Faculty of Enigeering

Cairo University

Communication Systems Engineering

Mixers and Modulators Research

Presented for ELC 3020

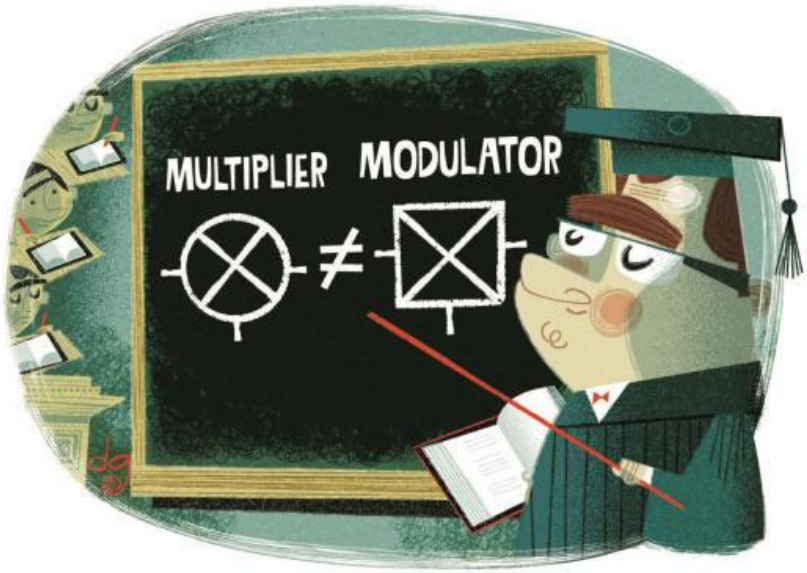
Presented to:

Dr. Ahmed Hesham

T.A: Mohamed Khalid

Email: ahesham.inquiries@gmail.com

TEAM MEMBERS (3rd Year Electronics and Electrical Communication Engineers)	
مجدي أحمد عباس عبد الحميد الابرق	Sec: 3 / I.D: 9210899 / BN: 36

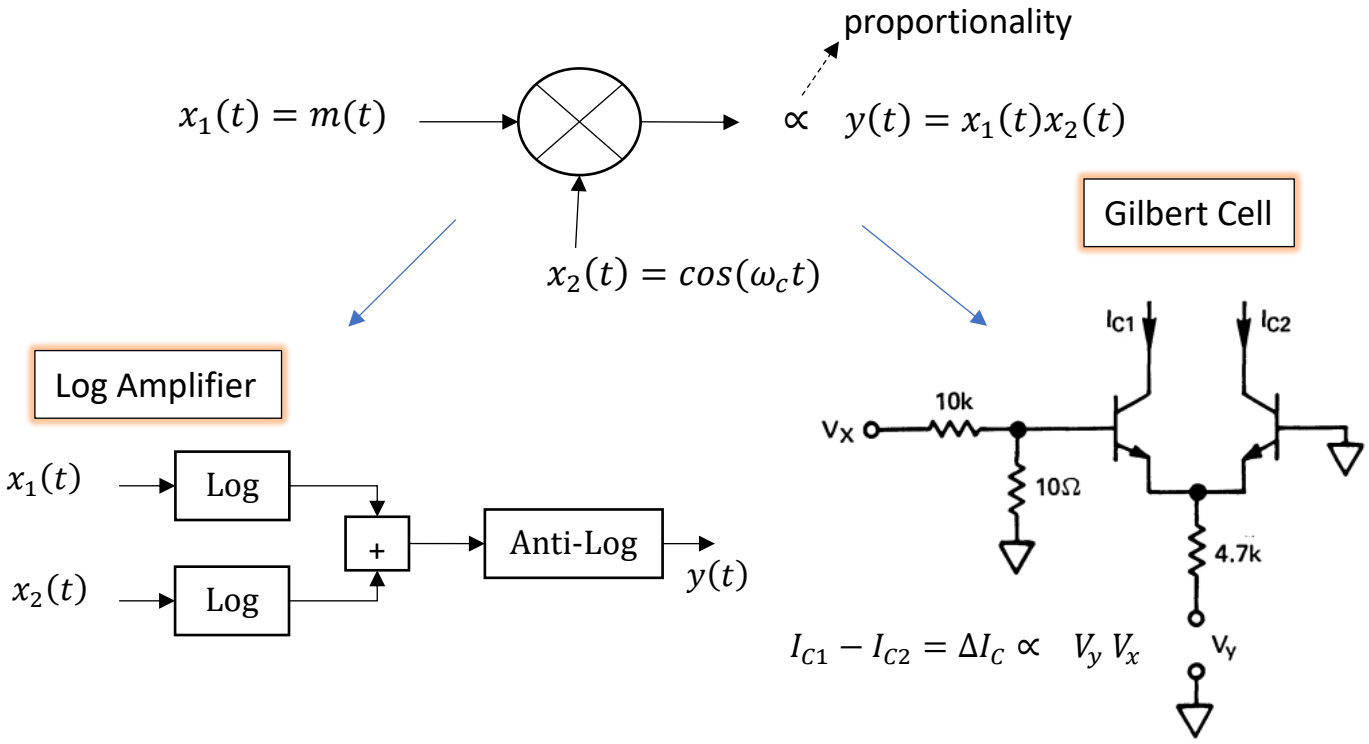


Types of AM Modulation

Product Modulator

In this type we multiply the message "modulating" signal directly by the carrier signal [in most cases we use this type when the carrier signal is sinusoidal].

Where we can generate this type of Modulators using log Amplifier or Gilbert cell.



Log Amplifier

The simplest electronic multipliers use logarithmic amplifiers.

The computation relies on the fact that the antilog of the sum of the logs of two numbers is the product of those numbers.

The disadvantages of this type of multiplication are the very limited bandwidth and single quadrant operation. A far better type of multiplier uses the "Gilbert Cell."

GILBERT CELL MULTIPLIER

There is a linear relationship between the collector current of a silicon junction transistor and its transconductance (gain) which is given by $\frac{dI_c}{dV_{BE}} = \frac{qI_c}{kT}$

Where $I_c =$ the collector current, $V_{BE} =$ the base-emitter voltage, $q =$ the electron charge $(1.6 \times 10^{-19}\ c)$, $k =$ Boltzmann's constant (1.38×10^{-23}), $T =$ the absolute temperature in kelvin $(°K = ℃ + 273)$.

This relationship may be exploited to construct a multiplier with a long-tailed pair of silicon transistors.

$$I_{C_1} - I_{C_2} = \Delta I_C = \frac{q}{kT}\left(\frac{V_y + V_{be}}{4.7 \times 10^3}\right)\left(\frac{10}{10{,}010}\right) V_x$$
$$= 8.3 \times 10^{-6}\left(V_y + 0.6\right) V_x \ @\ 25°C$$

From the above equation we notice that the Gilbert cell Multiplier is a rather poor multiplier because:

① The Y input is offset by the V_{BE} which changes nonlinearly with V_y.

② The X input is non-linear because of the exponential relationship between I_c & V_{BE}.

