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: Youssef Khaled Abdelraouf

Sec: 4   **BN:** 37

Table of contents

Table of figures

1. ***The transmitter***

This part contains the following tasks

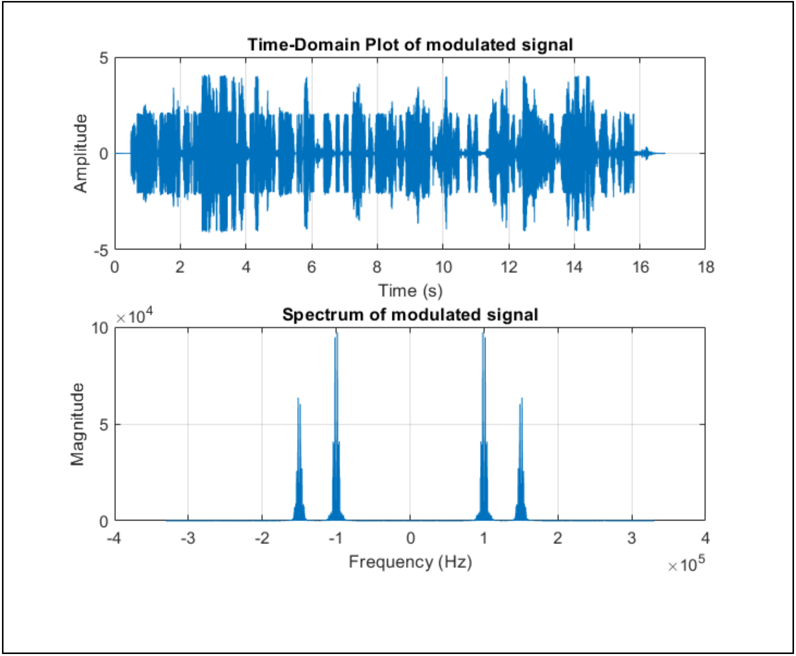
1. Reading monophonic audio signals into MATLAB.
2. Upsampling the audio signals.
3. Modulating the audio signals (each on a separate carrier).
4. Addition of the modulated signals.

Discussion

In the transmitter, we convert stereo signals to monophonic since the receiver processes single-channel signals. We then upsample to unify the sampling rate and meet Nyquist criterion. Each signal is modulated onto a unique carrier for frequency division multiplexing, and the modulated signals are added to create the transmitted signal.

Figures

Figure 1: The spectrum of the output of the transmitter



2. ***The RF stage***

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

Discussion

The main role of RF BPF is image rejection to avoid interference of image signal of desired channel after IF stage, we make it tunable by varying its center frequency to be at desired channel and a pass band small enough to reject image signal, so it was equal to bandwidth of desired signal. The mixer in this stage is necessary to choose certain signal by first moving it to intermediate frequency.

Figures

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

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

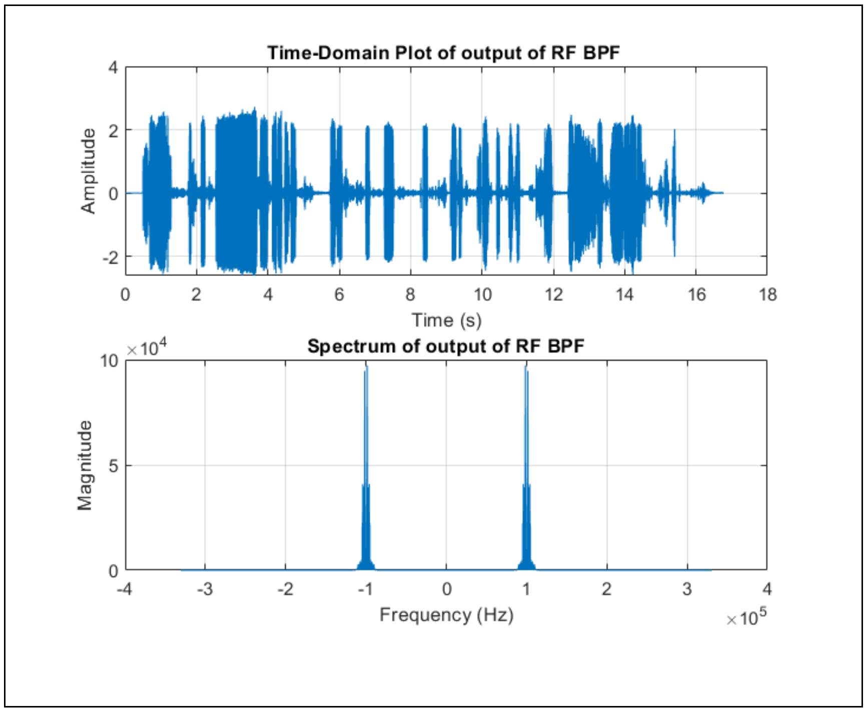
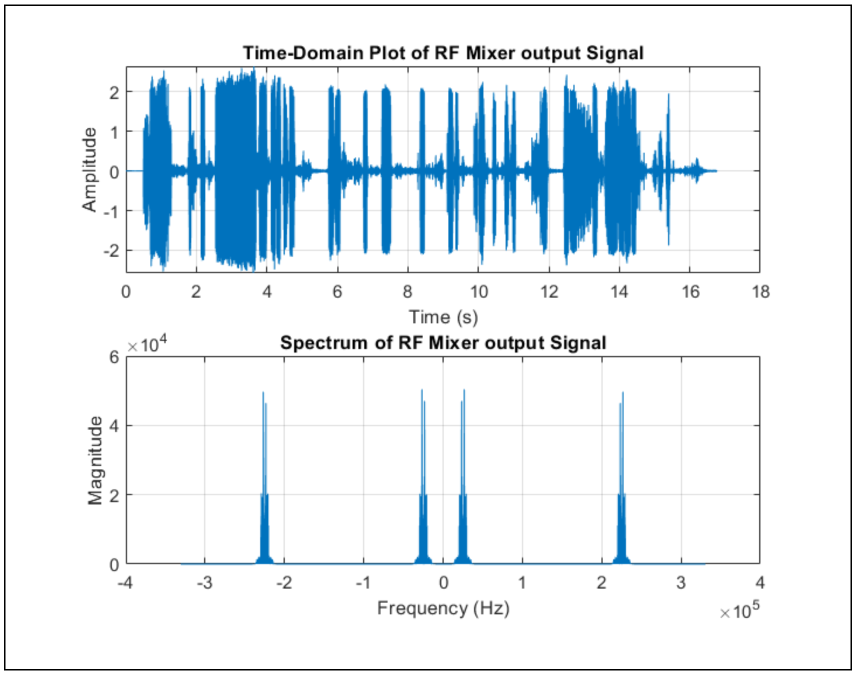


