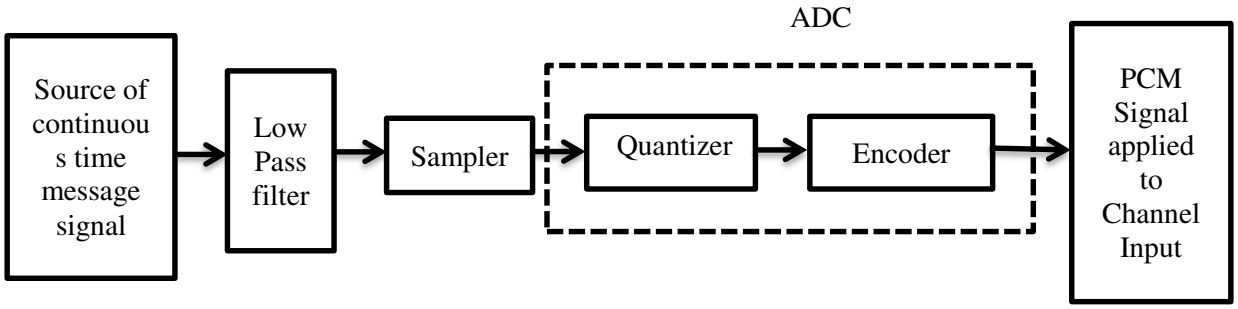
Aim:

Generate Pulse code Modulation and Demodulation.

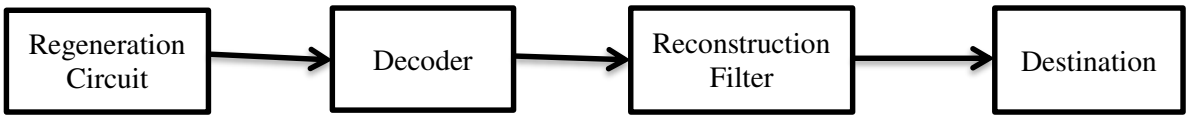
Experimental Requirements:

PC Loaded with MATLAB

Block Diagram:



(a) Transmission Path



(b) Receiver

MatLab Program:

% Pulse code modulation & Demodulation

```
clc;
clear all;
close all;
a=4;
fm=2;
fs=100*fm;
t=0:1/fs:1;
x=a*sin(2*pi*fm*t);
subplot(5,1,1);
plot(t,x,'r');
xlabel('time');
ylabel('amplitude');
title('Original message signal');
subplot(5,1,2);
```

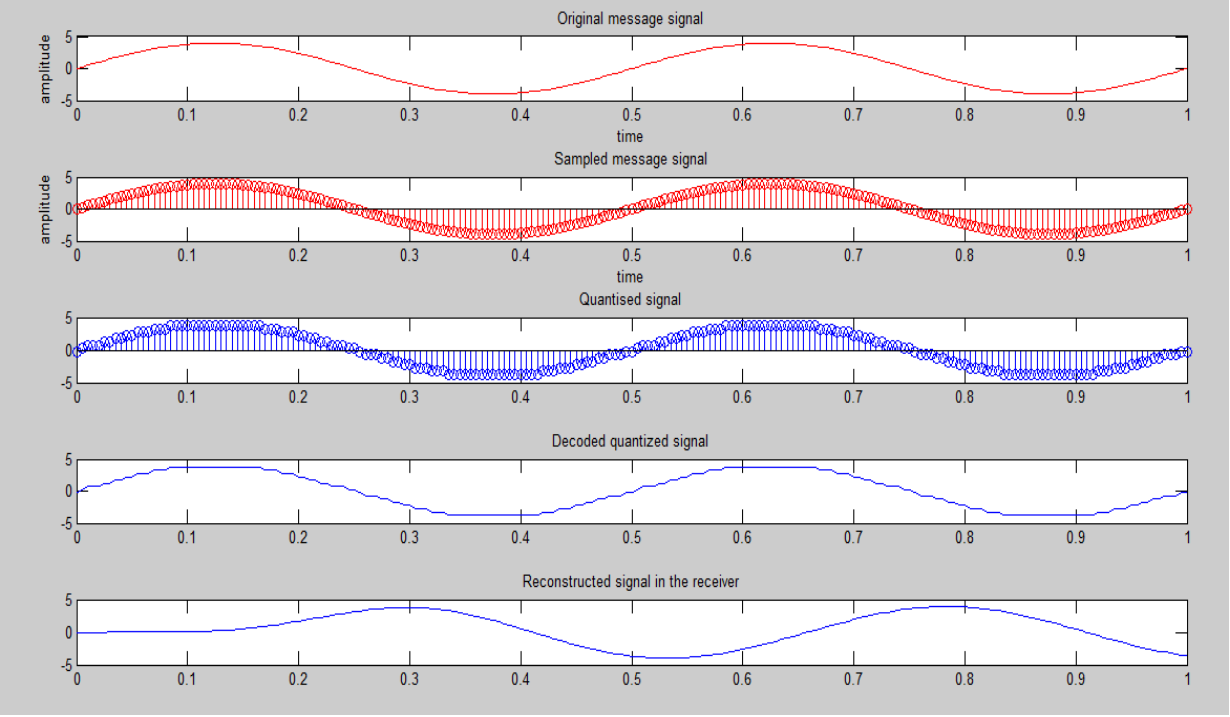
```
stem(t,x,'r');
xlabel('time');
ylabel('amplitude');
title('Sampled message signal');
enc=[];
for(i=1:length(x))
 if (x(i)>0 && x(i)<=1)
 e=[1 0 0];
 xq(i)=0.5;
 elseif (x(i)>1 && x(i)<=2)
 e=[1 0 1];
 xq(i)=1.5;
 elseif (x(i)>2 && x(i)<=3)
 e=[1 1 0];
 xq(i)=2.5;
 elseif (x(i)>3 && x(i)<=4)
 e=[1 1 1];
 xq(i)=3.5;
 elseif (x(i)>-4 && x(i)<=-3)
 e=[0 0 0];
 xq(i)=-3.5;
 elseif (x(i)>-3 && x(i)<=-2)
 e=[0 0 1];
 xq(i)=-2.5;
 elseif (x(i)>-2 && x(i)<=-1)
 e=[0 1 0];
 xq(i)=-1.5;
 else (x(i)>-1 && x(i)<=0)
 e=[0 1 1];
 xq(i)=-0.5;
 end
 enc=[enc e];
 end
 subplot(5,1,3);
 plot(t,xq,'b');
 title('Quantised signla');

% decoding(Receiver section)

 X_Q=[];
 for i=1:3:length(enc)-2
 if(enc(i)==0 && enc(i+1)==0 && enc(i+2)==0)
 x_q=-3.5;
 elseif(enc(i)==0 && enc(i+1)==0 && enc(i+2)==1)
 x_q=-2.5;
 elseif(enc(i)==0 && enc(i+1)==1 && enc(i+2)==0)
 x_q=-1.5;
 elseif(enc(i)==0 && enc(i+1)==1 && enc(i+2)==1)
 x_q=-0.5;
 elseif(enc(i)==1 && enc(i+1)==0 && enc(i+2)==0)
```

```
 x_q=0.5;
 elseif(enc(i)==1 && enc(i+1)==0 && enc(i+2)==1)
 x_q=1.5;
elseif(enc(i)==1 && enc(i+1)==1 && enc(i+2)==0)
 x_q=2.5;
 elseif(enc(i)==1 && enc(i+1)==1 && enc(i+2)==1)
 x_q=3.5;
 end
 X_Q=[X_Q x_q];
 end
 subplot(5,1,4);
 plot(t,X_Q);
 title('Decoded signal');
 [num,den]=butter(6,4*fm/fs);
 recon=filter(num,den,X_Q);
 subplot(5,1,5);
 plot(t,recon);
 title('Reconstructed signal in the receiver');
```

Wave forms:



Result:

Conclusion:

Experiment - 2

Delta Modulation

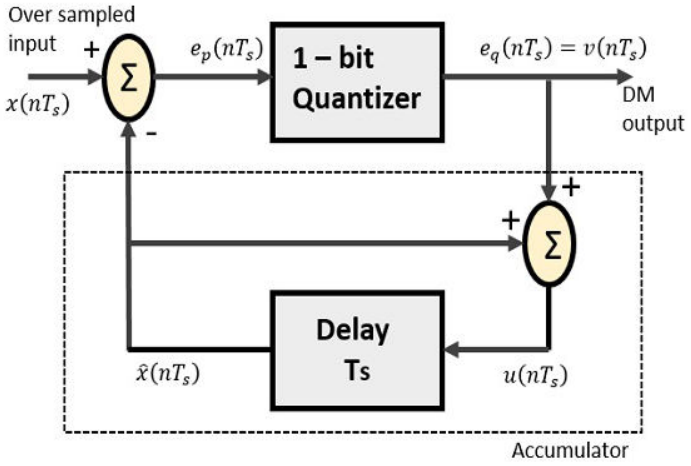
Aim:

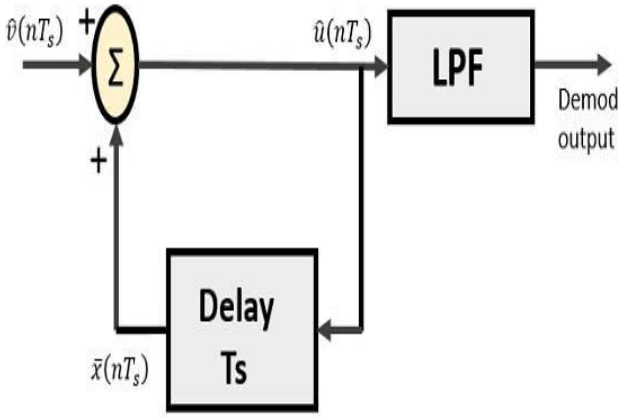
Implementation of Delta Modulated signal

Experimental Requirements:

PC Loaded with MATLAB

Block Diagram:





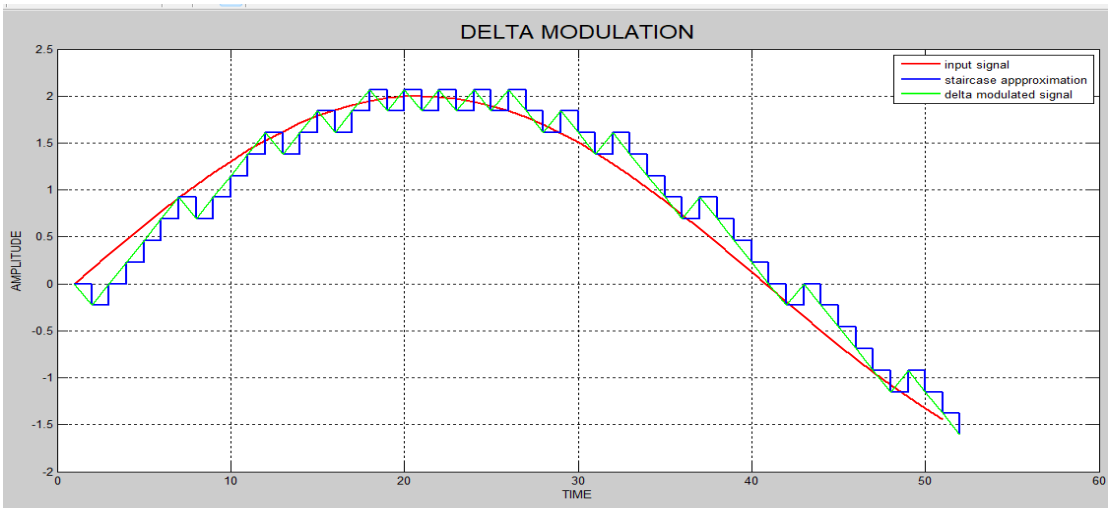
MatLab Program

CASE 1

```
clc;
clear
close all;
a=2;
t=0:2*pi/50:2*pi;
```

```
x=a*sin(2*pi/10*t);
l=length(x);
plot(x,'r','linewidth',2);
delta=0.2;
hold on
xn=0;
for i=1:l
if x(i)>xn(i)
d(i)=1;
xn(i+1)=xn(i)+delta;
else
d(i)=0;
xn(i+1)=xn(i)-delta;
end
end
stairs(xn,'b','linewidth',2)
hold on
plot(xn,'g','linewidth',2);
xlabel('TIME');
ylabel('AMPLITUDE');
title('DELTA MODULATION','fontsize',18);
legend('input signal','staircase appproximaton','delta modulated signal');
grid on
```

Waveform:

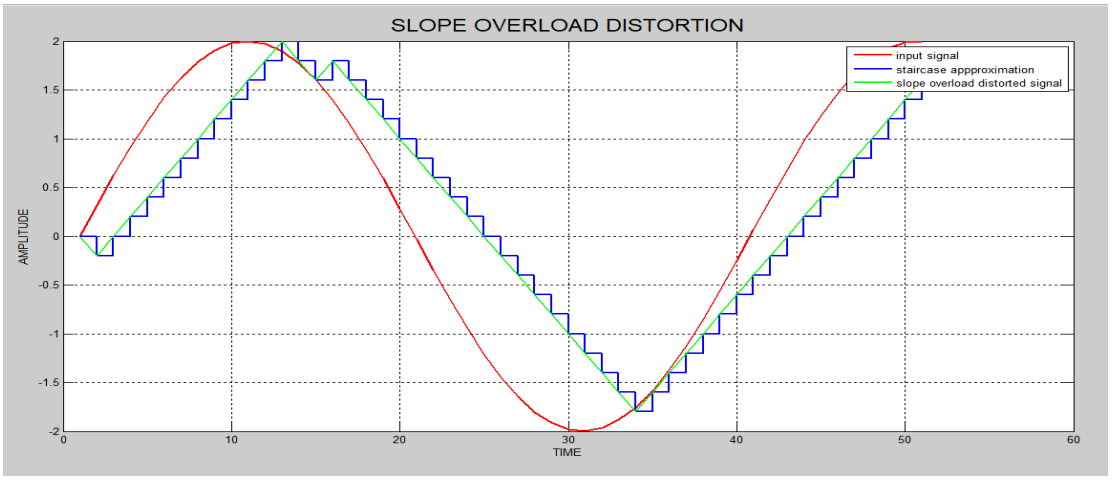


CASE 2

```
clc;
clear
close all;
a=2;
t=0:2*pi/50:2*pi;
x=a*sin(2*pi/5*t);
l=length(x);
```

```
plot(x,'r','linewidth',2);
delta=0.2;
hold on
xn=0;
for i=1:l
if x(i)>xn(i)
d(i)=1;
xn(i+1)=xn(i)+delta;
else
d(i)=0;
xn(i+1)=xn(i)-delta;
end
end
stairs(xn,'b','linewidth',2)
hold on
plot(xn,'g','linewidth',2);
xlabel('TIME');
ylabel('AMPLITUDE');
title('SLOPE OVERLOAD DISTORTION','fontsize',18);
legend('input signal','staircase appproximaton','slope overload distortedsignal');
grid on
```

Waveform:

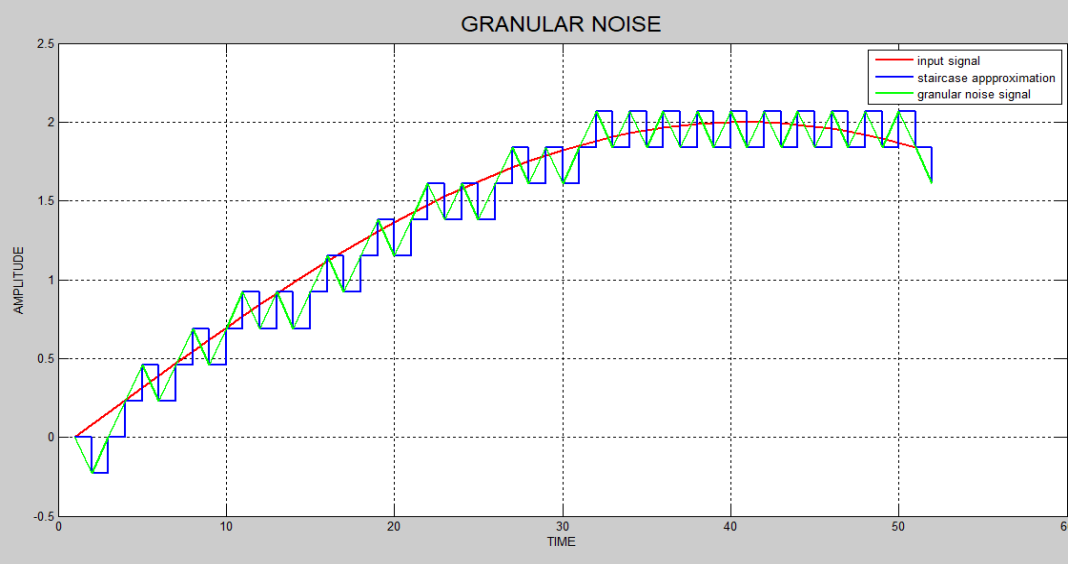


CASE 3

```
clc;
clear
close all;
a=2;
t=0:2*pi/50:2*pi;
x=a*sin(2*pi/20*t);
l=length(x);
plot(x,'r','linewidth',2);
delta=0.23;
```

```
hold on
xn=0;
for i=1:l
if x(i)>xn(i)
d(i)=1;
xn(i+1)=xn(i)+delta;
else
d(i)=0;
xn(i+1)=xn(i)-delta;
end
end
stairs(xn,'b','linewidth',2)
hold on
plot(xn,'g','linewidth',2);
xlabel('TIME');
ylabel('AMPLITUDE');
title('GRANULAR NOISE','fontsize',18);
legend('input signal','staircase appproximation','delta modulated signal');
grid on;
```

Waveform:



Result:

Conclusion:

Experiment - 3

Binary Phase Shift Keying

Aim:

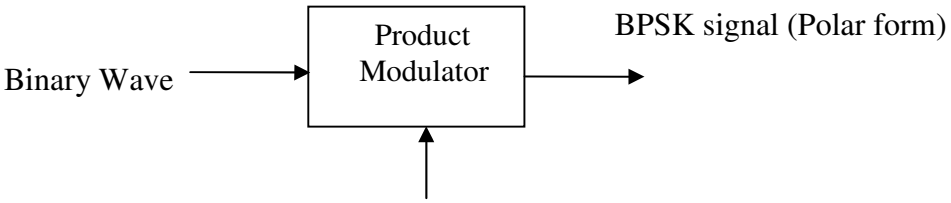
To generate and demodulate Binary phase shift keyed (BPSK) signal using MATLAB

Experimental Requirements:

PC loaded with MATLAB

Block Diagram:

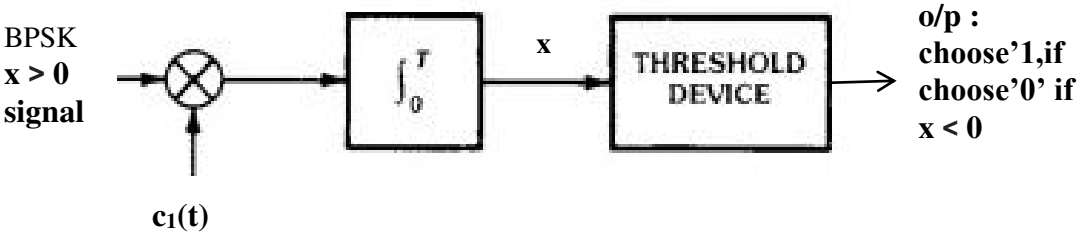
BPSK Transmitter:



$c_1(t) = \sqrt{2}/T_b \cos 2\pi f_c t$

BPSK Receiver:

BPSK Signal

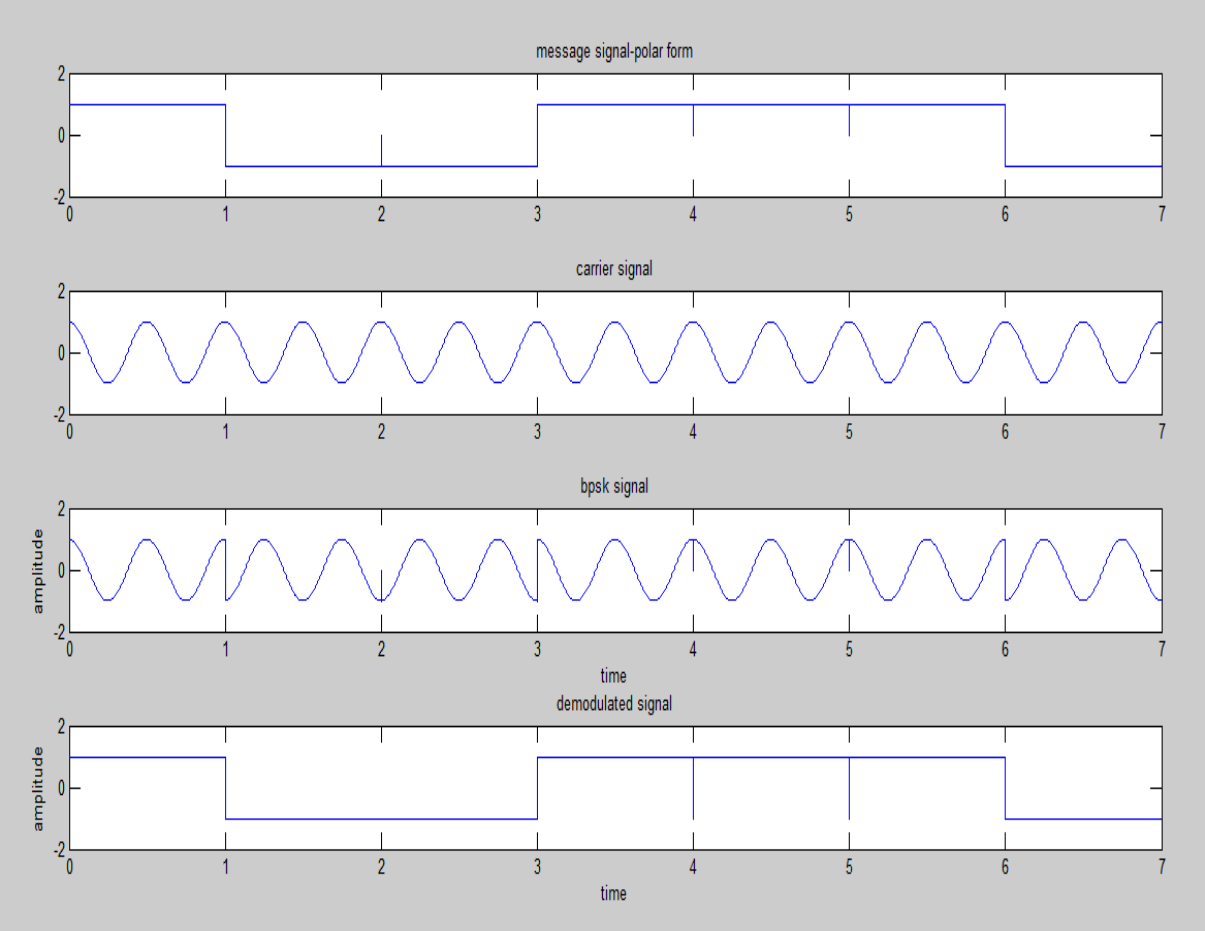


MatLab Program:

```
clc;
clear all;
close all;
N=7;
n=randi([10],1,N);
n=[1 0 0 1 1 1 0];
for ii=1:length(n)
if n(ii)==1
```

```
nn(ii)=1;
else
nn(ii)=-1;
end
end
i=1;
t=0:0.0001:length(n);
for j=1:length(t)
if t(j)<=i
y(j)=nn(i);
else
i=i+1;
end
end
subplot(4,1,1);
plot(t,y);
axis([0,length(n) -2 2]);
title('message signal-polar form');
c1=cos(2*pi*2*t);
subplot(4,1,2);
plot(t,c1);
axis([0,length(n) -2 2])
title('carrier signal')
x=y.*c1;
subplot(4,1,3);
plot(t,x);
axis([0,length(n) -2 2])
xlabel('time');
ylabel('amplitude');
title('bpsk signal');
for j=1:length(t)
if x(j)==c1(j)
det(j)=1;
else
det(j)=-1;
end
end
subplot(4,1,4);
plot(t,det);
axis([0,length(n) -2 2])
xlabel('time');
ylabel('amplitude');
title('demodulated signal');
```

Waveforms:



Result:

Conclusion:

Experiment - 4

Binary Frequency Shift Keying

Aim:

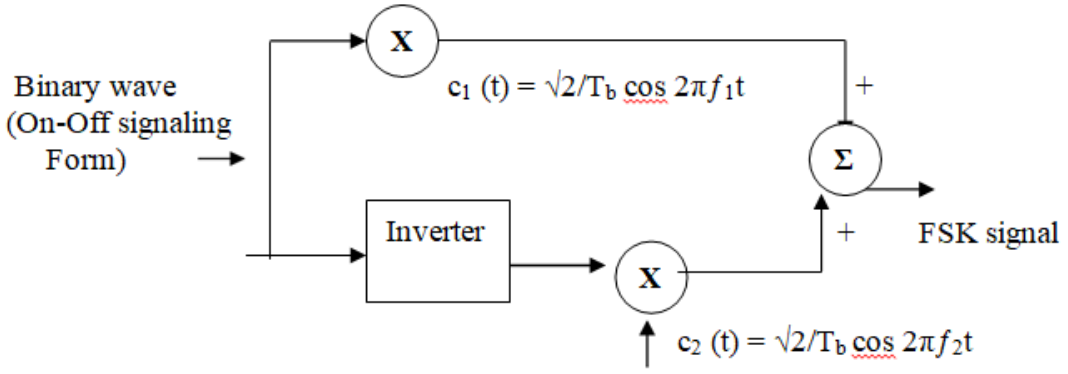
To generate and demodulate Binary Frequency shift keyed (BFSK) signal using MATLAB

Experimental Requirements:

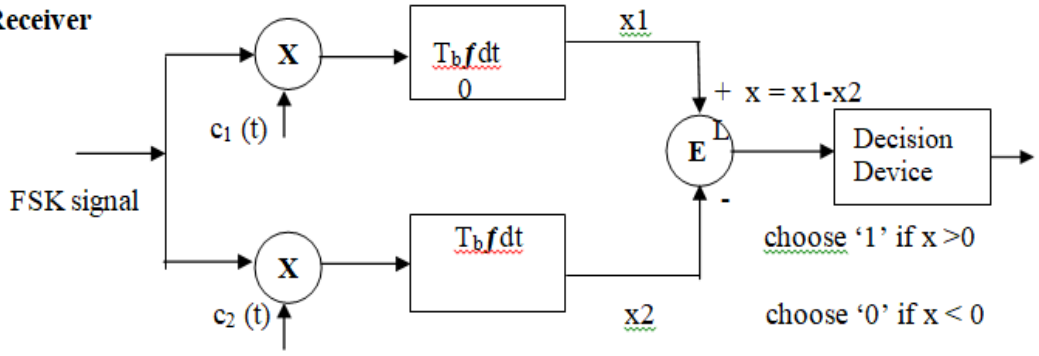
PC loaded with MATLAB

Block Diagram:

BFSK Transmitter



BFSK Receiver

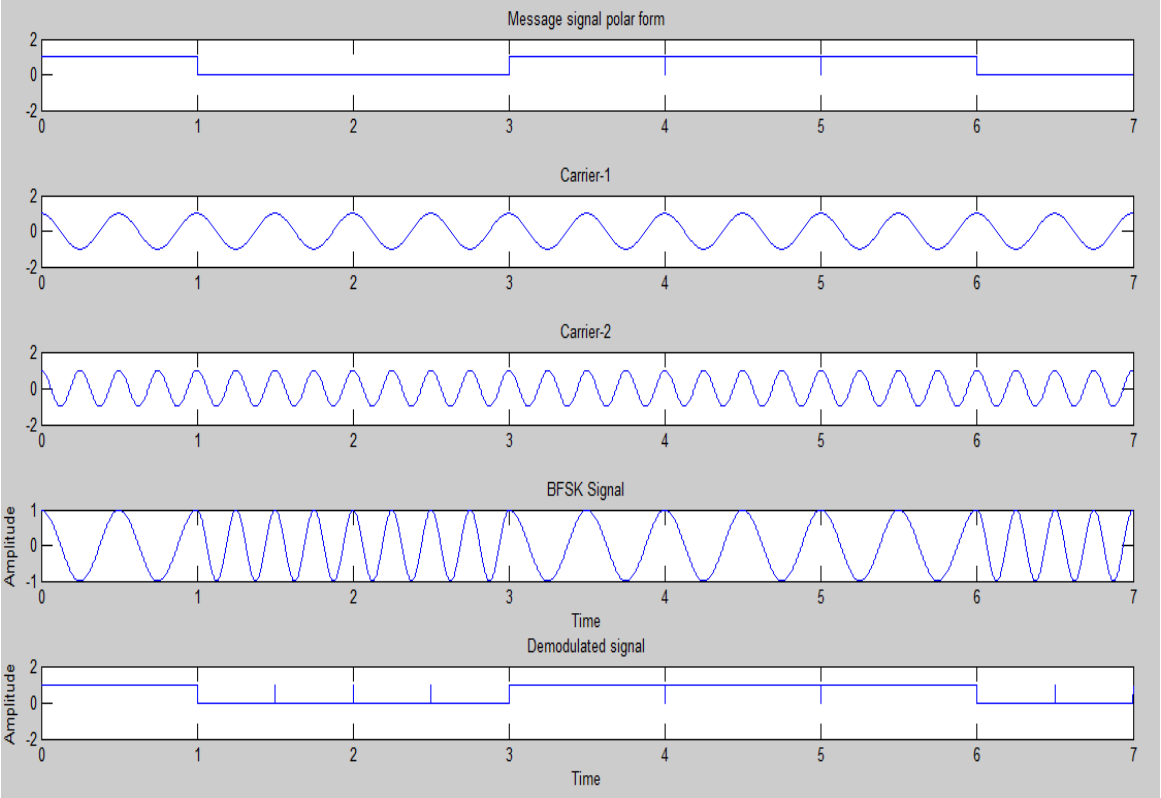


MatLab Program:

```
clc;
clear all;
close all;
%NRZ polar line coding
%input seqeunce
N=10;
n=randi([0 1] ,1,N);
%polar mapping
for ii=1:length(n)
if n(ii)==1;
nn(ii)=1
else
nn(ii)=-1;
end
end
%pulse shaping
i=1
t=0:0.01:length(n);
for j=1:length(t)
if t(j)<=i
y(j)=nn(i);
else
i=i+1;
end
end
%plotting
figure(1)
subplot(2,1,1);
plot(t,y);
title('message signal');
%carrier signal
c=cos(2*pi*2*t);
figure(1)
subplot(2,1,2);
plot(t,c);
title('carrier signal');
%BPSK modulation
x=y.*c;
figure(2)
subplot(2,1,1);
plot(t,x);
title('BPSK signal');
%detection and construction
for j=1:length(t)
if x(j)==c(j)
det(j)=1;
else
det(j)=0;
end
```

```
end
figure(2)
subplot(2,1,2);
plot(t,det);
title('demodulation');
```

Waveforms:



Result:

Conclusion:

Experiment - 5

Differential Phase Shift Keying (DPSK)

Aim:

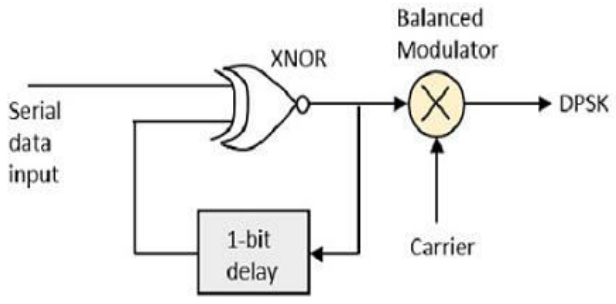
To verify the bit error rate performance of DPSK Modulatioin using MATLAB

Experimental Requirements:

PC loaded with MATLAB software

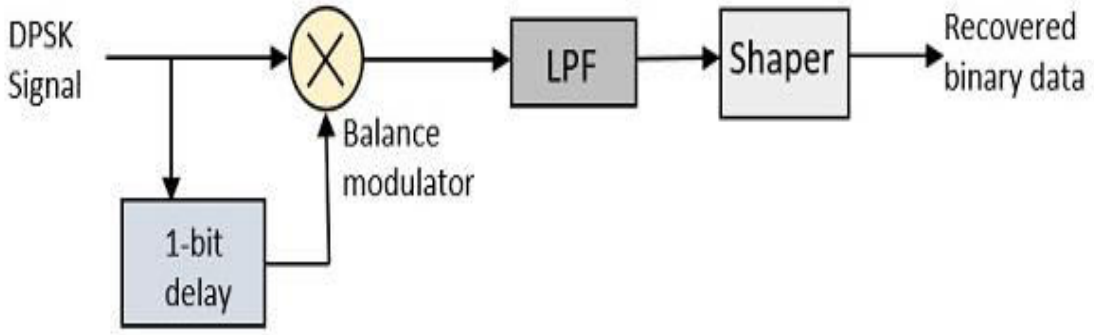
Block Diagram:

Modulator:



DPSK Modulator

Demodulator:



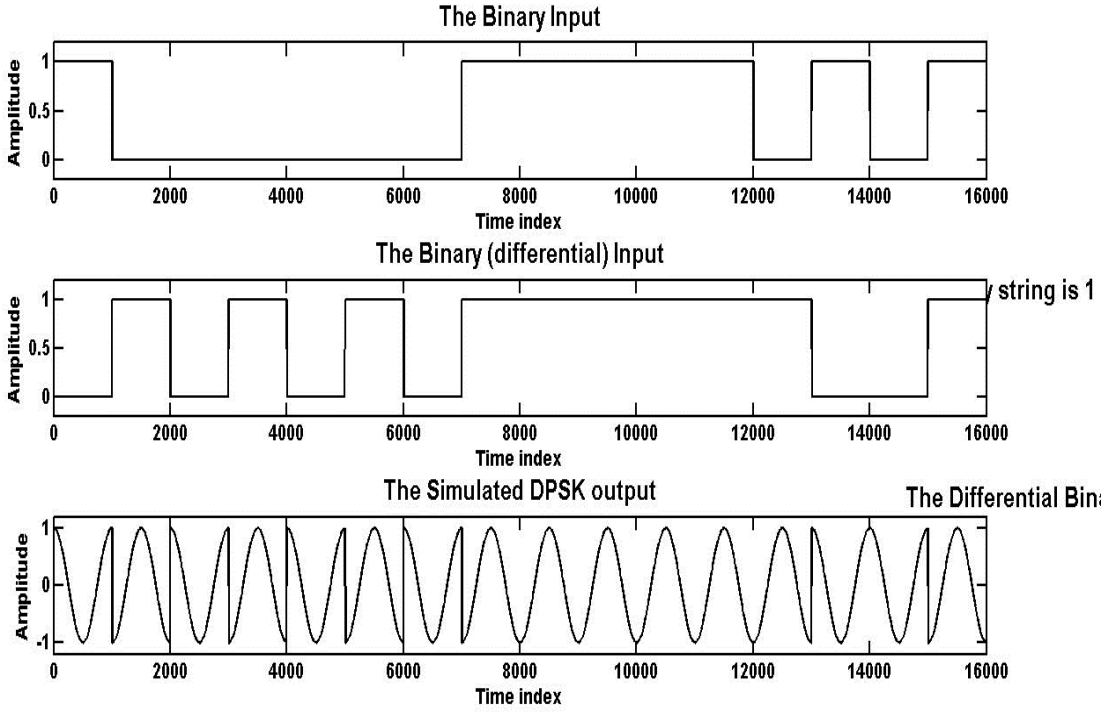
MatLab Program:

DPSK Program

```
clear all;
close all;
clc;
fc=1;% carrier frequency;
samp=1000;
t=linspace(0,2*pi,samp);
```

```
ph1=cos(fc*t);
ph2=-cos(fc*t);
b=[ 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1];
x=1;
nb(1)=not(xor(b(1),0));
for i=2:length(b)+1
nb(i)=not(xor(b(i-1),x));
x=nb(i);
end;
dpsk=[];bin1=[];bin2=[];for j=1:length(nb)
if nb(j)==0
dpsk=[dpsk,ph1];
bin1=[bin1,zeros(1,samp)];
elseif nb(j)==1
dpsk=[dpsk,ph2];
bin1=[bin1,ones(1,samp)];
end;%% end of if..
end;%% end of for
for k=1:length(b)
if b(k)==0
bin2=[bin2,zeros(1,samp)];
elseif b(k)==1
bin2=[bin2,ones(1,samp)];
end;
end;
subplot(311),plot(bin2,'k','LineWidth',3');
axis([0 samp*length(b) -0.2 1.2]);
xlabel('Time index'); ylabel('Amplitude');
title('The Binary Input','FontSize',12);
bn=num2str(b);
bx=['The Binary string is ',bn];
gtext(bx,'FontSize',12);
subplot(312),plot(bin1,'k','LineWidth',3');
axis([0 samp*length(b) -0.2 1.2]);
xlabel('Time index'); ylabel('Amplitude');
title('The Binary (differential) Input','FontSize',12);
bn=num2str(nb);
bx=['The Differential Binary string is ',bn];
gtext(bx,'FontSize',12);
subplot(313),plot(dpsk,'k','LineWidth',3');
axis([0 samp*length(b) -1.2 1.2]);
xlabel('Time index'); ylabel('Amplitude');
title('The Simulated DPSK output','FontSize',12);
```

Waveforms:



Result:

Conclusion:

Experiment - 6

Direct Sequence Spread Spectrum

Aim:

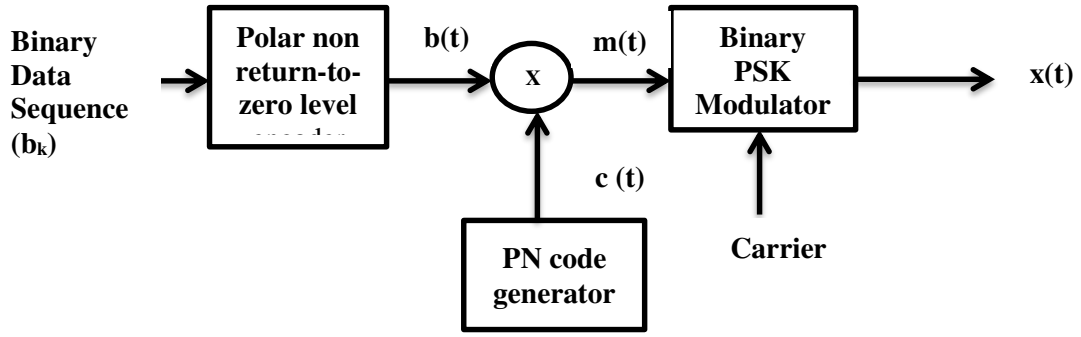
To implement Direct sequence spread spectrum

Experimental Requirements:

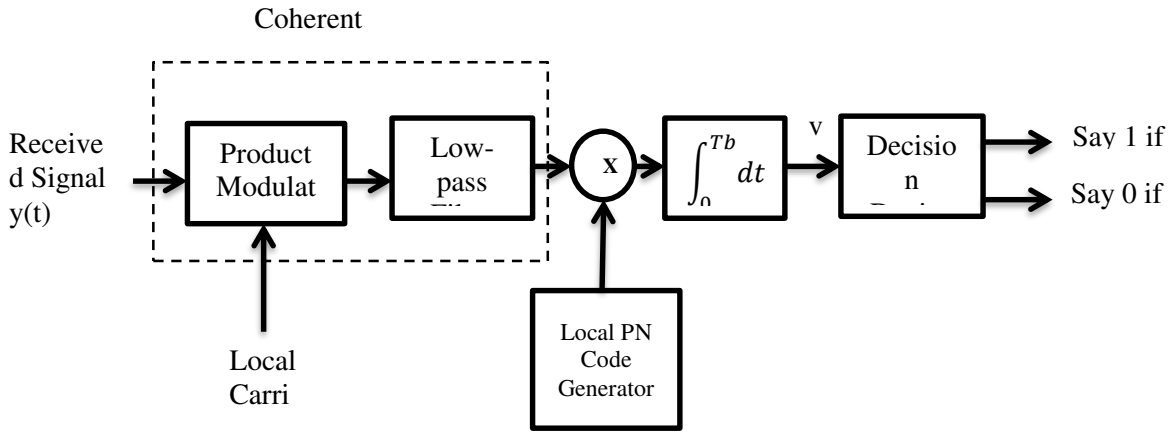
PC loaded with MATLAB

Block Diagram:

Transmitter



Receiver



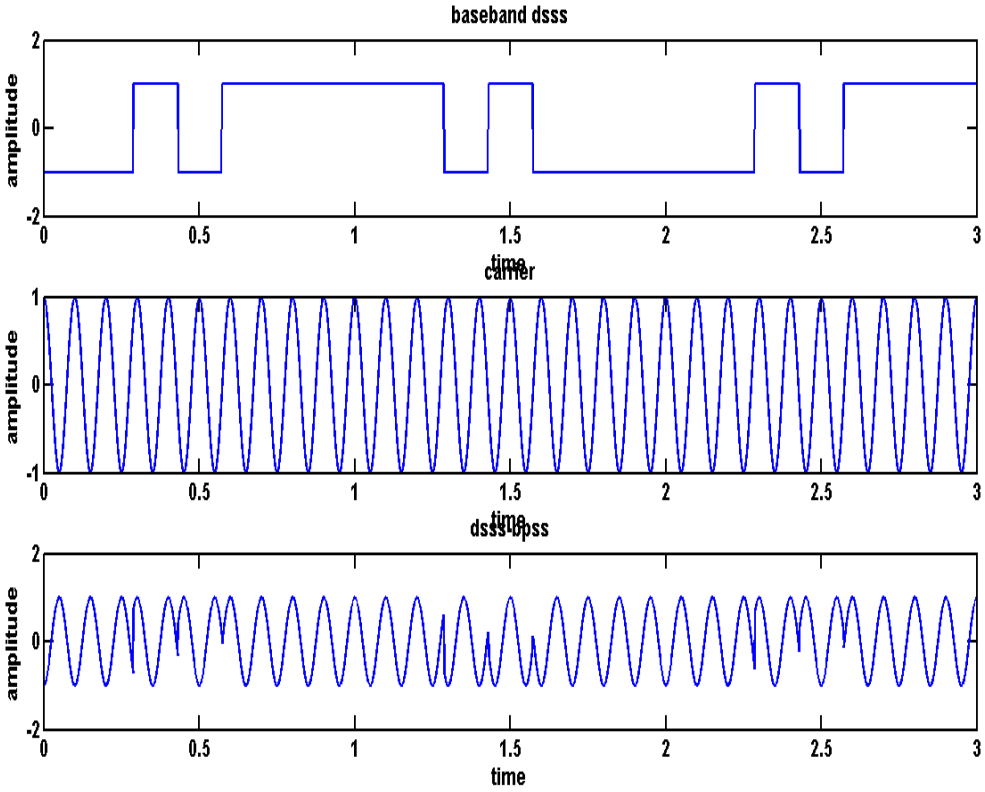
MatLab Program:

% Direct Sequence Spread Spectrum

```
clc;
clear all;
close all;
N=7;
c=[0 0 1 0 1 1 1];
for i=1:length(c)
if c(i)==0
cm(i)=-1;
else
cm(i)=1;
end
end
b=randi([01],1,5);
b=[1 0 1];
m=[];
for k=1:length(b)
if b(k)==0
mm=-cm;
else
mm=cm;
end
m=[m mm];
end
i=1;
h=(1/N);
s=700;
t=0:1/s:length(b)-1/s;
for j=1:length(t)
if t(j)<=h
y(j)= m(i);
else
y(j)=m(i);
i=i+1;
h=h+1/N;
end
end
subplot(3,1,1);
plot(t,y);
xlabel('time');
ylabel('amplitude');
title('baseband dsss');
axis([0 length(b) -2 2])
c1=cos(2*pi*10*t);
subplot(3,1,2);
plot(t,c1);
xlabel('time');
ylabel('amplitude');
```

```
title('carrier');
x=y.*c1;
subplot(3,1,3);
plot(t,x);
xlabel('time');
ylabel('amplitude');
axis([0 length(b) -2 2])
title('dsss-bpss');
```

Waveforms:



Result:

Conclusion:

Experiment - 7

Frequency Hopping Spread Spectrum

Aim:

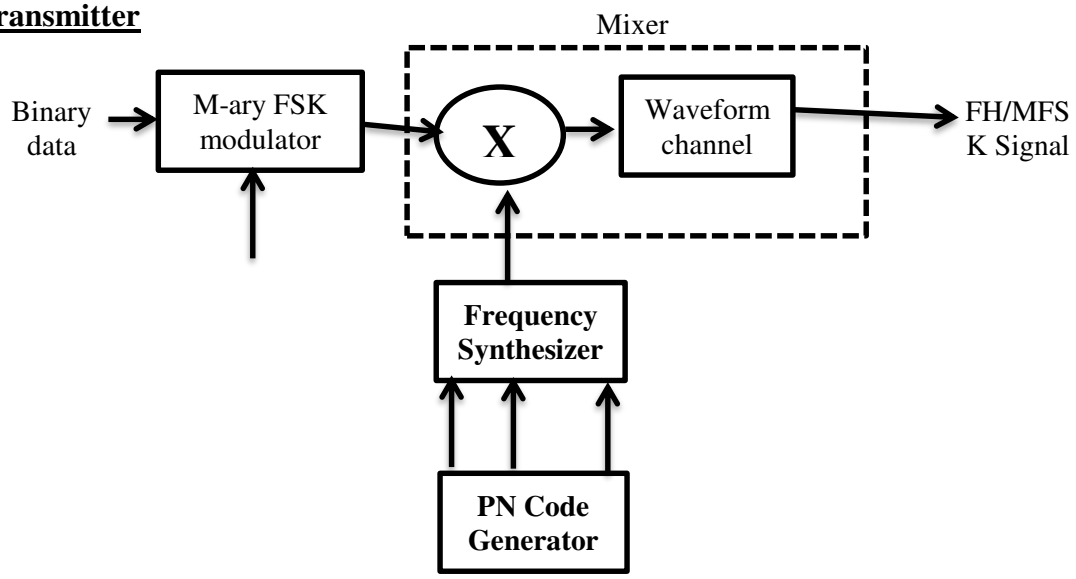
To implement frequency hopping spread spectrum in MATLAB

Experimental Requirements:

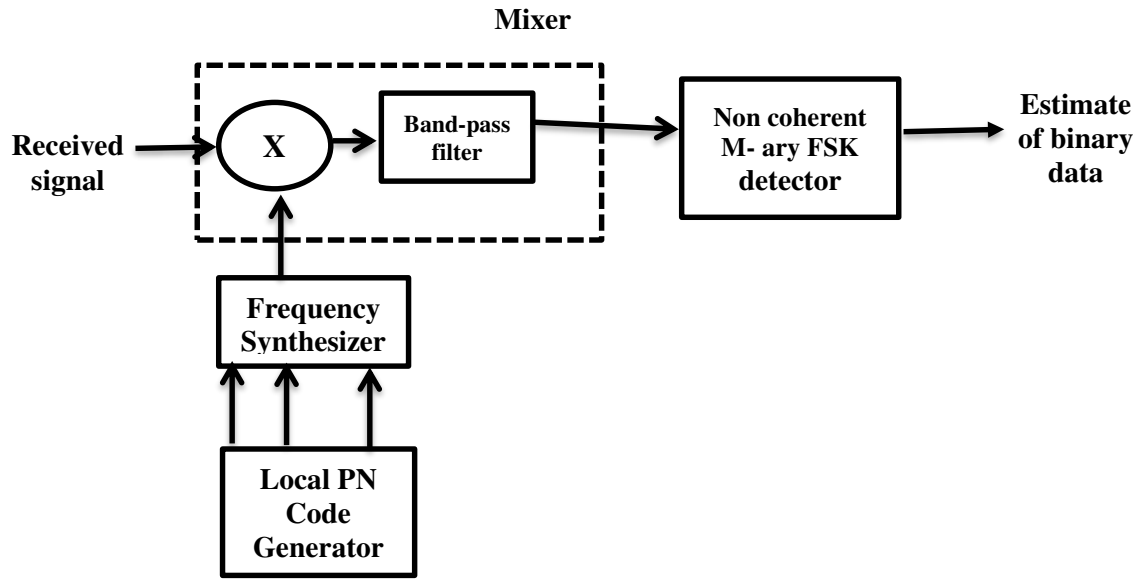
PC loaded with MATLAB

Block Diagram:

Transmitter



Receiver

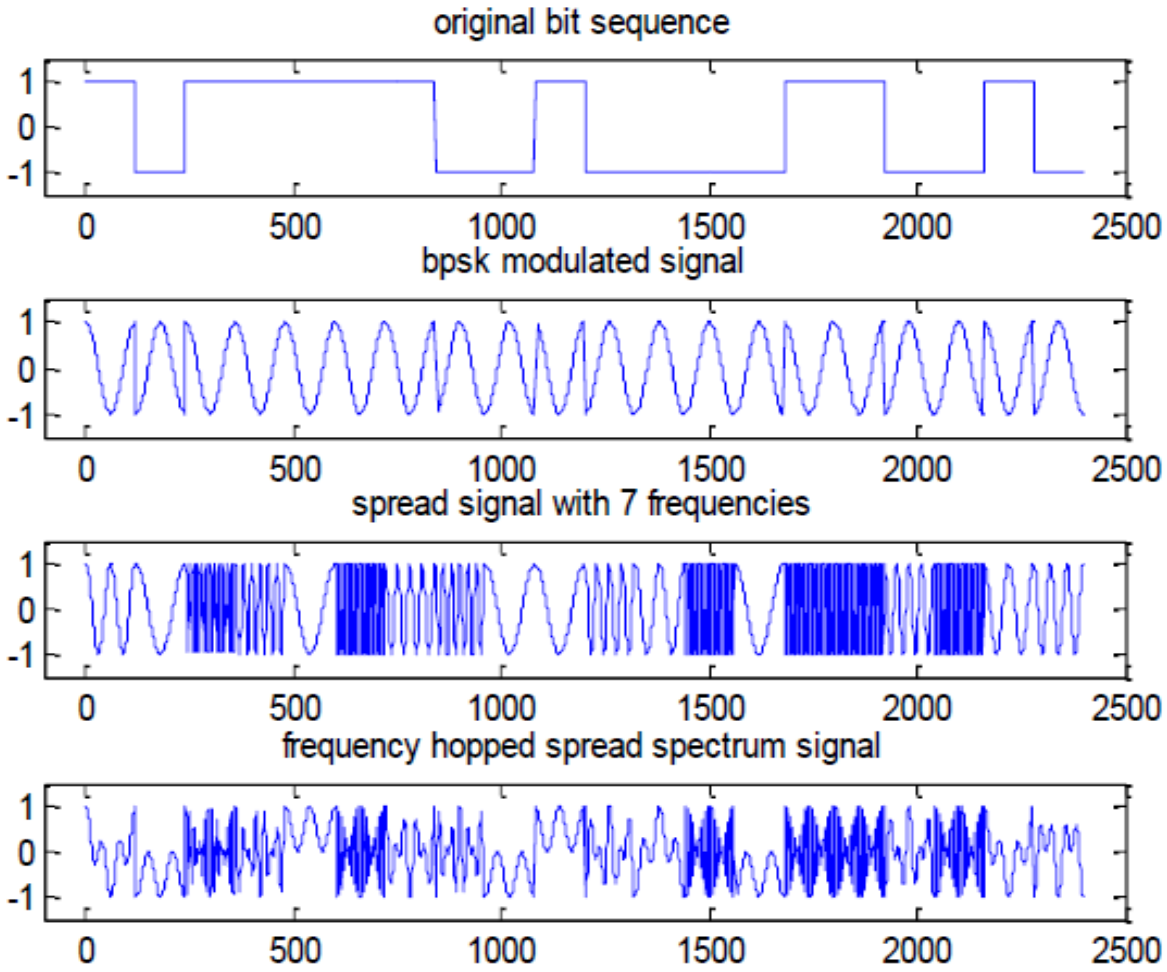


MatLab Program:

```
clc;
close all;
clear all;
even=1;
odd=1;
cp=randint(1,20,[1 7]);
for k=1:20
 if(mod(k,2))
 dualcode(1,k)=cp(1,k);
 evenbits(1,even)=cp(1,k);
 even=even+1;
 else
 oddbits(1,odd)=cp(1,k);
 odd=odd+1;
 const=bin2gray(cp(1,k),'psk',16);
 dualcode(1,k)=const;
 end
end
s=round(rand(1,20));
signal=[];
carrier=[];
t=[0:2*pi/119:2*pi];
for k=1:20
 if s(1,k)==0
 sig=-ones(1,120);
 else
 sig=ones(1,120);
 end
 c=cos(t);
 carrier=[carrier c];
 signal=[signal sig];
end
subplot(4,1,1);
plot(signal);
axis([-100 2500 -1.5 1.5]);
title('original bit sequence');
bpsk_sig=signal.*carrier;
subplot(4,1,2);
plot(bpsk_sig);
axis([-100 2500 -1.5 1.5]);
title('bpsk modulated signal');
t0=[0:2*pi/4:2*pi];
t1=[0:2*pi/9:2*pi];
t2=[0:2*pi/19:2*pi];
t3=[0:2*pi/29:2*pi];
t4=[0:2*pi/39:2*pi];
t5=[0:2*pi/59:2*pi];
t6=[0:2*pi/119:2*pi];
c0=cos(t0);
```

```
c0=[c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0 c0];
c1=cos(t1);
c1=[c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1];
c2=cos(t2);
c2=[c2 c2 c2 c2 c2 c2];
c3=cos(t3);
c3=[c3 c3 c3 c3];
c4=cos(t4);
c4=[c4 c4 c4];
c5=cos(t5);
c5=[c5 c5];
c6=cos(t6);
spread_signal=[];
for n=1:20
 c=dualcode(1,n);
 switch(c)
 case(1)
 spread_signal=[spread_signal c0];
 case(2)
 spread_signal=[spread_signal c1];
 case(3)
 spread_signal=[spread_signal c2];
 case(4)
 spread_signal=[spread_signal c3];
 case(5)
 spread_signal=[spread_signal c4];
 case(6)
 spread_signal=[spread_signal c5];
 case(7)
 spread_signal=[spread_signal c6];
 end
end
subplot(4,1,3);
plot([1:2400],spread_signal);
axis([-100 2500 -1.5 1.5]);
title('spread signal with 7 frequencies');
freq_hopped_sig=bpsk_sig.*spread_signal;
subplot(4,1,4);
plot([1:2400],freq_hopped_sig);
axis([-100 2500 -1.5 1.5]);
title('frequency hopped spread spectrum signal');
```