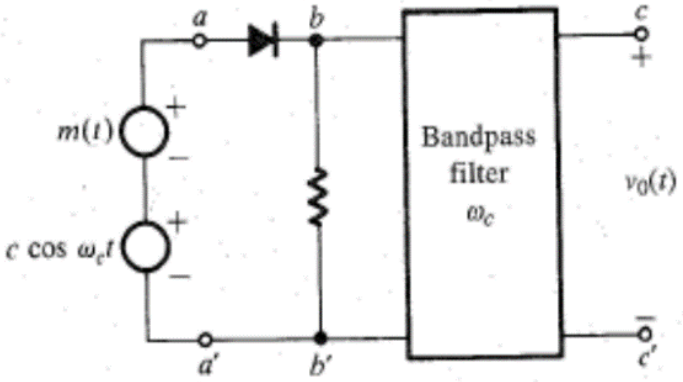
③ The scale factor varies with temperature.

Switching Modulator

In this type we multiply the message "modulating" signal by a periodic square pulse signal followed by a band pass filter to select the desired frequency only where there are three Methods to implement this Modulator.

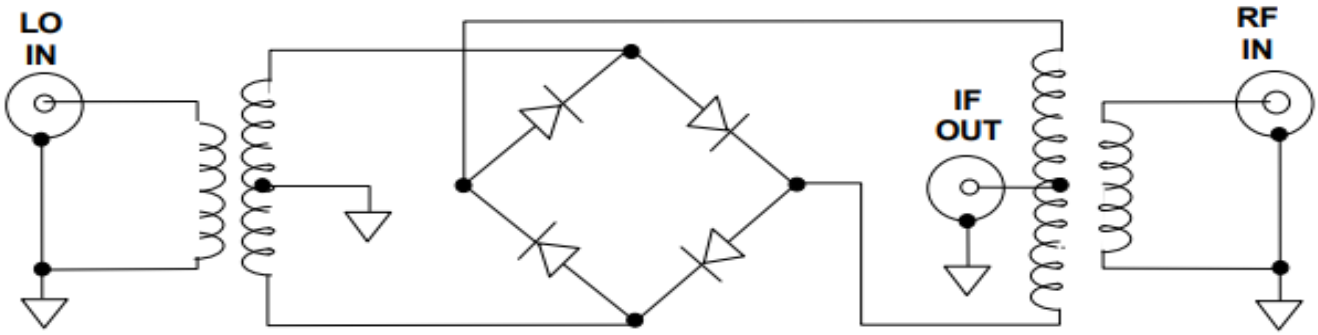
Diode

where the diode exists between the (a, b) terminals & if $A_c \gg m(t)$, the diode will act as a switch (turn on & off)



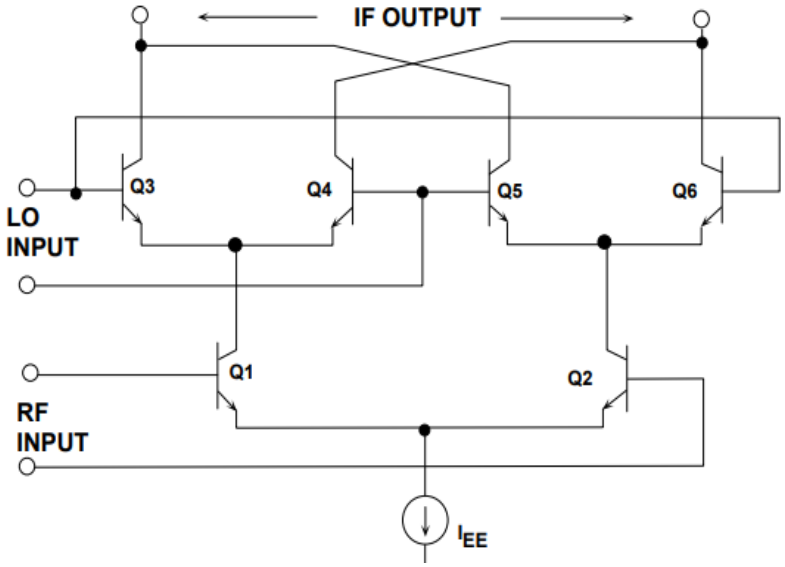
Diode Ring

For many years, the most common mixer topology for high-performance applications has been the diode-ring mixer. The diodes, which may be silicon junction, silicon Schottky-barrier or gallium-arsenide types, provide the essential switching action. but note in passing that the LO drive needs to be quite high in order to ensure that the diode conduction is strong enough to achieve low noise and to allow large signals to be converted without excessive spurious nonlinearity.



Classic Active Mixer

The diode-ring mixer not only has certain performance limitations, but it is also not amenable to fabrication using integrated circuit technologies. In the mid 1960's it was realized that the four diodes could be replaced by four transistors to perform essentially the same switching function.



The Classic active mixer is attractive for the following reasons: -

① It can be monolithically integrated with other signal processing circuitry.

② It can provide conversion gain, whereas a diode-ring mixer always has an insertion loss.

③ It requires much less power to drive the LO port.

④ It provides excellent isolation between the signal ports.

⑤ Is far less sensitive to load-matching, requiring neither diplexer nor broadband termination.

Cairo University

Faculty of Engineering

Electronics and Electrical Communications Engineering Department

Third Year

Analog Communications

Term Project

MATLAB implementation of a superheterodyne receiver

Student Name: مجدى أحمد عباس عبد الحميد

Sec: 3 / BN: 36 / ID: 9210899

Contents

Table of figures

1. The transmitter

This part contains the following tasks:

1. Reading monophonic audio signals into MATLAB.
2. Up sampling the audio signals.
3. Modulating the audio signals (each on a separate carrier).
4. Addition of the modulated signals.

Discussion

After reading the 5 Audio signals and sample them, we convert each one of them into single channel stream (Monophonic Receiver) in order for no need to two separate channels.

Then we fit the size of all audio signals to be with equal Length (Padding).

In modulation stage:

the goal for the modulation process is that we want to shift the signals at very high frequency to be compatible with the antenna size $\left(\lambda_{antenna} = \frac{c}{f}\right)$

First, we Increase the Sample rate of the signals 20 times the original sample frequency to achieve the Nyquist criteria (to avoid aliasing).