Figure 3: The output of the mixer



3. ***The IF stage***

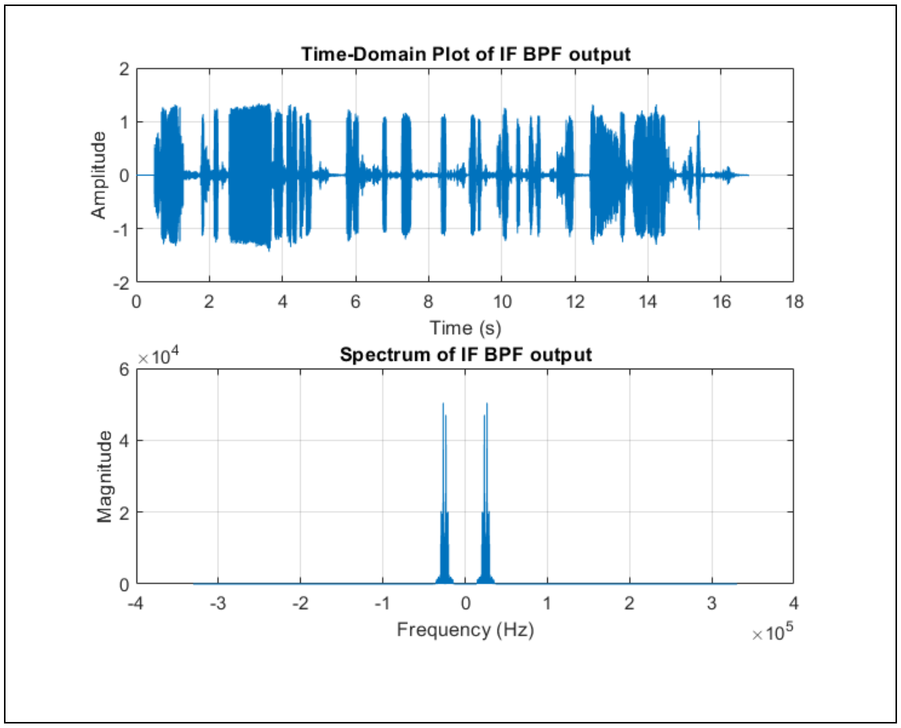
This part addresses the IF filter.

Discussion

The role of IF filter is to select certain signal from the channel we have now (after image rejection). This IF filter's sharpness can be higher than RF filter as it's designed at lower frequency. The filter was designed to have center frequency at the intermediate frequency and a pass band equal to bandwidth of desired signal.

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

The mixer at this stage multiplies the signal with a carrier of frequency equal to intermediate frequency (F_IF) to return the desired signal at baseband. It's then filtered by a low pass filter to get only the component at zero Hz and reject ones at (2*F_IF, -2*F_IF).

Figures

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

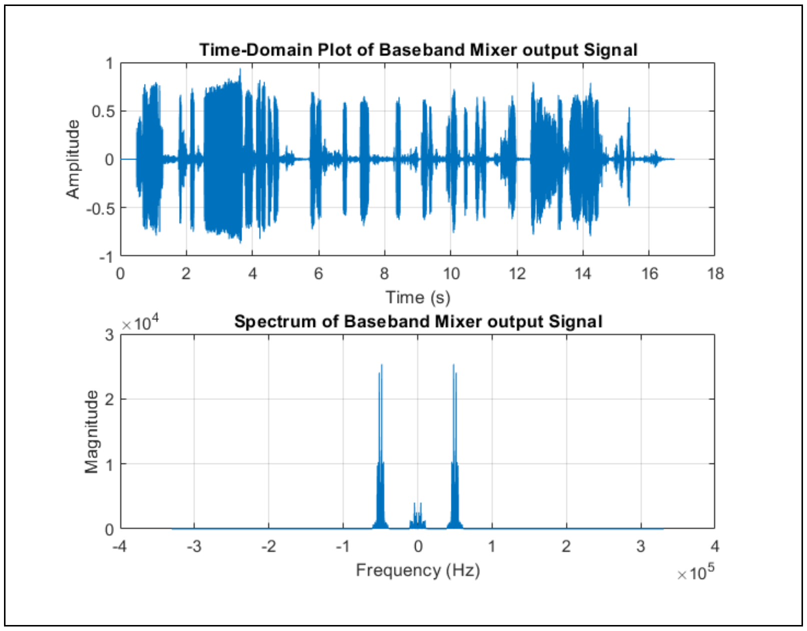
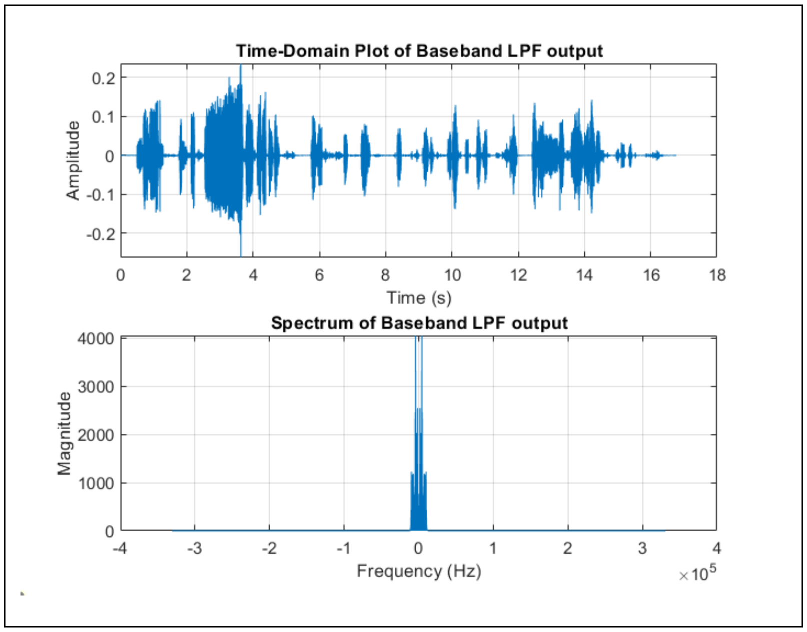


Figure 6: Output of the LPF



5. Performance evaluation without the RF stage

Figures

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

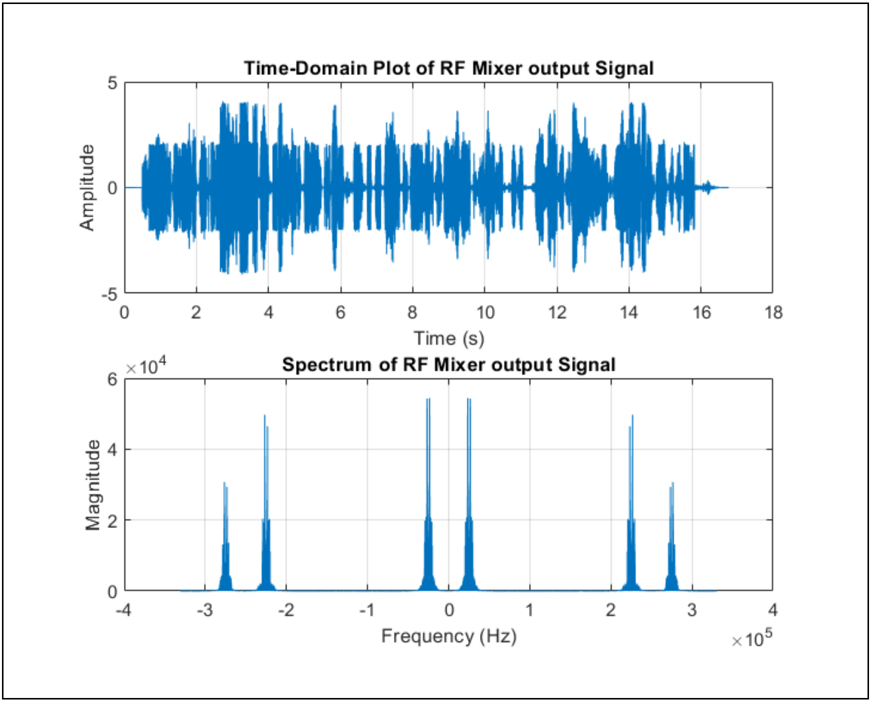


Figure 8: Output of the IF filter (no RF filter)

