**EE323 Complex Engineering Problem - Fall 2020**

* Ayesha Ishaq 18i-0772
* Muhammad Soban Younis 18i-0889

**Abstract:**

In this report we will cover the simulation using MATLAB of FM transmitter and receiver for part 1, with which we transmit our own sound messages and demodulate on the receiver end. The carrier frequencies were chosen between 80MHz to 83MHz with guardbands of 200KHz between them. Multiplexer was also implemented to send the messages on a single carrier. Superheterodyne receiver was used with Intermediate frequency at 10.7MHz and envelope detector to obtain original message. This implementation required reconstruction of message signal to have samples at selected Fs (sampling frequency) for which sinc function was used. [Actual implementation was done at carrier frequency equal to roll number KHz due to processing problems for reconstruction]. In part 2 of the CEP, we implemented a digital transmitter that uses BPSK modulation scheme. Accordingly, the correlation-type demodulator was used. AWGN was also added to simulate the effects of noise.

We found our FM system to be largely accurate but our demodulation simulation added some noise to the output sound, using the fmdemod command allowed for even more accurate output. The accuracy of our digital system relied on the strength of added noise and the signal power.

**Introduction and background:**

*FM modulation*

Our simulation model is closely based on the actual fm modulator. In FM modulation the frequency of the carrier signal is changed according to the amplitude of the message signal, therefore incorporating the message into the frequency of the carrier. The deviation constant determines the maximum change in frequency of carrier from its original frequency and is important when determining the modulation scheme, that is, wide band or narrow band FM. The deviation constant also helps determine the effective bandwidth of the modulated signal [1].

*Superheterodyne Receiver*

The important theory used in the construction of Superheterodyne receiver is that of frequency mixing which is a nonlinear method that allows frequencies to be translated. The receiver captures signals at the tuned frequency, filters it and translates it to the intermediate frequency by mixing it with a locally generated signal at a frequency less than or greater than signal frequency by IF. In doing so it needs to only detect the message signal always at IF. After translating, generally an envelope detector is used to obtain the message signal [2].

*Frequency Demodulation*

In order to demodulate the FM signal, the message signal must be converted from the frequency changes to the amplitude changes. After an AM demodulator can be employed to extract the message. Practically different kind of demodulators are used include phase locked loop, but for the simulation FM to AM conversion was done using simple differentiation [3].

*BPSK Digital Modulation Scheme*

The theory of BPSK uses the same idea as angle modulation in analog systems, including the message in the phase of the carrier. In BPSK using mostly two basis or in the case of M=2 only one base, the entire signal space is generated. Each BPSK has the same energy and is therefore equally far from origin in the signal space representation. For M signals where each signal carries log2M bits, there are M points on the circle in signal representation space [4].

**System Model**

*CEP part 1*

**

Figure 1. CEP Phase 1 Block Diagram

In our simulation we have used the following model. Our first block shows that first we sample the message signals. These are done at the allowed Fs of MATLAB and our device’s microphone. Then we reconstruct the signal so we have sampling of message at our required Fs. This step is done using summation of each sample multiplied by shifted sinc function by the same amount. After reconstruction our signal is frequency modulated. For this step we integrate our message signal and add it to the instantaneous frequency of the carrier signal while multiplying it with the correct deviation constant. All signals are similarly modulated and multiplexed. On the receiver end, the signal is first bandpass filtered at the chosen carrier frequency and then multiplied with a signal with frequency FLO (Fc + FIF) hence translating it to FIF. The signal is then filtered again through a passband equal to effective bandwidth. It is then differentiated to bring the message into the amplitude of the carrier. Next step is the envelope detector: it is rectified and passed through a lowpass filter. The obtained signal is the integral of the original message, so the last step is differentiation and we then obtain the original signal.

*CEP part 2*

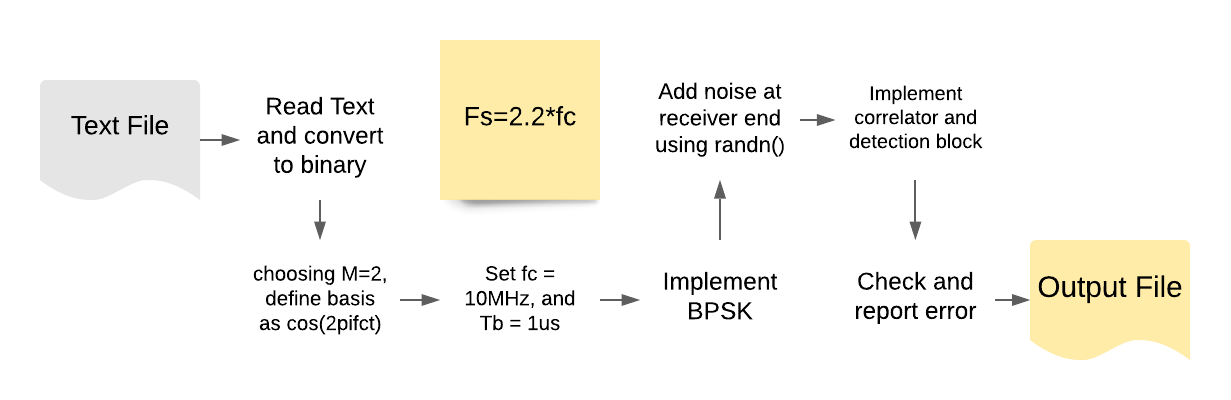


Figure 2. CEP Phase 2 Block Diagram

For the second part of the CEP, we have made the following model to simulate the BPSK scheme. First, we read text data from a text file, we then convert it to binary (the ASCII code representation). Having chosen M=2, that is one bit per signal, we have a signal base in the signal representation space. The representation of the bits 0 and 1 are 180 apart in phase which in cosine changes the sign of the signal. Therefore, each bit is represented by the appropriate signal and bit duration is chosen to be 1us. Next, AWGN is added to the modulated signal. At the demodulator end the signal is multiplied by the base function and the result is integrated. The result is used to determine whether received bit is 1 or 0.

**Equations and calculations:**

*CEP Part 1*

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

.

*CEP Part 2*

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

Receiver side:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

**Results and Discussion:**

*CEP Part 1*

*Magnitudes and Bandwidths of message signals*

**Note:** Message signals in figures are normalized.

Four audio messages are recorded at sampling frequency of 8KHz.