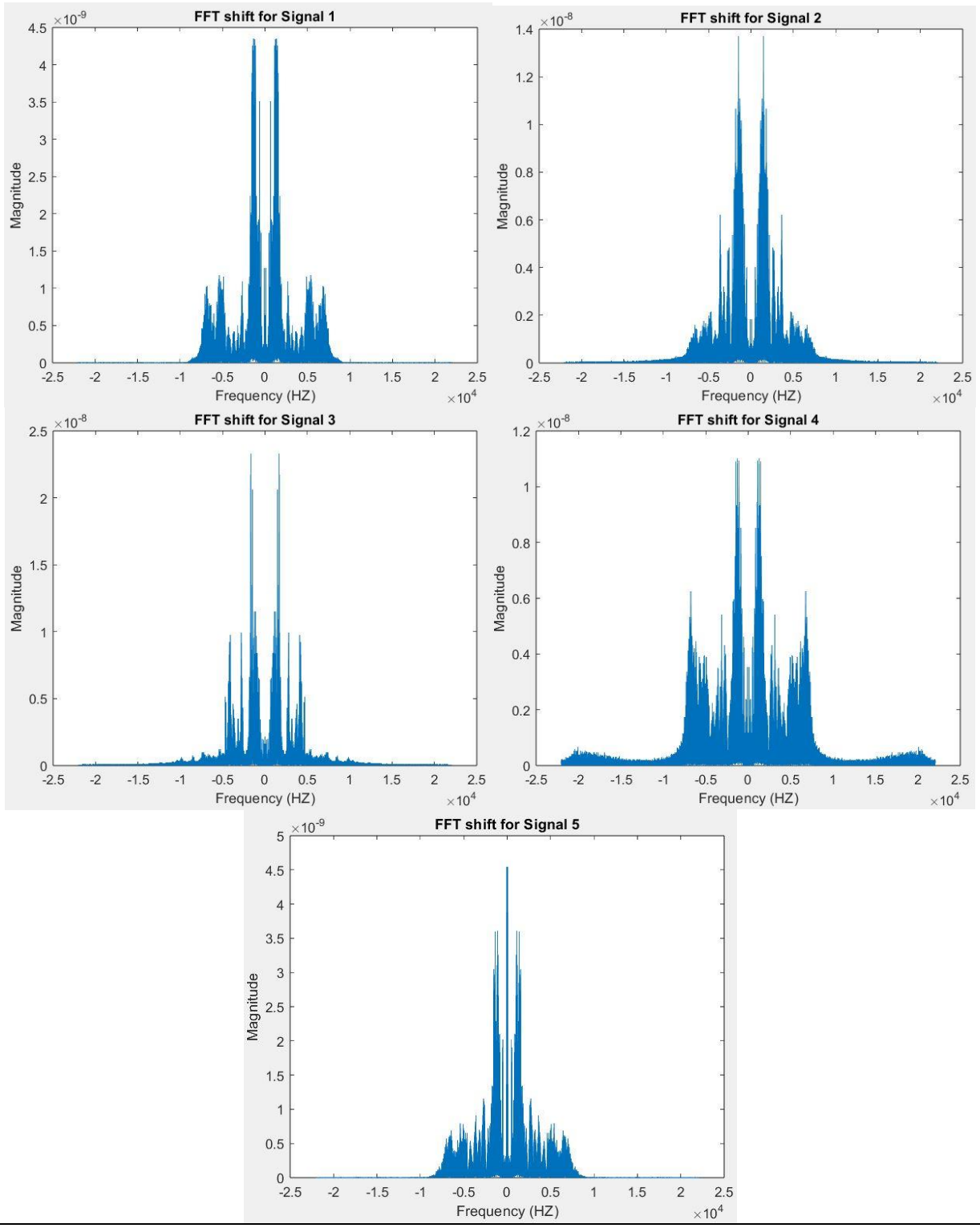
Second, we implement specific carrier for each signal with ΔF between them.

Finally, we multiple each signal with this specific carrier and add them to be sent to the T_X Antenna (Frequency Division Multiplexing).