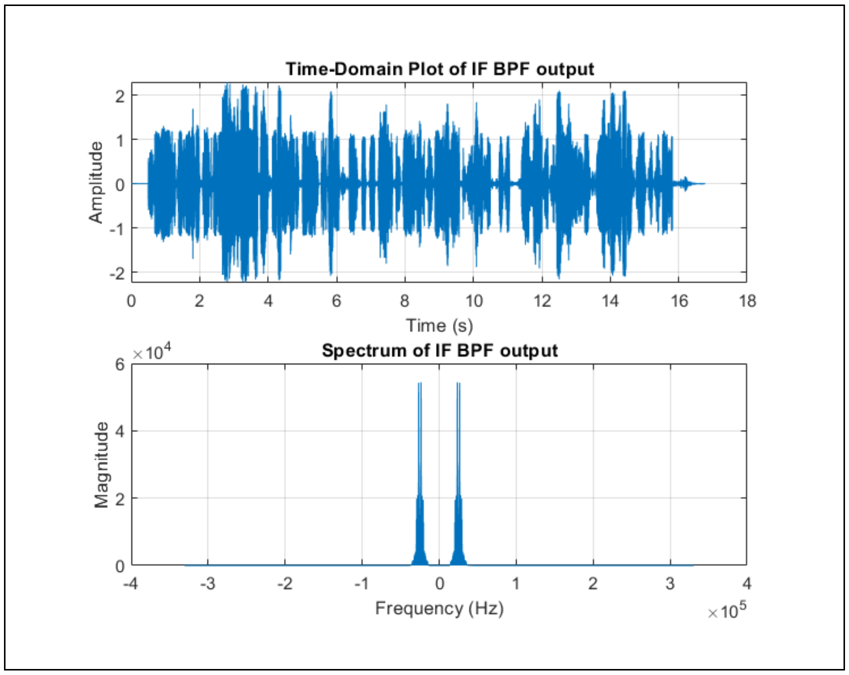


Figure 9: Output of the mixer before the LPF (no RF filter)

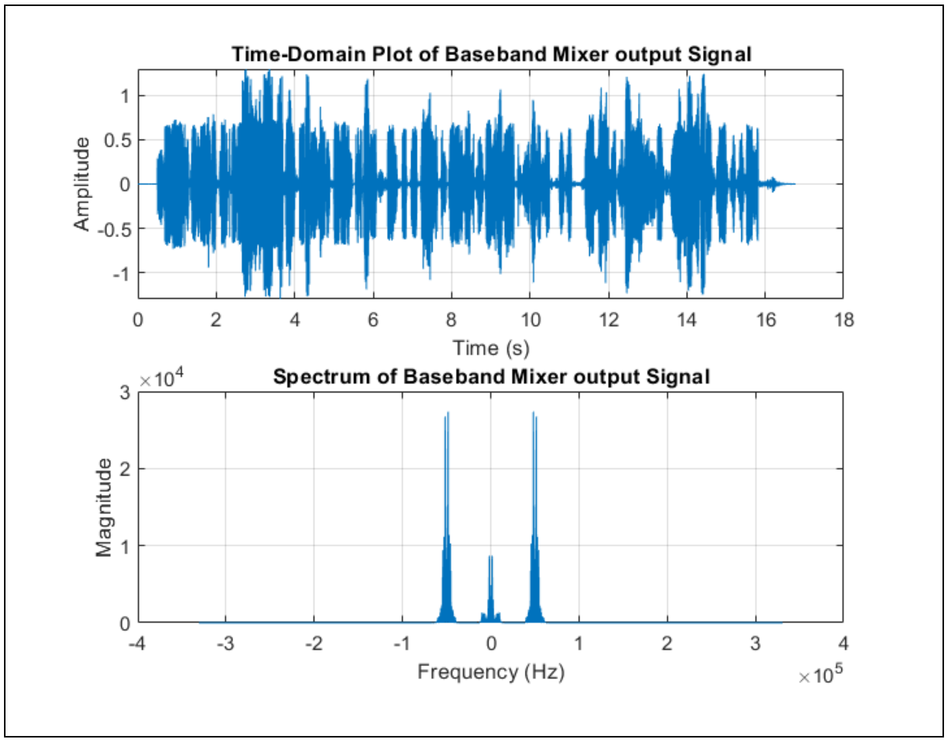
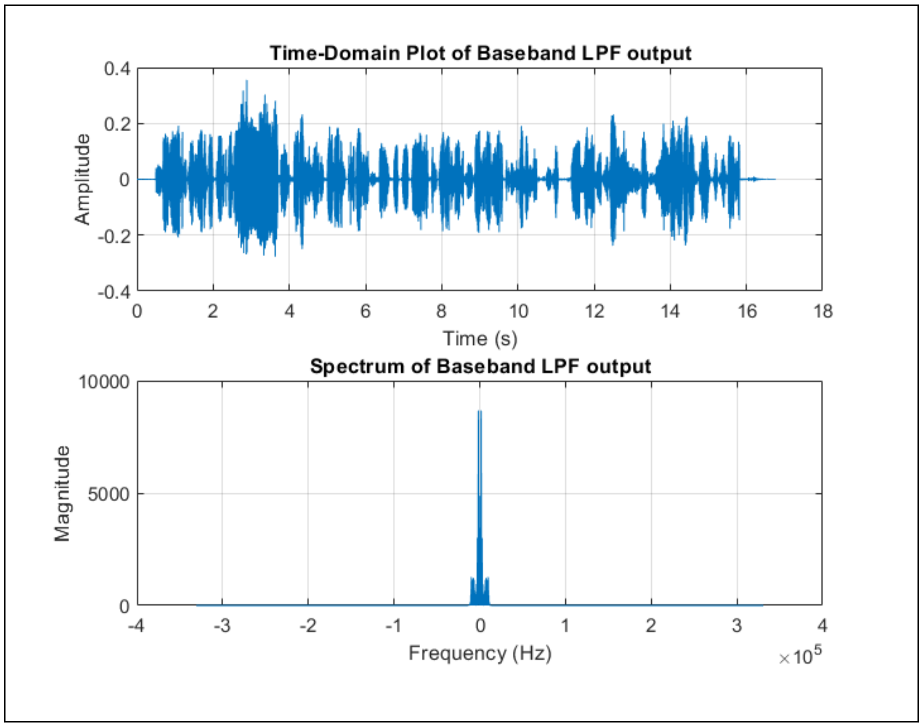


Figure 10: Output of the LPF (no RF filter)



6. ***Comment on the output sound***

In existence of RF BPF original sound/signal was clearly heard and restored/demodulated successfully as expected but after removing RF BPF the two audio signals were superimposed on each other due to interference of image signal which was at distance 2*F_IF = 50kHz from desired signal so both were added on each other at F_IF after the RF mixer and hence in baseband both were superimposed on each other.