*Message 1:*

Max amplitude = 0.5737 V

Bandwidth = 3.3 K Hz

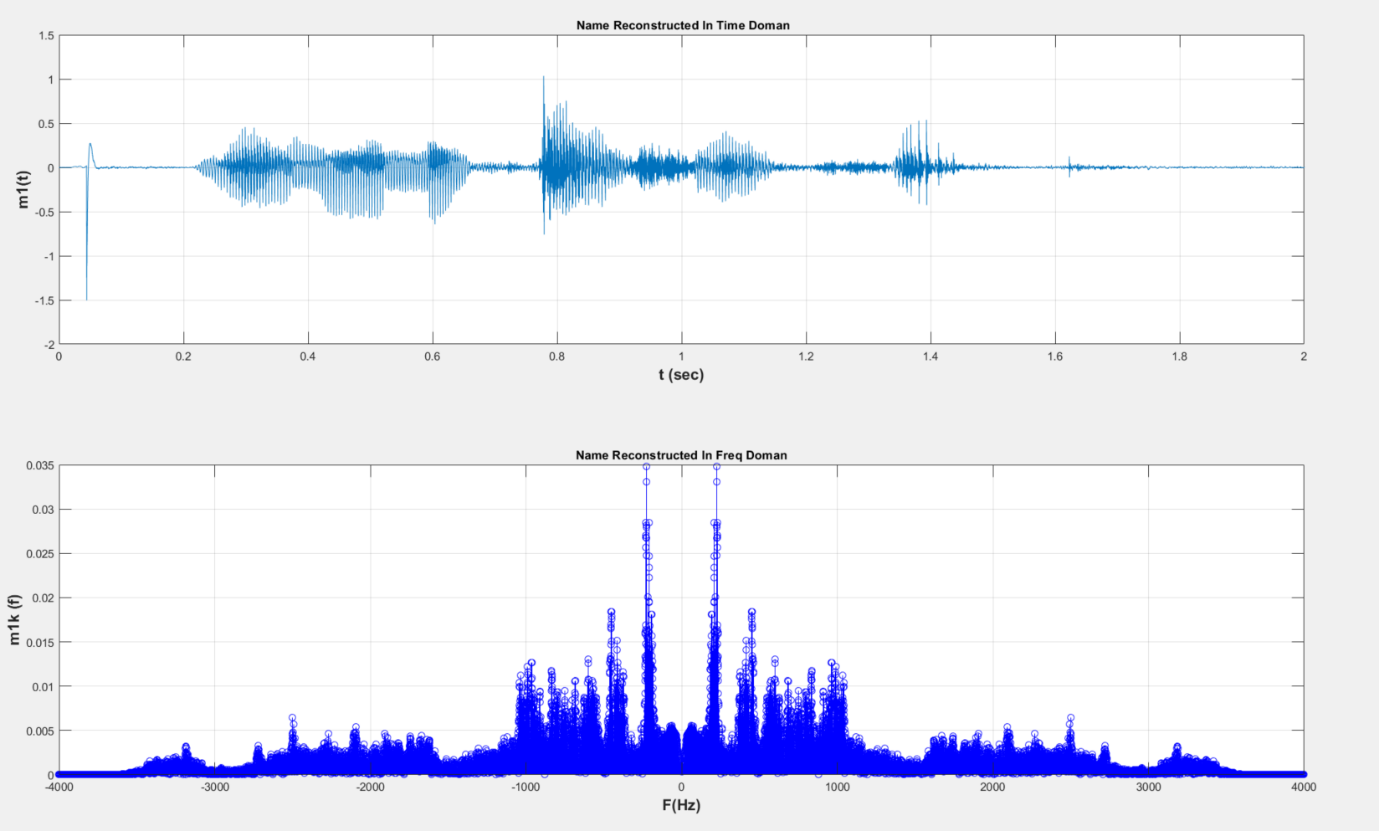


Figure 3. Message 1

*Message 2:*

Max amplitude = 0.4724 V

Bandwidth = 3.3 K Hz

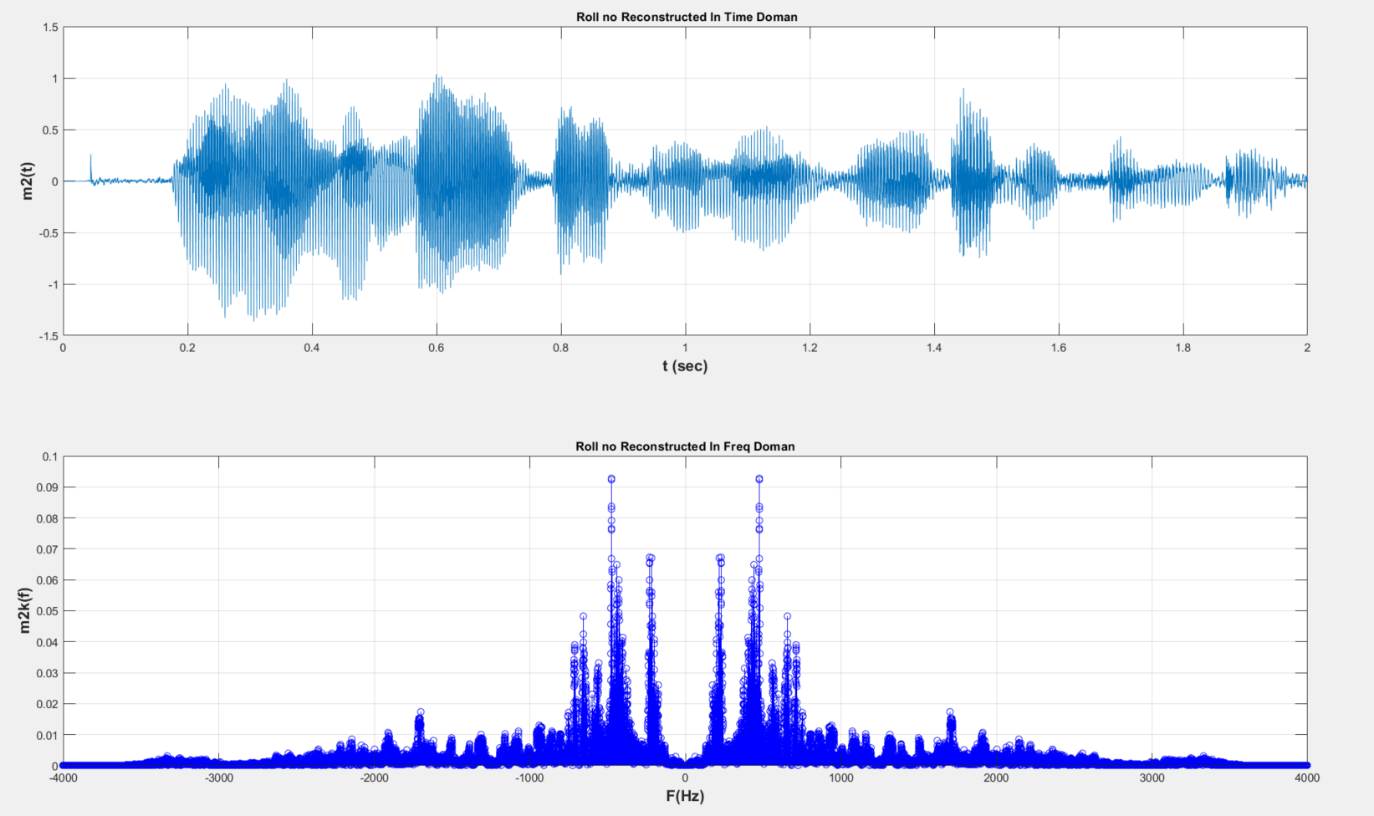