Waveforms:



Result:

Conclusion:

Experiment - 8

Implementation of Huffman Coding Using MatLab

Aim:

To implement Huffman coding using MATLAB

Experimental Requirements:

PC loaded with MATLAB software

MatLab Program:

```
clc; clearall;

s=input('Enter symbols- ')          %format ['a','b','c','d','e','f'];
p=input('Enter value of probabilty- ') %format [0.22,0.20,0.18,0.15,0.13,0.12];
if length(s)~=length(p)
error('Wrong entry.. enter again- ') end
i=1;
for m=1:length(p) for n=1:length(p) if(p(m)>p(n))
a=p(n); a1=s(n); p(n)=p(m);s(n)=s(m); p(m)=a; s(m)=a1;
end end end
display(p) %arranged prob. in descending order.
tempfinal=[0]; sumarray=[]; w=length(p); lengthp=[w]; b(i,:)=p;
while(length(p)>2) tempsum=p(length(p))+p(length(p)-1);
sumarray=[sumarray,tempsum];
p=[p(1:length(p)-2),tempsum]; p=sort(p,'descend');
i=i+1;
b(i,:)=[p,zeros(1,w-length(p))]; w1=0;
lengthp=[lengthp,length(p)];
for temp=1:length(p) if p(temp)==tempsum;
w1=temp;
end end
tempfinal=[w1,tempfinal]; % Find the place where tempsum has been inserted
display(p); end

sizeb(1:2)=size(b); tempdisplay=0;
temp2=[];
for i= 1:sizeb(2) temp2=[temp2,b(1,i)];
end sumarray=[0,sumarray]; var=[];
e=1;
for ifinal= 1:sizeb(2) code=[s(ifinal),'    ']
for j=1:sizeb(1) tempdisplay=0;
for i1=1:sizeb(2)
if( b(j,i1)==temp2(e)) tempdisplay=b(j,i1); end
if(tempdisplay==0 & b(j,i1)==sumarray(j)) tempdisplay=b(j,i1);
end end
var=[var,tempdisplay];
```

```
if tempdisplay==b(j,lengthp(j))  %assign 0 & 1 code=[code,'1'];
elseif tempdisplay==b(j,lengthp(j)-1) code=[code,'0'];
else code=[code,'']; end
temp2(e)=tempdisplay; end
display(code) %display final codeword e=e+1
end
```

Result:

Conclusion:

Experiment - 9

Shannon Fano Coding

Aim:

To implement Shannon Fano Coding in MATLAB

Experimental Requirements:

PC loaded with MATLAB

MatLab Program:

```
clc;
clear all;
close all;
m=input('Enter the no. of message ensembles : ');
z=[];

h=0;l=0;
display('Enter the probabilities in descending order');
for i=1:m
fprintf('Ensemble %d\n',i);
p(i)=input('');
end
%Finding each alpha values
a(1)=0;
for j=2:m;
a(j)=a(j-1)+p(j-1);
end
fprintf('\n Alpha Matrix');
display(a);
%Finding each code length
for i=1:m
n(i)= ceil(-1*(log2(p(i))));

end
fprintf('\n Code length matrix');
display(n);
%Computing each code
for i=1:m
int=a(i);
for j=1:n(i)
frac=int*2;
c=floor(frac);
frac=frac-c;
z=[z c];
int=frac;
end
```

```
fprintf('Codeword %d',i);
display(z);
z=[];

end
%Computing Avg. Code Length & Entropy
fprintf('Avg. Code Length');
for i=1:m
x=p(i)*n(i); l=l+x;
x=p(i)*log2(1/p(i));
h=h+x;
end
display(l);
fprintf('Entropy');
display(h);
%Computing Efficiency
fprintf('Efficiency');
display(100*h/l);
fprintf('Redundancy');
display(100-(100*h/l));
```

Result:

Conclusion:

Experiment - 10

Linear Block Codes

Aim:

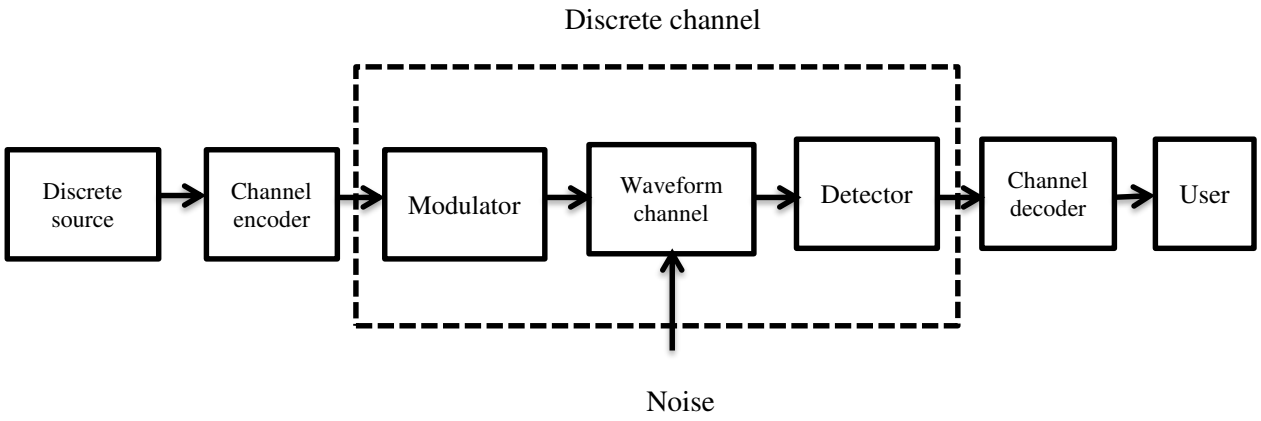
To implement Linear Block Codes in MATLAB

Experimental Requirements:

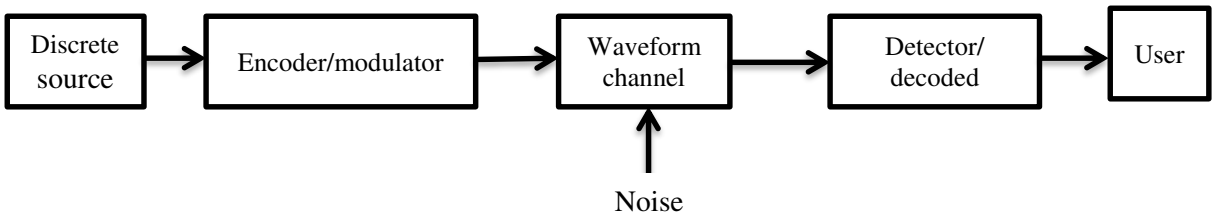
PC loaded with MATLAB

Block Diagram:

Coding and modulation performed separately



Coding and modulation combined



MatLab Program:

```
% Given H Matrix

H = [1 0 1 1 1 0 0;
     1 1 0 1 0 1 0;
     0 1 1 1 0 0 1]

k = 4;
n = 7;

% Generating G Matrix

% Taking the H Matrix Transpose
P = H';

% Making a copy of H Transpose Matrix
L = P;

% Taking the last 4 rows of L and storing
L((5:7), : ) = [];

% Creating a Identity matrix of size K x K
I = eye(k);

% Making a 4 x 7 Matrix
G = [I L]

% Generate U data vector, denoting all information sequences
no = 2 ^ k

% Iterate through an Unit-Spaced Vector
for i = 1 : 2^k

% Iterate through Vector with Specified Increment
% or in simple words here we are decrementing 4 till we get 1
for j = k : -1 : 1
if rem(i - 1, 2 ^ (-j + k + 1)) >= 2 ^ (-j + k)
u(i, j) = 1;
else
u(i, j) = 0;
end

% To avoid displaying each iteration/loop value
echo off;
end
end

echo on;
u
```

```
% Generate CodeWords
c = rem(u * G, 2)

% Find the min distance
w_min = min(sum((c(2 : 2^k, :))'))

% Given Received codeword
r = [0 0 0 1 0 0 0];
r

p = [G(:, n - k + 2 : n)];

%Find Syndrome
ht = transpose(H)

s = rem(r * ht, 2)

for i = 1 : 1 : size(ht)
if(ht(i,1:3)==s)
r(i) = 1-r(i);
break;
end
end

disp('The Error is in bit:')
disp(i)

disp('The Corrected Codeword is :')
disp(r)
```