7. ***Performance evaluation in presence of Offset at receiver***

Figures

A. Offset = 0.2kHz

Figure 11: Output of RF mixer (with 0.2kHz offset)

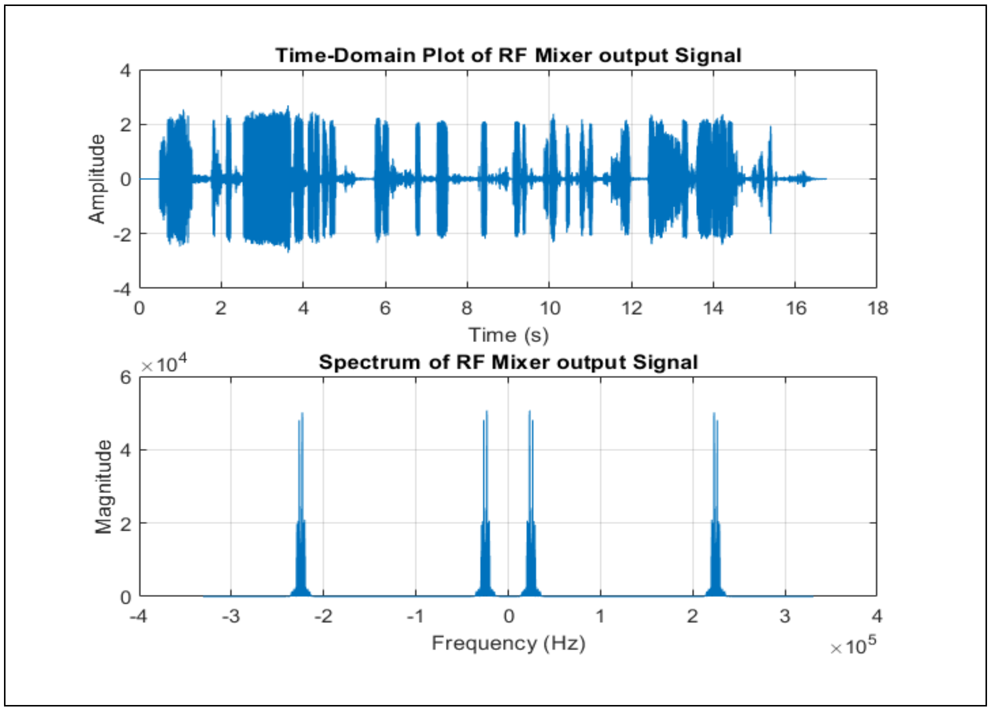


Figure 12 : Output of IF filter (with 0.2kHz offset)

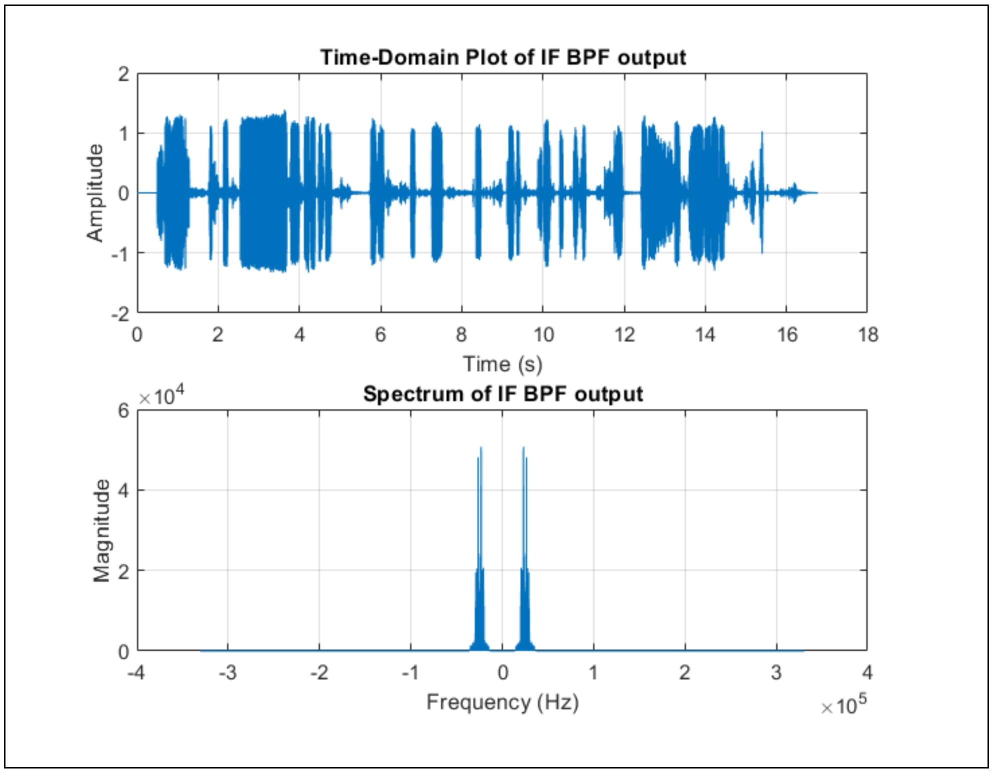


Figure 13: Output of the mixer before the LPF (with 0.2kHz offset)