Figure 4. Message 2

*Message 3:*

Max amplitude = 0.6786 V

Bandwidth = 3.3 K Hz

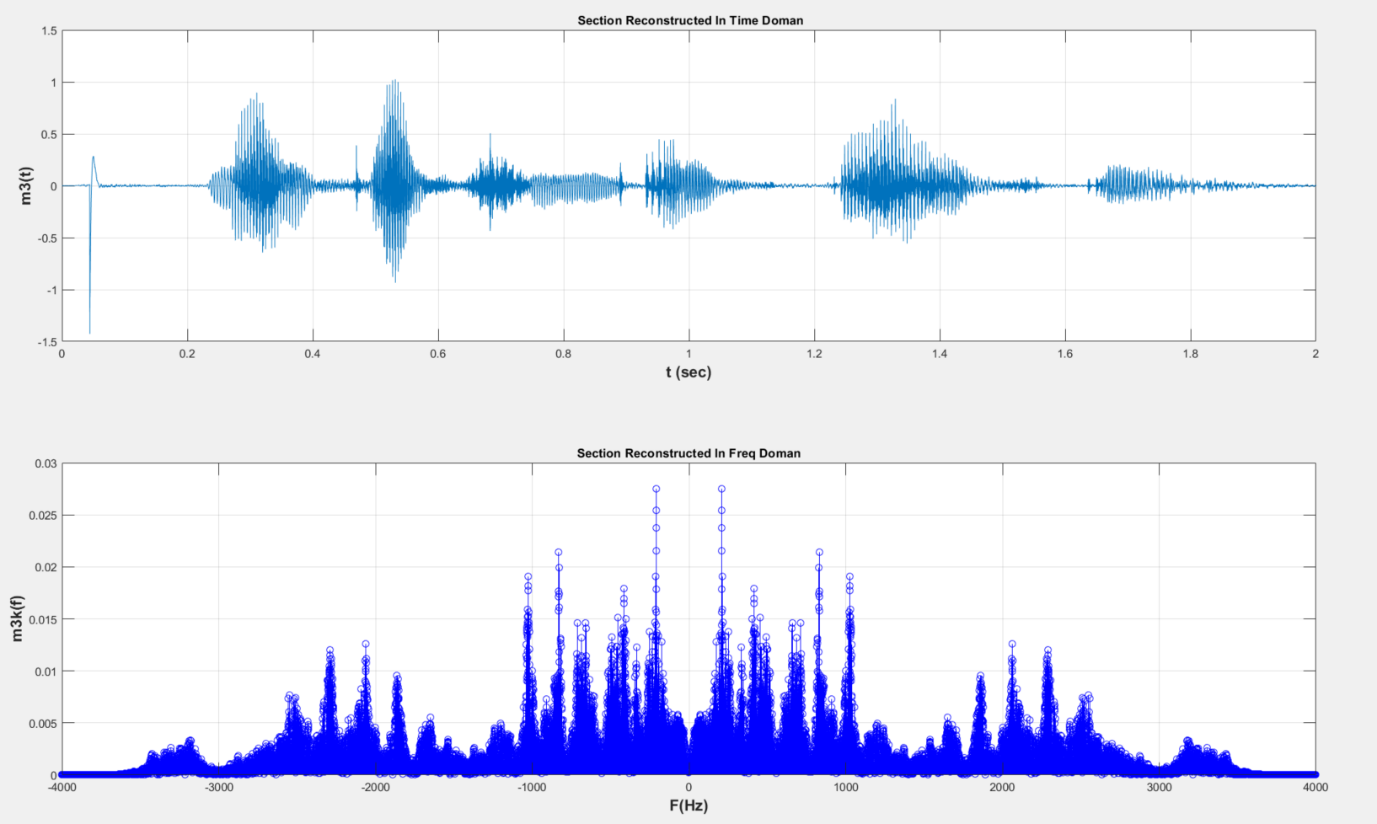


Figure 5. Message 3

*Message 4:*

Max amplitude = 0.3579 V

Bandwidth = 3.3 K Hz

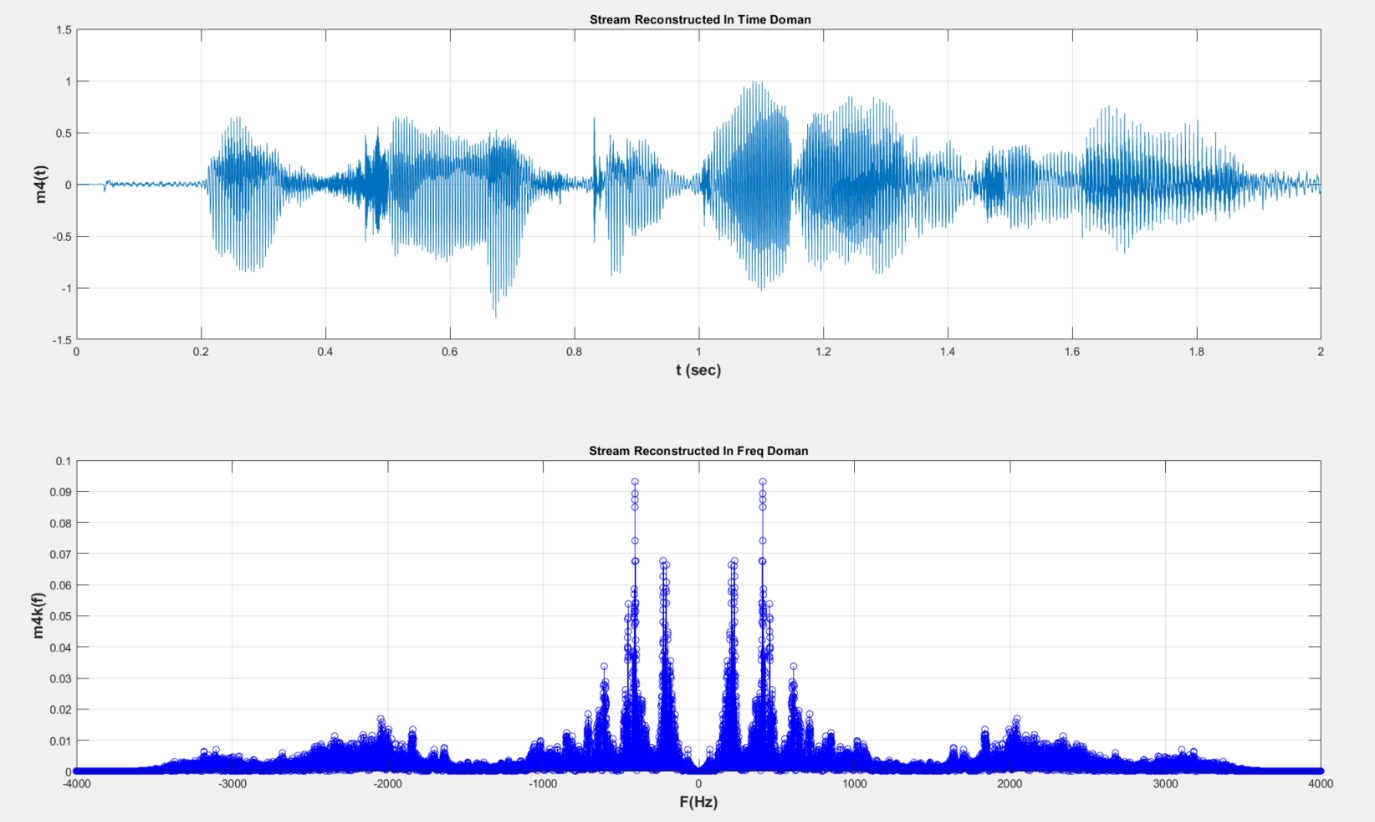
****

Figure 6. Message 4

*Modulation and Multiplexing (Figures for message 3)*

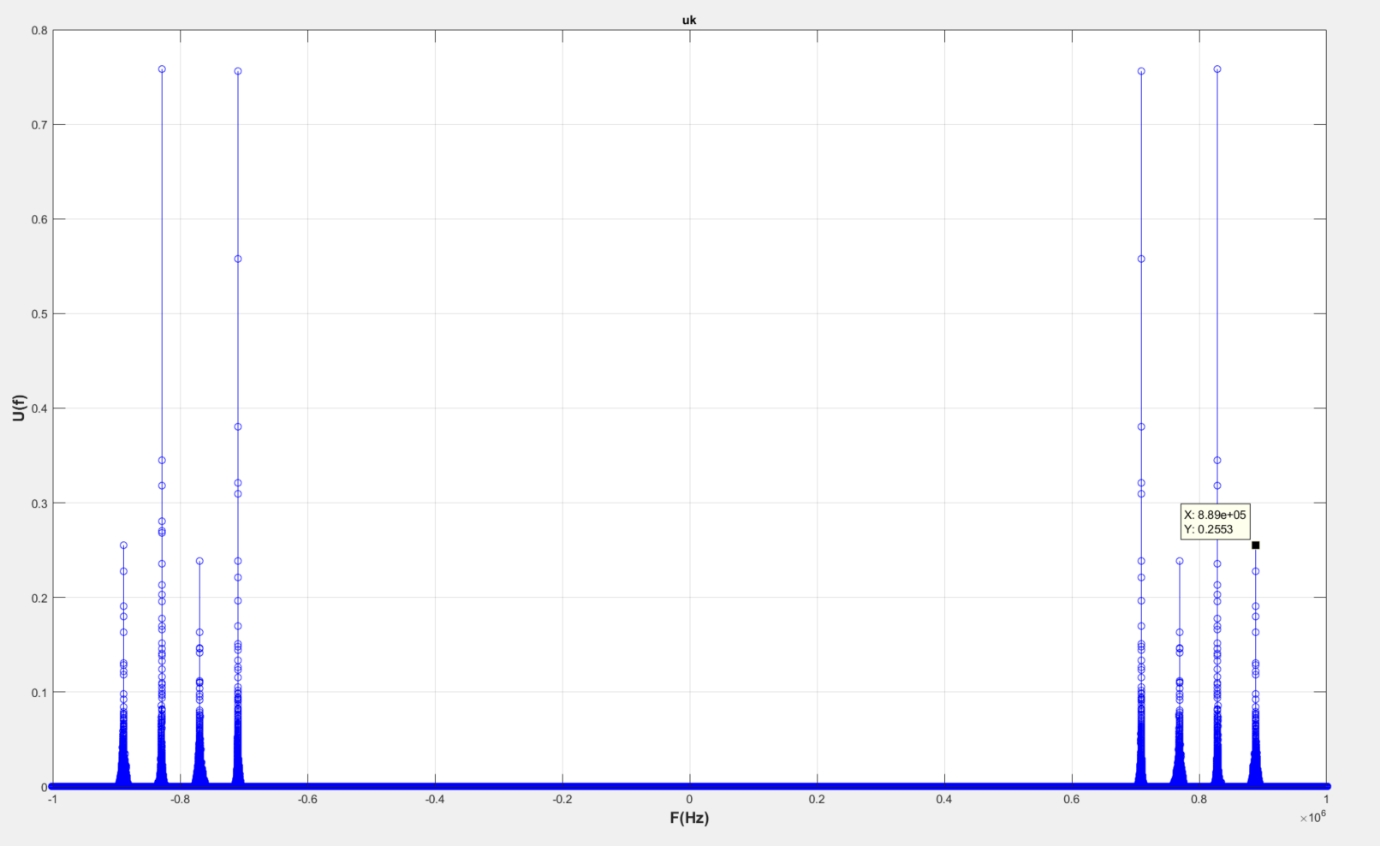


Figure 7. Transmitter

Message signals are modulated and multiplexed. This is the transmitter end.

*Shifting to Intermediate Frequency*

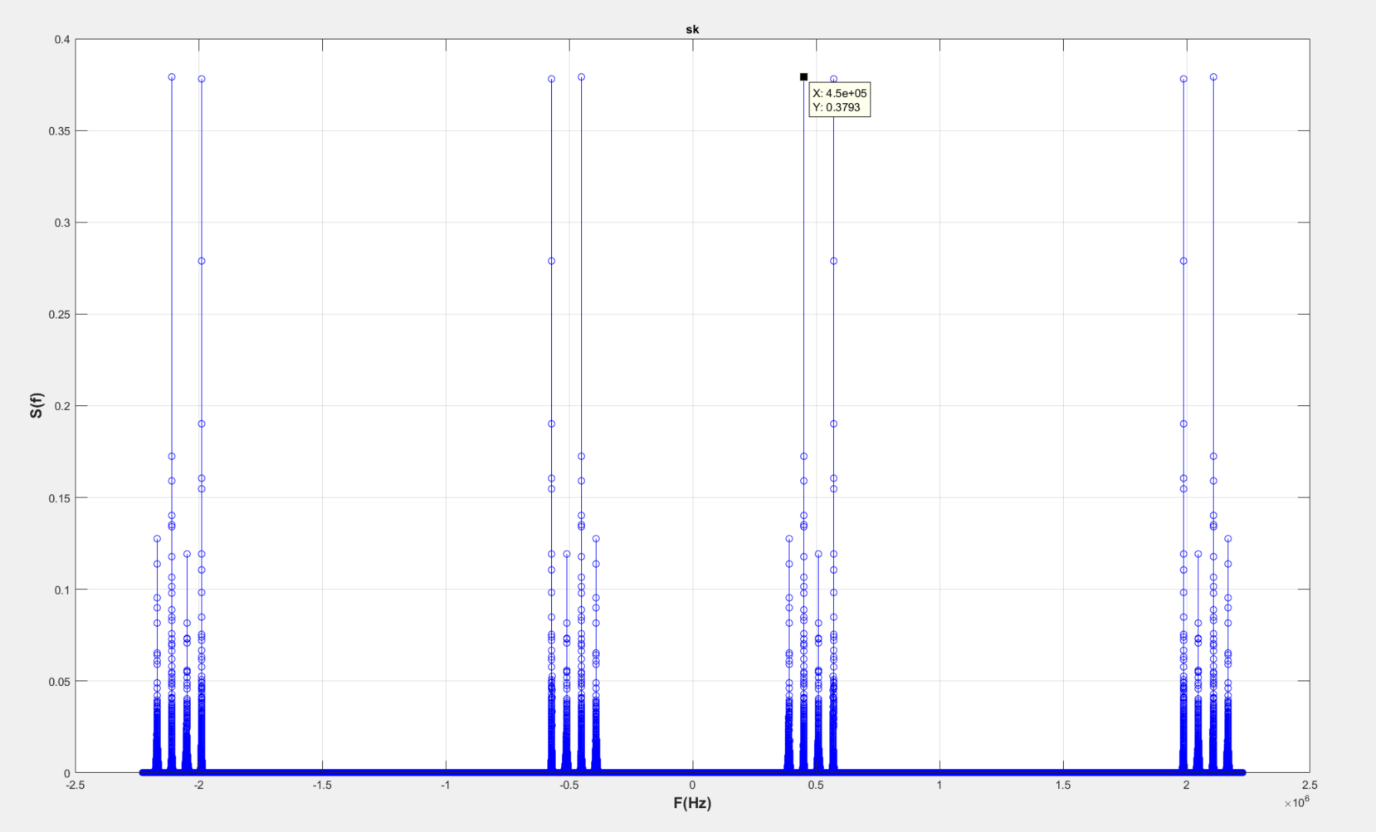


Figure 8. Shifting to Intermediate Frequency

Signal is received and the desired signal is moved to IF 450K. 2Fc + Fif component can also be seen in the figure.

*Bandpass Filtering at Fif*

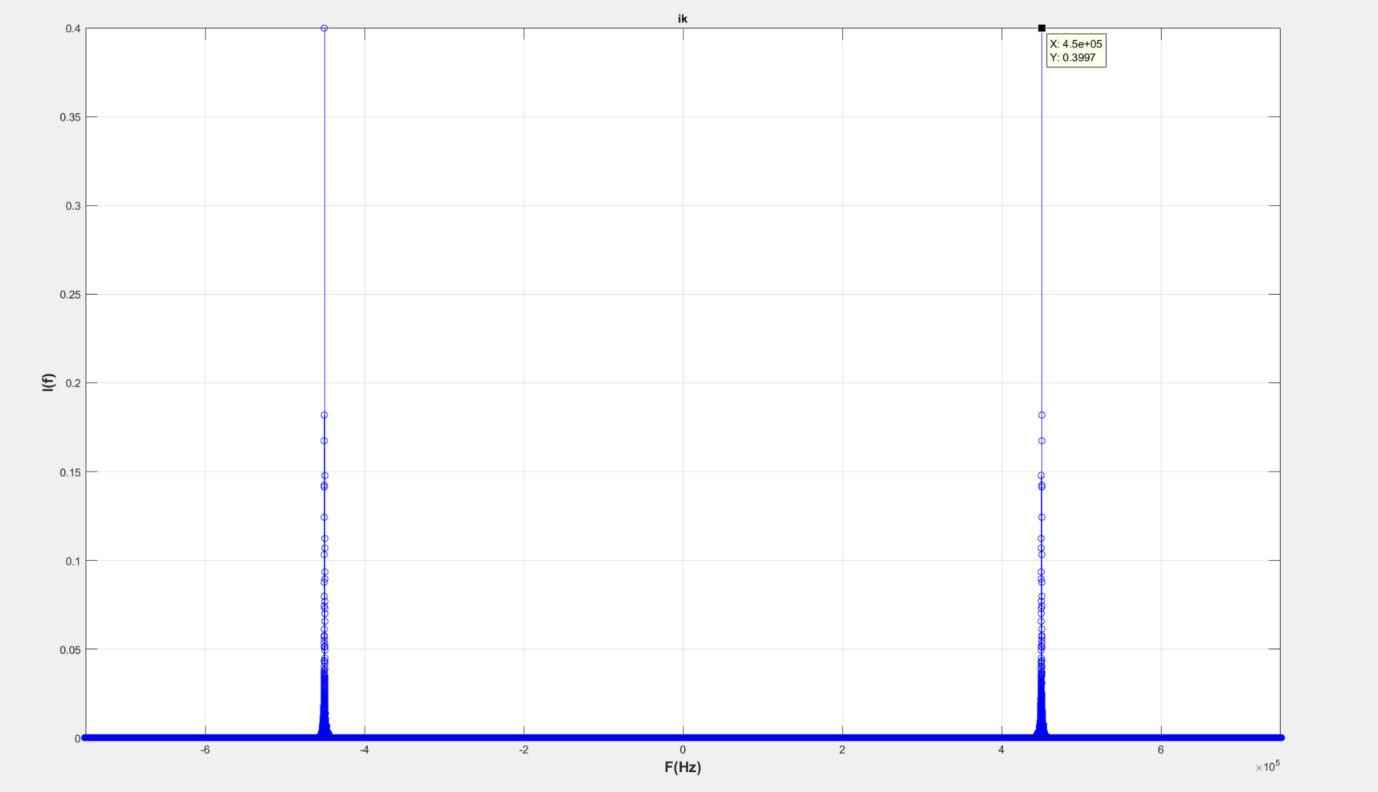


Figure 9. Bandpass Filtering at Fif

Signal at 450K Hz is bandpass filtered to get that signal only. Bandpass center Freq is 450 K Hz.

*Demodulation of signal*

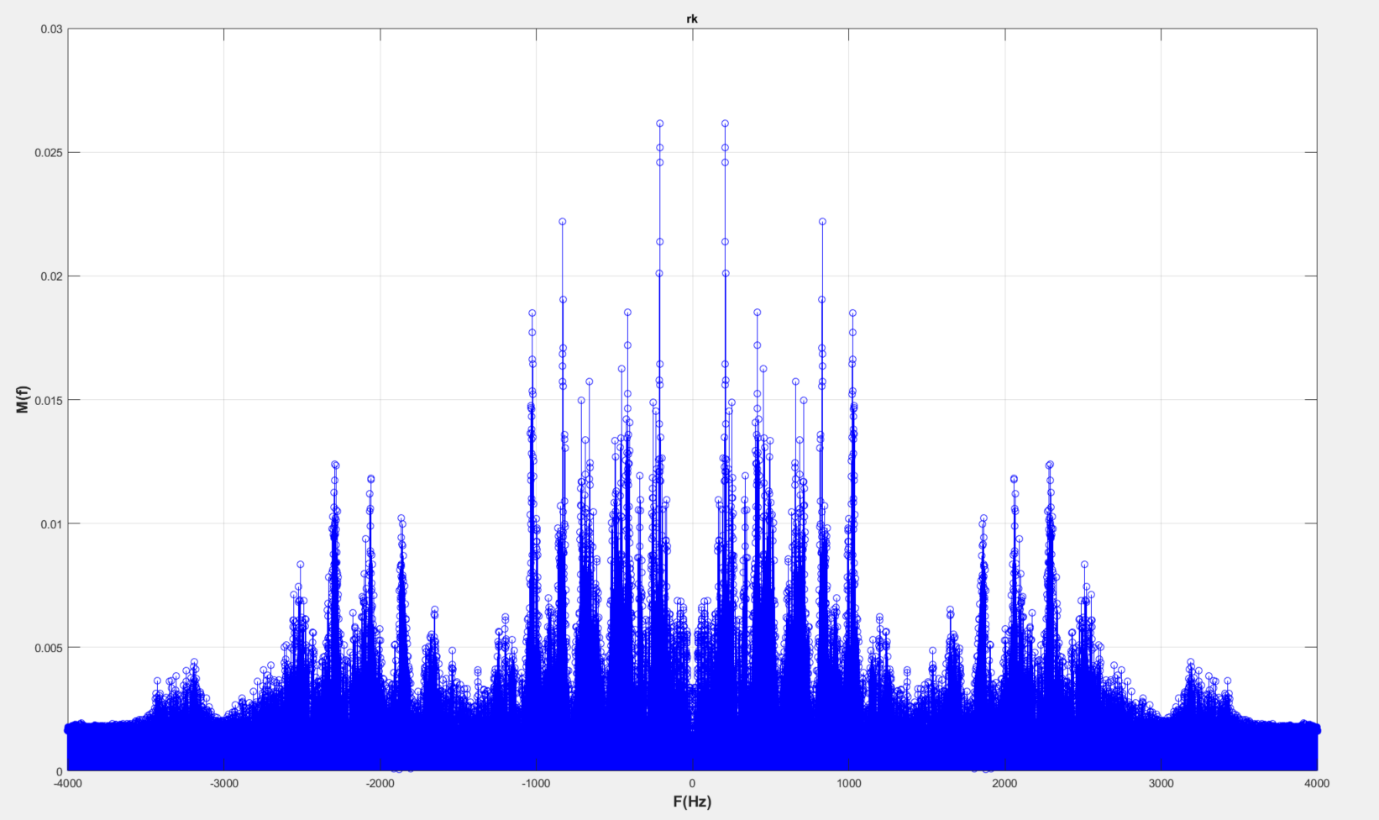


Figure 10. Demodulation of signal

Signal is demodulated to get the original message signal.

*CEP Part 2*

Character values are read from the text file and their binary values are converted into a binary vector for BPSK modulation. First a PAM signal is generated using the binary values and then the PAM signal is multiplied with a carrier with = 10MHz. Transmitted signal is passed through AWGS channel to see the effect of noise. Value of in equation [7] is changed to change the amplitude of the carrier to see the effect of noise immunity at the receiver.

****

Figure 11. PAM Signal

****

Figure 12. Transmitted Signal

*Received Signal with Additive Noise (var=5,000,000) (Es=1)*

****

Figure 13. Received Signal

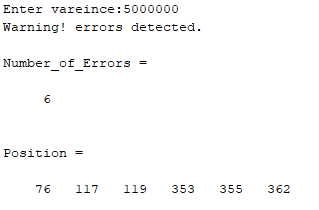
****

Figure 14. Bit errors

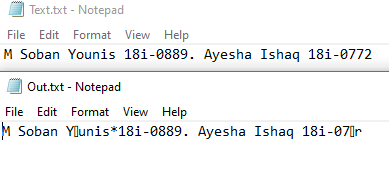
****

Figure 15. Output

*Received Signal with Additive Noise (var=5,000,000) (Es=5)*

****

Figure 16. Received Signal

****

Figure 17. Bit errors

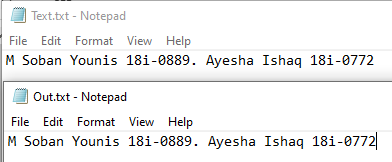
****

Figure 18. Output

It can be seen in the figures above that increasing the carrier amplitude decreases noise effect and the detector is able to give correct output.

**Matlab Code:**

*Part 1*

%%

% % Recording, Playing and Writing Audio File

% % clc;close all;clear all;

% % warning off

% % recObj = audiorecorder;% audiorecorder creates an 8000 Hz, 8-bit, 1 channel audiorecorder object.

% % audiorecorder(Fs, NBITS, NCHANS) creates an audiorecorder object with

% % sample rate Fs in Hertz, number of bits NBITS, and number of channels NCHANS.

% % Common sample rates are 8000, 11025, 22050, 44100, 48000, and 96000 Hz.

% % The number of bits must be 8, 16, or 24. The number of channels must

% % be 1 or 2 (mono or stereo).

% % audiorecorder(Fs, NBITS, NCHANS, ID) creates an audiorecorder object using

% % audio device identifier ID for input. If ID equals -1 the default input

% % device will be used.

% %

% % Fs =8000 ; % Sampling frequency in hertz8000, 11025, 22050, 44100, 48000, and 96000 Hz.

% % nBits = 16 ;% 8, 16, or 24

% % nChannels = 1 ; %Number of channels--2 options--1 (mono) or 2 (stereo)

% % ID = -1; % default audio input device like Microphone

% % recObj = audiorecorder(Fs,nBits,nChannels,ID);

% %

% % disp('Start speaking.')

% % recordblocking(recObj,0.5);

% % recordblocking(OBJ, T) records for length of time, T, in seconds;

% % does not return until recording is finished.

% % disp('End of Recording.');

% % play(recObj);

% % mySpeech = getaudiodata(recObj); % returns the recorded audio data as a double array

% % getaudiodata(OBJ, DATATYPE) returns the recorded audio data in

% % the data type as requested in string DATATYPE. Valid data types

% % are 'double', 'single', 'int16', 'uint8', and 'int8'.

% % Write audio file

% % audiowrite('test.wav',mySpeech,Fs);

% %

%Playing Recorded Audio file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Reading and Plotting Audio Signal with Noise %%%%%%%%%%%%%%%%%%%%%%%%%

% clc;clear all; close all

% [name,Fs] = audioread('myname.wav');

% [rollno,Fs] = audioread('myrollno.wav');

% [sec,Fs] = audioread('mysec.wav');

% [stream,Fs] = audioread('mystream.wav');

% samples=length(name);

% name=name/max(name);

% rollno=rollno/max(rollno);

% sec=sec/max(sec);

% stream=stream/max(stream);

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sound(name,Fs)

% pause(2.5)

% sound(rollno,Fs)

% pause(2.5)

% sound(sec,Fs)

% pause(2.5)

% sound(stream,Fs)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% n\_samples = name;

% r\_samples = rollno;

% s\_samples = sec;

% st\_samples = stream;

%%%%%

%starting reconstruction process

% % % name\_recon=0;

% % % roll\_recon=0;

% % % sec\_recon=0;

% % % stream\_recon=0;

% % % for k=0:length(n\_samples)-1

% % % l = linspace(k,-16000+k,4\*2228000); %Fs/2=2228000, our sound

% sample is 2 seconds

% % % name\_recon=name\_recon+n\_samples(k+1)\*sinc(l);

% % % roll\_recon=roll\_recon+r\_samples(k+1)\*sinc(l);

% % % sec\_recon=sec\_recon+s\_samples(k+1)\*sinc(l);

% % % stream\_recon=stream\_recon+st\_samples(k+1)\*sinc(l);

% % % end

% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%

%%% START FROM HERE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load WorkSpace\_889K\_x2

%

% Fs = length(name\_recon)/2; %each of our message is 2 seconds long

% sound(name\_recon(1:557:length(name\_recon)),8000) % 557 = 8912000 / 16000

% pause(2.5)

% sound(roll\_recon(1:557:length(roll\_recon)),8000)

% pause(2.5)

% sound(sec\_recon(1:557:length(sec\_recon)),8000)

% pause(2.5)

% sound(stream\_recon(1:557:length(stream\_recon)),8000)

% Message 1 figures

figure(1)

subplot(211)

plot(t,name\_recon)

xlabel('t (sec)','fontweight','bold','fontsize',14);

ylabel('m1(t)','fontweight','bold','fontsize',14);

grid on;

title(' Name Reconstructed In Time Doman')

m1k=fft(name\_recon)/Fs\*2;

m1k=fftshift(abs(m1k));

fx=-Fs/2:Fs/length(m1k):((Fs/2)-(Fs/length(m1k)));

subplot(212)

stem(fx,m1k,'b','LineWidth',0.1);

xlim([-4000 4000]);

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('m1k (f)','fontweight','bold','fontsize',14);

grid on;

title('Name Reconstructed In Freq Doman')

% Message 2 figures

figure(2)

subplot(211)

plot(t,roll\_recon)

xlabel('t (sec)','fontweight','bold','fontsize',14);

ylabel('m2(t)','fontweight','bold','fontsize',14);

grid on;

title(' Roll no Reconstructed In Time Doman')

m2k=fft(roll\_recon)/Fs\*2;

m2k=fftshift(abs(m2k));

fx=-Fs/2:Fs/length(m2k):((Fs/2)-(Fs/length(m2k)));

subplot(212)

stem(fx,m2k,'b','LineWidth',0.1);

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('m2k(f)','fontweight','bold','fontsize',14);

xlim([-4000 4000]);

grid on;

title('Roll no Reconstructed In Freq Doman')

% Message 3 figures

figure(3)

subplot(211)

plot(t,sec\_recon)

xlabel('t (sec)','fontweight','bold','fontsize',14);

ylabel('m3(t)','fontweight','bold','fontsize',14);

grid on;

title(' Section Reconstructed In Time Doman')

m3k=fft(sec\_recon)/Fs\*2;

m3k=fftshift(abs(m3k));

fx=-Fs/2:Fs/length(m3k):((Fs/2)-(Fs/length(m3k)));

subplot(212)

stem(fx,m3k,'b','LineWidth',0.1);

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('m3k(f)','fontweight','bold','fontsize',14);

xlim([-4000 4000]);

grid on;

title('Section Reconstructed In Freq Doman')

% Message 4 figures

figure(4)

subplot(211)

plot(t,stream\_recon)

xlabel('t (sec)','fontweight','bold','fontsize',14);

ylabel('m4(t)','fontweight','bold','fontsize',14);

grid on;

title(' Stream Reconstructed In Time Doman')

m4k=fft(stream\_recon)/Fs\*2;

m4k=fftshift(abs(m4k));

fx=-Fs/2:Fs/length(m4k):((Fs/2)-(Fs/length(m4k)));

subplot(212)

stem(fx,m4k,'b','LineWidth',0.1);

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('m4k(f)','fontweight','bold','fontsize',14);

xlim([-4000 4000]);

grid on;

title('Stream Reconstructed In Freq Doman')

%%

n=0:2\*Fs-1;

t=n\*(1/Fs);

B=5;

kf1 = 3300\*B/max(name\_recon);

kf2 = 3300\*B/max(roll\_recon);

kf3 = 3300\*B/max(sec\_recon);

kf4 = 3300\*B/max(stream\_recon);

freqdev1=kf1\*max(name\_recon);

freqdev2=kf2\*max(roll\_recon);

freqdev3=kf3\*max(sec\_recon);

freqdev4=kf4\*max(stream\_recon);

fc4=889000; %selected according to roll number

fc3=829000; %left a space of 60k between each fc

fc2=769000;

fc1=709000;

%Individual modulation

x1=cos(2\*pi\*fc1\*t + 2\*pi\*kf1\*cumtrapz(name\_recon)/length(name\_recon));

x2=cos(2\*pi\*fc2\*t + 2\*pi\*kf2\*cumtrapz(roll\_recon)/length(roll\_recon));

x3=cos(2\*pi\*fc3\*t + 2\*pi\*kf3\*cumtrapz(sec\_recon)/length(sec\_recon));

x4=cos(2\*pi\*fc4\*t + 2\*pi\*kf4\*cumtrapz(stream\_recon)/length(stream\_recon));

%% Multiplexing and Transimission

%Frequency multiplexing

u=x1+x2+x3+x4;

% Local oscillator Freqs at receiver end

fif = 450000;

flo1 = fc1 + fif;

flo2 = fc2 + fif;

flo3 = fc3 + fif;

flo4 = fc4 + fif;

%% User interface

string msg;

msgin=inputdlg('Please Enter your choice: 1,2,3 or 4 .','Input');

%msg = '1';

ch1 = strcmpi(msgin, '1') ;

ch2 = strcmpi(msgin, '2') ;

ch3 = strcmpi(msgin, '3') ;

ch4 = strcmpi(msgin, '4') ;

while ~(ch1 | ch2 | ch3 |ch4 )

msgin=inputdlg('Please Enter a valid choice: 1,2,3 or 4 .','Warning');

ch1 = strcmpi(msgin, '1') ;

ch2 = strcmpi(msgin, '2') ;

ch3 = strcmpi(msgin, '3') ;

ch4 = strcmpi(msgin, '4') ;

end

if ch1==1

flox = flo1;

freqdev=freqdev1;

else if ch2==1

flox = flo2;

freqdev=freqdev2;

else if ch3==1

flox = flo3;

freqdev=freqdev3;

else

flox = flo4;

freqdev=freqdev4;

end

end

end

%% Shifting to Fif, Bandpass filtering and Demodulation

s = u.\*cos(2\*pi\*flox\*t);

i = filter(inter,s); %'inter' filter created by filterDesign Tool

% passband BW = 65K Hz, filter is not ideal

% Low Fpass = 420K Hz & high Fpass = 485K Hz

% y=gradient(i);

% k=find(y<0);

% y(k)=0;

% [b,a]=butter(2,(3500/Fs/2));

% r=filter(b,a,y);

% r=50\*(r-mean(r));

% r=100\*gradient(r);

r=fmdemod(i,fif,Fs,freqdev);

r=r\*2;

sound(r(1:557:length(r)),8000);

%% Attempt at analog butter filter

%

% % % h\_band=gaurd/2 -7500;

% % % h=(fc4-h\_band)/Fs/2;

% % % l=(fc4+h\_band)/Fs/2;

% % % [bb,aa]=butter(5,[h l],'s');

% % % [z p k] = cheb2ap(4,20);

% % % [bp ap] = zp2tf(z,p,k);

% % [bp ap] = butter(10,fc1/(Fs/2));

% % [b,a]=lp2bp(bp,ap,(2\*pi\*fc1),2\*pi\*2\*h\_band);%wn=cuttofffreq/(Fs/2)

% % % [bb,aa]=fir1(48,[h l]);

% % freqz(bp,ap);

% % title('bp ap')

% % y4=filter(b,a,x4);

% %

% % r=fmdemod(u,fc1,Fs,freqdev);

% % sound(r(1:278:length(r)),8000)

%

%

%% Modulation and Multiplexing

%%% PLOT of U %%%

uk=fft(u)/Fs\*2;

uk=fftshift(abs(uk));

fx=-Fs/2:Fs/length(uk):((Fs/2)-(Fs/length(uk)));

figure(5)

stem(fx,uk,'b','LineWidth',0.1);

title('Multiplexed Signal')

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('U(f)','fontweight','bold','fontsize',14);

xlim([-1000000 1000000]);

grid on;

title('uk')

%% Shifting to Intermediate Frequency

%%% PLOT of S %%%

sk=fft(s)/Fs\*2;

sk=fftshift(abs(sk));

fx=-Fs/2:Fs/length(sk):((Fs/2)-(Fs/length(sk)));

figure(6)

stem(fx,sk,'b','LineWidth',0.1);

title('Output of IF mixer')

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('S(f)','fontweight','bold','fontsize',14);

% xlim([28000 50000]);

grid on;

title('sk')

%% Bandpass Filtering at Fif

%%% PLOT of I %%%

ik=fft(i)/Fs\*2;

ik=fftshift(abs(ik));

fx=-Fs/2:Fs/length(ik):((Fs/2)-(Fs/length(ik)));

figure(7)

stem(fx,ik,'b','LineWidth',0.1);

title('Output of bandpass centered at Fif')

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('I(f)','fontweight','bold','fontsize',14);

xlim([-750000 750000]);

grid on;

title('ik')

%%

%%% PLOT of M %%%

rk=fft(r)/Fs\*2;

rk=fftshift(abs(rk));

fx=-Fs/2:Fs/length(rk):((Fs/2)-(Fs/length(rk)));

figure(8)

stem(fx,rk,'b','LineWidth',0.1);

title('Message signal Spectrum')

xlabel('F(Hz)','fontweight','bold','fontsize',14);

ylabel('M(f)','fontweight','bold','fontsize',14);

xlim([-4000 4000]);

grid on;

title('rk')

*Part 2*

%% Reading Text file and binary representation

fidi = fopen('Text.txt','r'); %open file and get File ID

data=fread(fidi); %Read data in double format

fclose(fidi);

cdata=char(data); %Convert data into char (checking if data is read correctly)

bdata1=dec2bin(data,8)-'0'; %Convert double to binary(double format)

%check=bi2de(bdata1,'left-msb');

bdata=transpose(bdata1); %Transpose before reshape

x=reshape(bdata,1,length(cdata)\*8); %Vector of binary values

%% Transimitter

f =10000000; % f in Hz 10MHz

Fs =2.2\*f; % samples per symbol 22MHz

Tb =0.000001; % Bit duration

Ab =1; % Bit Amplitude.

%Code to define time t that contains N samples for Tb second(s)

Ts=1/Fs;

N=length(x)\*Tb\*Fs;

n=0:N-1; %Sampling Index

t=n\*Ts; %time vector

w = sqrt(2/Tb)\*cos(2\*pi\*f\*t); % Defining the Basis Function/ Carrier Signal

%Code to generate and plot BPSK\_Pulse

u=ones(1,length(t));

Fss=Fs\*Tb;

for i=1:length(x)

if x(i) == 0

if i ==1

u(1:1:Fss)= u(1:1:Fss)\*(-1);

else if i==length(x)

u(( (i-1)\*Fss+1):1:i\*Fss)=u(( (i-1)\*Fss+1):1:i\*Fss).\*(-1);

else

u(( (i-1)\*Fss+1):1:i\*Fss)=u(( (i-1)\*Fss+1):1:i\*Fss).\*(-1);

end

end

end

end

tt=t(1:1:length(t)/length(data));

figure(1) %Binary PAM Plot

plot(tt,u(1:1:length(tt)),'b','LineWidth',1)

grid on;

ylim( [-1.5 1.5])

title('Binary PAM')

xlabel('Time (sec)','fontweight','bold','fontsize',14);

ylabel('Amplitude (V)','fontweight','bold','fontsize',14);

u=u.\*w; %BPSK

figure(2)

plot(tt,u(1:1:length(tt)),'b','LineWidth',1)

grid on;

xlim( [0 3/1000000])

title('BPSK')

xlabel('Time (sec)','fontweight','bold','fontsize',14);

ylabel('Amplitude (V)','fontweight','bold','fontsize',14);

%% Noise

variance = input('Enter vareince:');

r=u+sqrt(variance)\*randn(1,length(u)); % Addition of Noise in the Channel or use awgn (r, SNR) command

figure(3)

plot(tt,r(1:1:length(tt)),'b','LineWidth',1)

grid on;

xlim( [0 3/1000000])

title('BPSK with noise')

xlabel('Time (sec)','fontweight','bold','fontsize',14);

ylabel('Amplitude (V)','fontweight','bold','fontsize',14);

%% Correlator Receiver and Detection Block

rec=0;

for i=1:length(x)

if i ==1

ss(1:1:Fss)= r(1:1:Fss).\*w(1:1:Fss);

s(i)=sum(ss(1:1:Fss));

else if i==length(x)

ss(( (i-1)\*Fss+1):1:i\*Fss)=r(( (i-1)\*Fss+1):1:i\*Fss).\*w(( (i-1)\*Fss+1):1:i\*Fss);

s(i)=sum(ss(( (i-1)\*Fss+1):1:i\*Fss));

else

ss(( (i-1)\*Fss+1):1:i\*Fss)=r(( (i-1)\*Fss+1):1:i\*Fss).\*w(( (i-1)\*Fss+1):1:i\*Fss);

s(i)=sum(ss(( (i-1)\*Fss+1):1:i\*Fss));

end

end

if s(i)>0

rec(i)=1;

else

rec(i)=0;

end

end

%% Error checking

err=0;

index=1;

for i=1:length(rec)

if x(i) ~= rec(i)

err(index)=i;

index = index+1;

end

end

if length(err)==1& err(1)==0

data\_bits = transpose(x);

Rcvd\_Bits = transpose(rec);

disp('Congragulations! Data received correctly')

else

data\_bits = transpose(x);

Rcvd\_Bits = transpose(rec);

disp('Warning! errors detected.')

%Total\_Bits = length(

Number\_of\_Errors = length(err)

Position = err

end

%% Convert binary back to char

y = reshape(rec,8,length(cdata));

y = transpose(y);

y = bi2de(y,'left-msb');

out = char(y);

disp(out')

%% Write to Output Text File

fid2 = fopen('Out.txt','w'); %open file and get File ID

fprintf(fid2,'%c',out);

%print('%c',transpose(out));

%output = transpose(out)

fclose(fid2);

**Refences:**

|  |  |
| --- | --- |
| [1] | <https://www.electronics-notes.com/articles/radio/modulation/frequency-modulation-fm.php> |
| [2] | <https://www.electronics-notes.com/articles/radio/superheterodyne-receiver/theory-principles.php> |
| [3] | <https://www.electronics-notes.com/articles/radio/modulation/fm-frequency-demodulation-detection-discrimination.php> |
| [4] | <https://www.google.com.pk/amp/s/www.gaussianwaves.com/2010/04/bpsk-modulation-and-demodulation-2/amp/> |