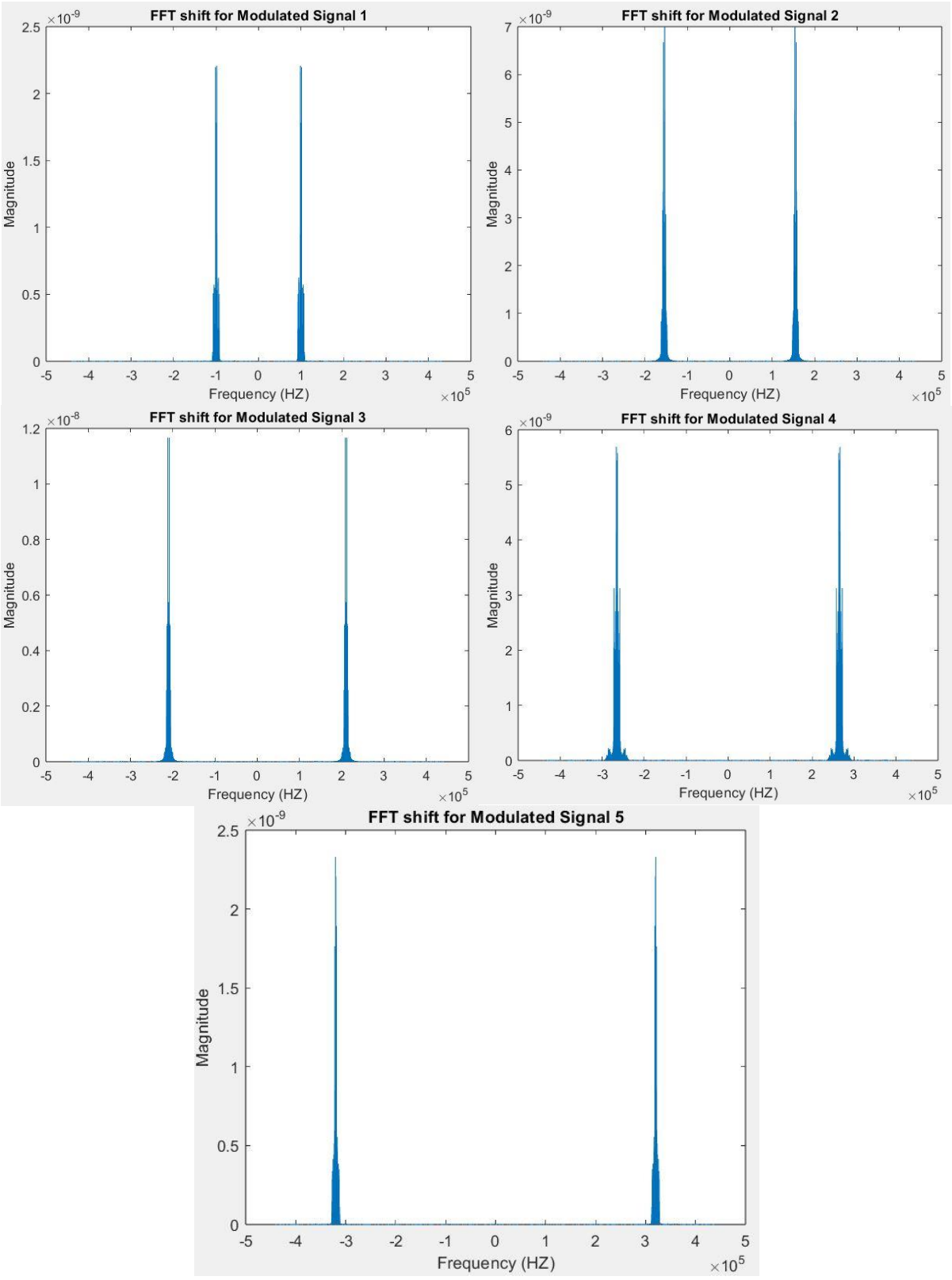
The figures

Figure 1: The spectrum of the output of the transmitter

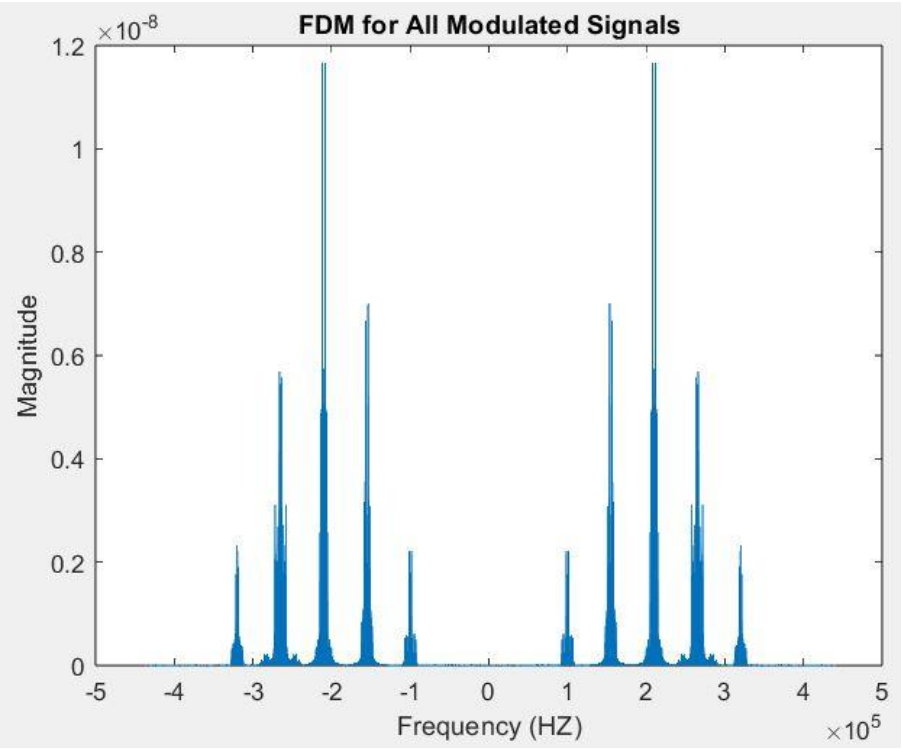
Task 1: Reading monophonic audio signals into MATLAB



Task 2: Up sampling the audio signals.
Task 3: modulating the audio signals (each on a separate carrier).



Task 4: Addition of the modulated signals (FFT Shifted FDM)



2. *The RF stage*

This part addresses the RF filter and the mixer following it.

Discussion

In this stage we use band bass filter to take one desired signal (from the user) and reject others which may cause imaging where the image signal occurs at $(\omega_c + 2 \times \omega_{IF})$.

First, we Increase the Sample rate of the signals 40 times the original sample frequency.

After that we use the Oscillator with Carrier frequency $\omega_c + \omega_{IF} + \omega_{offset}$ where $\omega_c = \omega_n + n \times \Delta F$ multiplied by the audio signal with a new sample factor In order to prevent the problems that we will face when we shift the selected signal in the baseband.

The figures

Assume we want to demodulate the first signal (at ω_o).

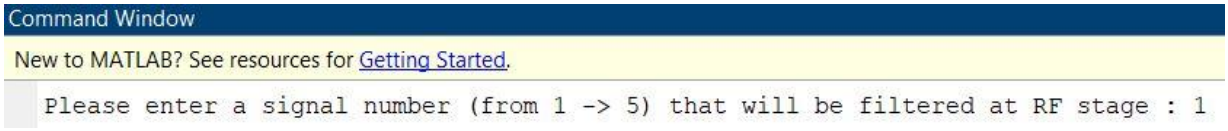


Figure 2: the output of the RF filter (before the mixer)

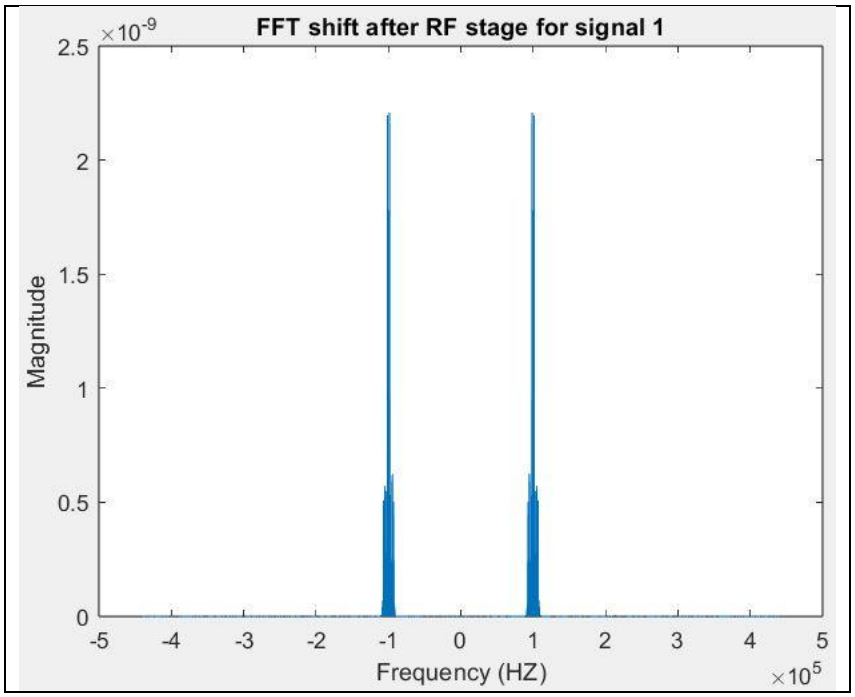
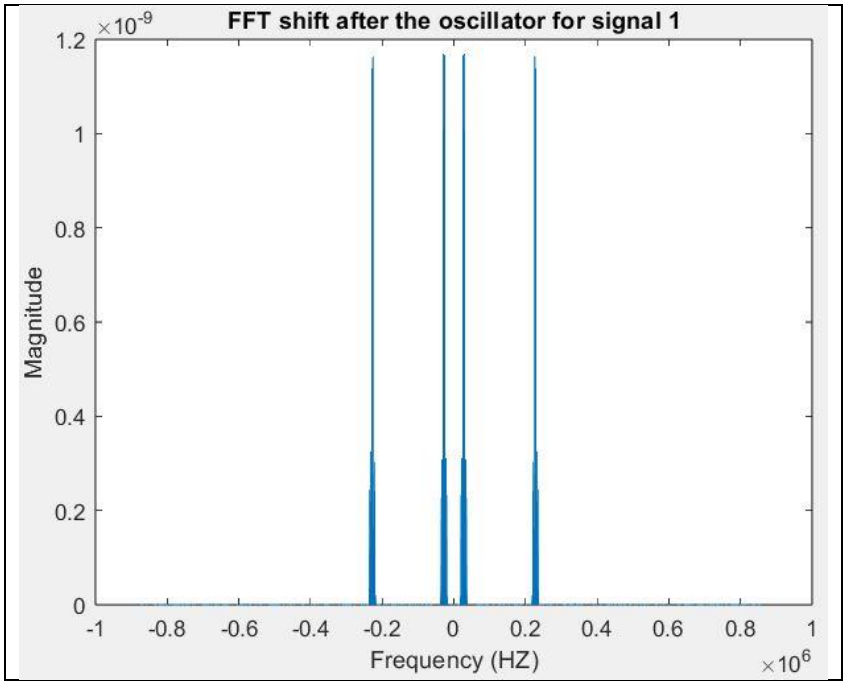


Figure 3: The output of the mixer



3. *The IF stage*

This part addresses the IF filter.

Discussion

The previous signal has carrier at high frequency and at intermediate.

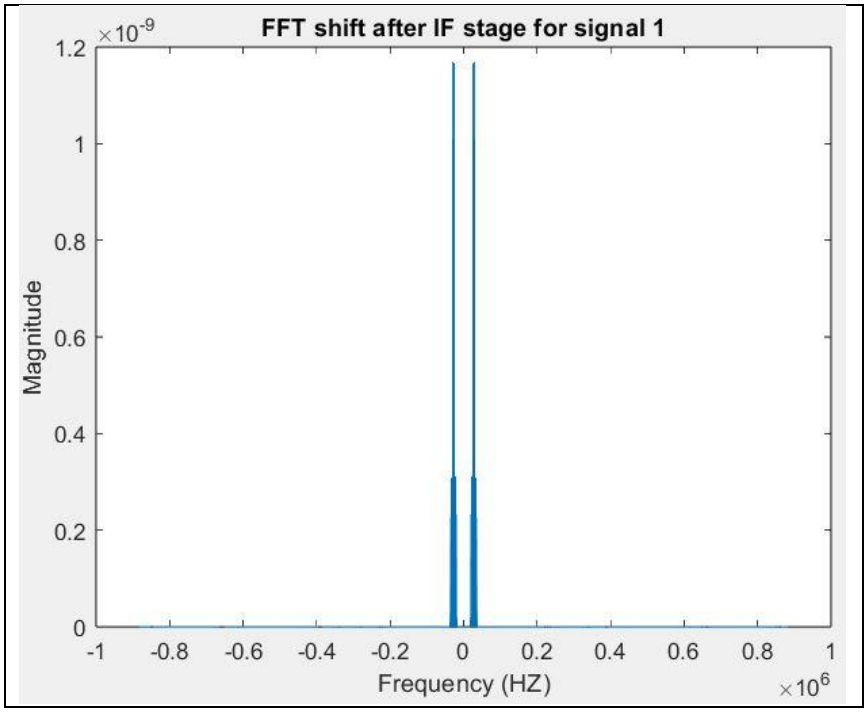
So, we need a bandpass filter centered at "ω_{IF}" and reject high frequency part.

We use this stage to prevent some problems when demodulate the message directly to the baseband where the baseband has some disadvantages like:

1) Local Oscillator (LO) Leakage (Causing DC offset and Self - Mixing)
2) Flicker Noise exist @ Baseband.
3) RF Circuits Linearity
4) Filter Selectivity (Frequency increase → Filter Selectivity Decrease)

The figures

Figure 4: Output of the IF filter



4. *The baseband demodulator*

This part addresses the coherent detector used to demodulate the signal from the IF stage.

Discussion

In this stage we return the signal back into baseband by multiplying it by a carrier with "IF" frequency. Then use a low-pass filter to reject high frequencies.

Then we successfully listen to the audio signal after multiplying it by a gain.
Where the gain is equal to the Multiplication inverse of the three mixers (Oscillator + Amplitude modulator + Demodulator "@ Baseband Stage") on the path of the original signal.

The figures

Figure 5: Output of the mixer (before the LPF)

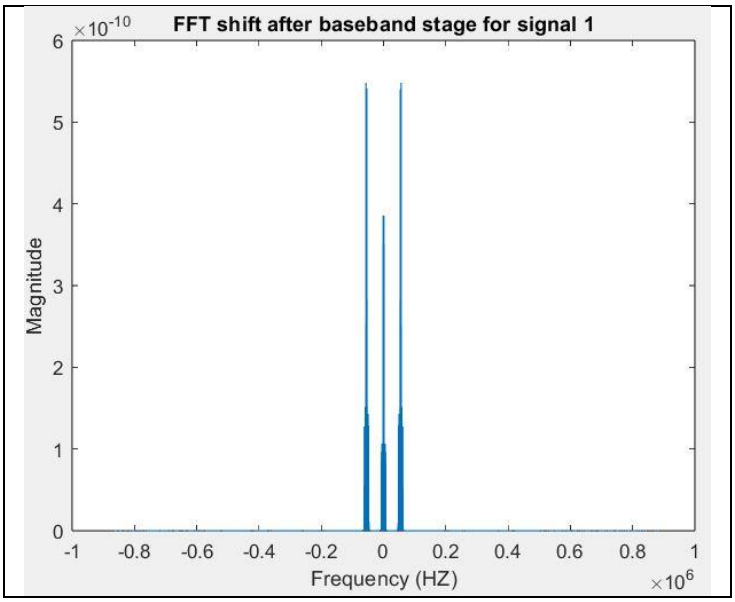
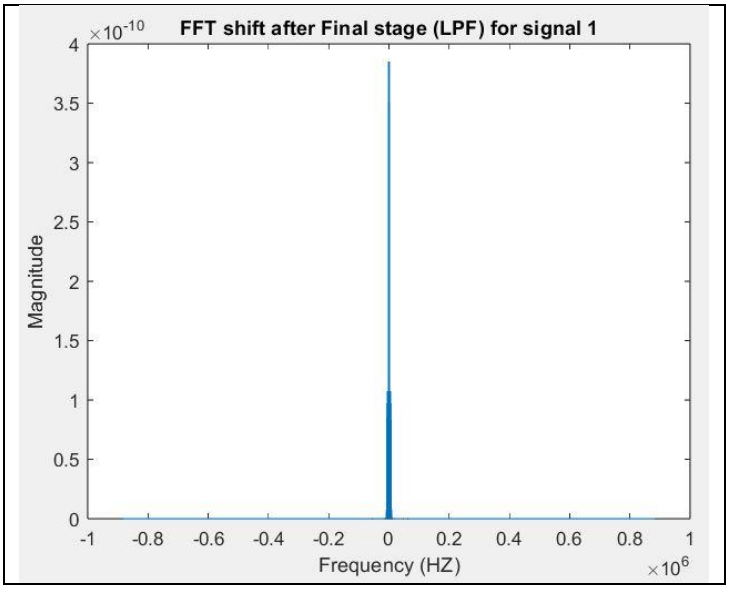


Figure 6: Output of the LPF



5. *Performance evaluation without the RF stage*

The figures

Figure 7: output of the RF mixer (no RF filter)

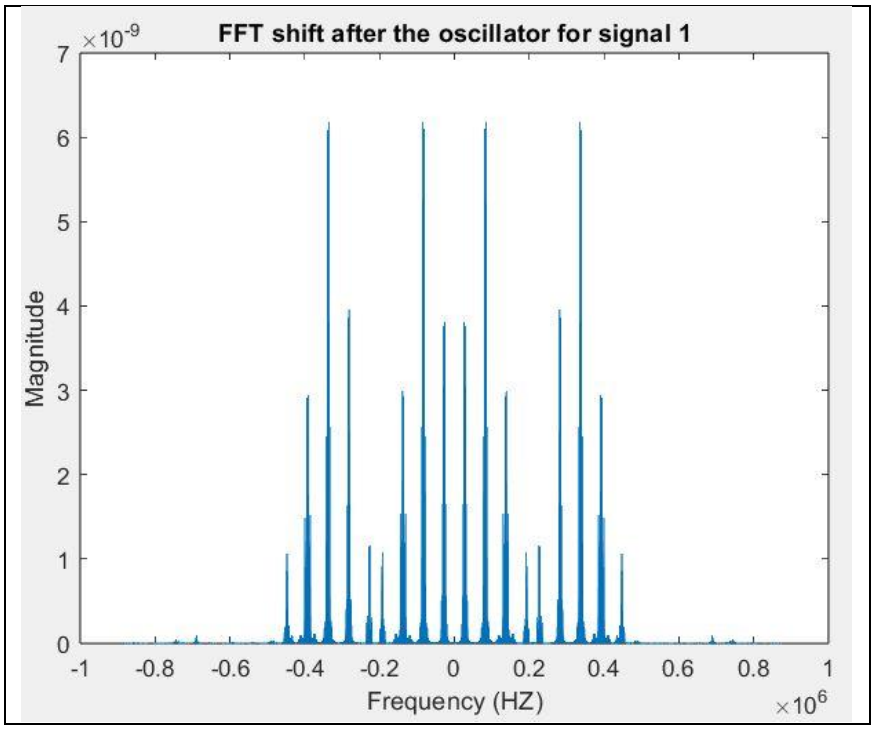


Figure 8: Output of the IF filter (no RF filter)

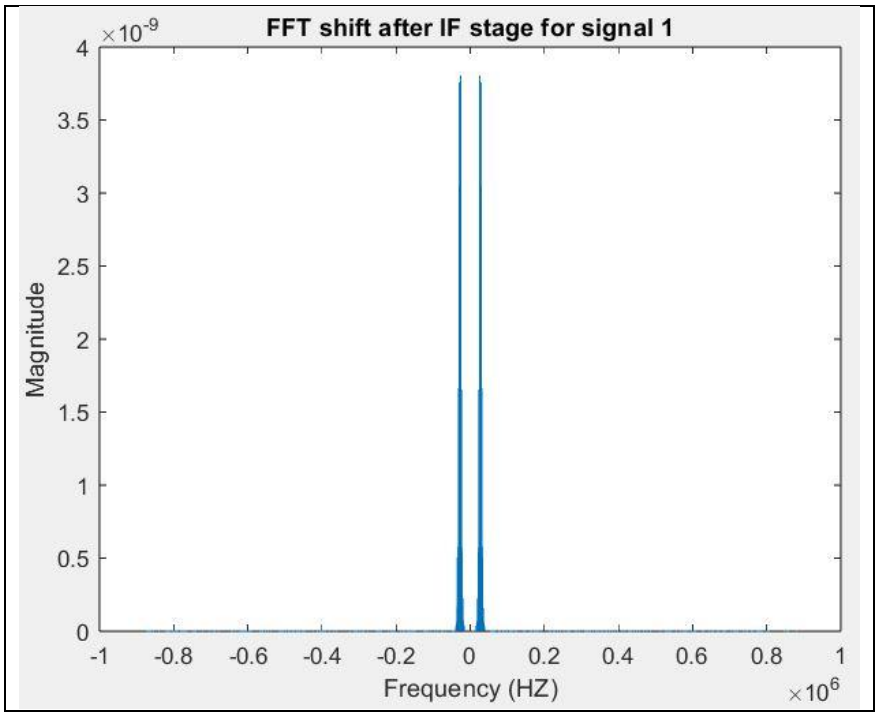


Figure 9: Output of the IF mixer before the LPF (no RF filter)

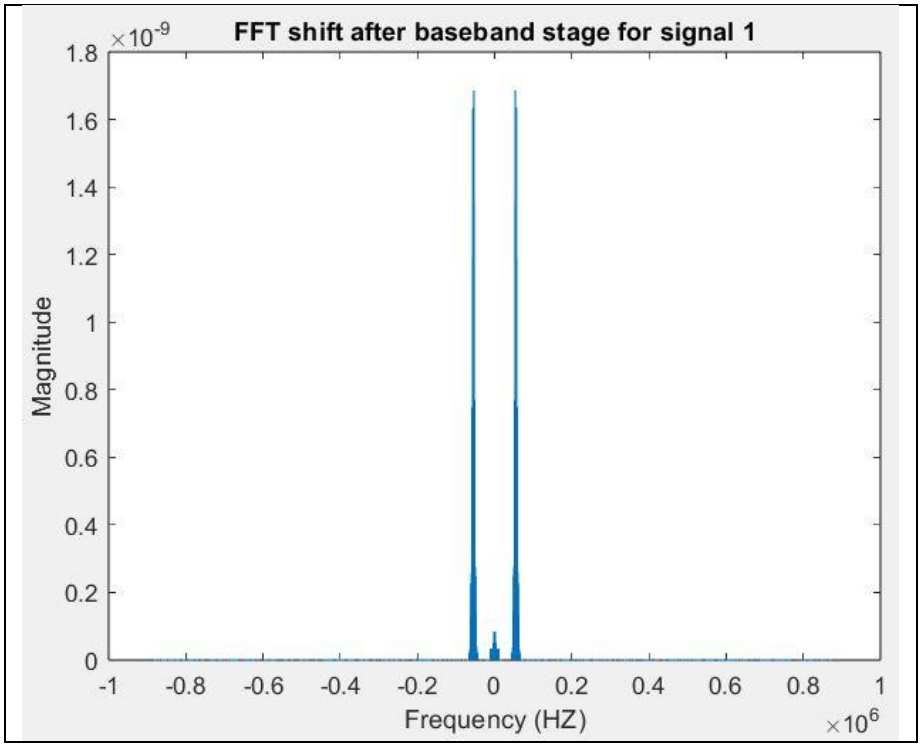
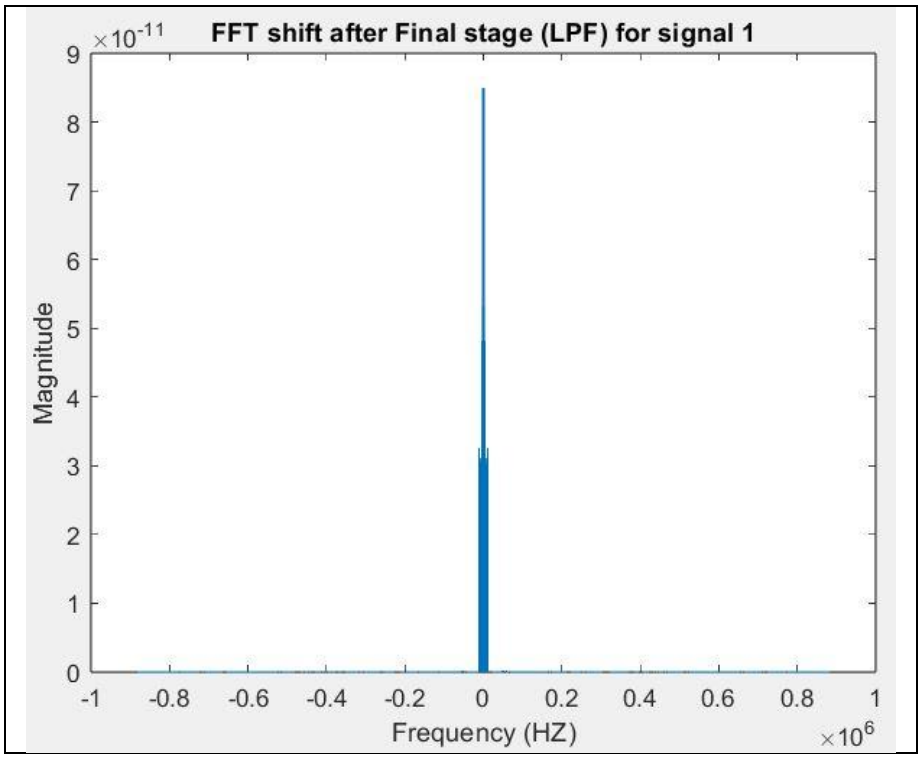


Figure 10: Output of the LPF (no RF filter)



6. *Comment on the output sound*

If the RF filter does not exist, the image problem occurs when remove the RF stage (RF bandpass filter) required to remove the image signal that the received audio interferences with other audio signal that it's carrier frequency $= \omega_c + 2\,\omega_{IF}$

While before remove the RF stage we use band pass filter to take **only** the desired signal and reject the others and its image.

What happens (in terms of spectrum and the sound quality) if the receiver oscillator has frequency offset by 0.1 KHz and 1 KHz.

The existence of frequency offset in oscillator led to distortion in the signal and bad quality of the sound.

The more offset, the more distortion increases, and the audio quality gets worse.

7. The code

```
clc
clear
close all;

%% Audio Signals (5 Messages)

% Reading Audio Signals
[message1_BBCArabic2,Fs]    = audioread('Short_BBCArabic2.wav');
                                          % 17 Seconds , Length = 740544
[message2_FM9090,~]         = audioread('Short_FM9090.wav');
                                          % 16 Seconds , Length = 697536
[message3_QuranPalestine,~] = audioread('Short_QuranPalestine.wav');
                                          % 17 Seconds , Length = 739200
[message4_RussianVoice,~]   = audioread('Short_RussianVoice.wav');
                                          % 16 Seconds , Length = 703360
[message5_SkyNewsArabia,~]  = audioread('Short_SkyNewsArabia.wav');
                                          % 17 Seconds , Length = 711872

% Max. Length for All Signals = 740544
Length = length(message1_BBCArabic2);

% Monophonic Receiver Implementation (Single Channel for each Signal)
mono_message1_BBCArabic2     = message1_BBCArabic2(:,1)     +
                                          message1_BBCArabic2(:,2);
mono_message2_FM9090         = message2_FM9090(:,1)         +
                                          message2_FM9090(:,2);
mono_message3_QuranPalestine = message3_QuranPalestine(:,1) +
                                          message3_QuranPalestine(:,2);
mono_message4_RussianVoice   = message4_RussianVoice(:,1)   +
                                          message4_RussianVoice(:,2);
mono_message5_SkyNewsArabia  = message5_SkyNewsArabia(:,1)  +
                                          message5_SkyNewsArabia(:,2);

% Signals Padding with Zeros so they have all Equal Length
audios_signals               = zeros(Length,5);

message1_BBCArabic2_PAD     = [mono_message1_BBCArabic2;
           zeros(Length-length(mono_message1_BBCArabic2),1)];
message2_FM9090_PAD         = [mono_message2_FM9090;
             zeros(Length-length(mono_message2_FM9090),1)];
message3_QuranPalestine_PAD = [mono_message3_QuranPalestine;
         zeros(Length-length(mono_message3_QuranPalestine),1)];
message4_RussianVoice_PAD   = [mono_message4_RussianVoice;
          zeros(Length-length(mono_message4_RussianVoice),1)];
message5_SkyNewsArabia_PAD  = [mono_message5_SkyNewsArabia;
          zeros(Length-length(mono_message5_SkyNewsArabia),1)];

% Filling The Audios Signal Array with the Padded Messages
audio_signals(:,1) = message1_BBCArabic2_PAD;
audio_signals(:,2) = message2_FM9090_PAD;
audio_signals(:,3) = message3_QuranPalestine_PAD;
audio_signals(:,4) = message4_RussianVoice_PAD;
audio_signals(:,5) = message5_SkyNewsArabia_PAD;
```

```
%% Plot The Audio Signals In Frequency Domain

Freq_range = (-Length/2:Length/2-1)*Fs/Length;

audio_signals_fft = zeros(Length,5);

for n = 1 : 5
    audio_signals_fft(:,n) = abs(fft(audio_signals(:,n))/Length);
end

% Plotting The Shifted Version for FFT of All Audio Signals
for n = 1 : 5
    figure
    plot(Freq_range,fftshift(audio_signals_fft(:,n))/Length);
    title("FFT shift for Signal " + n);
    xlabel('Frequency (HZ)');
    ylabel('Magnitude');
end

%% AM Modulator Stage

% Increase Number of Samples to avoid Nyquist Criteria (Aliasing)
audio_signals_interp = zeros(Length*20,5);
for n = 1 : 5
    audio_signals_interp(:,n) = interp(audio_signals(:,n),20);
% Where N = 20 Represent The New Sample Factor
end

% Audio Signals Carriers
Fc = 100000;
Delta_F = 55000;
Ts = 1/Fs;
Ts_New = (1/20)*Ts;
T = 0:Ts_New:(20*Length-1)*Ts_New;

audio_signals_carriers = zeros(Length*20,5);
for n = 0 : 4
    audio_signals_carriers(:,n+1) = (cos(2*pi*(Fc+n*Delta_F)*T))';
end

% Modulated Signals
modulated_audio_signals = zeros(Length*20,5);
for n = 1 : 5
    modulated_audio_signals(:,n) =
                audio_signals_carriers(:,n).*audio_signals_interp(:,n);
end

%% Plot The Modulated Audio Signals In Frequency Domain

New_Freq_range = (-20*Length/2:20*Length/2-1)*Fs/Length;

modulated_audio_signals_fft = zeros(Length*20,5);
for n = 1 : 5
    modulated_audio_signals_fft(:,n) =
                    abs(fft(modulated_audio_signals(:,n))/(20*Length));
end
```

```
% Plotting The Shifted Version for FFT of All Audio Signals
for n = 1 : 5
    figure

plot(New_Freq_range,fftshift(modulated_audio_signals_fft(:,n))/Length);
    title("FFT shift for Modulated Signal " + n);
    xlabel('Frequency (HZ)');
    ylabel('Magnitude');
end

%% Frequency Division Multiplexing (FDM)

FDM_audio_signals = zeros(Length*20,1);

for n = 1 : 5
    FDM_audio_signals = FDM_audio_signals + modulated_audio_signals(:,n);
end

FDM_audio_signals_fft = abs(fft(FDM_audio_signals)/(20*Length));

% FDM Plotting
figure
plot(New_Freq_range,fftshift(FDM_audio_signals_fft)/Length);
title('FDM for All Modulated Signals');
xlabel('Frequency (HZ)');
ylabel('Magnitude');

%% Bandwidth Calculation From Audio Signals Figures (After Padding)

BB_BW_audio_signals = 22050;

%% The RF Stage

% infinite loop to force the user to enter a correct value from 1 --> 5
while (1)
  signal_number = input('Please enter a signal number (from 1 -> 5) that
will be filtered at RF stage : ');
    if(signal_number<1 || signal_number>5)
        disp( 'wrong input !! , please try again' );
    else
        break;
    end
end
signal_number = signal_number-1;
% signal_number will be vary from 0 --> 4 to be used directly in the fc
expression

% RF_filtered_audio_signal = zeros(Length*20,1);
fstop1  = (Fc+signal_number*Delta_F) - BB_BW_audio_signals/2 - 1000;
% Margin = 1kHz as filter is Not Ideal
fpass1  = (Fc+signal_number*Delta_F) - BB_BW_audio_signals/2;
fpass2  = (Fc+signal_number*Delta_F) + BB_BW_audio_signals/2;
fstop2  = (Fc+signal_number*Delta_F) + BB_BW_audio_signals/2 + 1000;
% Margin = 1kHz as filter is Not Ideal
BPF_OBJ1 = Bandpass_Filter_1(fstop1,fpass1,fpass2,fstop2);
% create instance from Band pass filter function
RF_filtered_audio_signal = filter(BPF_OBJ1, FDM_audio_signals);
```

```
%% Plot The Filtered Audio Signal after the RF Stage

RF_filtered_audio_signal_fft =
                            abs(fft(RF_filtered_audio_signal))/(20*Length);



% Plotting The Shifted Version for FFT of selected signal at RF stage
figure
plot(New_Freq_range,fftshift(RF_filtered_audio_signal_fft)/Length);
title("FFT shift after RF stage for signal " + (signal_number + 1));
xlabel('Frequency (HZ)');
ylabel('Magnitude');

%% The Oscillator Stage

F_IF = 27500;          % The IF Frequency
Ts_IF = Ts_New * 1/2; % Interpolation Factor = 2
T_IF = 0:Ts_IF:(40*Length-1)*Ts_IF;
F_offset = 0;

osc_audio_signal_interp = interp(RF_filtered_audio_signal,2);
% Where N = 2 Represent The New Sample Factor
osc_audio_signal = osc_audio_signal_interp .*
            ((cos(2*pi*(Fc+signal_number*Delta_F+F_IF+F_offset)*T_IF))');

%% Plot The Filtered Audio Signals after Oscillator

IF_Freq_range = (-40*Length/2:40*Length/2-1)*Fs/Length;
osc_audio_signal_fft = abs(fft(osc_audio_signal))/(40*Length);

% Plotting FFT of selected Audio Signal after the oscillator
figure
plot(IF_Freq_range,fftshift(osc_audio_signal_fft)/Length);
title("FFT shift after the oscillator for signal " + (signal_number +
1));
xlabel('Frequency (HZ)');
ylabel('Magnitude');

%% The IF Stage

fstop1   = F_IF - BB_BW_audio_signals/2 - 1000;
% Margin = 1khz as filter is Not Ideal
fpass1   = F_IF - BB_BW_audio_signals/2;
fpass2   = F_IF + BB_BW_audio_signals/2;
fstop2   = F_IF + BB_BW_audio_signals/2 + 1000;
% Margin = 1khz as filter is Not Ideal
BPF_OBJ2 = Bandpass_Filter_2(fstop1,fpass1,fpass2,fstop2);
% create instance from Band pass filter function
IF_filtered_audio_signal = filter(BPF_OBJ2, osc_audio_signal);

%% Plot The Filtered Audio Signal after the IF Stage

IF_filtered_audio_signal_fft =
                            abs(fft(IF_filtered_audio_signal))/(40*Length);
```

```
% Plotting The Shifted Version for FFT of selected signal at IF stage
figure
plot(IF_Freq_range,fftshift(IF_filtered_audio_signal_fft)/Length);
title("FFT shift after IF stage for signal " + (signal_number + 1));
xlabel('Frequency (HZ)');
ylabel('Magnitude');


%% Baseband detection stage

base_band_audio_signal = IF_filtered_audio_signal .*
                                                    ((cos(2*pi*F_IF*T_IF))');
base_band_audio_signal_fft =
abs(fft(base_band_audio_signal))/(40*Length);

% Plotting The Shifted Version for FFT of selected signal at baseband
stage
figure
plot(IF_Freq_range,fftshift(base_band_audio_signal_fft)/Length);
title("FFT shift after baseband stage for signal " + (signal_number +
1));
xlabel('Frequency (HZ)');
ylabel('Magnitude');

%% Low pass filter stage
fpass = BB_BW_audio_signals;
fstop = BB_BW_audio_signals + 1000;
% where the 1000 Hz is a margin value
LPF_OBJ = lowpass_filter(fpass,fstop);
% create instance from low pass filter function
Output_signal = filter(LPF_OBJ, base_band_audio_signal);

%% Plot The Filtered Audio Signal at the baseband Stage
(after low pass filter)

Output_signal_fft = abs(fft(Output_signal))/(40*Length);

% Plotting The Shifted Version for FFT of selected signal at IF stage
figure
plot(IF_Freq_range,fftshift(Output_signal_fft)/Length);
title("FFT shift after Final stage (LPF) for signal " + (signal_number
+1));
xlabel('Frequency (HZ)');
ylabel('Magnitude');

%% Test the output signal sound using sound function

Output_signal = 8 .* Output_signal;
% where gain = 8 as we have three mixers on the path of the original
signal each path decrease the amplitude by 1/2
Output_signal = decimate(Output_signal,40);

% the decimate function is used to downsample a signal by a factor of L
"in this case L = 40"

audiowrite('filtered_audio_signal_1.wav',Output_signal,Fs);

sound(Output_signal,Fs);
```

```
%% 1st Band Pass Filter Function (RF Stage)

function Hd = Bandpass_Filter_1(fstop1,fpass1,fpass2,fstop2)
% BANDPASS_FILTER_1 Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 9.10 and Signal Processing Toolbox 8.6.

% Chebyshev Type II Bandpass filter designed using FDESIGN.BANDPASS.

% All frequency values are in Hz.
Fs = 882000;
% Sampling Frequency = 20 * Fs = 20 * 44100 = 882000 Hz

Fstop1 = fstop1;      % First Stopband Frequency
Fpass1 = fpass1;      % First Passband Frequency
Fpass2 = fpass2;      % Second Passband Frequency
Fstop2 = fstop2;      % Second Stopband Frequency
Astop1 = 100;         % First Stopband Attenuation (dB)
Apass  = 1;           % Passband Ripple (dB)
Astop2 = 100;         % Second Stopband Attenuation (dB)
match  = 'passband';  % Band to match exactly

% Construct an FDESIGN object and call its CHEBY2 method.
h  = fdesign.bandpass(Fstop1, Fpass1, Fpass2, Fstop2, Astop1, Apass, ...
                      Astop2, Fs);
Hd = design(h, 'cheby2', 'MatchExactly', match);
end

%% 2nd Band Pass Filter Function (IF Stage)

function Hd = Bandpass_Filter_2(fstop1,fpass1,fpass2,fstop2)
% BANDPASS_FILTER_2 Returns a discrete-time filter object.
% MATLAB Code
% Generated by MATLAB(R) 9.10 and Signal Processing Toolbox 8.6.

% Chebyshev Type II Bandpass filter designed using FDESIGN.BANDPASS.

% All frequency values are in Hz.
Fs = 1764000;
% Sampling Frequency = 2 * 20 * Fs = 2 * 20 * 44100 = 1764000 Hz

Fstop1 = fstop1;      % First Stopband Frequency
Fpass1 = fpass1;      % First Passband Frequency
Fpass2 = fpass2;      % Second Passband Frequency
Fstop2 = fstop2;      % Second Stopband Frequency
Astop1 = 100;         % First Stopband Attenuation (dB)
Apass  = 1;           % Passband Ripple (dB)
Astop2 = 100;         % Second Stopband Attenuation (dB)
match  = 'passband';  % Band to match exactly

% Construct an FDESIGN object and call its CHEBY2 method.
h  = fdesign.bandpass(Fstop1, Fpass1, Fpass2, Fstop2, Astop1, Apass, ...
                      Astop2, Fs);
Hd = design(h, 'cheby2', 'MatchExactly', match);
end
```

```
%% Low pass filter function (baseband stage)

function Hd = lowpass_filter(fpass,fstop)
%LOWPASS_FILTER Returns a discrete-time filter object.

% MATLAB Code
% Generated by MATLAB(R) 9.10 and Signal Processing Toolbox 8.6.

% Chebyshev Type II Lowpass filter designed using FDESIGN.LOWPASS.

% All frequency values are in Hz.
Fs = 1764000;         % Sampling Frequency = 40 * 44100
[where the resample factor =40]

Fpass = fpass;        % Passband Frequency
Fstop = fstop;        % Stopband Frequency
Apass = 1;            % Passband Ripple (dB)
Astop = 100;          % Stopband Attenuation (dB)
match = 'passband';   % Band to match exactly

% Construct an FDESIGN object and call its CHEBY2 method.
h  = fdesign.lowpass(Fpass, Fstop, Apass, Astop, Fs);
Hd = design(h, 'cheby2', 'MatchExactly', match);
end
```