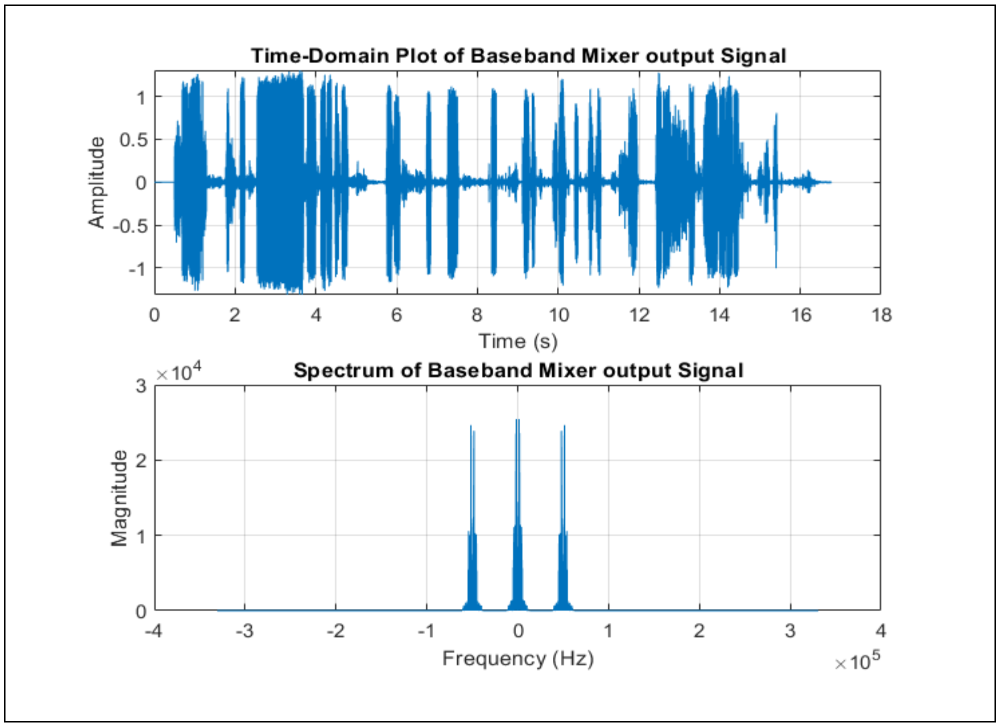
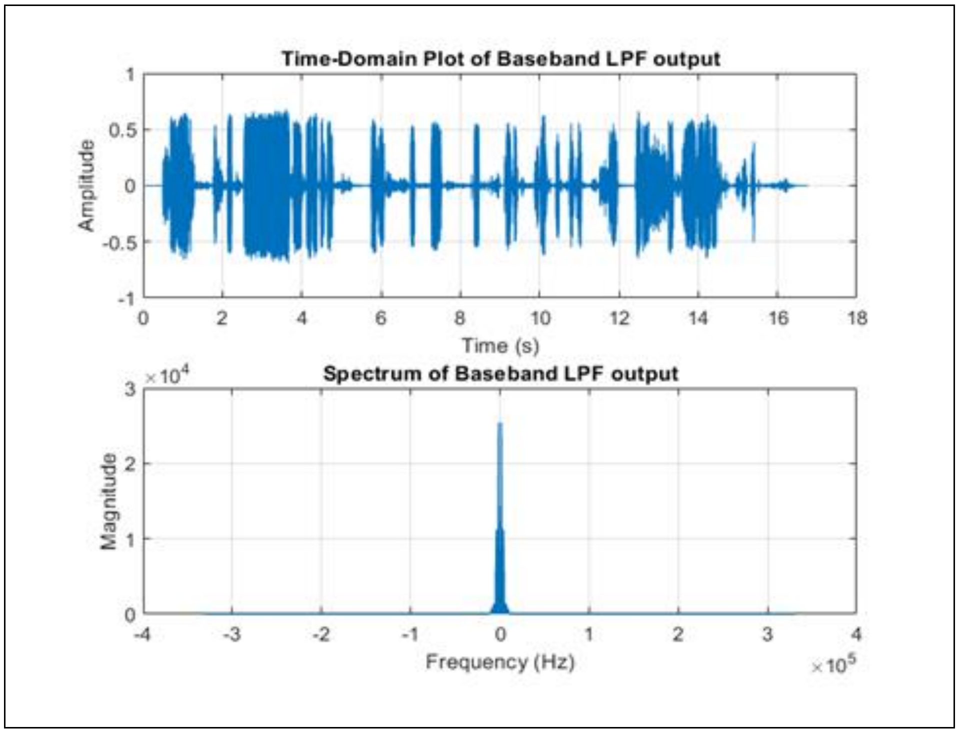


Figure 14: Output of the LPF (with 0.2kHz offset)



B. Offset = 1.2kHz

Figure 15 : Output of RF mixer (with 1.2kHz offset)

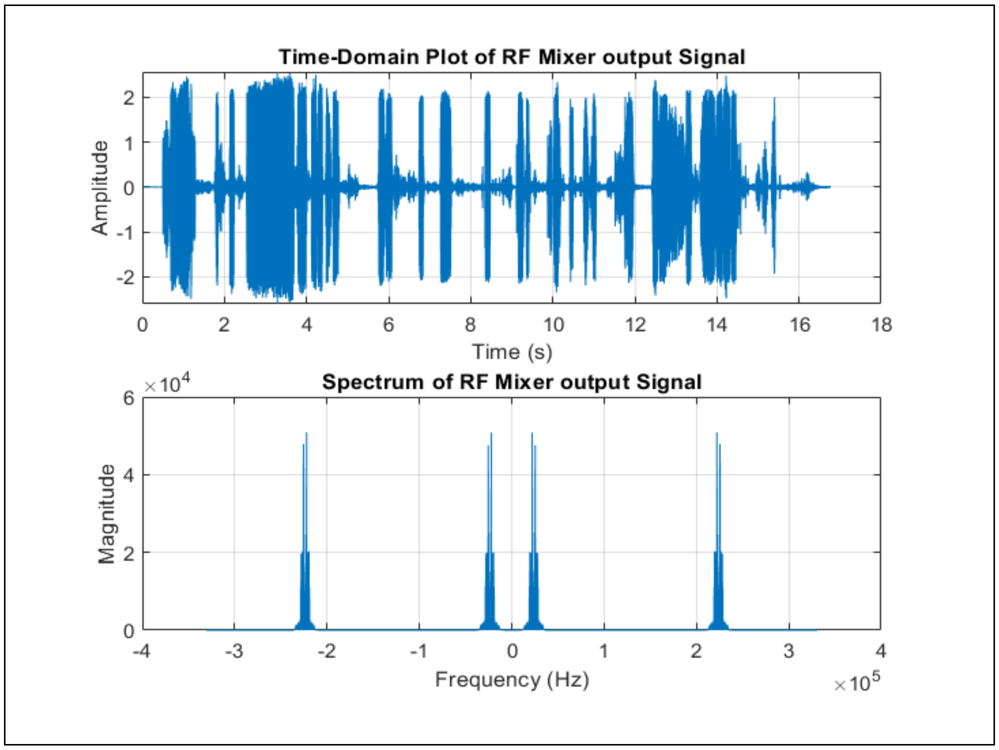


Figure 16: Output of IF filter (with 1.2kHz offset)

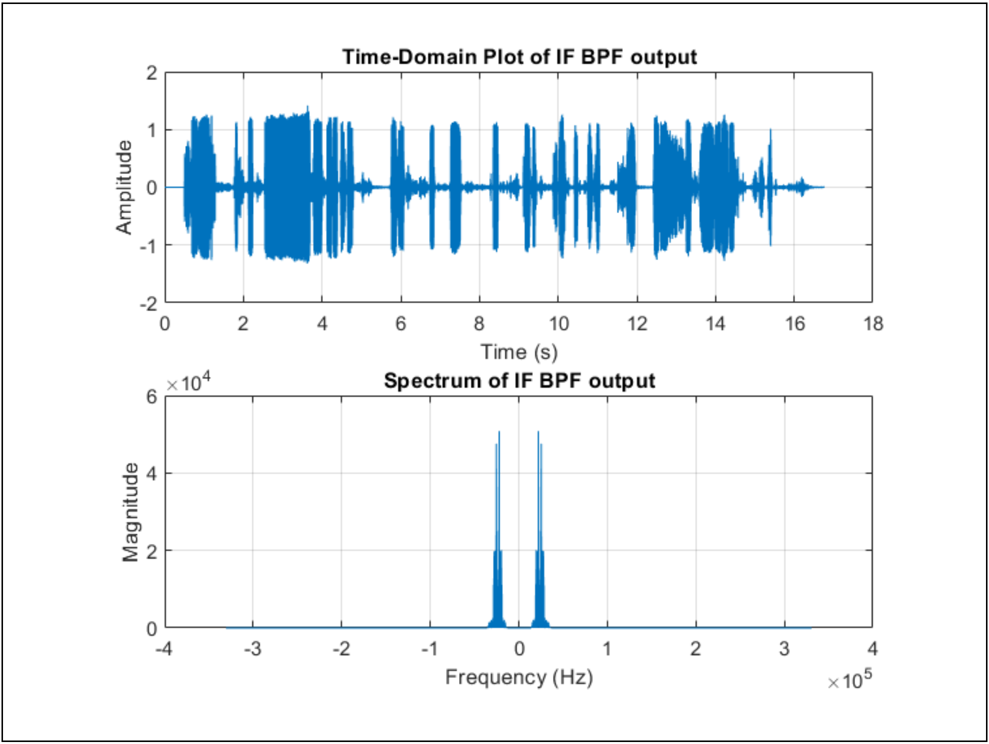


Figure 17: Output of the mixer before the LPF (with 1.2kHz)