Result:

Conclusion:

Experiment - 11

Implementation of Cyclic Code Encoder Using Matlab

Aim:

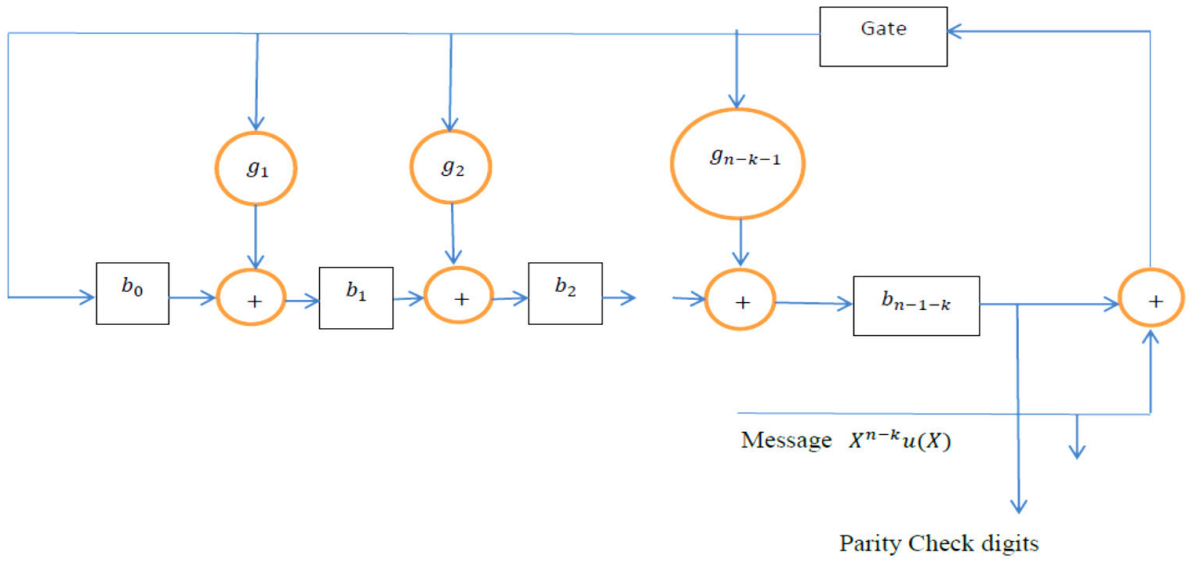
To implement cyclic code encoder using MATLAB

Experimental Requirements:

PC loaded with MATLAB software

Block diagram:

Cyclic Encoder



MatLab Program:

```
clc;
clearall;
sym=input('Enter the Message--- ');
l=length(sym); x=sort(sym);
a=seqwordcount(sym,'a'); %counting no.of character
h=seqwordcount(sym,'h');
i=seqwordcount(sym,'i');
m=seqwordcount(sym,'m');
s=seqwordcount(sym,'s');
b=seqwordcount(sym,'_');
prob_a=a./l; %Finding probability of each symbol
```

```
prob_h=h./l; prob_i=i./l;
prob_m=m./l; prob_s=s./l; prob_b=b./l;
msg=[1,2,3,4,5,6];
prob=[prob_a,prob_h,prob_i,prob_m,prob_s,prob_b];
dict = huffmandict(msg,prob); %arrange symbols according to probability
code_a = huffmanenco(1,dict); %Huffman coding of symbols
code_a = (code_a)';
code_h = huffmanenco(2,dict);
code_h= (code_h)';
code_i = huffmanenco(3,dict);
code_i = (code_i)';
code_m = huffmanenco(4,dict);
code_m = (code_m)';
code_s = huffmanenco(5,dict);
code_s = (code_s)';
code_b = huffmanenco(6,dict);
code_b = (code_b)';
for i=1:length(prob)
z(i)=prob(i).*log2(1./prob(i)); %Calculation of entropy
end entropy=sum(z)
code_array= [code_i code_b code_a code_m code_b code_a code_s code_h code_i code_s
code_h]
sz= size(code_array);
n=35; %code vector to b transmitted
k=29; %message-code_array
p=n-k; %parity bit
pol=cyclpoly(n,k);
[parmat,genmat]=cyclgen(n,pol);
msg= code_array;
codeword=mod(msg*genmat,2)
trt = syndtable(parmat); % Produce decoding table.
recd= input('enter the recieved vector- ');
syndrome = rem(recd*parmat',2);
syndrome_de = bi2de(syndrome,'left-msb'); % Convert to decimal.
```

```
errorvect = trt(1+syndrome_de,:);
correctedcode = rem(errorvect+recd,2);
if recd == codeword
disp('No Error- ')
disp(sym)
else
disp('Error in message....')
end
```

Result:

Conclusion:

Experiment - 12

Convolutional Codes

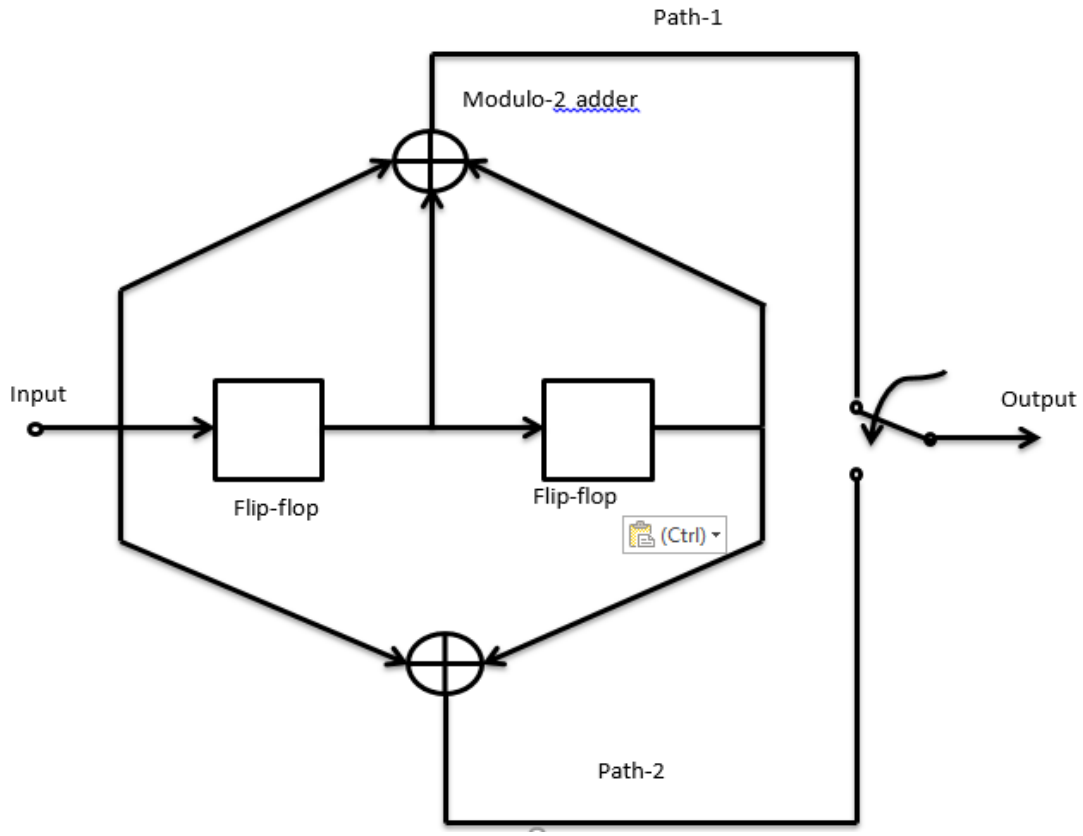
Aim:

To implement Convolutional encoder in MA TLAB

Experimental Requirements:

PC loaded with MATLAB

Block Diagram:



MatLab Program:

```
% (2,1,2) Convolutional Encoder
m=[1 0 1 1];% message sequence
p=2;% no of FFs
z=zeros(1,p);
mm=horzcat(m,z);% zeros added to message sequence
d1=0; d2=0; % Initializing content of FFs
x=[ ]; % FF states
c=[ ];% code vector
for i=1:1:length(mm)
d1(i+1)=mm(i)
d2(i+1)=d1(i);
x=[x; d1(i) d2(i)];% FF states are displayed
u(i)=xor(x(i,1), x(i,2));
c1(i)=xor(u(i), mm(i));
c2(i)=xor(mm(i),x(i,2));
c=[c; c1(i) c2(i)];
end
%  disp(' states of the shift Reg');
%   x
disp(' code vector');
c
```

Result:

Conclusion: