COMMUNICATION ENGINEERING LAB

Debagnik Kar (1804373) (ETC-06)

SCHOOL OF ELECTRONICS ENGINEERING, KIIT Deemed to be University

Index Page

Experiment Number	01
Date of Experiment	03/08/2020
Date of Submission	17/08/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

Generation and detection of Amplitude Modulation and Demodulation, DSBSC and SSB-SC modulation.

Equipment / Software Required:-

- MATLAB R2018a

Theory:

Amplitude modulation: AM is a modulating technique which is generally used to transmit Radio signals over a carrier signal, where the amplitude of the modulated changes in response of the signal.

Mathematically,

If the carrier signal is,

$$c(t) = Ac.\cos(2\pi.fc.t)$$

And the message signal is,

$$m(t) = Am.cos\ (2\pi.fm.t)$$

Then the equation of the modulated signal is,

$$s(t) = [Ac + \cos(2.\pi.fm.t)].\cos(2.\pi.fc.t)$$

If we solve this equation further, we get,

$$s(t) = Ac\left[1 + \frac{Am}{Ac}.\cos(2.\pi.fm.t)\right].\cos(2.\pi.fc.t)$$

$$s(t) = Ac[1 + \mu.\cos(2.\pi.fm.t)].\cos(2.\pi.fc.t), \quad Where,\ \mu = \frac{Am}{Ac}$$

Modulating Index(μ), also known as modulation depth, of a modulation scheme describes by how much the modulated variable of the carrier signal varies around its unmodulated level. It is defined differently in each modulation scheme.

DSBSC: The transmission of a signal, which contains a carrier along with two sidebands can be termed as Double Sideband Full Carrier system or simply DSBSC.

Mathematically,

If the carrier Signal is,

$$c(t) = Ac.\cos(2.\pi.fc.t)$$

And the message signal is,

$$m(t) = Am.\cos(2.\pi.fm.t)$$

Then the modulating signal will be,

$$s(t) = m(t).c(t)$$

$$or, \quad s(t) = Am.Ac.\cos(2.\pi.fm.t).\cos(2.\pi.fc.t)$$

SSBSC: The process of suppressing one of the sidebands along with the carrier and transmitting a single sideband is called as Single Sideband Suppressed Carrier system or simply SSBSC.

Mathematically,

If the carrier Signal is,

$$c(t) = Ac.\cos(2.\pi.fc.t)$$

And the message signal is,

$$m(t) = Am.\cos(2.\pi.fm.t)$$

Then the modulating signal will be,

$$s(t) = \frac{Am.Ac}{2}.\cos[2.\pi.(fc + fm).t] \quad for\ the\ upper\ Sideband$$

$$s(t) = \frac{Am.Ac}{2}.\cos[2.\pi.(fc - fm).t] \quad For\ the\ lower\ Sideband$$

Code:-

<<<File: Generate Comment: This code will amplitude modulate and then demodulate a sine wave.>>>

```
%generating user defined AM Signals
%Written By Debagnik Kar 1804373

clc;
clear all;
close all;

t = linspace(0,1,1000) %Time of 1 secs divided by 1000 times
%Carrier wave
fc = input('Enter fc = ')
ac = input('Enter ac = ')
xc= cos(2*pi*fc*t)

%Message signal
fm = input('Enter fm = ');
am = input('enter am = ');

xm = cos(2*pi*fm*t);

%Amplitude modulation

y = [ac + am*xm].*xc;

%plot AM
subplot(4,1,1)
plot(t,xc);
xlabel("Time -->")
ylabel("Amplitude -->")
title("Carrier Wave")
subplot(4,1,2)
plot(t,xm)
xlabel("Time -->")
ylabel("Amplitude -->")
title("Message Wave")
subplot(4,1,3)
plot(t,y)
xlabel("Time -->")
ylabel("Amplitude -->")
title("Modulated Wave")

%if else statement

mu = am/ac;

if mu==1
```

```
    disp('Critical modulation');
elseif mu>1
   disp('Over modulated signal');
elseif mu<1
   disp('under modulated signal');
end

%Demodutating The wave
dm = y.^2;
[b,a] = butter(10,0.1);
xd = filter(b,a,dm);

subplot(4,1,4)
plot(t,xd);
xlabel("Time -->")
ylabel("amplitude")
title("Demodulated Wave")
```

<<<File:DSBSC.m Comment: This code will generate a DSBCS Modulated signal>>>

```
%DSBCS Generation
%Written by Debagnik Kar 1804373
clc;
clear all;
close all;
t = linspace(0,4,1000);

fc = input('Enter the carrier frequency: ')
ac = input('Enter the carrier amplitude: ')
fm = input('Enter the message frequency: ')
am = input('Enter the message amplitude: ')

y = am*cos(2*pi*fm*t) %message signal
z = ac*cos(2*pi*fc*t) %carrier signal

w = ((am*ac)/2).*(cos(2*pi*(fc+fm)*t)+cos(2*pi*(fc-fm)*t))
%DSBSC Modulation
subplot(3,1,1)
plot(t,z)
xlabel("time -->")
ylabel("magnitude -->")
title("Carrier Signal")
subplot(3,1,2)
plot(t,y)
xlabel("time -->")
ylabel("magnitude -->")
title("Message Signal")
subplot(3,1,3)
plot(t,w)
xlabel("time -->")
ylabel("magnitude -->")
```

```
title("DSBSC Signal")
```

<<<File: SSBSC.m Comment: Generates an upper sideband, a lower side band SSBSC>>>

```
% Generation of SSB-SC Signal
% Written by Debagnik Kar
clear all
close all
clc

fc = input('Enter the frequency of Carrier: ')
ac = input('Enter the amplitude of Carrier: ')
fm = input('Enter the frequency of Message: ')
am = input('Enter the amplitude of Message: ')

t = linspace(0,1,1000)

m = am*cos(2*pi*fm*t) %Message signal
c = ac*cos(2*pi*fc*t) %carrier Signal

Susb = ((am*ac)/2).*cos(2*pi*(fc+fm)*t) %upper Sideband
Slsb = ((am*ac)/2).*cos(2*pi*(fc-fm)*t) %lower Sideband

subplot(4,1,1)
plot(t,c,'r')
xlabel("Time -->")
ylabel("Amplitude-->")
title("Carrier Wave")

subplot(4,1,2)
plot(t,m,'g')
xlabel("Time -->")
ylabel("Amplitude-->")
title("Message Wave")

subplot(4,1,3)
plot(t,Susb,'b')
xlabel("Time -->")
ylabel("Amplitude-->")
title("Upper Sideband SSB-SC Signal")

subplot(4,1,4)
plot(t,Slsb,'k')
xlabel("Time -->")
ylabel("Amplitude-->")
title("Lower Sideband SSB-SC Signal")
```

Output/Graph:-

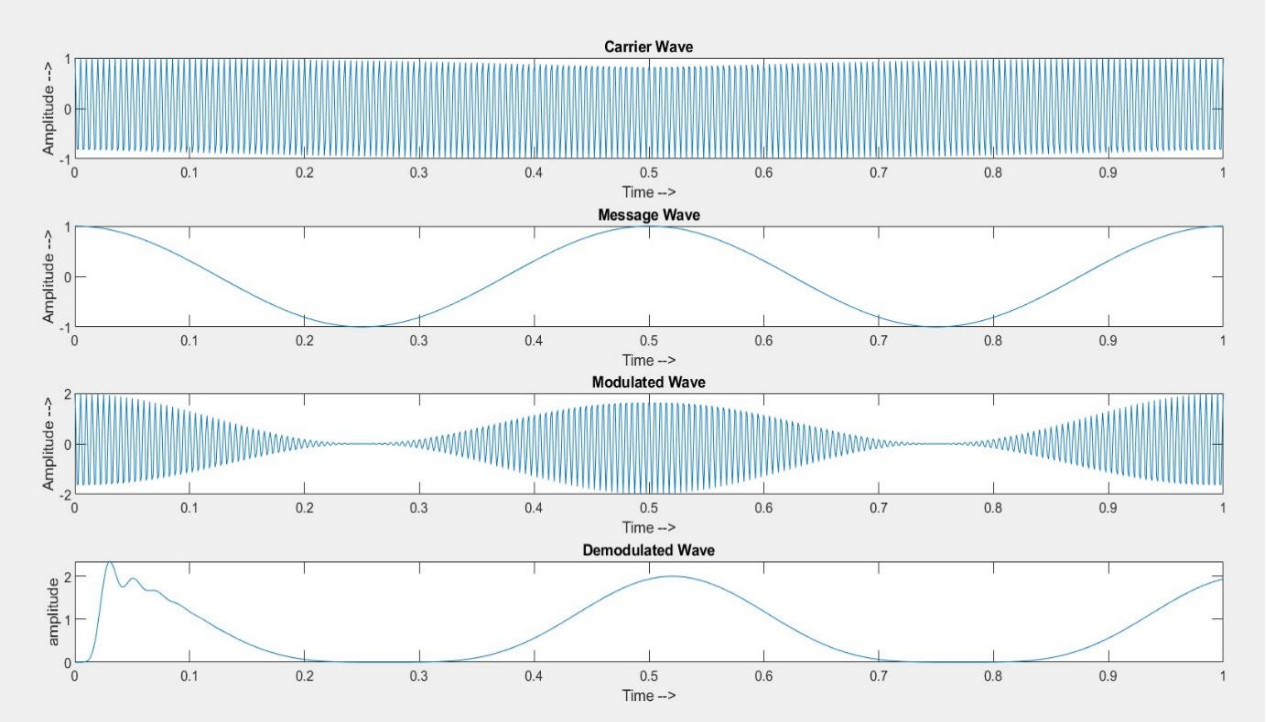


Fig 1.1 : Amplitude modulating and Demodulating a Sine wave of 2Hz

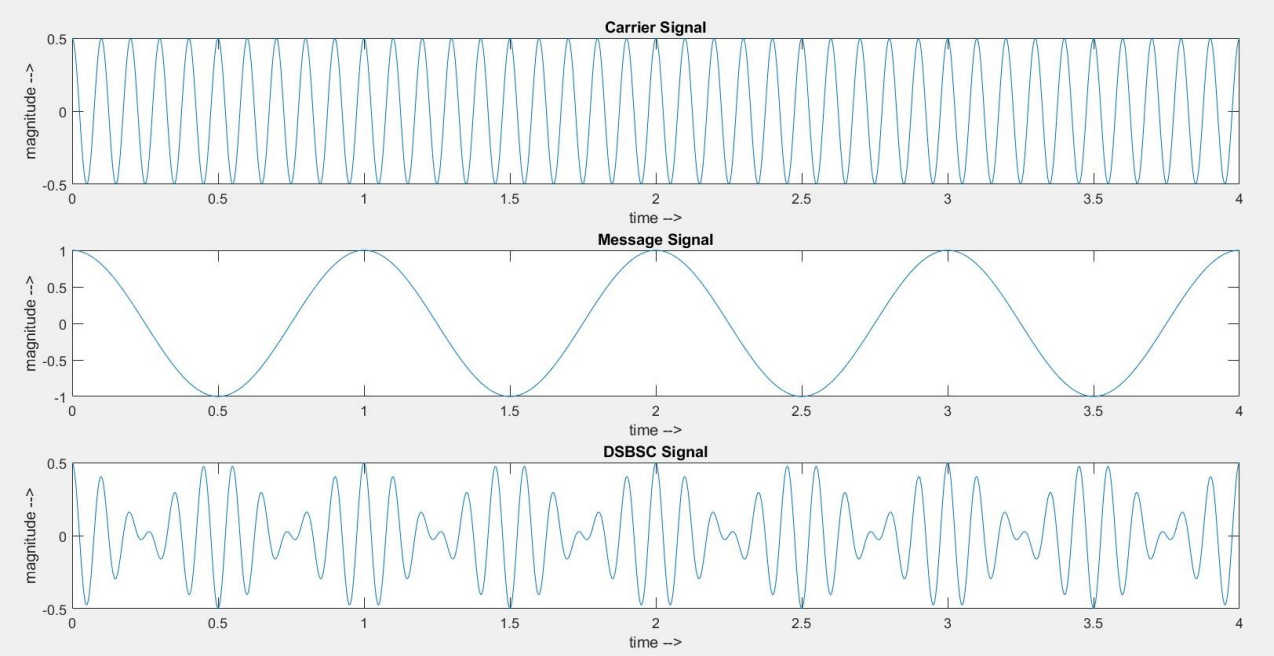


Fig 1.2: Generation of DSBSC Signal

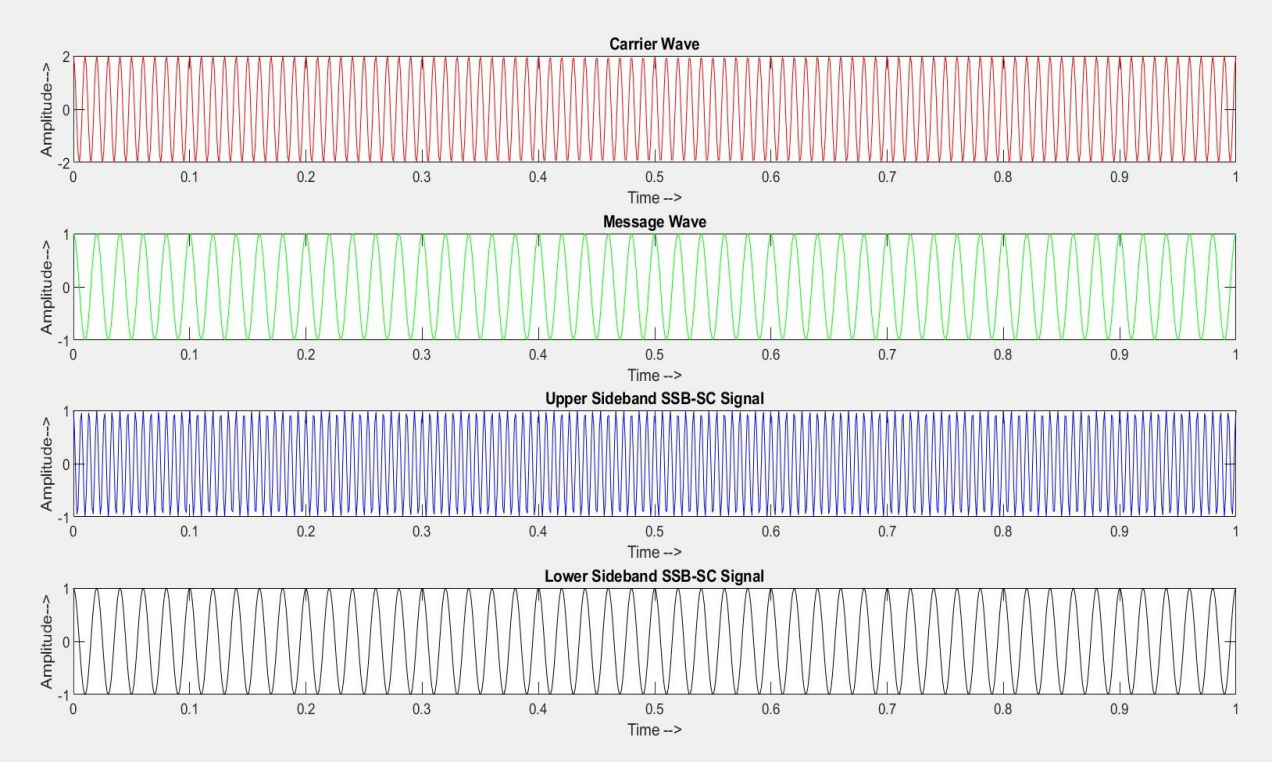


Fig 1.3: Generation of SSBSC Signal

Discussion or Inference of the experiment

This experiment taught me about different kind of amplitude modulation and demodulation techniques that is practiced in analog radio communication technology. It also helped me visualize the difference between the different techniques of transmission

Conclusion:-

The simulation of experiment is done successfully using MATLAB Software.

Experiment Number	02
Date of Experiment	10/08/2020
Date of Submission	17/08/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

Study of Frequency modulation and Demodulation Techniques.

Equipment / Software Required:-

- MATLAB R2018a

Theory

Frequency Modulation (FM) is a form of modulation in which changes in the carrier wave frequency correspond directly to changes in the baseband signal.

Mathematically,

If the Carrier signal is

$$c(t) = \cos(2.\pi.fc.t)$$

And the message signal is

$$m(t) = \sin(2.\pi.fm.t)$$

Then the modulating signal will be,

$$s(t) = \cos[2.\pi.fc.t - \{\mu.\sin(2.\pi.fm.t)\}]$$

$$Where\ \mu = \frac{Am}{Ac}$$

Code:-

<<<File:FMModDemod.m Comment: This code generates a FM signal and Demodulates it>>

```
%Generating a FM Signal and Demodulating it
%Written by Debagnik Kar 1804373

clc;
clear all;
close all;

fc=input('Enter the carrier signal: ');
fm=input('Enter the message signal: ');
mu=input('Modulation index ');

t=linspace(0,1,1000);

c=cos(2*pi*fc*t);%carrier signal
m=sin(2*pi*fm*t);%message signal

subplot(4,1,1);
plot(t,c,'r'); %plotting the carrier signal
ylabel('amplitude');
xlabel('time');
title('Carrier signal');

subplot(4,1,2);
plot(t,m,'g'); %plotting the message signal
ylabel('amplitude');
xlabel('time');
title('Message signal');

y=cos(2*pi*fc*t-(mu*cos(2*pi*fm*t))); % FM Generation

subplot(4,1,3);
plot(t,y,'b'); % Plotting the FM Generation
ylabel('amplitude');
xlabel('time');
title('Frequency Modulated signal');

%FM Demodulation
dem = diff(y);
dem = [0,dem];
rect_dem = abs(dem)
b,a]=butter(10,0.06);
rec = filter(b,a,rect_dem);
subplot(4,1,4)
plot(t,rec,'k')
xlabel('Time')
ylabel('Amplitude')
title('Demodulated Signal')
```

Output/Graph:-

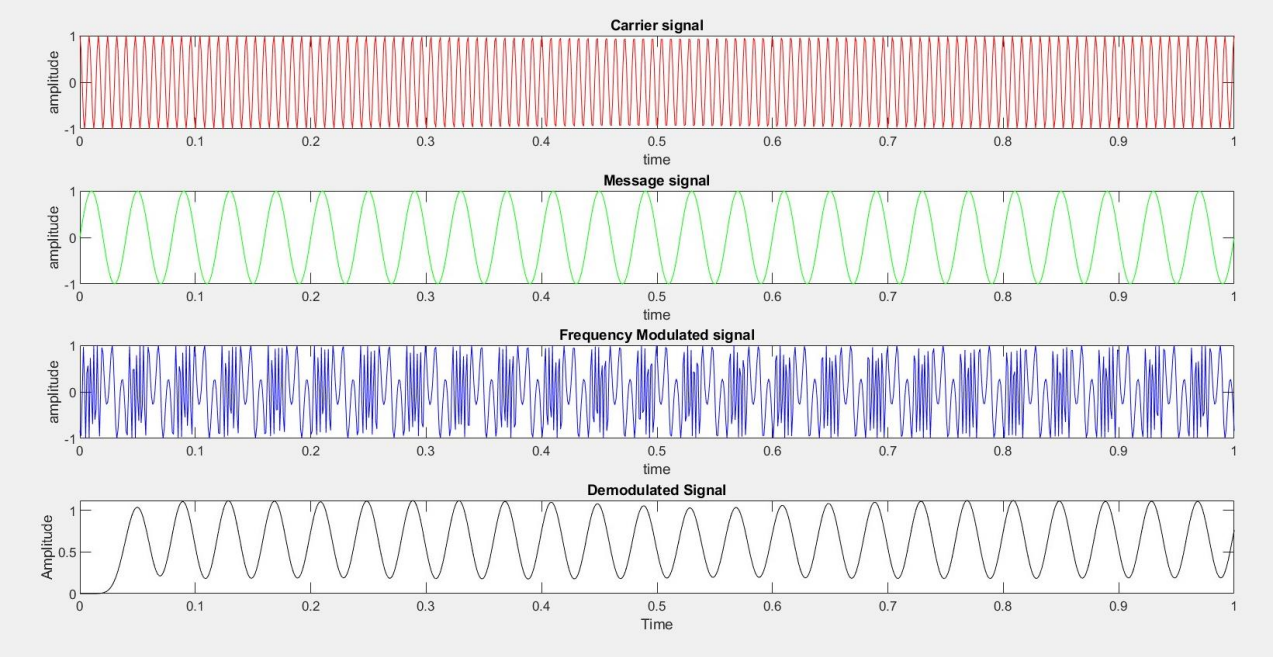


Fig 2.1: Generation of FM Signal and demodulating it.

Discussion or Inference of the experiment

This experiment taught me techniques of radio transmission that are currently being practiced. It also taught me the reasons that FM is better than AM in radio signal transmissions. I also learnt to demodulate a FM signals.

Conclusion:-

Simulation of experiment is done successfully

Experiment Number	03
Date of Experiment	24/08/2020
Date of Submission	29/08/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC - 06

Aim of The Experiment :-

Generation and detection of PAM, PWM and PPM techniques

Equipment / Software Required:-

MATLAB R2018a

Theory

Pulse Amplitude Modulation (PAM): is an analog modulating scheme in which the amplitude of the pulse carrier waves varies proportionally to the instantaneous amplitude of the message signal. There are two types of PAM 1. Natural PAM 2. Flat top PAM.

1. Natural PAM: a signalled at Nyquist rate is reconstructed, by passing it though an efficient Low pass Frequency with exact cut-off frequency.
2. Flat top PAM: Although the natural PAM signal is passed through an LPF, it cannot recover the signal without distortion. Hence to avoid this noise, flat-top sampling is done. Flat-top sampling is the process in which sampled signal can be represented in pulses for which the amplitude of the signal cannot be changed with respect to the analog signal, to be sampled. The tops of amplitude remain flat. This process simplifies the circuit design.

Pulse Width Modulation (PWM): is an analog modulating scheme in which the duration or width or time of the pulse carrier varies proportional to the instantaneous amplitude of the message signal.

The width of the pulse varies in this method, but the amplitude of the signal remains constant. Amplitude limiters are used to make the amplitude of the signal constant. These circuits clip off the amplitude, to a desired level and hence the noise is limited. It is mainly used in the semiconductor industries to transmit signals between microcontrollers or actuators.

Pulse Position Modulation: s an analog modulating scheme in which the amplitude and width of the pulses are kept constant, while the position of each pulse, with reference to the position of a reference pulse varies according to the instantaneous sampled value of the message signal.

The transmitter has to send synchronizing pulses (or simply sync pulses) to keep the transmitter and receiver in synchronism. These sync pulses help maintain the position of the pulses. The following figures explain the Pulse Position Modulation.

PAM	PWM	PPM
Amplitude is varied	Width is varied	Position is varied
Bandwidth depends on the width of the pulse	Bandwidth depends on the rise time of the pulse	Bandwidth depends on the rise time of the pulse
Instantaneous transmitter power varies with	Instantaneous transmitter power varies with the amplitude and the width of the pulse	Instantaneous transmitter power remains constant with the width of the pulses
System complexity is high	System complexity is low	System complexity is low
Noise interference is high	Noise interference is low	Noise interference is low
It is similar to amplitude modulation	It is similar to amplitude modulation	It is similar to phase modulation

Table 3.1: Comparison of different types of modulations

Demodulation: For demodulating all the signals. A Butterworth filters.

Code:-

<<< File: PulseAmpMod.m Comment: Generated a PAM signal and then Demodulates it. Also asks the user if he/she wants to continue to generate a PPM Signal from the PWM signal.>>>

```
%PULSE AMPLITUDE MODULATION PAM
%Written by Debagnik Kar 1804373
clc
close all
clear all

t = 1:0.0001:2
f = input('Enter the value of frequency: ')

x=sawtooth(2*pi*f*t)
ts=0.02
%PULSE GENERATION
for k=1:length(t)
  if mod(t(1,k),ts)==0
      pulse(1,k)=1 %PULSE
  else
      pulse(1,k)=0;
  end
end
natural_pam = x.*pulse %Natural Pulse amplitude modulation

subplot(5,1,1)
plot(t,x,'r')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Message')
subplot(5,1,2)
plot(t,pulse,'b')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Pulse Signal')
subplot(5,1,3)
plot(t,natural_pam,'g')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Natural PAM')

%FLATTOP PAM
FlatTop_pam =zeros(1,length(t));
k=1;

while k <length(t)
if natural_pam(1,k)~=0
  FlatTop_pam(1,k:k+49)=natural_pam(1,k)*ones(1,50); % pulse
duration is 50*0.001
  k=k+49;
```

```
else FlatTop_pam(1,k)=0;
  k=k+1;
 end
end

subplot(5,1,4)
plot(t,FlatTop_pam,'k')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Flat top PAM')

%Demodulation
[b,a]=butter(7,0.004)
demod=filter(b,a,FlatTop_pam)
subplot(5,1,5)
plot(t,demod,'c')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Demodulated PAM')
```

<<<File: PulseWidthMod.m Comment: Generation and demodulation of PWM>>>

```
%Generation of PWM
%Written by Debagnik Kar 1804373

clc
clear all
close all

fc = input('Carrier frequency in Hz: ') %user input prompt
fm = input('Message Freguency in Hz: ') %user input prompt

t = 0:0.001:1 % TIME

carrier = sawtooth(2*pi*fc*t)

subplot 411
plot(t,carrier)
title('Carrier wave')
xlabel('Time')
ylabel('Amplitude')

% Modulating waveform
message = cos(2*pi*fm*t)
subplot 412
plot(t,message)
title('Message wave')
xlabel('Time')
ylabel('Amplitude')

% PWM waveform generation
```

```
p=find(carrier>=message)
    pwm(p) = -1
q=find(carrier<message)
     pwm(q) = 1

subplot 413
plot(t,pwm)
axis([0 1 0 2])
title('PWM waveform')
xlabel('Time')
ylabel('Amplitude')

[b, a] = butter(6,0.009)
demod = filter(b,a,pwm)
subplot 414
plot(t,demod,'r')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Demodulated Wave')

prompt = input('Do you want to continue to PPM? (1 - yes/0 -
no):  ')
if prompt == 1
    PulsePosMod(t,pwm)
elseif prompt == 0
    disp('Okay!')
    exit()
else
    disp('Wrong choice, Exiting')
    goto(46)
end
```

<<<File: PulsePosMod.m Comment: Extension of PulseWidthMod.m , It generates a PPM Signal from the PWM from the above signal>>>

```
%PPM Extension Function to PWM
%Written By Debagnik Kar

function PulsePosMod(t,pwm,f)
clc
figure(2)
dip=diff(pwm)
dip = [0,dip]
ppm=zeros(1,length(dip))
k=1
%Positive edge
while k<length(dip)
    if dip(1,k) == 2 % take -2 for negative edge triggering
        ppm(1,k:k+9)=ones(1,10) % I took to block
        k=k+10
    else
```

```
        k=k+1
    end
end
subplot 311
plot(t,cos(2*pi*f*t),'r')
xlabel('Time -->')
ylabel('Amplitude -->')
title('Message signal')
subplot 312
plot(t,pwm,'c')
axis([0 1 0 2])
xlabel('Time -->')
ylabel('Amplitude -->')
title('PWM Signal')
subplot 313
plot(t,ppm,'g')
axis([0 1 0 2])
xlabel('Time -->')
ylabel('Amplitude -->')
title('PPM Signal')
end
```

Output/Graph:-

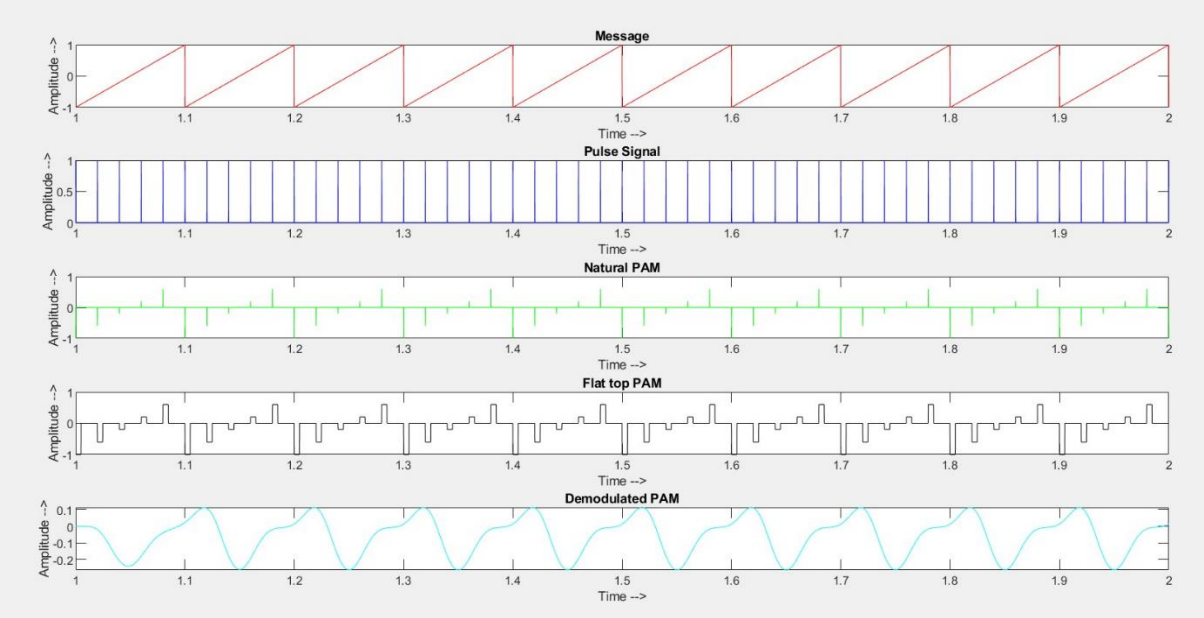


Fig 3.1: Modulation and Demodulation of Natural PAM and Flat-top PAM.

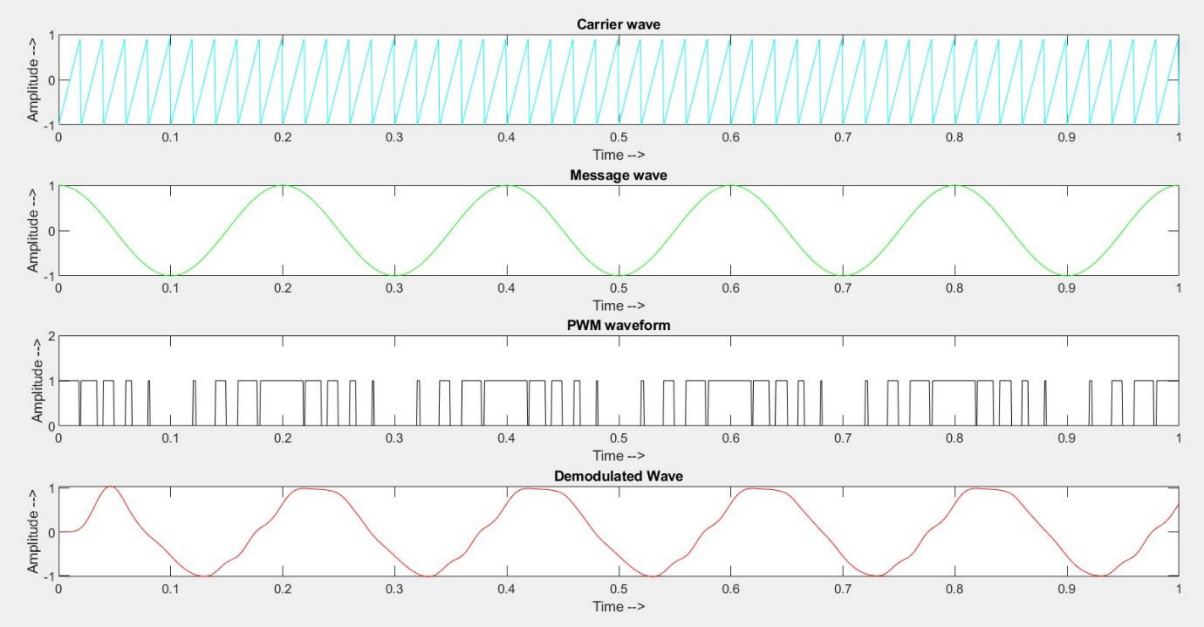


Fig 3.2: Modulation and Demodulation of PWM Signal.

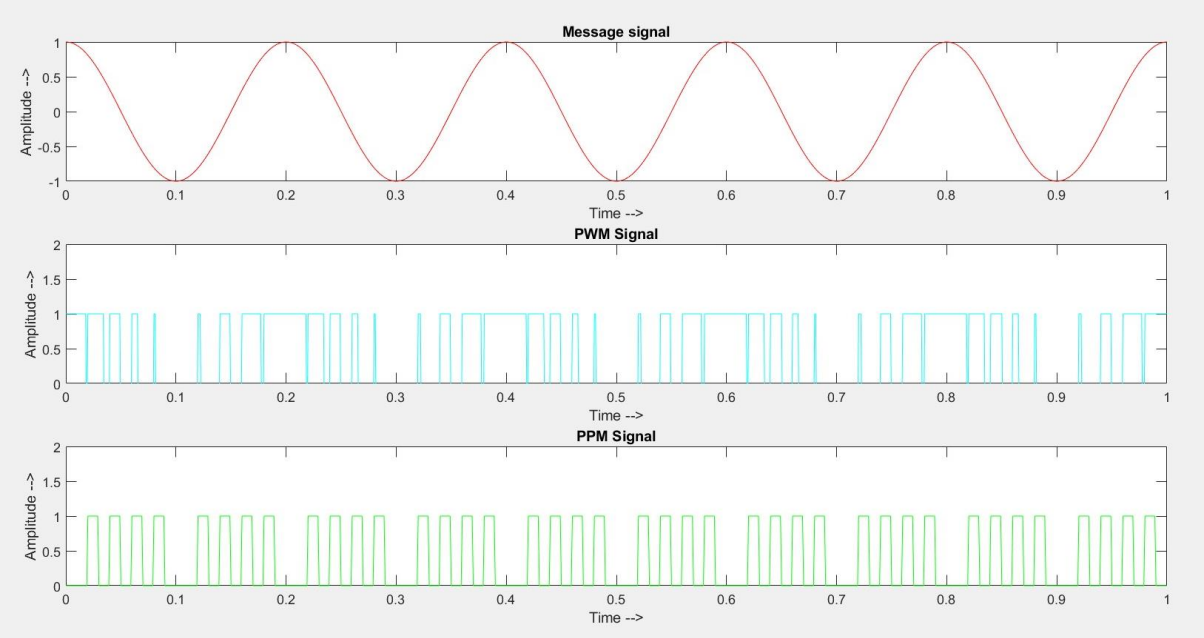


Fig 3.3: Modulation of PPM.

Discussion or Inference of the experiment

This Experiment taught me different pulse modulation techniques such as PAM, PWM, PPM. I also realized advantages of Pulse Position Modulation, over Pulse Amplitude Modulation or Pulse Width Modulation.

By understanding, Pulse position modulation has low noise interference when compared to PAM because amplitude and width of the pulses are made constant during modulation. Noise removal and separation is also very easy in pulse position modulation. I also learned about differentiator and inverter circuit, and its use during generation of PPM signal, through PWM signal.

Conclusion:-

In this experiment, for generating and demodulating PAM, PWM and PPM signals were visualized, simulated and Demultiplexed/Demodulated (respectively) in the MATLAB software with the help of our prerequisite knowledge.

Experiment Number	04
Date of Experiment	31/08/2020
Date of Submission	15/08/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

Study of Pulse Code Modulation (PCM) and demodulation. Multiplexing of signal using Time Division Multiplexing (TDM) technique.

Equipment / Software Required:-

MATLAB R2018a

Theory

- **TDM**: Time division multiplexing is a technique of multiplexing, where the users are allowed the total available bandwidth on time sharing basis. Here the time domain is divided into several recurrent slots of fixed length, and each signal is allotted a time slot on a round-robin basis.

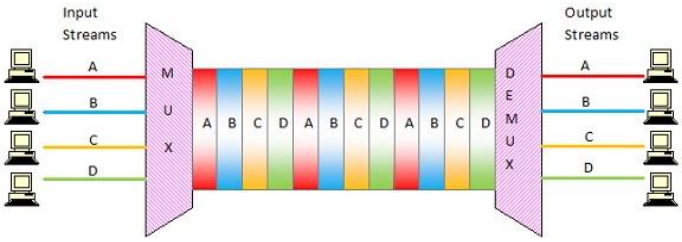


Fig 4.1: Visualizing TDM

- **PCM**: Pulse-code modulation (PCM) is a method used to digitally represent sampled analog signals. It is the standard form of digital audio in computers, compact discs, digital telephony and other digital audio applications. In a PCM stream, the amplitude of the analog signal is sampled regularly at uniform intervals, and each sample is quantized to the nearest value within a range of digital steps.

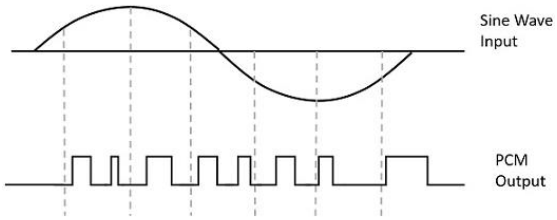


Fig 4.2: Visualizing PCM

Code:-

<<<File: TDM.m Comment: Generation and demultiplexing of TDM signals>>>

```
%Generation of Time domain Multiplexing(TDM)
%Written by Debagnik Kar 1804373
clc
clear all
close all
f1=700
f2=50
s=0.005 %sampling rate

x=pulstran(0:1/f1:1,0:1/f2:1,'rectpuls',s)
y=(1/2)*pulstran(0:1/f1:1,0.01:1/f2:1,'rectpuls',s)
t=0:1/f1:1

subplot 411
plot(t,x)
title('Train of pulse [1]')
ylabel('Amplitude')
xlabel('Time')
subplot 412
plot (t,y,'r')
title('Train of pulse [2]')
ylabel('Amplitude')
xlabel('Time')

%message signal
y1=20*sin(2*pi*2*t)
y2=20*sin(2*pi*4*t)
subplot 413
plot(t,y1,'g')
hold on
plot(t,y2,'r')
ylabel('Amplitude')
xlabel('Time')
title('message signal')

Pam1=x.*y1
Pam2=y.*y2
y3=Pam1+Pam2

subplot(4,1,4)
plot(t,y3,'k')
title('Tdm signals')
ylabel('Amplitude')
xlabel('Time')

figure(2)
demux=y3.*x
```

```
[b,a]=butter(7,0.02)
s1=filter(b,a,demux)
subplot 313
plot(t,s1,'g')
hold on
title('Demuxed and demodulated signal')
demux2=y3.*y
[b,a]=butter(7,0.02);
s2=filter(b,a,demux2)
plot(t,s2,'r');
hold off
ylabel('Amplitude')
xlabel('Time')

subplot 311
plot(t,y1,'g')
hold on
plot(t,y2,'r')
ylabel('Amplitude')
xlabel('Time')
title('message signal')

subplot 312
plot(t,y3)
ylabel('Amplitude')
xlabel('Time')
title('TDM signal')
```

<<<File: PCM.m Comment: Generation and demodulation of PCM>>>

```
%Generation of PCM Signals and Demodulating it
%written by Debagnik 1804373
clc
close all
clear all

n=input('Enter n values for n-bit Pcm system: ')
n1=input('Enter number of samples in a period : ')

%Sampling Operation

x=0:2*pi/n1:4*pi

s=8*sin(x)
subplot 411
plot(s,'r')
axis([0 50 -10 10])
grid on
title('Analog Signal');
ylabel('Amplitude')
xlabel('Time')
```

```
subplot 412
stem(s,'y')
axis([0 50 -10 10])
grid on
title('Sampled Signal')
ylabel('Amplitude')
xlabel('Time')

%Quantization Process
vmax=8
vmin=-vmax
del=(vmax-vmin)/(2^n)
part=vmin:del:vmax
code=vmin-(del/2):del:vmax+(del/2)
[ind,q]=quantiz(s,part,code)

l1=length(ind)
l2=length(q)

for i=1:l1
    if(ind(i)~=0)
        ind(i)=ind(i)-1;
    end
    i=i+1
end
for i=1:12
    if(q(i)==vmin-(del/2))
        q(i)=vmin+(del/2);
    end
end

subplot 413
stem(q,'b')
grid on
title('Quantized Signal')
ylabel('Amplitude')
xlabel('Time')
axis([0 50 -10 10])
%Encoding process

code=de2bi(ind)

k=1
for i=1:11
  for j=1:n
    coded(k)=code(i,j);
    j=j+1;
    k=k+1;
  end
  i=i+1;
```

```
end


subplot 414
grid on
stairs(coded,'g')
axis([0 180 -0.5 1.5])
grid on
title('Encoded Signal');
ylabel('Amplitude')
xlabel('Time')

%Demodulation of pcm signals

index=bi2de(code)
q=del*index+vmin+(del/2);
figure(2)
subplot 311
plot(s,'k')
axis([0 50 -10 10])
grid on
title('Analog Signal');
ylabel('Amplitude')
xlabel('Time')

subplot 312
grid on
stairs(coded,'c')
axis([0 180 -0.5 1.5])
grid on
title('Encoded Signal');
ylabel('Amplitude')
xlabel('Time')

subplot 313
plot(q,'m')
axis([0 50 -10 10])
grid on
title('Demodulated Signal')
ylabel('Amplitude')
xlabel('Time')
```

Output/Graph:-

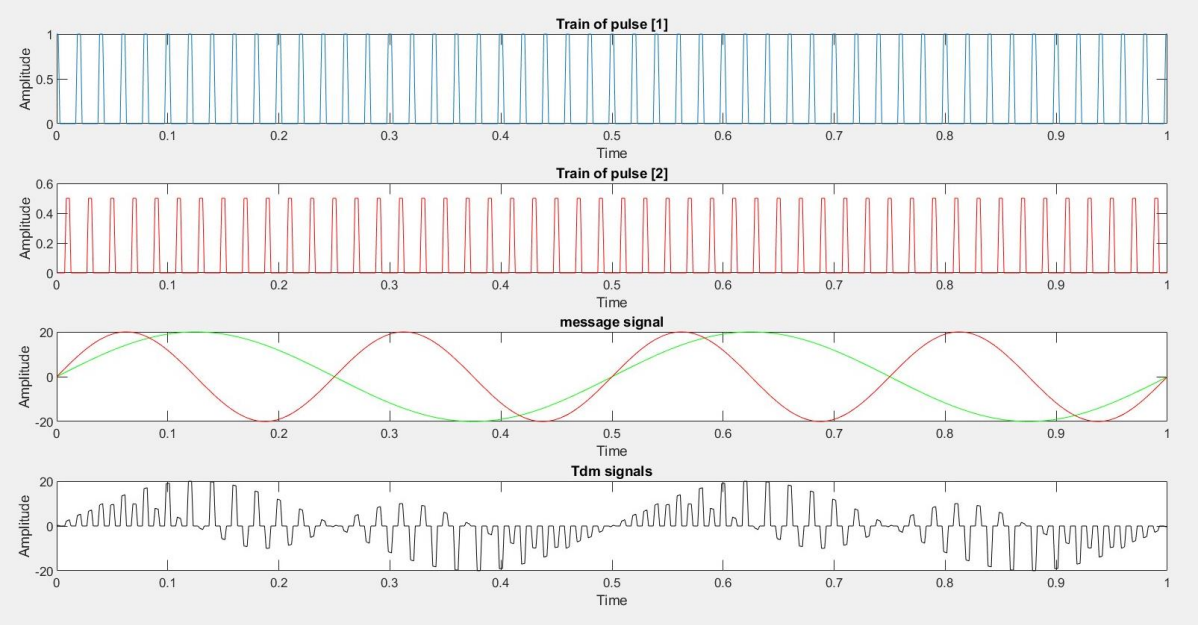


Fig 4.3: Time Domain Multiplexing

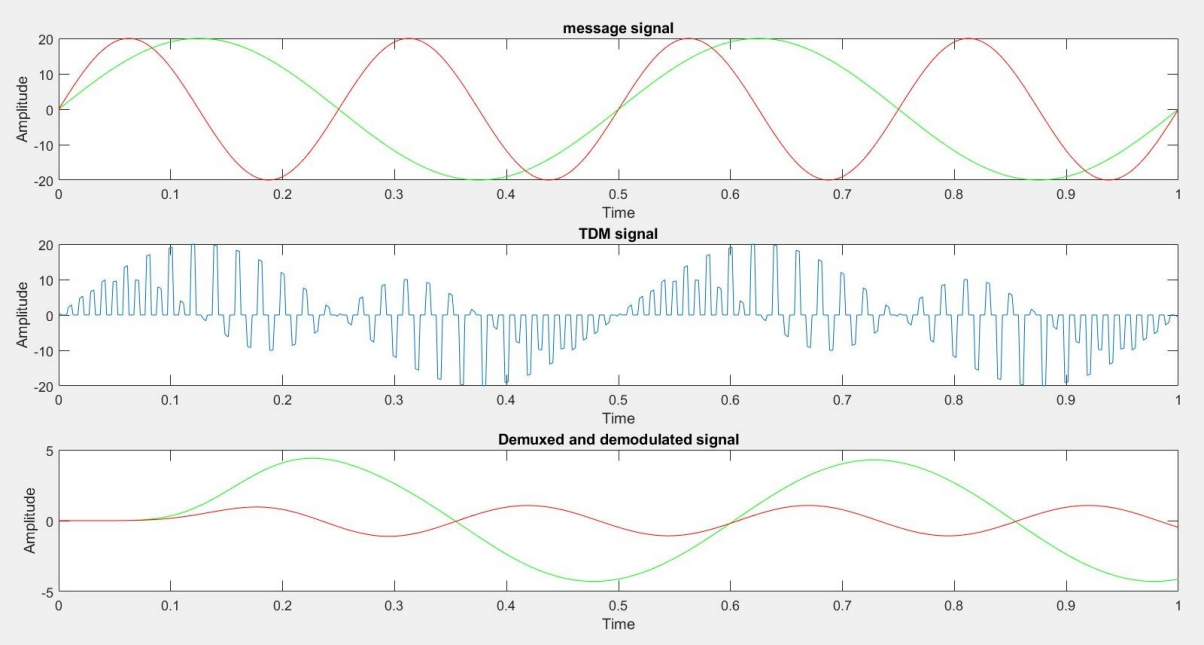


Fig 4.4: TDM Signal Demultiplexing

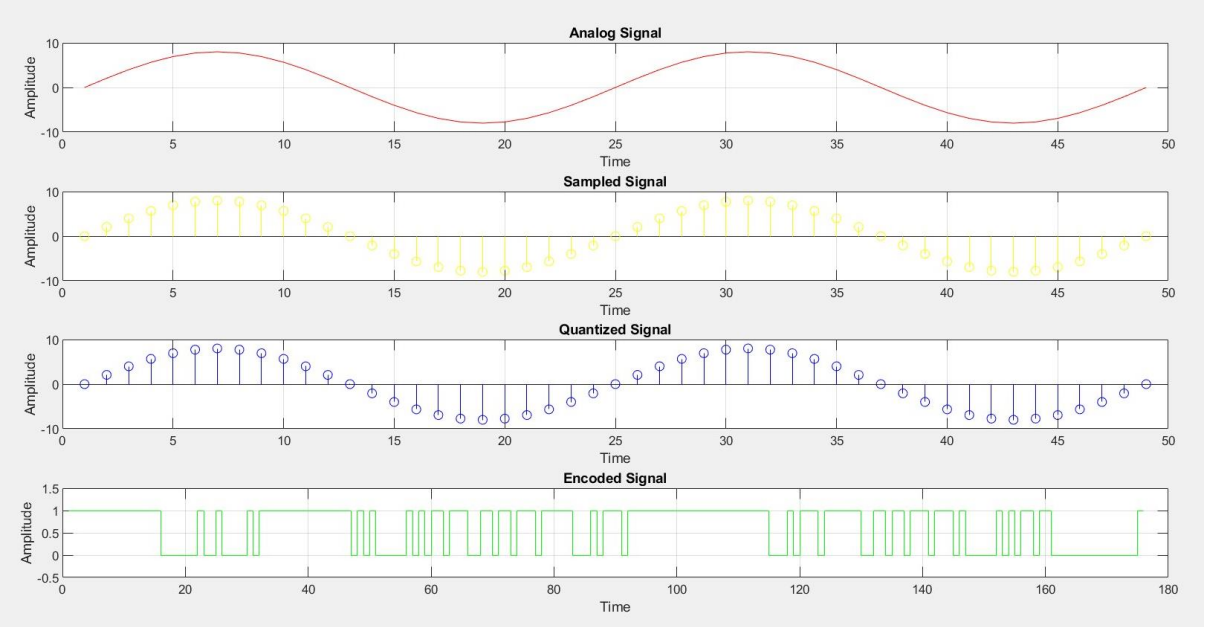


Fig 4.5: Generation of PCM Signals

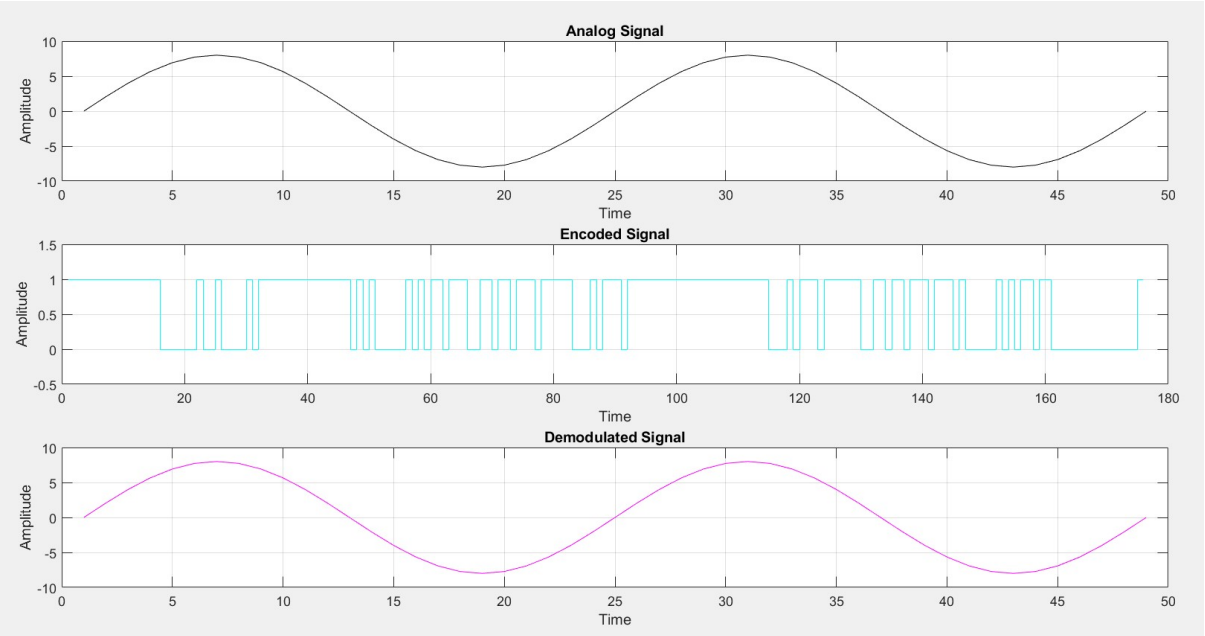


Fig 4.6: Demodulating PCM Signals

Discussion or Inference of the experiment

This experiment taught me the implementation of TDM and PCM, as we learned in our theory.

The experiment helped me understand the signal quantization process, and its implementation in MATLAB software.

Also, I understood.

1. Time-division multiplexing systems are more flexible than frequency division multiplexing.
2. Time division multiplexing circuitry is not complex.
3. Problem of cross talk is not severe in TDM.

4. Full available channel bandwidth can be utilized for each channel in TDM.
5. The PCM (pulse code modulation) convenient for long-distance communication.
6. PCM has a higher transmitter efficiency.
7. PCM has a higher noise immunity

Conclusion:-

In this experiment, Time Division Multiplexing(TDM) and Pulse Code Modulation(PCM) were visualized, simulated and Demultiplexed/Demodulated (respectively) in the MATLAB software with the help of our prerequisite knowledge.

Experiment Number	05
Date of Experiment	14/09/2020
Date of Submission	05/09/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

Generation and detection of Delta modulation Technique

Equipment / Software Required:-

MATLAB R2018a

Theory

The type of modulation, where the sampling rate is much higher and in which the step-size after quantization is of a smaller value, such a modulation is termed as delta modulation.

Some Features of Delta Modulations:-

- An over-sampled input is taken to make full use of the signal correlation.
- The quantization design is simple.
- The input sequence is much higher than the Nyquist rate.
- The quality is moderate.
- The design of the modulator and the demodulator is simple.
- The stair-case approximation of output waveform.
- The step-size is very small, i.e., Δ delta.
- The bit rate can be decided by the user.
- This involves simpler implementation.

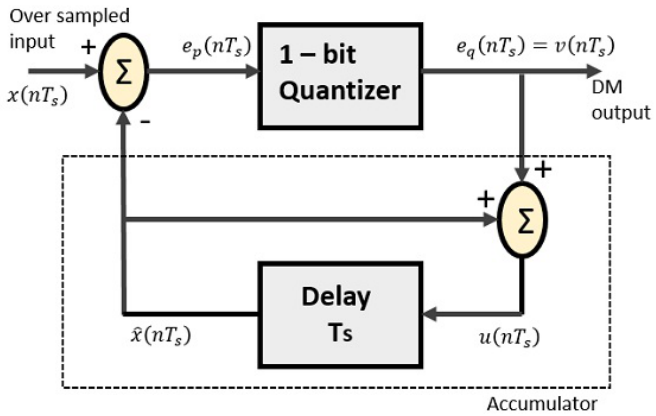


Fig 5.1: A Block diagram of Delta Modulator

Code:-

```
%generation of Delta Modulated Signals
%Written by Debagnik Kar 1804373

clc
close all
clear all
am=5          %Amplitude
fm=2          %Frequency of the signal
fs=100        %No. Of samples
t=0:1/fs:2    %Time
x=am*sawtooth(2*pi*fm*t) %Message signal (Sawtooth)
subplot(3,1,1)
plot(t,x,'r')
xlabel('time')
ylabel('Amplitude')
title('Message Signal')
d=(2*pi*am*fm)/fs
for n=1:length(x)
  if n==1
    e(n)=x(n);
    eq(n)=d*sign(e(n));
    xq(n)=eq(n);
  else
    e(n)=x(n)-xq(n-1);
    eq(n)=d*sign(e(n));
    xq(n)=eq(n)+xq(n-1);
  end
end
subplot(3,1,2)
stairs(t,xq,'b')
hold on
plot(t,x,'y')
hold off
xlabel('time')
ylabel('Amplitude')
title('Quantized signal')
for i=1:length(x)
    if e(i)>0
        dm(i)=1
    else
        dm(i)=0
    end
end
subplot(3,1,3);
stairs(t,dm,'g')
axis([0 2 -1 2])
xlabel('time')
ylabel('amplitude')
title('Delta modulated signal')
```

Output/Graph:-

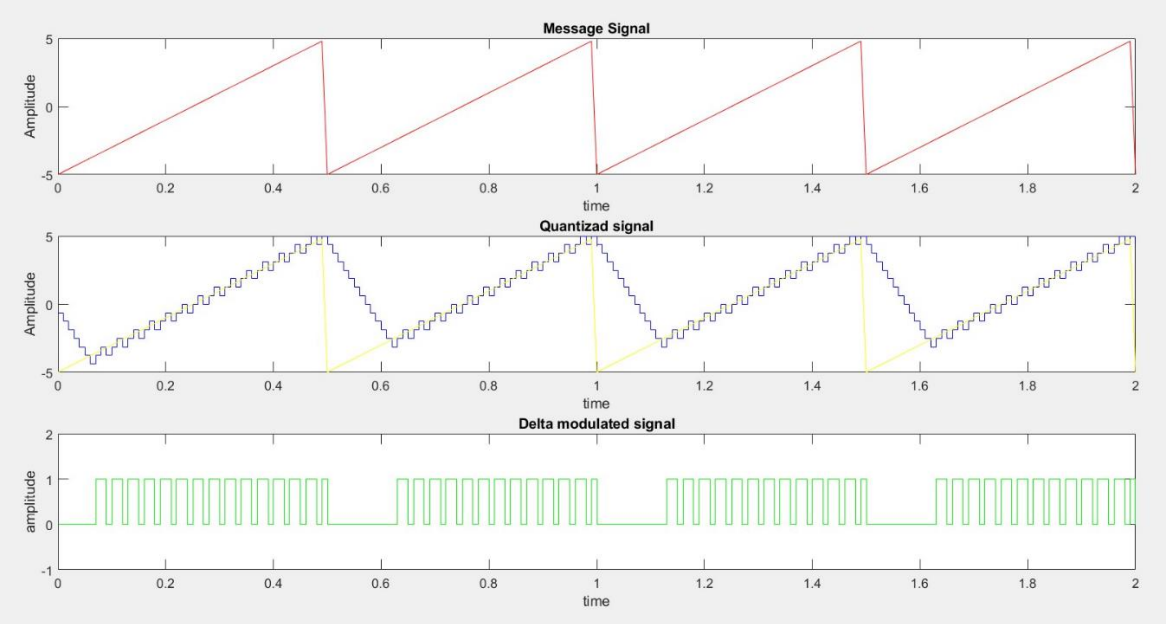


Fig 5.2: Delta modulation

Discussion or Inference of the experiment

In this experiment I came to know that Delta Modulation is simplified PCM. Here I also learnt the concept that instead of sending real value of every sample at each time, variances of present and previous samples are sent.

Conclusion:-

After successfully simulating the experiment through MATLAB I was able to generate the Delta modulated signal graph which is at par with the desired theory result.

Experiment Number	06
Date of Experiment	28/09/2020
Date of Submission	05/10/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC - 06

Aim of The Experiment :-

i. Study of different Data formatting techniques.
ii. Generation and Detection of Amplitude Shift Keying (ASK)

Software Required:-

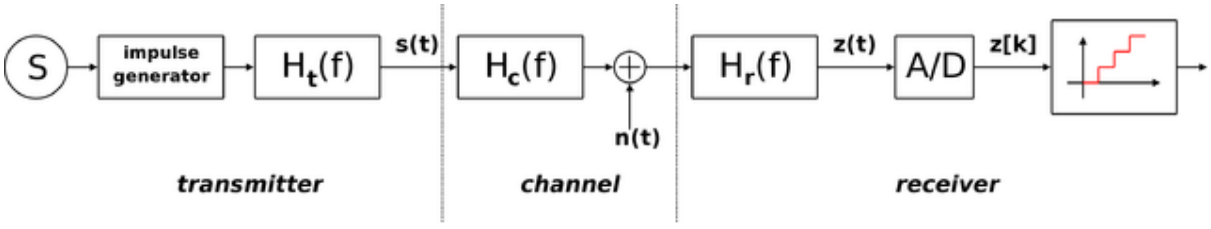
- MATLAB R2018a

Theory

Data formatting techniques are the methods to send computer information over transmission channels that require analog signals, like fibre-optic networks or computer modems. In each of the systems an electromagnetic carrier wave is used to carry the information over great distances and connect digital information users at remote locations. The digital data is used to modulate one or more of the parameters of the carrier wave, This basic process is given the name "Shift-Keying" to differentiate it from the purely analog systems like AM and FM. As with analog modulation, there are three parameters of the carrier wave to vary and therefore three basic types of shift keying:

1. Amplitude shift keying (ASK)
2. Frequency Shift Keying (FSK)
3. Phase Shift Keying (PSK)

ASK: is a form of amplitude modulation that represents digital data as variations in the amplitude of a carrier wave. In an ASK system, the binary symbol 1 is represented by transmitting a fixed-amplitude carrier wave and fixed frequency for a bit duration of T seconds. If the signal value is 1 then the carrier signal will be transmitted; otherwise, a signal value of 0 will be transmitted.



Code:

<<<File: ask.m Comment: Generation of ASK signals>>>

```
%generating ASK signal
%Written By Debagnik Kar 1804373

clc
clear all
close all

time = linspace(0,2,5000)

am = input('Enter amplitude of message signal:    ')
fm = input('Enter frequency of message signal:    ')
ac = input('Enter amplitude of carrier signal (more than
message):    ')
fc = input('Enter frequency of carrier signal (more than
message):    ')

data = am*(square(2*pi*fm*time)+1)

carrier = ac*cos(2*pi*fc*time)

signal = data.*carrier

subplot 311
title('message signal')
plot(time, data)
xlabel('Time')
ylabel('Magnitude')

subplot 312
title('carrier signal')
plot(time, carrier)
xlabel('Time')
ylabel('Magnitude')

subplot 313
title('Modulated signal')
plot(time, signal)
xlabel('Time')
ylabel('Magnitude')
```

Output/Graph:-

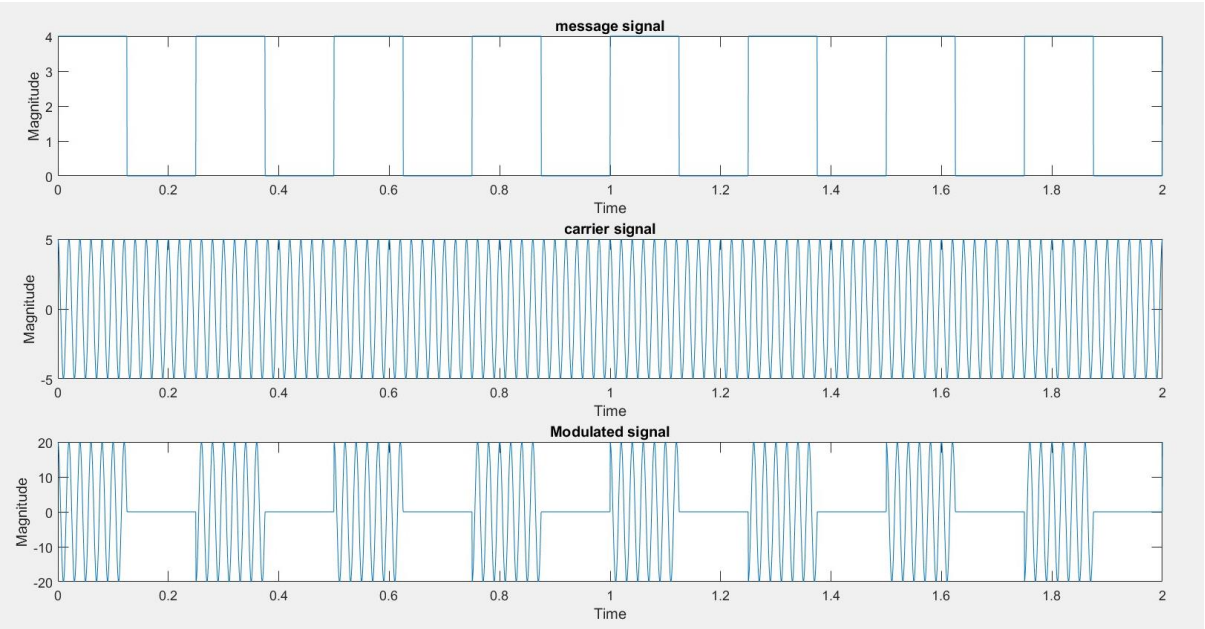


Fig 6.1: Generation of ASK Signal.

Discussion or Inference of the experiment

In this experiment we learnt about different digital modulation techniques, and realized how it provides more information capacity, high data security, quicker system availability with great quality communication. Also, its capacity to convey larger amounts of data than analog modulation techniques.

We also learnt about ASK technique in detail through MATLAB simulation, and how it provides high bandwidth efficiency. In the process we also realized that It has simple receiver design.

Conclusion:-

The generation and detection of ASK data modulation technique was successful using MATLAB R2018a software. The simulation matched predicted output as we learnt in our theory classes.

Experiment Number	07
Date of Experiment	05/10/2020
Date of Submission	12/10/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

i. Generation and Detection of Frequency Shift Keying (FSK).
ii. Generation and Detection of Binary Phase Shift Keying (BPSK)

Equipment / Software Required:-

- MATLAB R2018a

Theory

Frequency-shift keying (FSK) is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal. The technology is used for communication systems such as telemetry, weather balloon radiosondes, caller ID, garage door openers, and low frequency radio transmission in the VLF and ELF bands. The simplest FSK is binary FSK (BFSK). BFSK uses a pair of discrete frequencies to transmit binary (0s and 1s) information. With this scheme, the "1" is called the mark frequency and the "0" is called the space frequency.

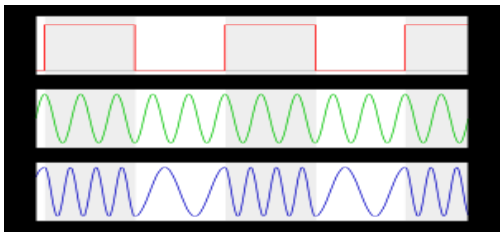


Fig 7.1: FSK Signal

Phase-shift keying (PSK) is a digital modulation process which conveys data by changing (modulating) the phase of a constant frequency reference signal (the carrier wave). The modulation is accomplished by varying the sine and cosine inputs at a precise time. It is widely used for wireless LANs, RFID and Bluetooth communication.

BPSK (also sometimes called PRK, phase reversal keying, or 2PSK) is the simplest form of phase shift keying (PSK). It uses two phases which are separated by 180° and so can also be termed 2-PSK. It does not particularly matter exactly where the constellation points are positioned, and in this figure they are shown on the real axis, at 0° and 180°. Therefore, it

handles the highest noise level or distortion before the demodulator reaches an incorrect decision. That makes it the most robust of all the PSKs. It is, however, only able to modulate at 1 bit/symbol (as seen in the figure) and so is unsuitable for high data-rate applications.

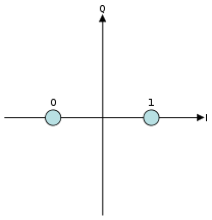


Fig 7.2: BPSK constellation diagram

Code:-

<<<File BFSK.m Comment: Generate and demodulate BFSK Signal>>>

```
% Generation and demodulation of BFSK signal
% Written by Debagnik Kar 1804373

clc
clear all
close all


%Psuedo-random Message generation in binary


n =input('Enter the no. of bits:     ')
data = randi([0 1], 1, n)


for i=1:n
    if data(i) ==1
        Data(i)=1
    else
        Data(i)=0
    end
end


s = 100
r = 1
t = 0:1/s:n


for j=1:length(t)
    if t(j)<=r
        y(j) = Data(r)
    else
        y(j) = Data(r)
```

```
        r = r+1
    end
end

subplot 411
plot(t, y,'b')

axis([0 n -0.5 1.5])
title('Message signal')
ylabel('Amplitude')
xlabel('Time')

%Generation of 2 Carrier signal

f1 = 5
f2 = 10

c1 = cos(2*pi*f1*t)
c2 = cos(2*pi*f2*t)

subplot 412

plot(t, c1, 'g')
axis([0 n -1.5 1.5])
title('Low frequency carrier signal')
ylabel('amplitude')
xlabel('Time')

subplot 413
plot(t, c2, 'm')
axis([0 n -1.5 1.5])
title('High frequency carrier signal')
ylabel('Amplitude')
xlabel('Time')

%Modulation of the message signal to BFSK

for j=1:length(t)
    if y(j)==1
        x(j)=c1(j)
    else
        x(j)=c2(j)
    end
end

subplot 414
plot(t, x, 'k')

axis([0 n -1.5 1.5])
title('Modulated signal')
ylabel('amplitude')
```

```
xlabel('Time')

% Demodulation of the BFSK signal

l = x
l1 = l.*c1

Int1 = []
for I=0:s:length(l1)-s
    Int = (1/s)*trapz(l1(I+1:I+s))
    Int1 = [Int1 Int]
end

l2 = l.*c2

Int2 = []
for I=0:s:length(l2)-s
    INT = (1/s)*trapz(l2(I+1:I+s))
    Int2 = [Int2 INT]
end

demod = Int1 - Int2
disp('Detected Bits')
det = (demod>0) %Recieved bits

figure(2)

subplot 311
plot(t, y, 'r')

title('Message signal')
axis ([0 n -0.5 1.5])
ylabel('Amplitude')
xlabel('Time')

subplot 312
plot(t, x, 'g')

axis([0 n -1.5 1.5])
title('Modulated signal')
ylabel('Amplitude')
xlabel('Time')

subplot 313
stairs(0:n, [det data(n)])

axis ([0 n -0.5 1.5])
title('Demodulated signal')
ylabel('Amplitude')
xlabel('Time')
```

<<<File: BPSK.m Comment: Generate and restore BPSK signals>>>

```
%Generate and demodulate BPSK Signals
%Written by Debagnik Kar 1804373

clc
clear all
close all

%Generating a pseudo-random number as message signal

n = input('Enter the length of data in bits:    ')
data = randi([0 1], 1, n)


for i=1:length(data)
    if data(i) == 1
        Data(i)=1
    else
        Data(i)= -1
    end
end


r=1
t = 0:0.01:length(data)


for j = 1:length(t)
    if t(j) <=r
        y(j) = Data(r)
    else
        y(j) = Data(r)
        r = r+1
    end
end

subplot 411
plot(t,y,'g')

title('Message Signal')
axis([0 n -1.5 1.5])
grid on
xlabel('Time')
ylabel('Amplitude')

%carrier Signal
fc =20
carrier = cos(2*pi*pi*t)
subplot 412
plot(t, carrier, 'r')
```

```
title('Carrier Signal')
axis([0 n -1.5 1.5])
grid on
xlabel('Time')
ylabel('Amplitude')

%Modulation
psk = y.*carrier
subplot 413
plot(t, psk, 'm')

title('Modulated Signal')
axis([0 n -1.5 1.5])
grid on
xlabel('Time')
ylabel('Amplitude')


%Demodulation


for j=1:length(t)
    if psk(j) == carrier(j)
        demod(j) = 1
    else
        demod(j) = 0
    end
end


subplot 414
plot(t,demod)

title('Demodulation Signal')
axis([0 n -0.5 1.5])
grid on
xlabel('Time')
ylabel('Amplitude')
```

Output/Graph:-

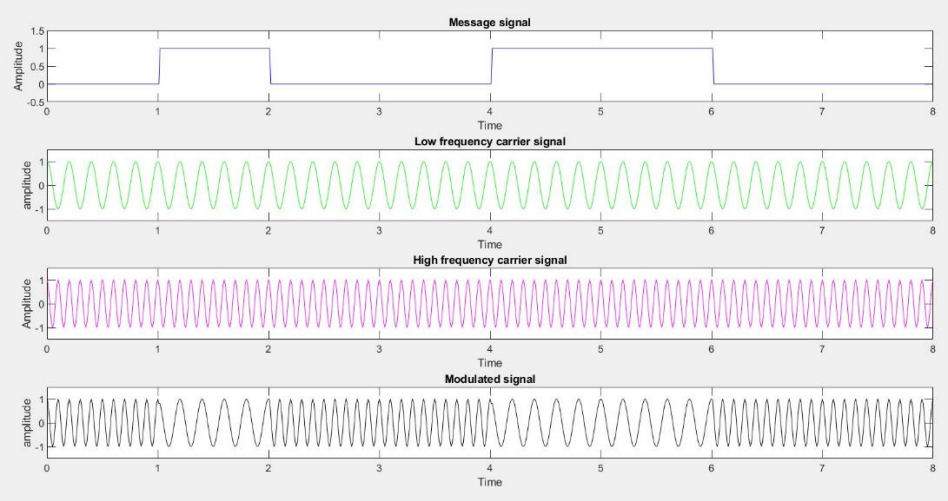


Fig 7.3: Modulation of the message signal using BFSK

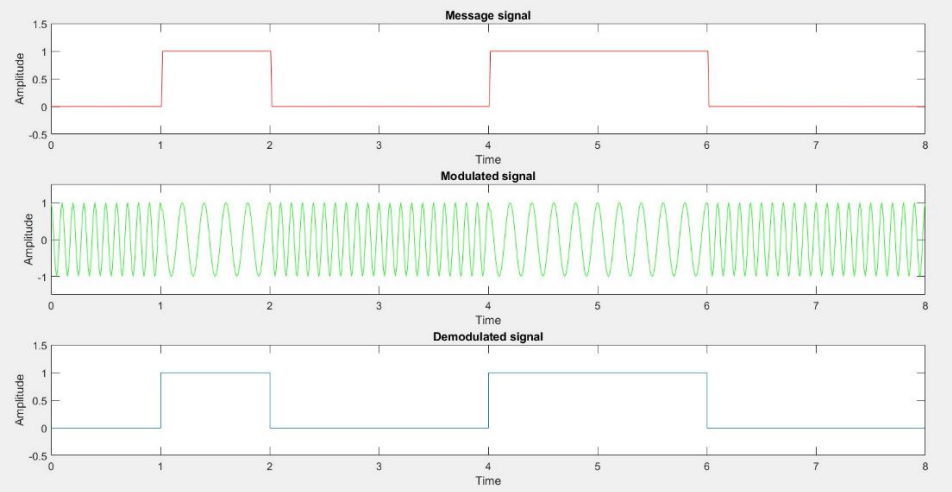


Fig 7.4: Demodulating the modulated signal back

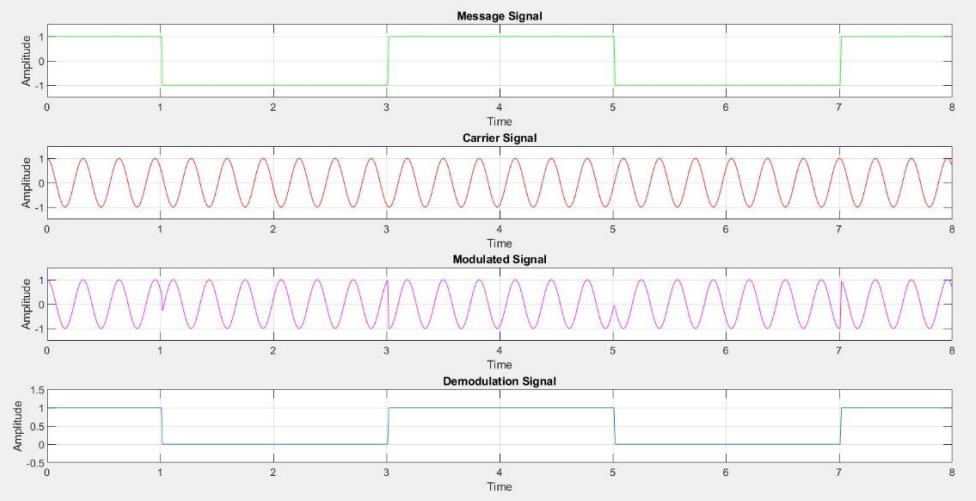


Fig 7.5: Modulating and demodulating a Message using BPSK

Discussion or Inference of the experiment

In this experiment we learnt about two of the most prominent data modulation technique, that is FSK and BPSK, We learnt that Frequency Shift Keying (FSK) is the digital modulation technique the frequency of the carrier signal varies according to the discrete digital changes while in BPSK the sine wave carrier takes two phase reversals such as 0° and 180°, thereby also called as 2-phase PSK (or) Phase Reversal Keying.

Conclusion:-

The generation and detection of BPSK and FSK data modulation technique was successful using MATLAB R2018a software. The simulation matched predicted output as we learnt in our theory classes.

Experiment Number	08
Date of Experiment	12/10/2020
Date of Submission	19/10/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC-06

Aim of The Experiment :-

Generation and Detection of Quadrature Phase Shift Keying (QPSK).

Software Required:-

MATLAB R2018a

Theory

The Quadrature Phase Shift Keying QPSK is a variation of BPSK, and it is also a Double Side Band Suppressed Carrier DSBSC modulation scheme, which sends two bits of digital information at a time, called as bigits.

Instead of the conversion of digital bits into a series of digital stream, it converts them into bit pairs. This decreases the data bit rate to half, which allows space for the other users.

The QPSK Modulator uses a bit-splitter, two multipliers with local oscillator, a 2-bit serial to parallel converter, and a summer circuit. Following is the block diagram for the same.

At the modulator's input, the message signal's even bits (i.e., 2nd bit, 4th bit, 6th bit, etc.) and odd bits (i.e., 1st bit, 3rd bit, 5th bit, etc.) are separated by the bits splitter and are multiplied with the same carrier to generate odd BPSK (called as PSKI) and even BPSK (called as PSKQ). The PSKQ signal is anyhow phase shifted by 90° before being modulated.

The QPSK waveform for two-bits input is as follows, which shows the modulated result for different instances of binary inputs.

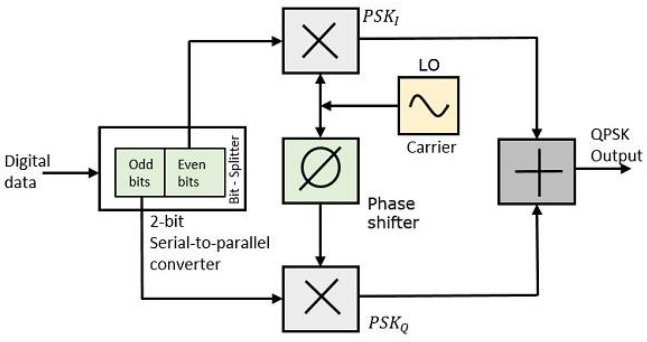


Fig 8.1: QPSK Modulator Block diagram

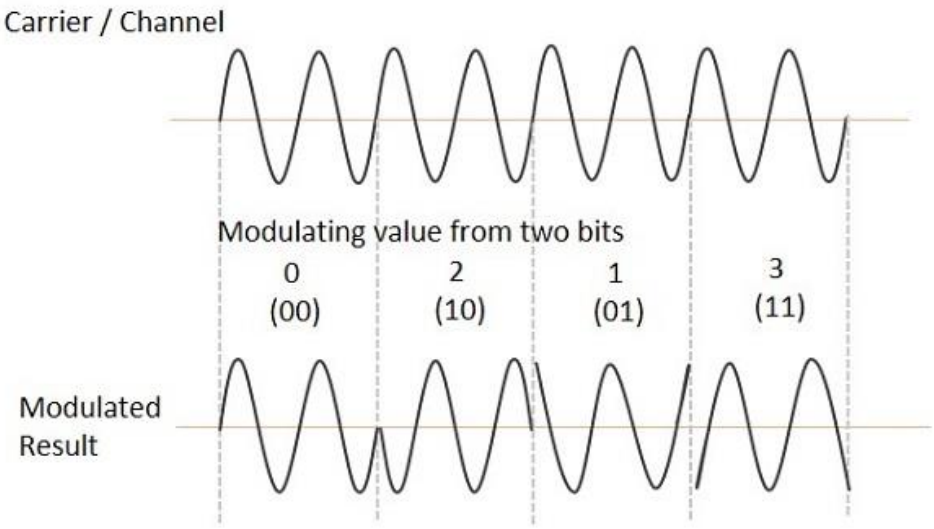


Fig 8.2: QPSK Modulated signal

Code:-

<<<File:QPSK.m Comment: Generation of QPSK using a pseudo-random 8–bit Binary number>>>

```
%Generate and demodulate QPSK Signals
%Written by Debagnik Kar 1804373

clc
clear all
close all

%Generating a psuedo-random number as message signal

n = input('Enter the length of data in bits:    ')
f = input('Enter the frequency of carrier:    ')
data = randi([0 1], 1, n)

for i=1:length(data)
    if data(i) == 1
        Data(i)=1
    else
        Data(i)= -1
    end
end

r=1
t = 0:0.01:length(data)
for j = 1:length(t)
    if t(j) <=r
        y(j) = Data(r)
```

```
     else
        y(j) = Data(r)
        r = r+1
    end
end

s1 = 0
s2 = 0
for i = 1:length(data)
    if mod(i,2) == 0
        s1(i) = Data(i)
    elseif mod(i,2) ~= 0
        s2(i) = Data(i)
    end
end

r=1
for j = 1:length(t)
    if t(j) <=r
        Even(j) = s1(r)
    else
        Even(j) = s1(r)
        r = r+1
    end
end

s2 = [s2 0]
r=1

for j = 1:length(t)
    if t(j) <=r
        Odd(j) = s2(r)
    else
        Odd(j) = s2(r)
        r = r+1
    end
end


c1 = Odd.*sin(2*pi*f*t)
c2 = Even.*cos(2*pi*f*t)

subplot 511
plot(t,y)
grid on
title('Message signal')
xlabel('Time')
ylabel('Amplitude')
axis([0 n -1.5 1.5])

subplot 512
```

```
plot(t,cos(2*pi*f*t),'k')
grid on
title('Carrier signal')
xlabel('Time')
ylabel('Amplitude')

subplot 513
plot(t,c1,'b')
grid on
title('Odd Phase shift signal')
xlabel('Time')
ylabel('Amplitude')

subplot 514
plot(t,c2,'r')
grid on
title('Even phase shift')
xlabel('Time')
ylabel('Amplitude')

c = c1+c2

subplot 515
plot(t,c,'m')
grid on
title('QPSK Modulated signal')
xlabel('Time')
ylabel('Amplitude')
```

Output/Graph:-

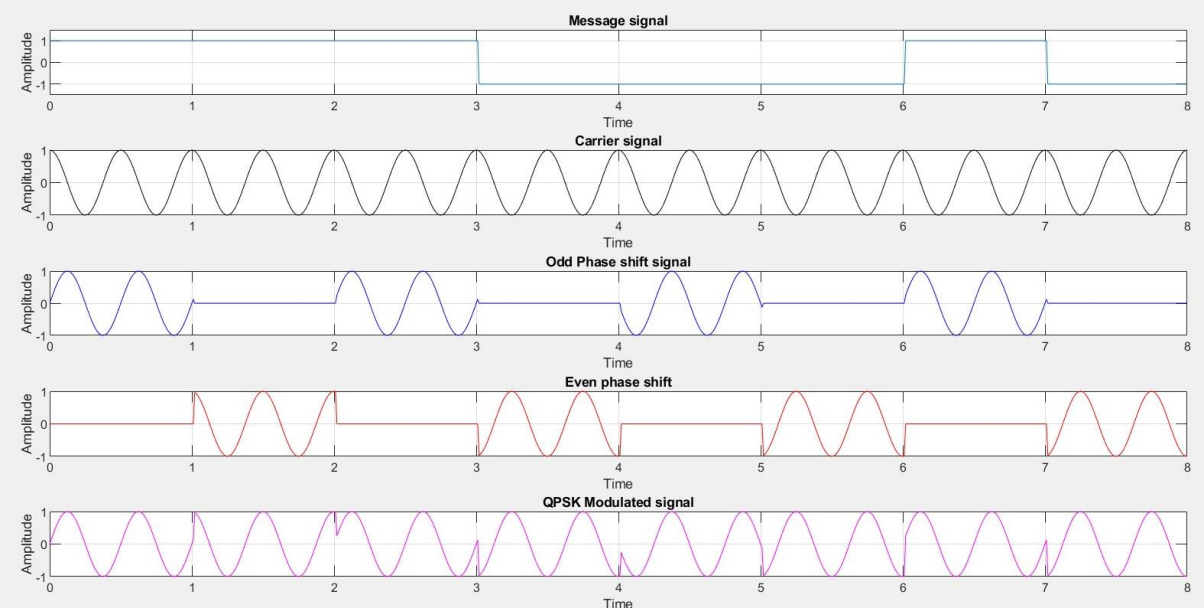


Fig 8.3: QPSK Simulation

Discussion or Inference of the experiment

In this experiment we learnt about another digital modulation technique that is QPSK, in which the sine wave carrier takes four phase reversals such as 0°, 90°, 180°, and 270°

We realised that QPSK is a variation of BPSK, and it is also a DSB-SC (Double Sideband Suppressed Carrier) modulation scheme, which send two bits of digital information at a time, called as bigits.

The difference is Instead of the conversion of digital bits into a series of digital stream, it converts them into bit-pairs. This decreases the data bit rate to half, which allows space for the other users.

Conclusion:-

The generation and detection of QPSK data modulation technique was successful using MATLAB R2018a software. The simulation matched predicted output as we learnt in our theory classes

Aim:

a. To import an audio file, add noise and then reduce noise

b. To send a FM modulated signal add noise and reduce the noise at the recieving end.

Software Needed:

1. Google Colab Hosted IPython Notebook service Or Jupyter Notebook
2. Numpy, Scipy, Soundfile, Noisereduce Libraries
3. Audio Files (Get here)

Theory:

Noise reduction is the process of removing noise from a signal. Noise reduction techniques exists for audio and images. Noise reduction algorithms tends to alter signals to an extent that the noise in the signal reduces. Thus algotitms like this have been worked with several times the past in the feilds of signal processing, statistics, informaton theory and even machine learning.

Noise is an unwanted signal which interferes with the original message signal and corrupts the parameters of the message signal. Noise is generally a high frequency signal that is additive in nature. When a signal is transmitted through a channel (transmission media) noise is added to it and therefore the data that we receive at the receiver end is noisy. It is type of a communication impairment.

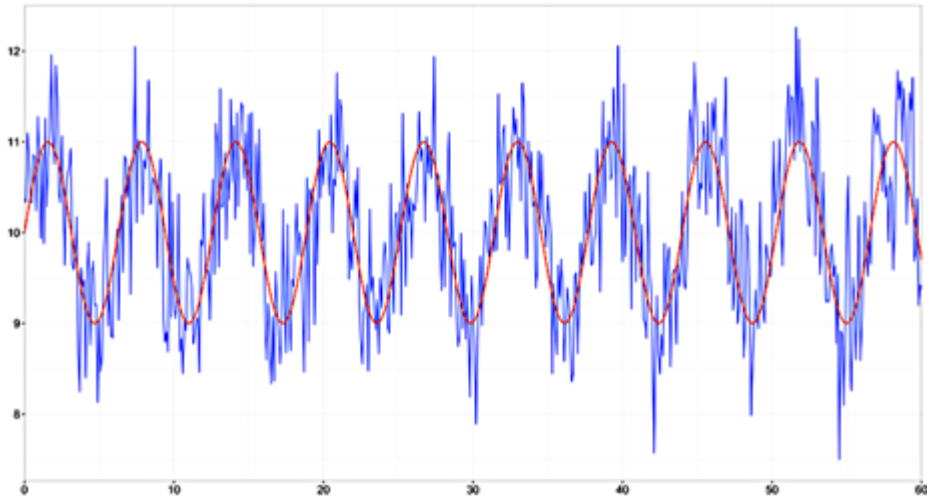


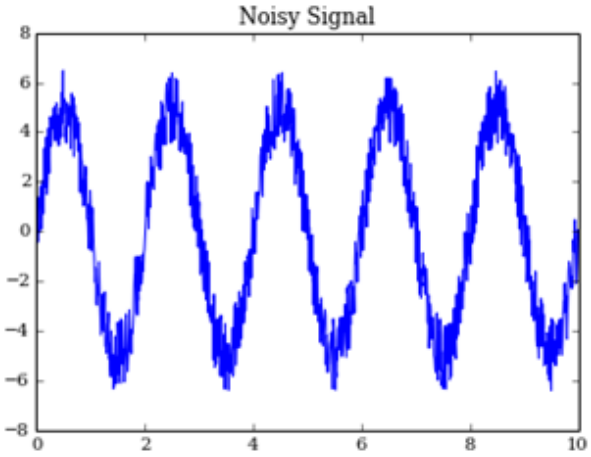
Fig: An example of noisy Signal

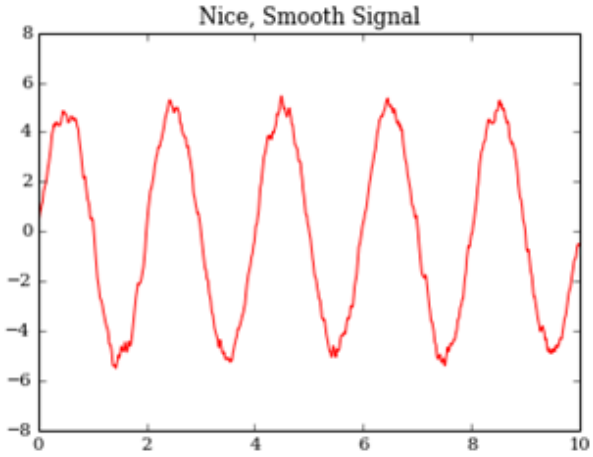
The best way to deal with it is by improving the SNR of the signal i.e. the signal to noise ratio

$$SNR = \frac{Power_{Signal}}{Power_{Noise}} = \left(\frac{Amplitude_{Signal(RMS)}}{Amplitude_{Noise(RMS)}}\right)^2$$

To improve the SNR, we can either increase the signal power or decrease the noise power.

Noise reduction There are many different types of filters that are used to reduce the effects of noise. Like a low pass filter which takes the lower frequencies and rejects the higher frequencies, moving average filter that takes average values at the time and the FIR filter which allows low frequency components.





Noise reduction in python

Here we will be using python code to simulate the waveforms of the signals and show how noise that is added during transmission of the signal is reduced using a python script by importing the library noisereduce. Apart from that we will be using tensorflow gpu library to speedup the fft and the goussia convolution algorithms.

The noisereduce alorithm require two inputs:

1. A noise signal
2. A message signal

Main steps of the algorithm

1. An FFT is calculated over the noise signal.
2. Statistics are calculated over the FFT of the noise in frequency.
3. A threshold is calculated based upon the statistics of the noise.
4. An FFT is calculated based over signal

5. A mask is determined by comparing the signal FFT to the threshold.
6. The mask is smoothed with a filter over frequency and time.
7. The mask is applied to signal and is inverted.

Code and output

Written By:

1. Debagnik Kar 1804373
2. Sayani Ghoroi 1804406

Installing required python packages

In [ ]:
```
!pip install tensorflow-gpu==2.0.0-beta
!pip install noisereduce
!pip install soundfile
```

Importing various library

In [2]:
```
import IPython
from scipy.io import wavfile as wav
from scipy import signal as sp
import noisereduce as nr
import soundfile as sf
from noisereduce.generate_noise import band_limited_noise
import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive
import math as m
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/noisereduce/noisereduce.py:5: TqdmE
xperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use `
tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  from tqdm.autonotebook import tqdm
```

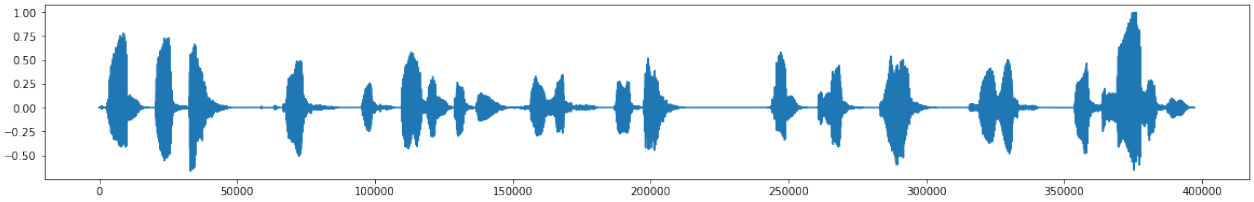
In [3]:
```
drive.mount('/content/drive')
data, rate = sf.read('/content/drive/My Drive/Music/music1.wav')
data = data
IPython.display.Audio(data=data, rate=rate)
```

```
Mounted at /content/drive
```

Out[3]: Your browser does not support the audio element.

In [4]:
```
fig, ax = plt.subplots(figsize=(20,3))
ax.plot(data)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7fa0bafc1400>]
```

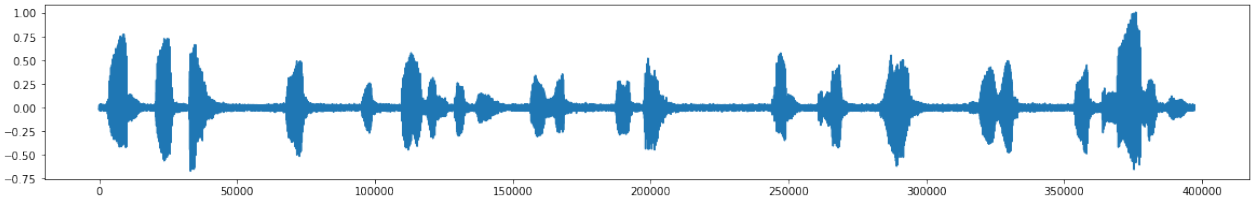


Adding noise

In [5]:
```
noise_len = 2
noise = band_limited_noise(min_freq=2000, max_freq = 12000, samples=len(da
ta), samplerate=rate)*10
noise_clip = noise[:rate*noise_len]
audio_clip_band_limited = data + noise
```

In [6]:
```
fig, ax = plt.subplots(figsize=(20,3))
ax.plot(audio_clip_band_limited)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x7fa0b94ceac8>]
```

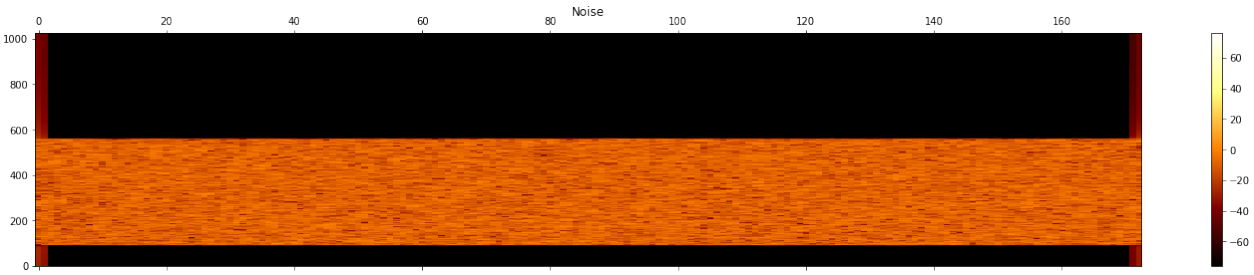


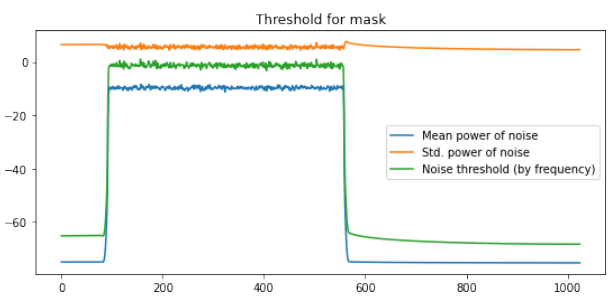
In [7]:
```
IPython.display.Audio(data=audio_clip_band_limited, rate=rate)
```

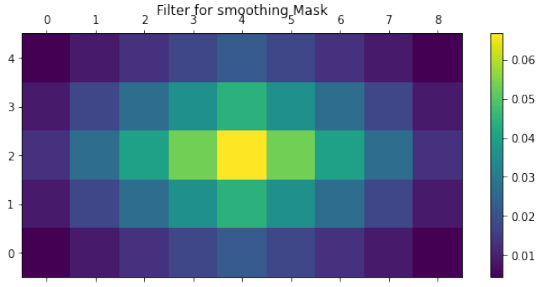
Out[7]: Your browser does not support the audio element.

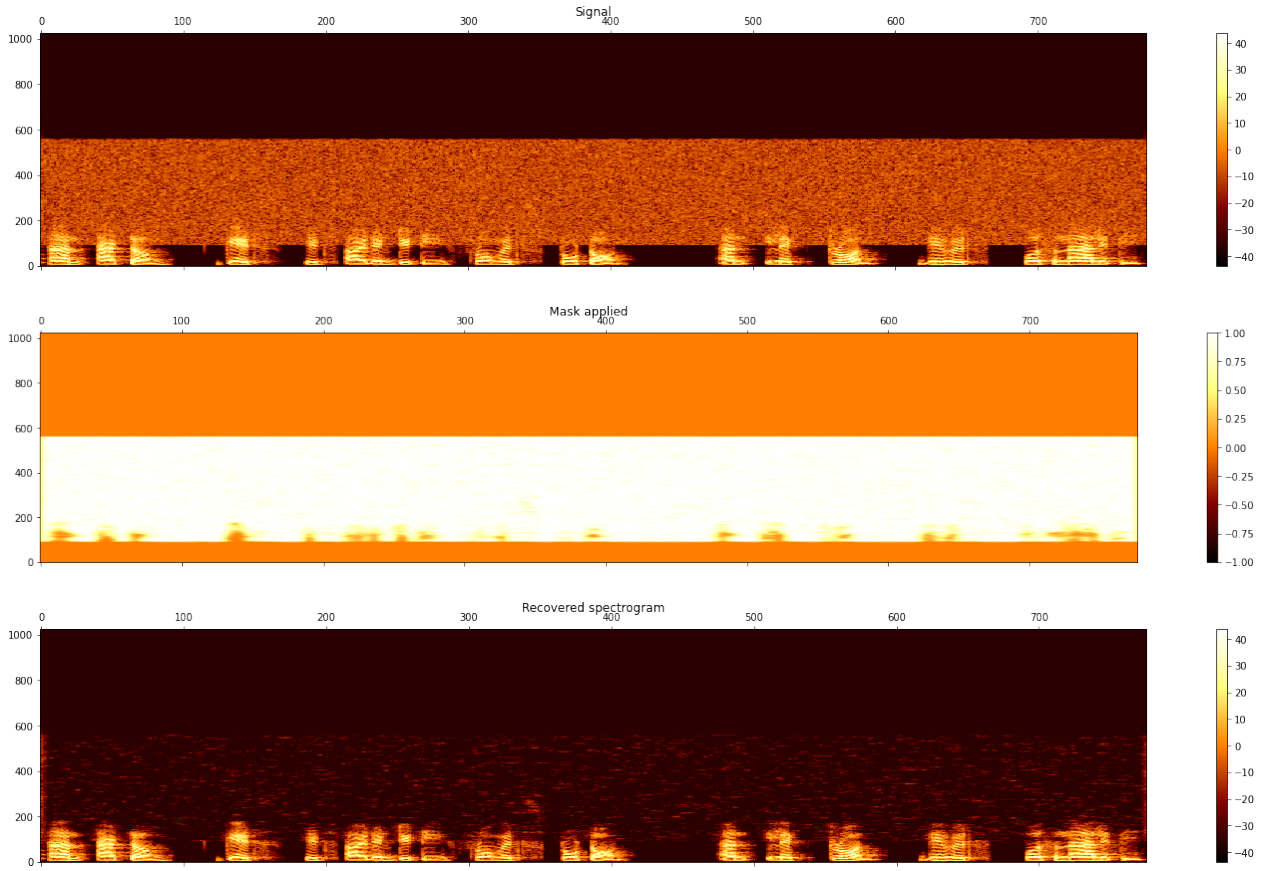
Lets reduce the noise

In [8]:
```
noise_reduced = nr.reduce_noise(audio_clip=audio_clip_band_limited, noise_
clip=noise_clip, prop_decrease=1.0, verbose=True)
```







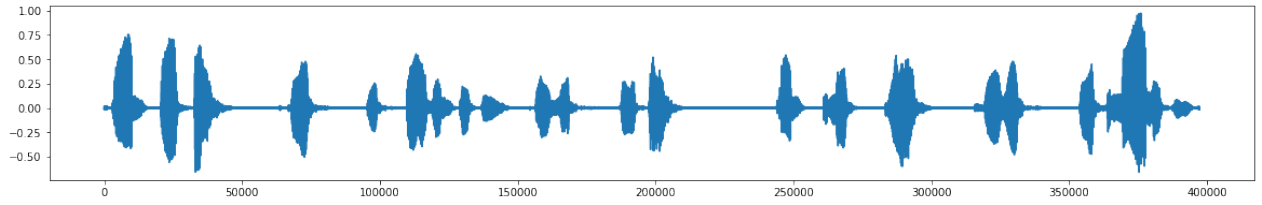


In [9]:
```
IPython.display.Audio(data=noise_reduced, rate=rate)
```

Out[9]: Your browser does not support the audio element.

In [10]:
```
fig, ax = plt.subplots(figsize=(20,3))
ax.plot(noise_reduced)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x7fa0b417de48>]
```



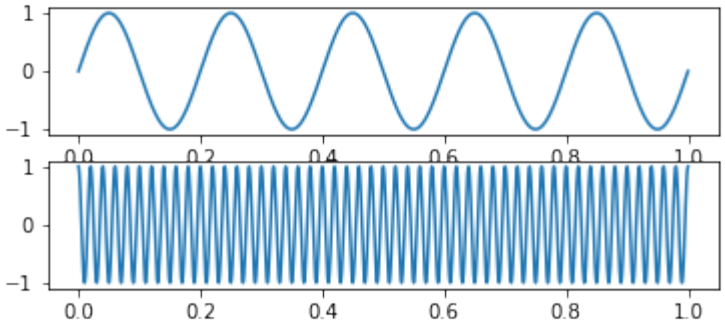
Modulating the signal in FM

We took a 5Hz Sine wave message signal and a 50 Hz carrier signal to simulate

In [11]:
```
time = np.linspace(0,1,2000)
message = np.sin(2*m.pi*5*time)
carrier = np.cos(2*m.pi*50*time)
```

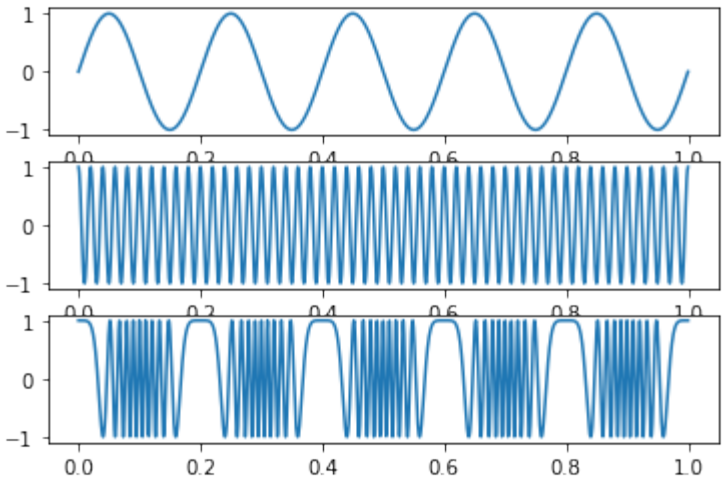
In [12]:
```
plt.subplot(3,1,1)
plt.plot(time,message)
plt.subplot(3,1,2)
plt.plot(time,carrier)
```

```
plt.show()
```



In [13]:
```
fmsignal = np.cos(2*m.pi*50*time-(5*(np.sin(2*m.pi*5*time)+np.sin(2*m.pi*5
*time)))) #mu=5 taken
plt.subplot(3,1,1)
plt.plot(time,message)
plt.subplot(3,1,2)
plt.plot(time,carrier)
plt.subplot(313)
plt.plot(time,fmsignal)
```

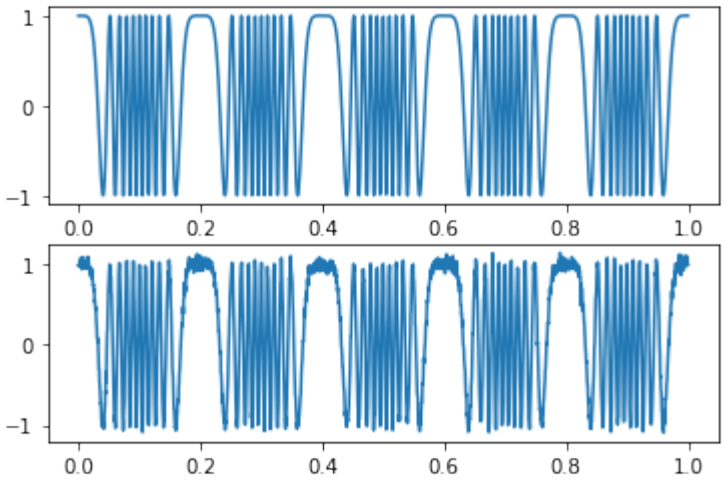
Out[13]: [<matplotlib.lines.Line2D at 0x7fa0b400f5f8>]



Adding random noise

In [14]:
```
noise = np.random.normal(0,0.05,len(time))
noisysignal = fmsignal+noise
plt.subplot(2,1,1)
plt.plot(time,fmsignal)
plt.subplot(2,1,2)
plt.plot(time,noisysignal)
```

Out[14]: [<matplotlib.lines.Line2D at 0x7fa0b4455ef0>]



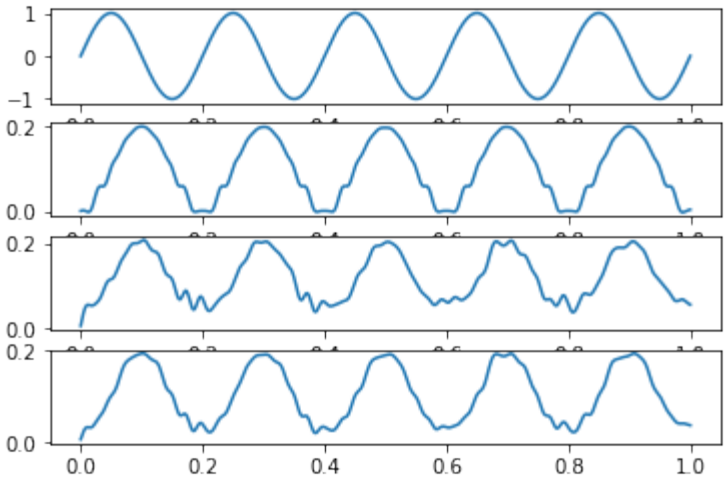
In [15]:
```
noisyclip = noise[:len(time)]
reducednoise = nr.reduce_noise(audio_clip=noisysignal,noise_clip=noisyclip
 ,verbose=False)
```

In [16]:
```
dem0 = np.diff(fmsignal)
dem1 = np.diff(noisysignal)
dem2 = np.diff(reducednoise)

dem0 = np.append(dem0, 0)
dem1 = np.append(dem1, 0)
dem2 = np.append(dem2, 0)

rect_dem0 = abs(dem0)
rect_dem1 = abs(dem1)
rect_dem2 = abs(dem2)
```

In [17]:
```
b, a = sp.butter(10, 0.0499, 'low')
demod0 = sp.filtfilt(b, a, rect_dem0)
demod1 = sp.filtfilt(b, a, rect_dem1)
demod2 = sp.filtfilt(b, a, rect_dem2)
```

In [18]:
```
plt.subplot(4,1,1)
plt.plot(time,message)
plt.subplot(4,1,2)
plt.plot(time,demod0)
plt.subplot(4,1,3)
plt.plot(time, demod1)
plt.subplot(4,1,4)
plt.plot(time, demod2)
```

Out[18]: [<matplotlib.lines.Line2D at 0x7fa0b41650f0>]



Inference

From this open ended project we learned about the effect of noice on a message signal during communication through a channel .we generated a frequency modulated message signal. we also were able to successfully reduce the noise that was added to the signal with the help of a software algorithm that used the library noisereduce , in python. The demodulation of the signals were also done with the help of low pass butterworth filter and all the waveforms were simulated as expected in theory.

Conclusion

From this project we successfully performed noise reduction of a signal using software algorithm in Python and simulated there waveforms.

This notebook is hosted at Here

Experiment Number	10
Date of Experiment	02/11/2020
Date of Submission	07/11/2020
Name of the student	Debagnik Kar
Roll Number	1804373
Section	ETC - 06

Aim of The Experiment :-

Open Ended 2
Application of Frequency division multiplexing using audio signal

Software Required:-

1. MATLAB r2018a
2. Audio files (Get it here)

Theory

Multiplexing is a process by which a number of signals can be transmitted through a single channel without interference.

There are mainly two type of multiplexing techniques for analog signals :

- Time division multiplexing
- Frequency division multiplexing.

In this project we will perform application of frequency division multiplexing using 3 audio clips.

Frequency-division multiplexing (FDM) is a technique by which the total bandwidth available in a communication medium is divided into a series of non-overlapping frequency bands, each of which is used to carry a separate signal.

FDM uses a carrier signal at a discrete frequency for each data stream and then combines many modulated signals. When FDM is used to allow multiple users to share a single physical communications medium (i.e. not broadcast through the air), the technology is called frequency-division multiple access (FDMA).

Here frequency division multiplexing is performed on 3 SSBSC modulated signals (audio clips) with different carrier frequencies. Each modulated signal is allocated with different frequency spectrum created at lets say fc1,fc2,and fc3 and therefore signal m1,m2 and m3 do not interfere with each other.

Code:-

<<<SimpleFDM.m Comment: Generates SSBSC signals from the audio files then transmit it through a noisy in frequency division multiplexed way, upon receiving noise is reduced, demultiplex and then SSBSC demodulated. Then played the message back>>>

```
%Experiment 10: Open ended 2:
%Application of FDM Multiplexing and Demultiplexing
%Collaboration of
%Debagnik Kar (1804373) and Sayani Ghoroi (1804406)
clear all
clc
close all
%PARAMETERS
bandwidth = 4000; % bandwidth for each frequency band in Hz
media_guard = 300; %
signal_to_noise_ratio = 20;
modudation_ssb = 1;% 1 for Single SSB modulation, 0 for AM
%The first signal will be placed in the third channel, the
second in the
%fourth and the third in the fifth.
frec_carrier1 = bandwidth*3;% Carrier frequency in Hz
frec_carrier2 = bandwidth*4;
frec_carrier3 = bandwidth*5;
Fs = frec_carrier3*2+5000;
cutoff_freq_passfilter = 2500;
% 1 show graphics, 0 don't
graphics = 1;
% 1 play sounds, 0 don't
sounds = 1;
%Define filters
[B,A] = butter(4,cutoff_freq_passfilter/(Fs/2));
low_pass = @(S) filter(B,A,S); %function handeling filter() in
low_pass variable to know more about it go to
[C1,D1] = butter(2,[bandwidth*2+media_guard bandwidth*3-
media_guard]/(Fs/2));
band_filter3 = @(S) filter(C1,D1,S);
[C2,D2] = butter(2,[bandwidth*3+media_guard bandwidth*4-
media_guard]/(Fs/2));
band_filter4 = @(S) filter(C2,D2,S);
[C3,D3] = butter(2,[bandwidth*4+media_guard bandwidth*5-
media_guard]/(Fs/2));
band_filter5 = @(S) filter(C3,D3,S);
%upload the files
[s1, g1] = audioread('1.wav');
len1 = length(s1);
[s2, g2]= audioread('2.wav');
len2 = length(s2);
[s3, g3]= audioread('3.wav'); % you got rick rolled lol in
2020
len3 = length(s3);
```

```
[beep, g4]= audioread('beep-8.wav');
playerbeep = audioplayer(beep,44100);
%are truncated to the length of the minor
min_len = min([len1 len2]);
t = linspace(0,5, min_len);
s1 = s1(1:min_len);
s2 = s2(1:min_len);
s3 = s3(1:min_len);
FLAG = input('STEP 1, the signals are reproduced as they
arrive');
%playing sounds
if (sounds > 0)
    player = audioplayer(s1,g1);
    playblocking(player);
    playblocking(playerbeep);
    player2 = audioplayer(s2,g2);
    playblocking(player2);
    playblocking(playerbeep);
    player3 = audioplayer(s3,g3);
    playblocking(player3);
end
FLAG = input('STEP 2, plot the spectra of the signals as they
arrive');
if (graphics > 0)
    figure
    esps1=abs(fft(s1));
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Signal
Spectrum 1');xlabel("Frequency, Hz");ylabel("Amplitude, dB");
    esps2=abs(fft(s2));
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Signal
Spectrum 2');xlabel("Frequency, Hz");ylabel("Amplitude, dB");
    esps3=abs(fft(s3));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Signal
Spectrum 3');xlabel("Frequency, Hz");ylabel("Amplitude, dB");
end;
FLAG = input('STEP 3, Signals are passed through a low pass
filter and plotted');
%they go through the low pass filter
s1 = low_pass(s1); %function handling by a variable
s2 = low_pass(s2);
s3 = low_pass(s3);
%Plot
if (graphics > 0)
    figure
    esps1=abs(fft(s1));
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Signal
spectrum 1 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps2=abs(fft(s2));
```

```
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Signal
spectrum 2 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps3=abs(fft(s3));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Signal
spectrum 3 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
end
FLAG = input('STEP 4, reproduce the signals after passing them
through the filter');
%Played again
if (sounds > 0)
    playerbeep = audioplayer(beep,44100);
    player = audioplayer(s1,g1);
    playblocking(player);
    playblocking(playerbeep);
    player2 = audioplayer(s2,g2);
    playblocking(player2);
    playblocking(playerbeep);
    player3 = audioplayer(s3,g3);
    playblocking(player3);
    playblocking(playerbeep);
end
FLAG = input('STEP 5, Signals are modulated to different
carriers');
%Modulate
if ( modudation_ssb  > 0)
    s1mod = ssbmod(s1,frec_carrier1,Fs);%modulates
    s2mod = ssbmod(s2,frec_carrier2,Fs);%modulates
    s3mod = ssbmod(s3,frec_carrier3,Fs);%modulates
else
    s1mod = ammod(s1,frec_carrier1,Fs);%modulates
    s2mod = ammod(s2,frec_carrier2,Fs);%modulates
    s3mod = ammod(s3,frec_carrier3,Fs);%modulates
end
%plotted
if (graphics > 0)
    figure
    esps1=abs(fft(s1mod));
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Signal
spectrum1 modulated');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps2=abs(fft(s2mod));
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Signal
spectrum2 modulated');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps3=abs(fft(s3mod));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Signal
spectrum2 modulated');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
end
```

```
FLAG = input('STEP 6, The modulated signals are filtered in
the determined band and summed');
fs1 = s1mod;
fs2 = s2mod;
fs3 = s3mod;
%added
x = fs1+fs2+fs3;
%plotted again
if (graphics > 0)
    figure
    esps1=abs(fft(fs1));
    subplot(4,1,1),plot(esps1),grid on,zoom,title('Signal
spectrum1 modulated and filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps2=abs(fft(fs2));
    subplot(4,1,2),plot(esps2),grid on,zoom,title('Signal
spectrum2 modulated and filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps3=abs(fft(fs3));
    subplot(4,1,3),plot(esps3),grid on,zoom,title('Signal
spectrum3 modulated and filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    espf=abs(fft(x));
    subplot(4,1,4),plot(espf),grid on,zoom,title('Summed
Spectrum');xlabel("Frequency, Hz");ylabel("Amplitude, dB");
end
FLAG = input('STEP 7, add some noise to the transmitted
signal');
if (graphics > 0)
    figure
    esps1=abs(fft(x));
    subplot(2,1,1),plot(esps1),grid on,zoom,title('Full signal
spectrum');xlabel("Frequency, Hz");ylabel("Amplitude, dB");
end
x = awgn(x, signal_to_noise_ratio );
if (graphics > 0)
    esps1=abs(fft(x));
    subplot(2,1,2),plot(esps1),grid on,zoom,title('Full signal
spectrum plus some noise'); xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
end
FLAG = input('STEP 8, upon arrival each band is filtered');
%signals are received and filtered
demuxs1 = band_filter3(x);
demuxs2 = band_filter4(x);
demuxs3 = band_filter5(x);
%Plotted again
if (graphics > 0)
    figure
    esps1=abs(fft(demuxs1));
```

```
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Signal
spectrum1 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps2=abs(fft(demuxs2));
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Signal
spectrum2 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
    esps3=abs(fft(demuxs3));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Signal
spectrum3 filtered');xlabel("Frequency,
Hz");ylabel("Amplitude, dB");
end
FLAG = input('STEP 9, each recovered band is demodulated to
return the signal to the indicated frequency');
%Demodulate
if ( modudation_ssb  > 0)
    demods1 = ssbdemod(demuxs1, frec_carrier1,Fs);
    demods2 = ssbdemod(demuxs2, frec_carrier2,Fs);
    demods3 = ssbdemod(demuxs3, frec_carrier3,Fs);
else
    demods1 = amdemod(demuxs1, frec_carrier1,Fs);
    demods2 = amdemod(demuxs2, frec_carrier2,Fs);
    demods3 = amdemod(demuxs3, frec_carrier3,Fs);
end;
%Plotting
if (graphics > 0)
    figure
    esps1=abs(fft(demods1));
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Demodulated
signal1 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
    esps2=abs(fft(demods2));
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Demodulated
signal2 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
    esps3=abs(fft(demods3));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Demodulated
signal3 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
end
FLAG = input('STEP 10, the recovered signal is passed through
a low pass filter');
%played
demods1 = low_pass(demods1);
demods2 = low_pass(demods2);
demods3 = low_pass(demods3);
if (graphics > 0)
    figure
    esps1=abs(fft(demods1));
```

```
    subplot(3,1,1),plot(esps1),grid on,zoom,title('Demodulated
signal1 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
    esps2=abs(fft(demods2));
    subplot(3,1,2),plot(esps2),grid on,zoom,title('Demodulated
signal2 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
    esps3=abs(fft(demods3));
    subplot(3,1,3),plot(esps3),grid on,zoom,title('Demodulated
signal3 spectrum');xlabel("Frequency, Hz");ylabel("Amplitude,
dB");
end
FLAG = input('STEP 11, Signal reproduce the signal after
transmission');
player4 = audioplayer(demods1,g2);
playblocking(player4);
playblocking(playerbeep);
player5 = audioplayer(demods2,g2);
playblocking(player5);
playblocking(playerbeep);
player6 = audioplayer(demods3,g3);
playblocking(player6);
```

Output/Graph:-

STEP 1, the signals are reproduced as they arrive:

**Three Audios plays*

STEP 2, plot the spectra of the signals as they arrive

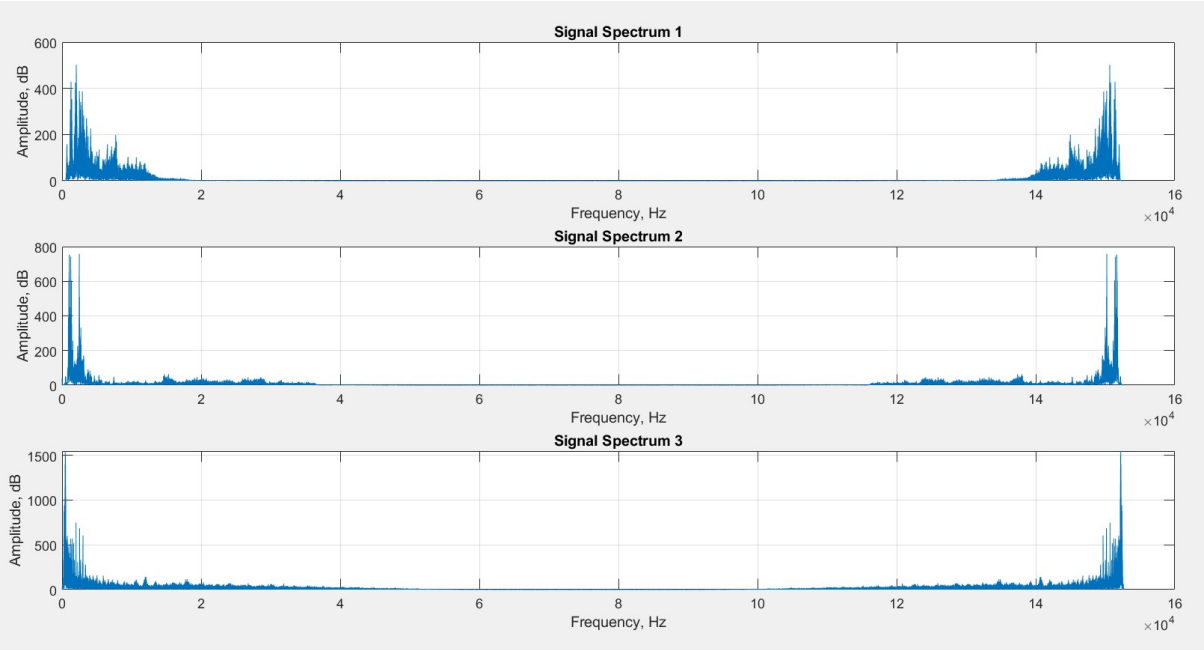


Fig 10.1: Step 2, spectral plot for the three signals

STEP 3, Signals are passed through a low pass filter and plotted

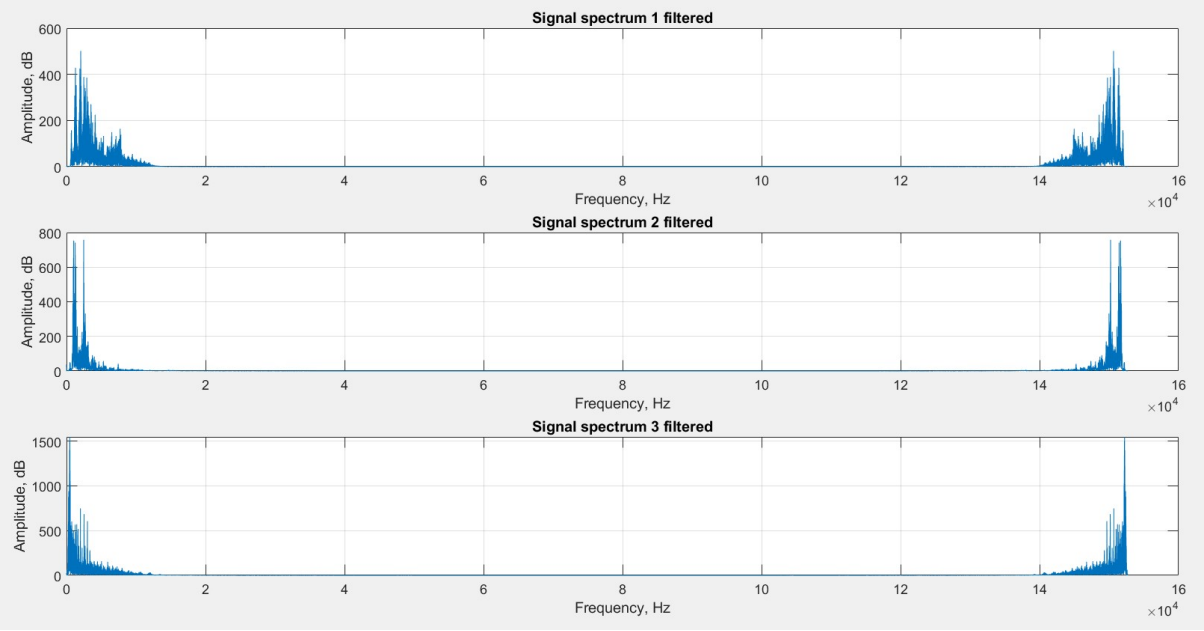


Fig 10.2: Step 3, Spectral plot for filtered signals

STEP 4, reproduce the signals after passing them through the filter

***Filtered audio signal plays*

STEP 5, Signals are modulated to different carriers

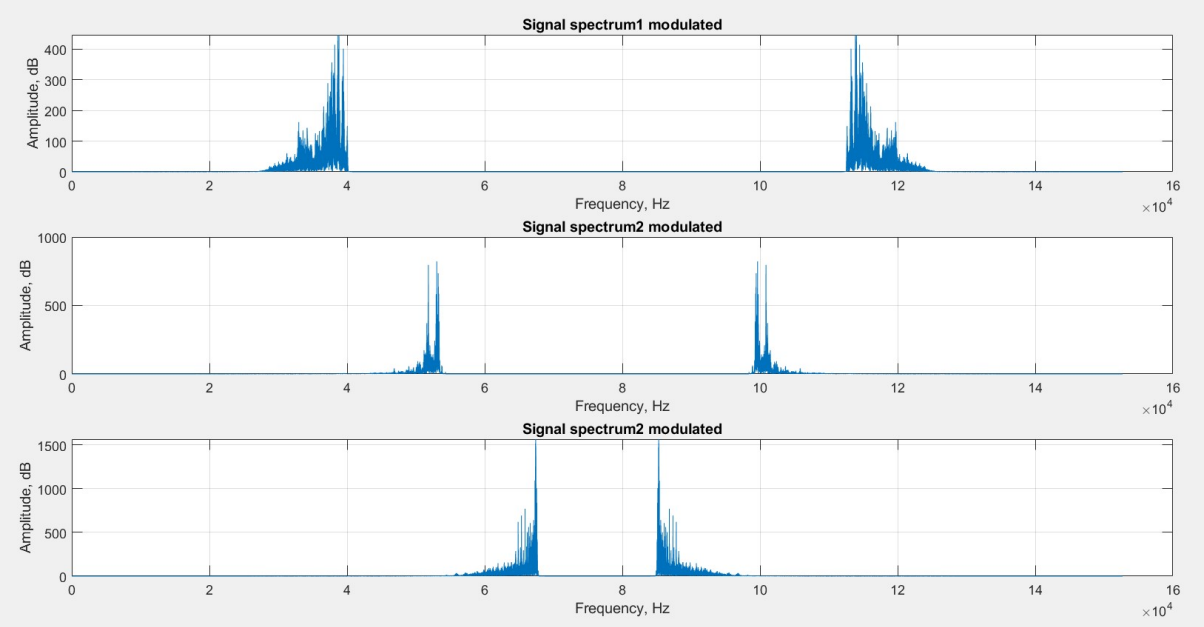


Fig 10.3: Step 5, SSBSC Modulated and filtered spectral plot

STEP 6, The modulated signals are filtered in the determined band and summed

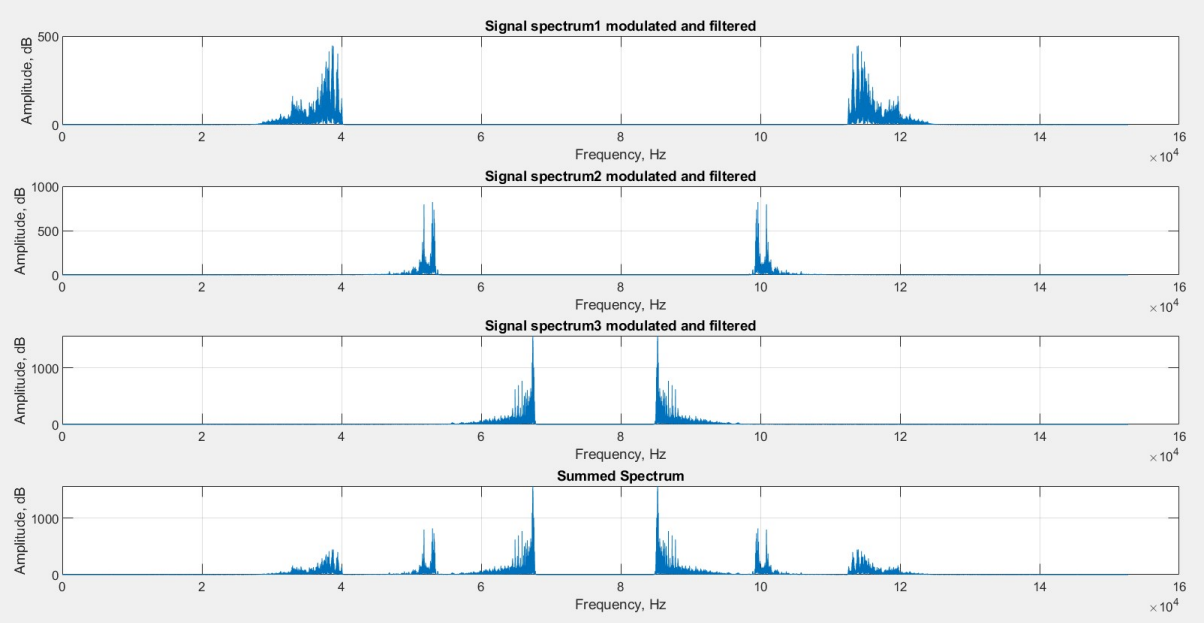


Fig 10.4: The three signals are frequency multiplexed

STEP 7, add some noise to the transmitted signal

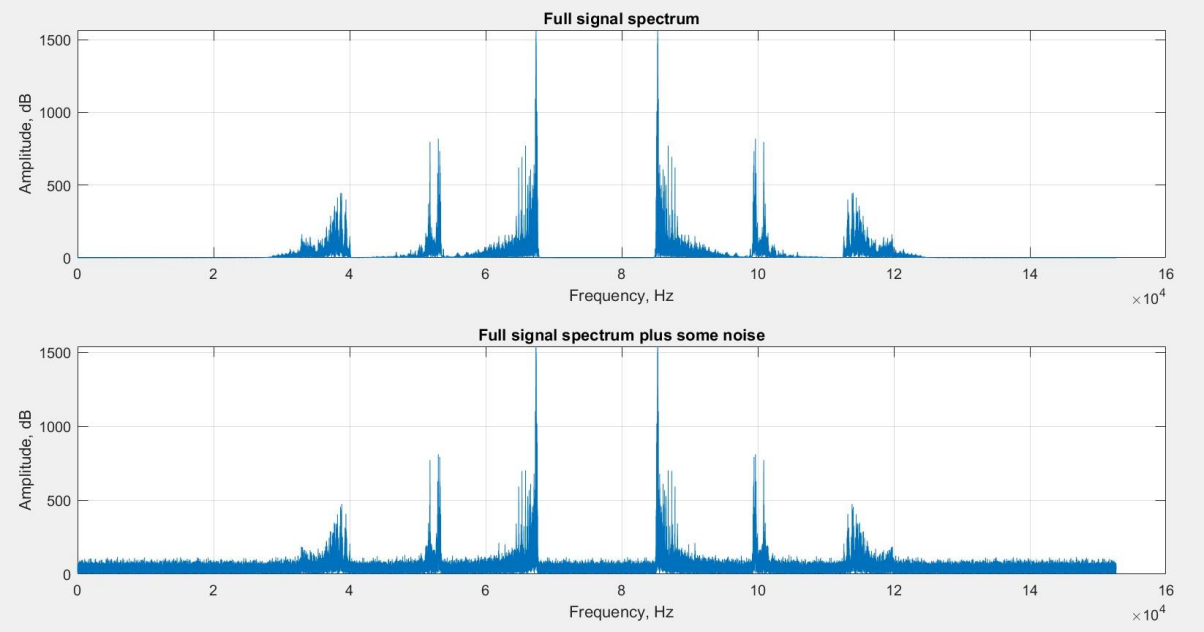


Fig 10.5: Multiplexed signal spectral plot with and without noise.

STEP 8, upon arrival each band is filtered

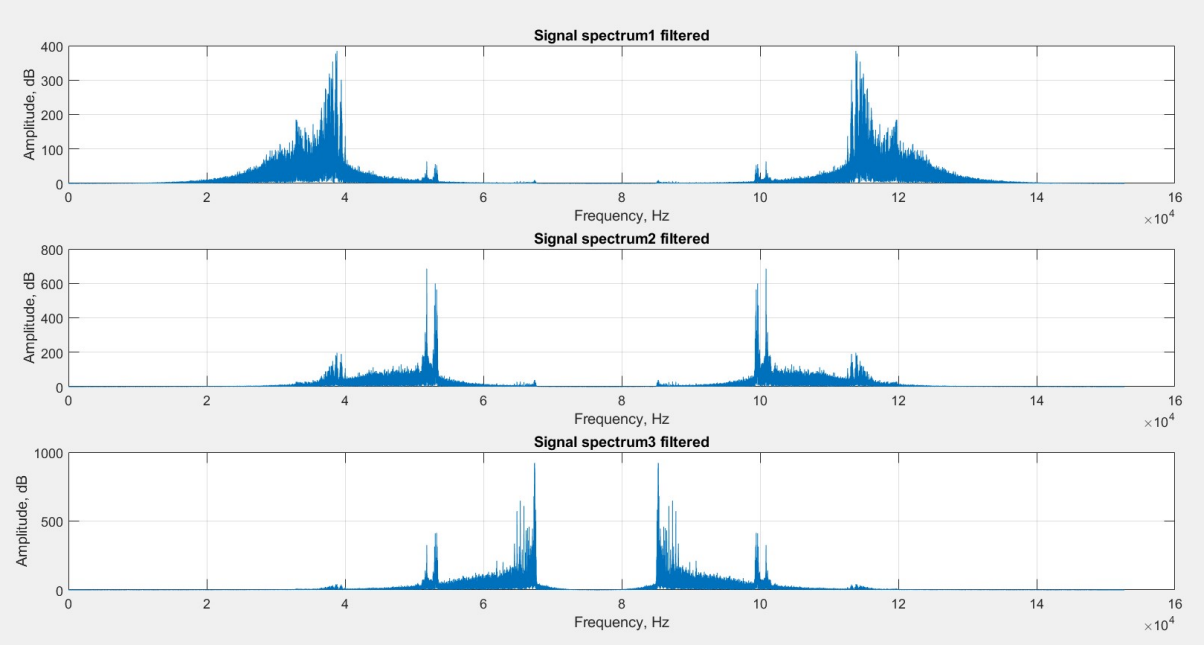


Fig 10.6: spectral plot of demultiplexed signals upon receiving.

STEP 9:, each recovered band is demodulated to return the signal to the indicated frequency

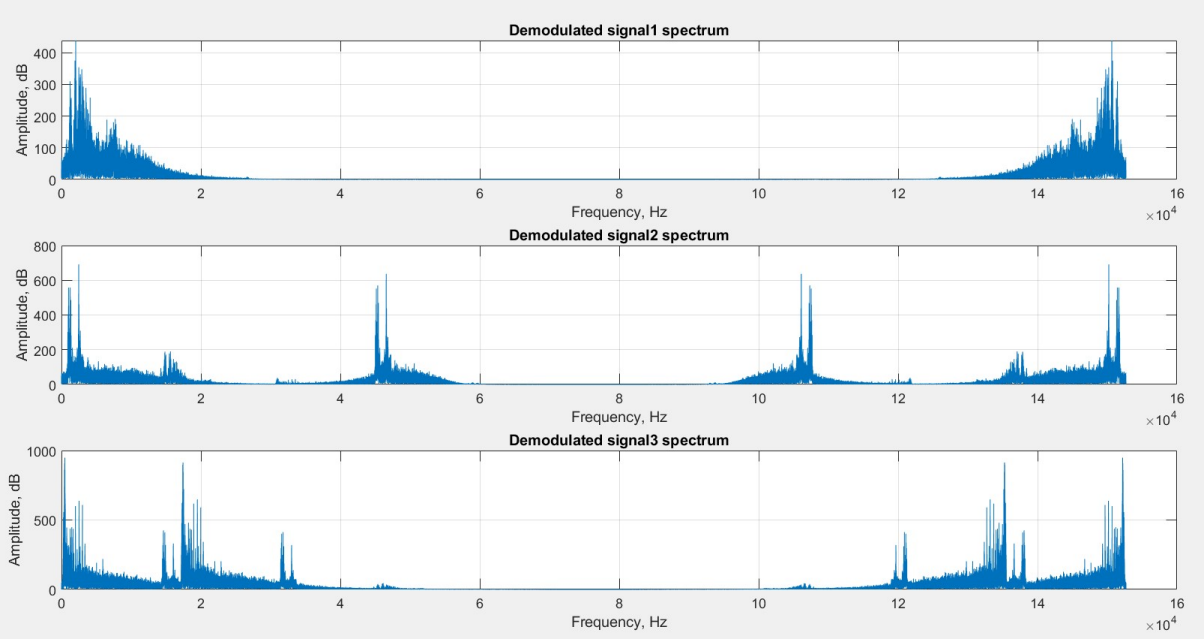


Fig 10.6: the signals are SSBSC demodulated

STEP 10, the recovered signal is passed through a low-pass filter

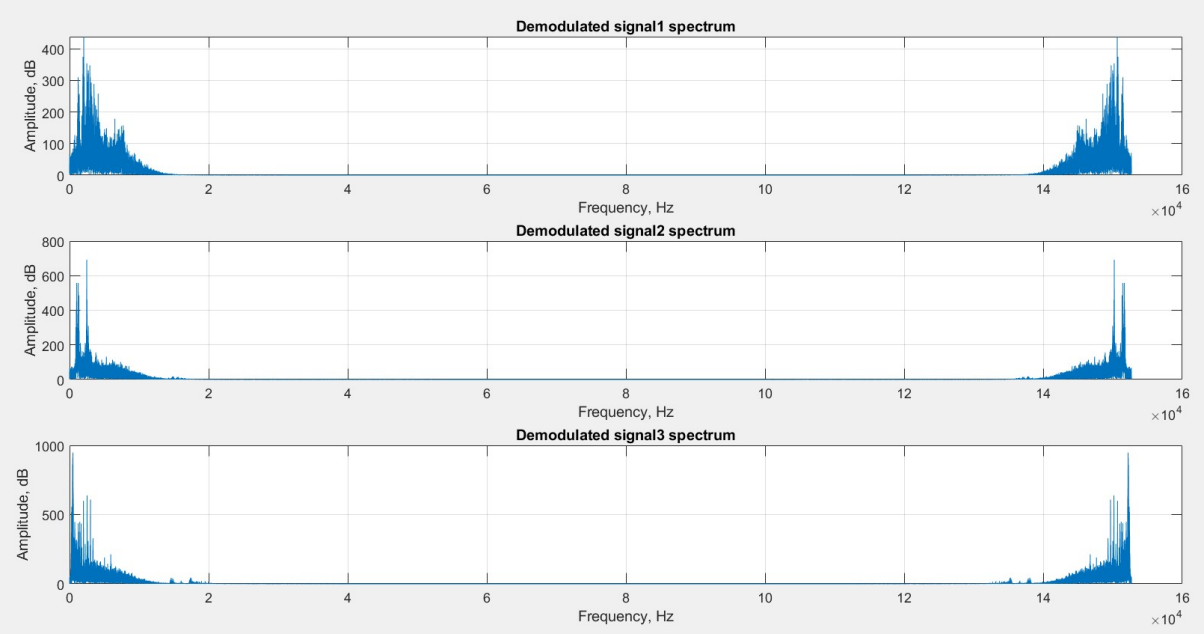


Fig 10.7: spectral plot of the recovered signals

STEP 11, Signal reproduce the signal after transmission

***The received and recovered signals are played*

Discussion or Inference of the experiment

From this open-ended project, we learned about the frequency division multiplexing and demultiplexing of signals during communication through a channel and we applied it generate 3 SSBSC modulated message signals (audio clips) with different carriers . we also were able to successfully multiplex ,demultiplex and demodulate the signal with the help of MATLAB to get back the message signals (audio clips) . Also, all the waveforms were simulated as expected in theory.

Conclusion:-

From this project we successfully performed Frequency division multiplexing using audio signal and its demultiplexing using MATLAB and simulated their waveforms.