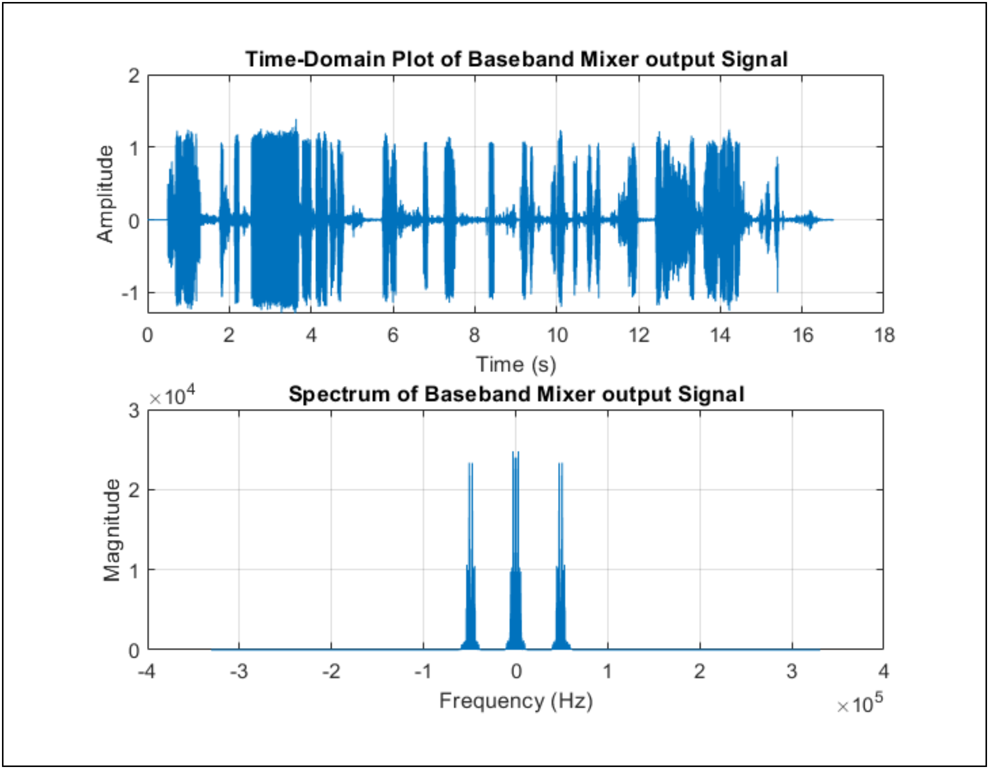
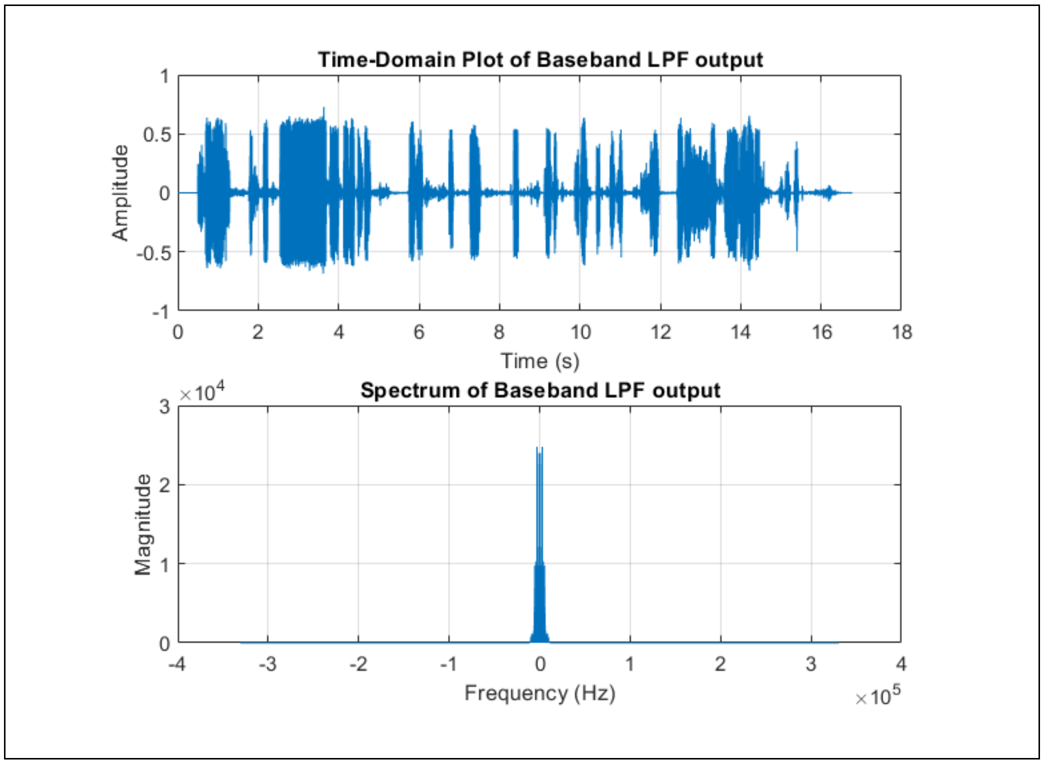


Figure 18: Output of the LPF (with 1.2kHz offset)



Discussion

What happens (in terms of spectrum and the sound quality) if the receiver oscillator has frequency offset by 0.2 KHz and 1.2 KHz

Spectrum is shifted (assuming selection of first signal) after the RF mixer to be at 25.2kHz,225.2kHz for frequency offset of 0.2kHz and 26.2kHz ,226.2kHz frequency offset of 1.2kHz at the RF mixer oscillator. That caused the sound quality to be more magnificent as the IF filter was centered at 25kHz and passband of 10kHz around, so some frequency components of the signal were filtered and hence lost.

8. ***Performance evaluation in presence of noise***

Figures

Figure 19: Output of RF filter (with noise)

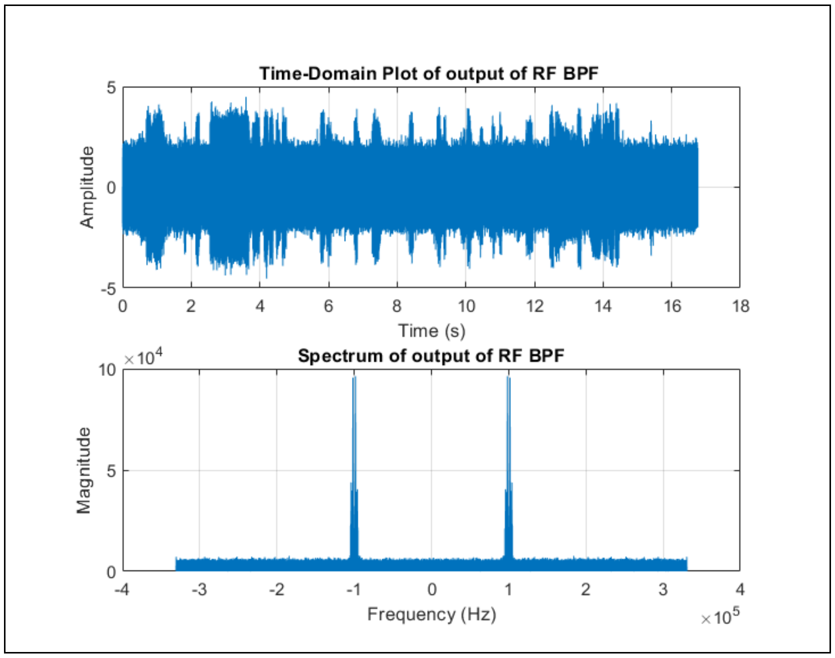
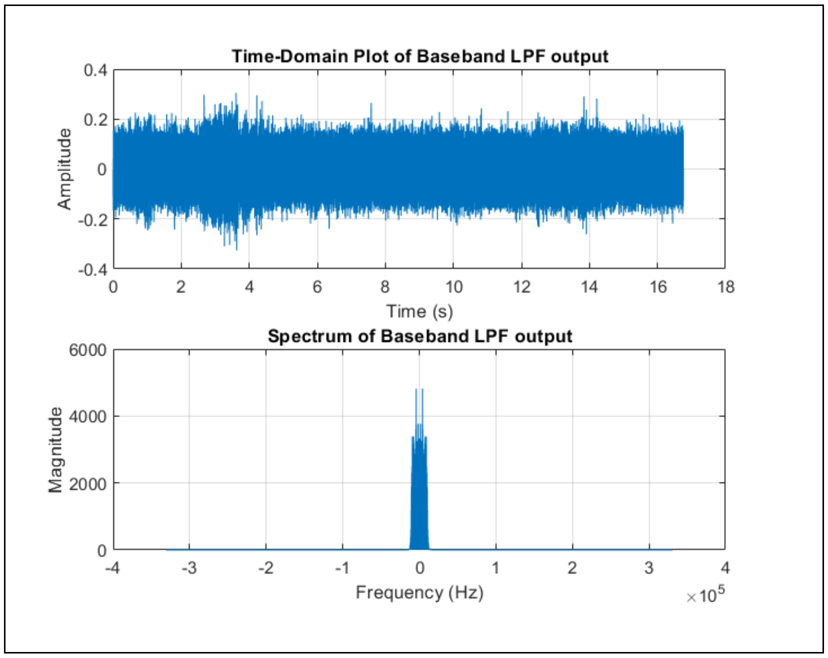


Figure 20: Output of the LPF (with noise)



Discussion

Noise of SNR = 5dB was added. Its effect was like loud hissing due to noise and very small sound was heard from original signal. It's also obvious from previous figures the effect of noise in time domain (signal was buried in the noise) and in frequency domain (noise floor is added to spectrum).

9. The code

```
clc; clear; close all;

% File paths for the input signals
file1 = 'Short_QuranPalestine.wav';
file2 = 'Short_FM9090.wav';

% Read the stereo signals
[stereo1, fs1] = audioread(file1);
[stereo2, fs2] = audioread(file2);

% Convert stereo to mono by averaging the two channels
mono1 = stereo1(:,1) + stereo1(:,2);
mono2 = stereo2(:,1) + stereo2(:,2);

% Zero-pad the shorter signal to match the length of the longer one
maxLength = max([length(mono1), length(mono2)]);
mono1 = [mono1; zeros(maxLength - length(mono1), 1)];
mono2 = [mono2; zeros(maxLength - length(mono2), 1)];

% Create time vectors for mono signals
t1_orig = (0:length(mono1)-1) / fs1; % Time vector for mono1
t2_orig = (0:length(mono2)-1) / fs2; % Time vector for mono2

% Define the upsampling factor
r = 15; % Increase sampling rate by 15 times

% Calculate new sampling frequency
new_fs = r * fs1; % New sampling frequency

% Resample signals
mono1_resampled = interp(mono1, r); % Upsample mono1 by factor r
mono2_resampled = interp(mono2, r); % Upsample mono2 by factor r

N_resampled = max([length(mono1_resampled), length(mono2_resampled)]);
% Create new time vectors for resampled signals
t_new = (0:N_resampled-1) / new_fs; % New time axis for mono
f_resampled = (-N_resampled/2:N_resampled/2-1) * (new_fs / N_resampled);

% --- Plot Signal 1 AFTER resampling: Time-Domain and Frequency-Domain ---
figure;
subplot(2,1,1);
plot(t_new, mono1_resampled);
title('Time-Domain Plot of Resampled Signal 1 (After resampling)');
xlabel('Time (s)');
```

```
ylabel('Amplitude');
grid on;

% FFT for resampled mono1
N_resampled = length(mono1_resampled);
Y1_resampled = fft(mono1_resampled);
Y1_resampled_shifted = fftshift(Y1_resampled);

subplot(2,1,2);
plot(f_resampled, abs(Y1_resampled_shifted));
title('Spectrum of Resampled Signal 1 (After resampling)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% --- Plot Signal 2 AFTER resampling: Time-Domain and Frequency-Domain ---
figure;
subplot(2,1,1);
plot(t_new, mono2_resampled);
title('Time-Domain Plot of Resampled Signal 2 (After resampling)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% FFT for resampled mono2
modulated2_freq = fft(mono2_resampled);
modulated2_freq_shifted = fftshift(modulated2_freq);

subplot(2,1,2);
plot(f_resampled, abs(modulated2_freq_shifted));
title('Spectrum of Resampled Signal 2 (After resampling)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

BW_arr = [10e3 10e3]; % Baseband BW of signals

% AM
% Carrier frequencies for DSB-SC modulation
fc1 = 100e3; % Carrier frequency for signal 1 (100 kHz)
fc2 = 150e3; % Carrier frequency for signal 2 (100 + Δf with Δf = 50 kHz)

% Generate carriers
carrier1 = cos(2 * pi * fc1 * t_new); % Carrier for signal 1
carrier2 = cos(2 * pi * fc2 * t_new); % Carrier for signal 2

% Modulate the signals using DSB-SC
modulated1 = mono1_resampled .* carrier1'; % Signal 1 modulated at 100 kHz
```

```
modulated2 = mono2_resampled .* carrier2'; % Signal 2 modulated at 150 kHz

% Construct the FDM signal by summing the modulated signals
fdm_signal = modulated1 + modulated2;

% % --- Plot FDM signal: Time-Domain and Frequency-Domain ---
figure;
subplot(2,1,1);
plot(t_new, fdm_signal);
title('Time-Domain Plot of modulated signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% FFT for resampled mono2
modulated_freq = fft(fdm_signal);
modulated_freq_shifted = fftshift(modulated_freq);

subplot(2,1,2);
plot(f_resampled, abs(modulated_freq_shifted));
title('Spectrum of modulated signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% RF BPF
signal_num = input("enter signal number (0 for first signal, 1 for second signal)\n");
BandPassFilt_RF = RF_BPF(signal_num, BW_arr,new_fs);

% Apply the Bandpass Filter to FDM Signal
RF_BPF_out = filter(BandPassFilt_RF, fdm_signal);

% Adding noise to signal after RF BPF
% RF_BPF_out = awgn(RF_BPF_out, 5);

% Removing RF BPF
% RF_BPF_out = fdm_signal;

% --- Plot Filtered Signal: Time-Domain and Frequency-Domain ---
figure;

% Time-Domain Plot
subplot(2,1,1);
plot(t_new, RF_BPF_out);
title('Time-Domain Plot of output of RF BPF ');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

```
% FFT of Filtered Signal
N_filtered = length(RF_BPF_out);
filtered_fft = fft(RF_BPF_out);
filtered_fft_shifted = fftshift(filtered_fft);
f_filtered = (-N_filtered/2:N_filtered/2-1) * (new_fs / N_filtered);

% Frequency-Domain Plot
subplot(2,1,2);
plot(f_filtered, abs(filtered_fft_shifted));
title('Spectrum of output of RF BPF');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Define Intermediate Frequency
f_IF = 25e3; % Intermediate frequency (25 kHz)

% Define Oscillator Frequency
f_osc = (100e3 + signal_num * 50e3) + f_IF;

% Adding offset (0.2 kHz and 1.2kHz)
% f_osc = f_osc - 0.2e3; % for frequency offset of 0.2kHz
% f_osc = f_osc - 1.2e3; % for frequency offset of 1.2kHz

% Generate Oscillator Signal
oscillator_signal = cos(2 * pi * f_osc * t_new);

% --- Mix Filtered Signal with Oscillator Signal ---
IF_signal = RF_BPF_out .* oscillator_signal';

% --- Plot Intermediate Frequency (IF) Signal ---
figure;

% Time-Domain Plot
subplot(2,1,1);
plot(t_new, IF_signal);
title('Time-Domain Plot of RF Mixer output Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% FFT of IF Signal
IF_fft = fft(IF_signal);
IF_fft_shifted = fftshift(IF_fft);

% Frequency-Domain Plot
subplot(2,1,2);
```

```
plot(f_resampled, abs(IF_fft_shifted));
title('Spectrum of RF Mixer output Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% IF BPF
BandPassFilt_IF = IF_BPF(signal_num, BW_arr, new_fs);

% Apply the Bandpass Filter to FDM Signal
IF_BPF_out = filter(BandPassFilt_IF, IF_signal);

% --- Plot Intermediate Frequency (IF) Signal ---
figure;

% Time-Domain Plot
subplot(2,1,1);
plot(t_new, IF_BPF_out);
title('Time-Domain Plot of IF BPF output');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% FFT of IF Signal
IF_BPF_out_fft = fft(IF_BPF_out);
IF_BPF_out_fft_shift = fftshift(IF_BPF_out_fft);

% Frequency-Domain Plot
subplot(2,1,2);
plot(f_resampled, abs(IF_BPF_out_fft_shift));
title('Spectrum of IF BPF output');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Baseband detection

% --- Mix with Local Carrier (for Baseband Detection) ---
local_carrier = cos(2 * pi * f_IF * t_new); % Local oscillator signal

baseband_mixed = IF_BPF_out .* local_carrier'; % Mix IF signal with local carrier

figure;
% Time-Domain Plot
subplot(2, 1, 1);
plot(t_new, baseband_mixed);
title('Time-Domain Plot of Baseband Mixer output Signal');
xlabel('Time (s)');
```

```
ylabel('Amplitude');
grid on;

% FFT of Baseband Signal
baseband_mixed_fft = fft(baseband_mixed);
baseband_mixed_fft_shift = fftshift(baseband_mixed_fft);

% Frequency-Domain Plot
subplot(2, 1, 2);
plot(f_resampled, abs(baseband_mixed_fft_shift));
title('Spectrum of Baseband Mixer output Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% --- Define LPF Specifications ---
F_stop = 20e3;  % Stopband frequency = 20 kHz
F_pass = 10e3;  % Passband frequency = 10 kHz
A_stop = 60;    % Attenuation in the stopband = 60 dB
A_pass = 1;     % Amount of ripple allowed in the passband = 1 dB

% Create Low-Pass Filter Specification Object
LPFSpecObj = fdesign.lowpass('Fp,Fst,Ap,Ast', F_pass, F_stop, A_pass, A_stop, new_fs);

% Design the LPF using Butterworth Filter
LPF = design(LPFSpecObj, 'butter');

% --- Visualize the Low-Pass Filter Characteristics ---
%fvtool(LPF); % to view LPF response

% --- Apply Low-Pass Filter to Extract Baseband Signal ---
baseband_signal = filter(LPF, baseband_mixed);

% --- Plot Baseband Signal: Time-Domain and Frequency-Domain ---
figure;

% Time-Domain Plot
subplot(2, 1, 1);
plot(t_new, baseband_signal);
title('Time-Domain Plot of Baseband LPF output');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% FFT of Baseband Signal
baseband_fft = fft(baseband_signal);
baseband_fft_shifted = fftshift(baseband_fft);
```

```
% Frequency-Domain Plot
subplot(2, 1, 2);
plot(f_resampled, abs(baseband_fft_shifted));
title('Spectrum of Baseband LPF output');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Normalize the baseband signal
baseband_signal_normalized = baseband_signal / max(abs(baseband_signal));

% Downsample to 44.1 kHz
fs_downsampled = 44100; % Target sampling frequency
downsample_factor = new_fs / fs_downsampled;
baseband_downsampled = downsample(baseband_signal_normalized, round(downsample_factor));

% Play the downsampled signal
sound(baseband_downsampled, fs_downsampled);


% RF BPF
function BandPassFilt_RF = RF_BPF(signal_num, BW_arr,new_fs)

fc = 100e3 + signal_num * (50e3);
BW = BW_arr(1,signal_num + 1);

% Define Bandpass Filter Specifications
F_pass1 = (fc - BW);        % Edge of the passband
F_pass2 = (fc + BW);        % Closing edge of the passband
F_stop1 = F_pass1 - 10e3;           % Edge of the stopband
F_stop2 = F_pass2 + 10e3;           % Edge of the second stopband
A_stop1 = 60;               % Attenuation in the first stopband
A_stop2 = 60;               % Attenuation in the second stopband
A_pass = 1;                 % Amount of ripple allowed in the passband

BandPassSpecObj_RF =  fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', ...
                    F_stop1, F_pass1, F_pass2, F_stop2, A_stop1, A_pass, ...
                    A_stop2, new_fs);

BandPassFilt_RF = design(BandPassSpecObj_RF, 'butter');

% --- Plot the Filter Characteristics ---
%fvtool(BandPassFilt_RF); % Visualize the filter's frequency and phase response

end


% IF BPF
```

```
function BandPassFilt_IF = IF_BPF(signal_num, BW_arr,new_fs)

% fc = 100e3 + signal_num * (50e3);
BW = BW_arr(1,signal_num + 1);

% IF BPF

% Define Bandpass Filter Specifications
IF_F_pass1 = 25e3 - BW;
IF_F_pass2 = 25e3 + BW;
IF_F_stop1 = IF_F_pass1 - 10e3;
IF_F_stop2 = IF_F_pass2 + 10e3;
IF_A_stop1 = 60;
IF_A_stop2 = 60;
IF_A_pass = 1;

BandPassSpecObj_IF =  fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', ...
                     IF_F_stop1, IF_F_pass1, IF_F_pass2, IF_F_stop2, IF_A_stop1, IF_A_pass, ...
                     IF_A_stop2, new_fs);

BandPassFilt_IF = design(BandPassSpecObj_IF, 'butter');

% --- Plot the Filter Characteristics ---
%fvtool(BandPassFilt_IF); % Visualize the filter's frequency and phase response

end
```