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:

شهاب أبو زيد عبدالعال أبو زيد

Sec: 2 BN: 29 id: 9230477

Contents

1. The transmitter 3
   Discussion 3
   The figures 3
2. The RF stage 4
   Discussion 4
   The figures 4
3. The IF stage 6
   Discussion 6
   The figures 6
4. The baseband demodulator 7
   Discussion 7
   The figures 7
5. Performance evaluation without the RF stage 9
   The figures 9
6. Comment on the output sound 11
7. The code 12

Table of figures

Figure 1: The spectrum of the output of the transmitter 3
Figure 2: the output of the RF filter (before the mixer) 4
Figure 3: The output of the mixer 5
Figure 4: Output of the IF filter 6
Figure 5: Output of the mixer (before the LPF) 7
Figure 6: Output of the LPF 8
Figure 7: output of the RF mixer (no RF filter) 9
Figure 8: Output of the IF filter (no RF filter) 9
Figure 9: Output of the IF mixer before the LPF (no RF filter) 10
Figure 10: Ouptut of the LPF (no RF filter) 10
Figure 11:Output with 0.1KHZ frequency offset 11
Figure 12:Output with 1KHZ frequency offset 11

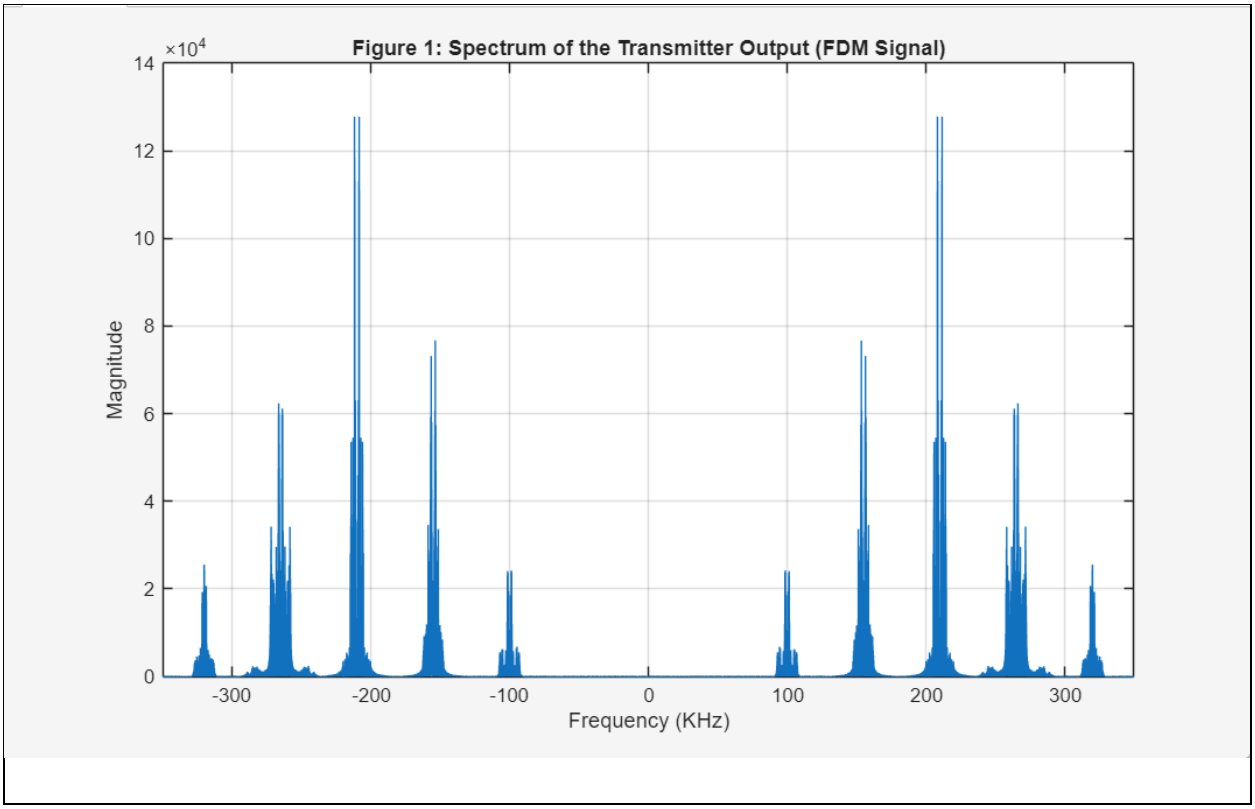
1. *The transmitter*

Discussion

I first pre-processed the five audio signals by converting them to mono, padding them to equal lengths, and scaling their amplitudes. I then **upsampled** (interpolated) the signals to increase the sampling rate, which is necessary to avoid aliasing since the carrier frequencies (100–320 KHz) are much higher than the original audio rate. Finally, we modulated each signal using DSB-SC on distinct carrier frequencies and summed them to create the FDM signal.

The 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

Role: The RF stage acts as a tunable Band-Pass Filter (BPF). In this step, we tuned the filter to the first station's carrier frequency (100 KHz) with a bandwidth of 20 KHz. Following the filter, a mixer multiplies the signal with a local oscillator frequency (ω_{LO} = 127.5 KHz) to shift the desired signal to the Intermediate Frequency (IF).
Why we need it: The RF filter is essential for **image rejection.** It isolates the desired station and suppresses other signals (like the station at 155 KHz) before mixing. If we do not filter first, other stations could interfere with our signal after the mixing process.

The figures

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

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

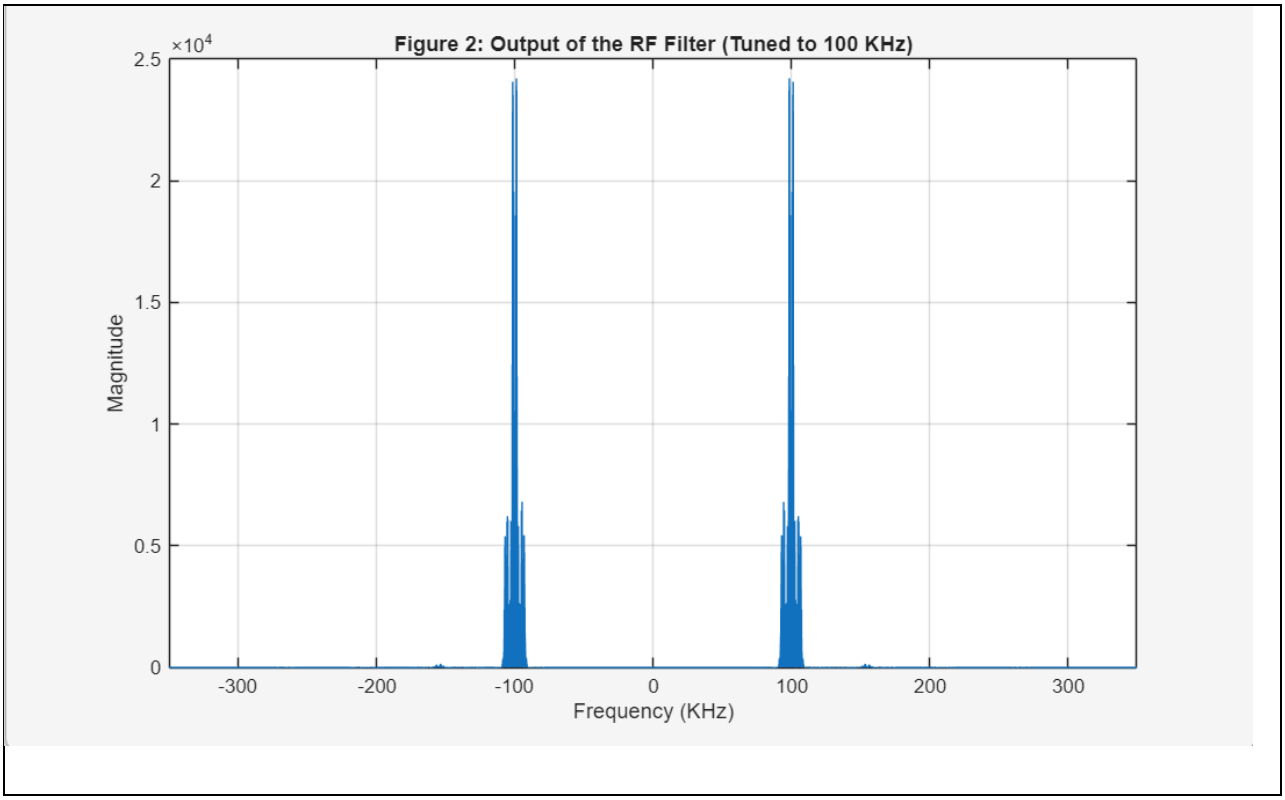
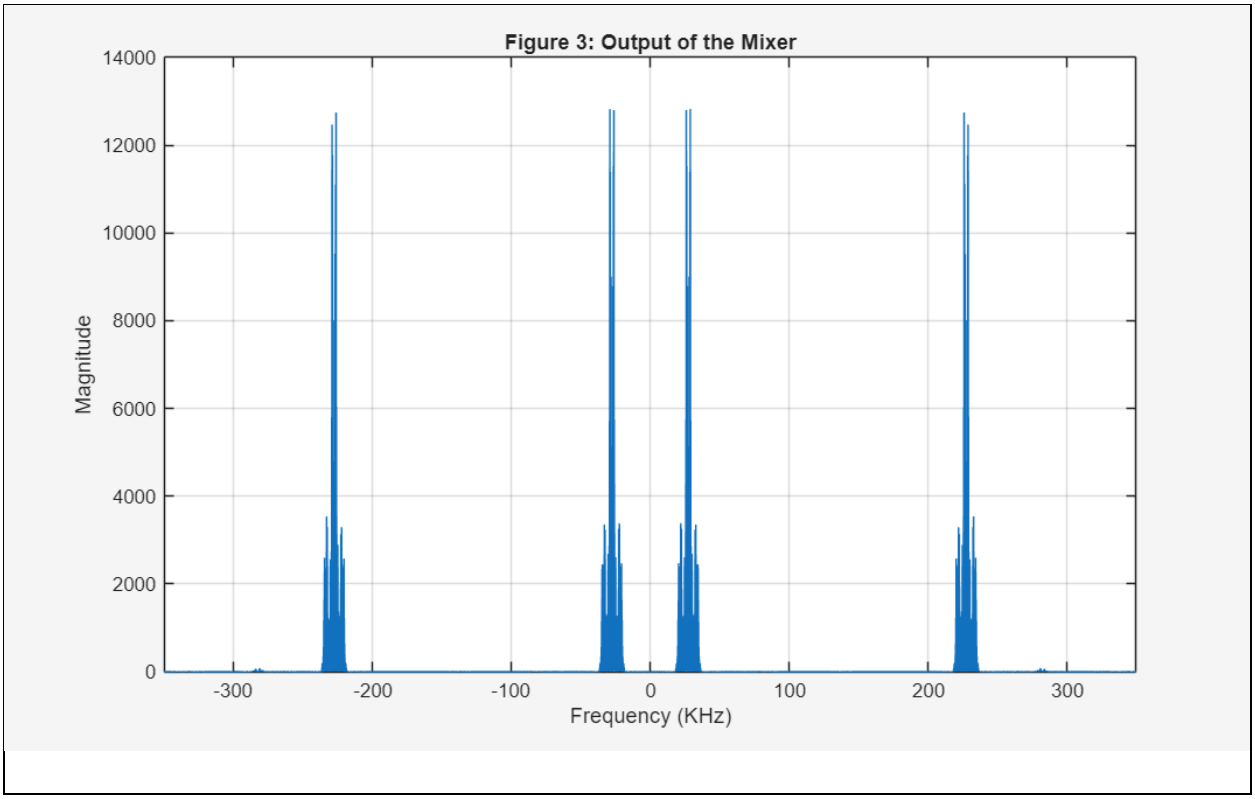


Figure 3: The output of the mixer



3. *The IF stage*

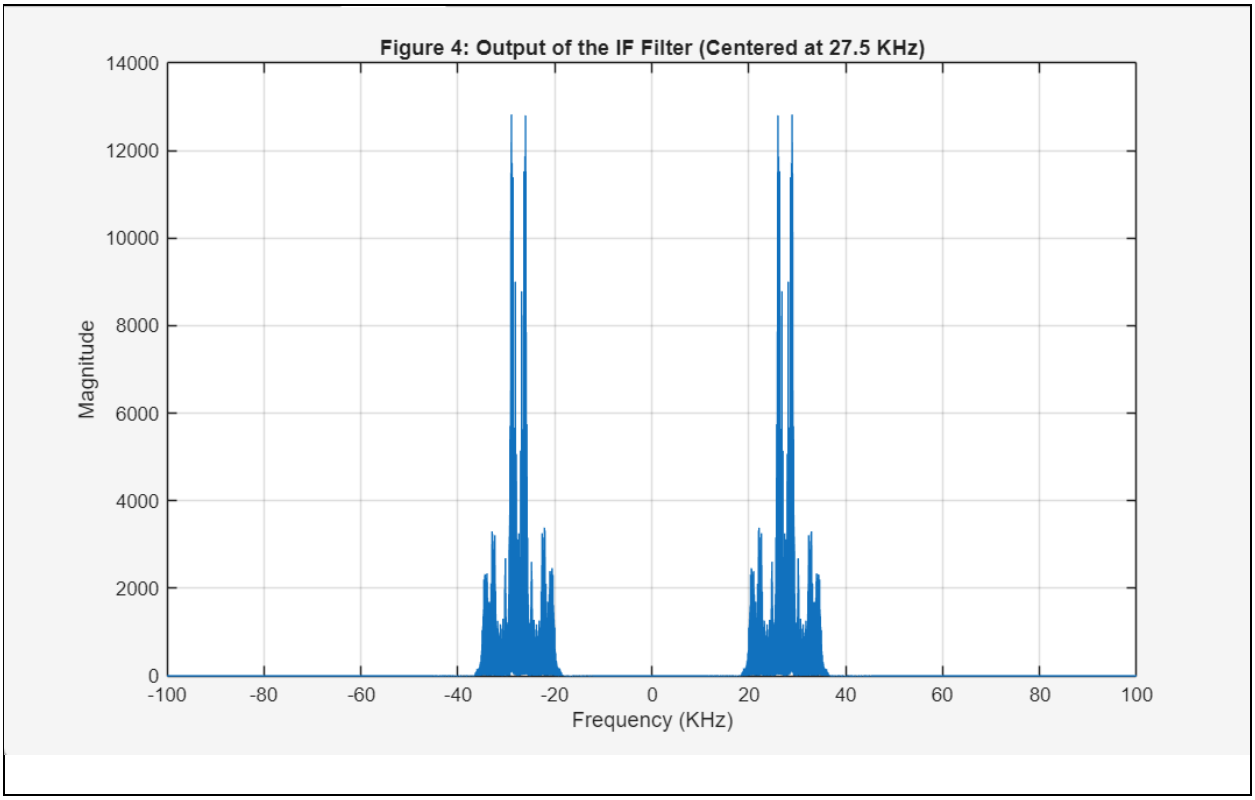
This part addresses the IF filter.

Discussion

Role: The Intermediate Frequency (IF) stage consists of a high-quality Band-Pass Filter that is fixed at $\{\omega_{IF}\}$ = 27.5 KHz. Regardless of which station we tune to in the RF stage, the signal is always shifted to this frequency before reaching this filter.
Why we need it: This stage provides the main **selectivity** of the receiver. Because the filter frequency is fixed, we can design it to be very sharp and precise. It removes the high-frequency components (sum frequencies) generated by the mixer and any remaining interference, leaving only the desired signal ready for demodulation.

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

Role: The Baseband Demodulator performs **coherent detection**[2]. The signal from the IF stage (at 27.5 KHz) is mixed with a local carrier of the exact same frequency (ω_{IF} = 27.5 KHz). This multiplication shifts the spectrum to two locations: the difference frequency (Baseband/0 Hz) and the sum frequency ($2w_{IF}$ = 55 KHz).
LPF Role: A Low-Pass Filter (LPF) with a cutoff of 10 KHz is used to filter this output. It removes the unwanted high-frequency component at 55 KHz, leaving only the original baseband audio signal, which is then sent to the speakers
.

The figures

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

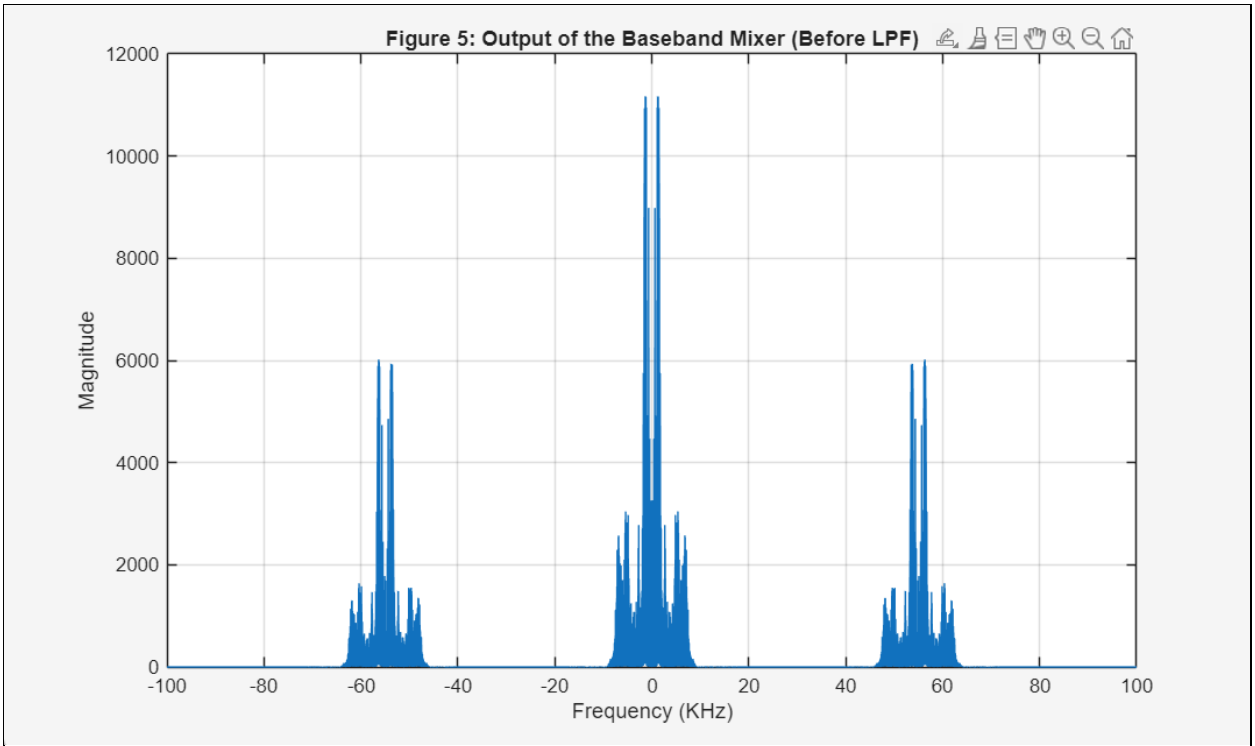
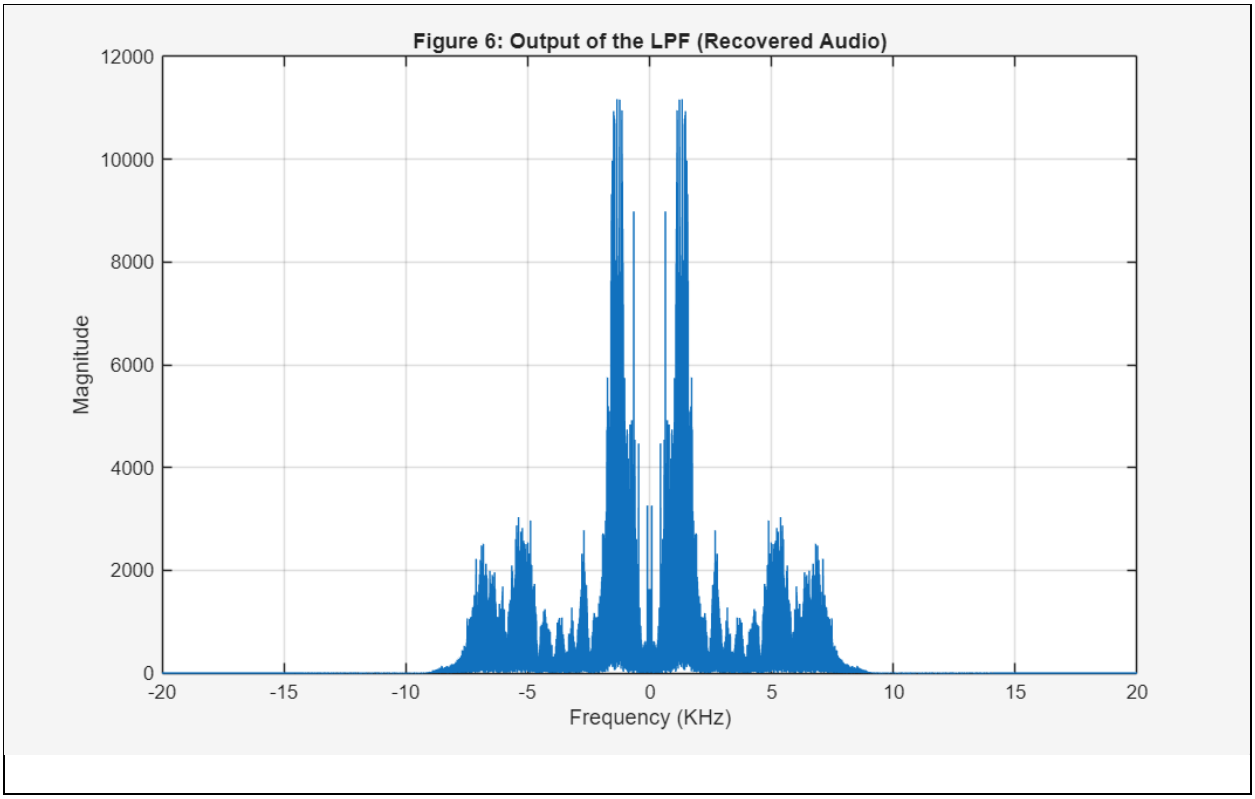


Figure 6: Output of the LPF



5. ***Performance evaluation without the RF stage***

The figures

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

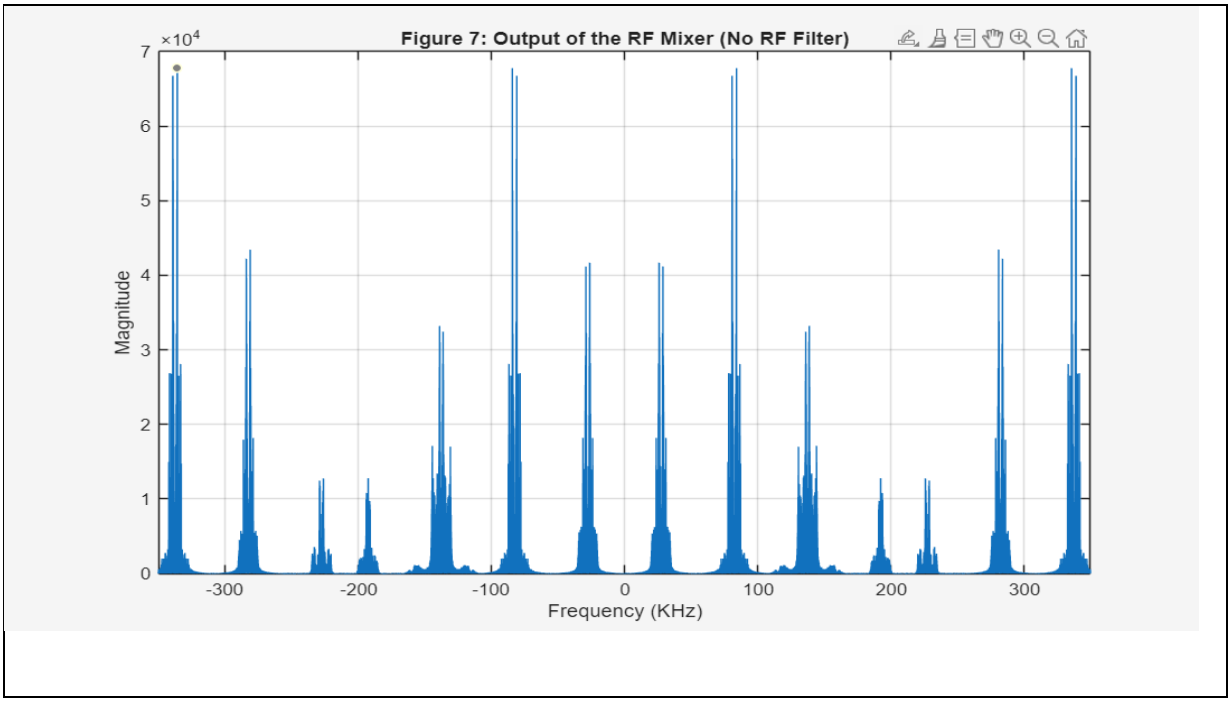


Figure 8: Output of the IF filter (no RF filter)

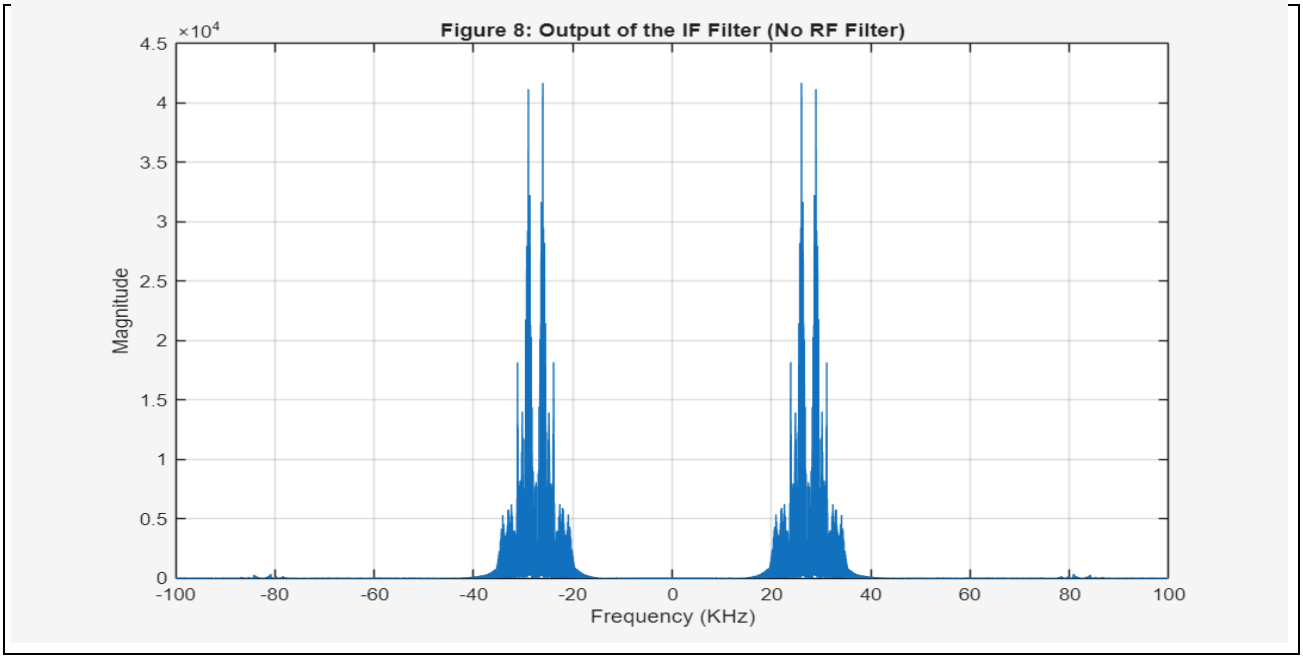


Figure 9: Output of the IF mixer before the LPF (no RF filter)

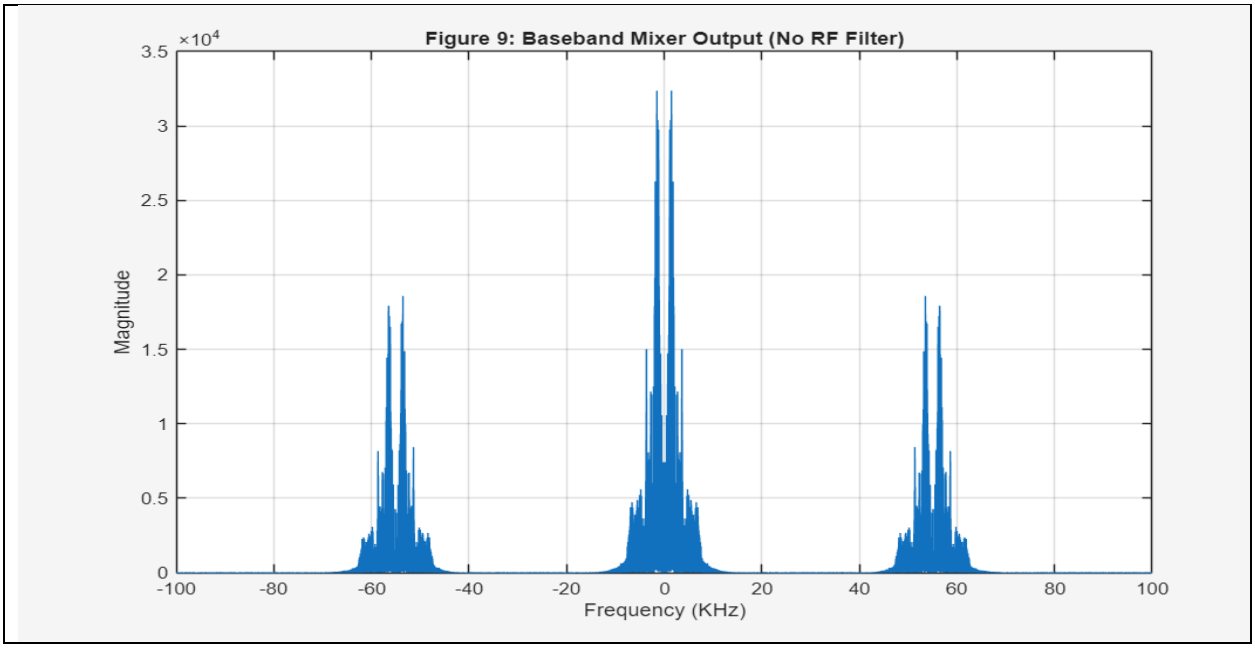
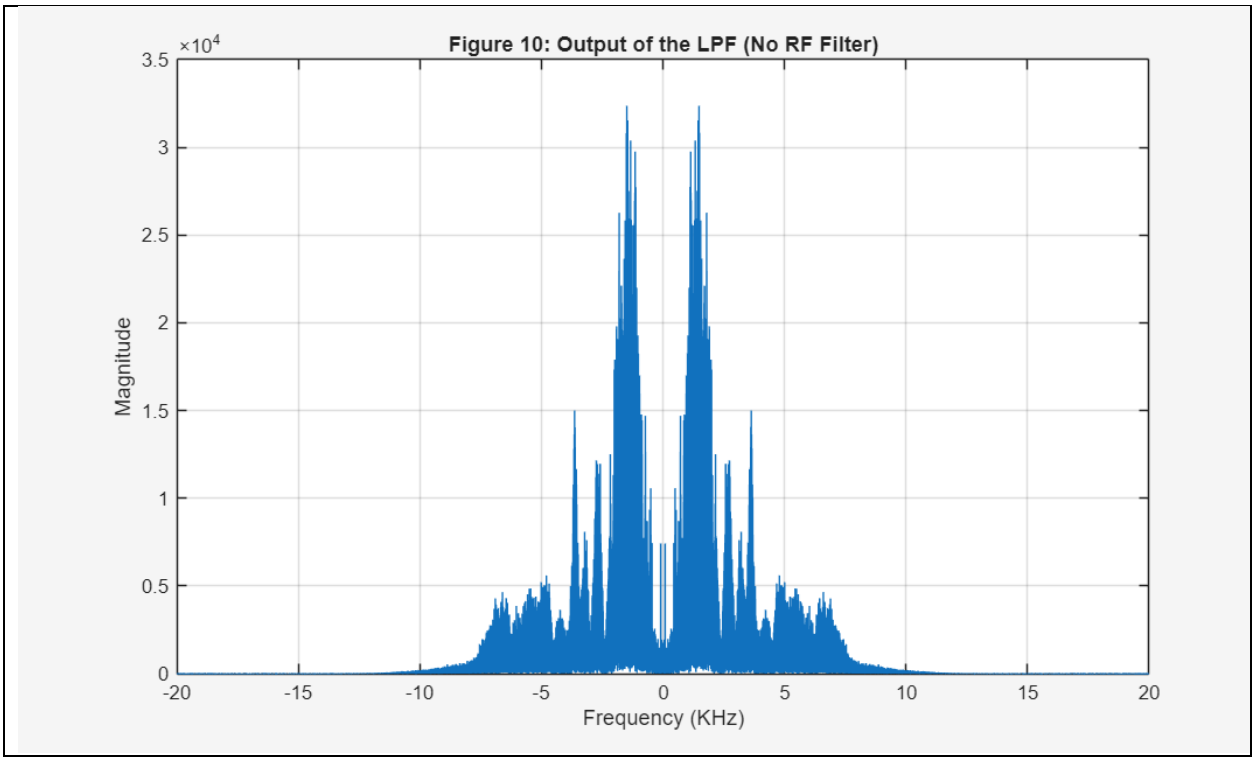


Figure 10: Ouptut of the LPF (no RF filter)



6. *Comment on the output sound*

Observation: When the RF stage (BPF) was removed, I heard **two stations playing simultaneously**. I heard the desired station (BBC Arabic at 100 KHz) mixed with the second station (FM9090 at 155 KHz).
Explanation (The Image Problem):
This occurred because of the Image Frequency phenomenon.

- We tuned our oscillator to ω_{LO} = 127.5 KHz to catch the 100 KHz station (|100 - 127.5| = 27.5 KHz).
- However, the second station is at 155 KHz. When mixed, it produces a difference frequency of |155 - 127.5| = 27.5 KHz.
- Since both stations land on exactly 27.5 KHz (the IF frequency), the IF filter cannot separate them. The RF filter is therefore strictly required to reject this "Image" (155 KHz) *before* it reaches the mixer.

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

We simulated the frequency offsets in MATLAB:

- **With 0.1 KHz offset:** The spectrum shifted by 100 Hz. The audio sounded like it was vibrating or pulsating (beating effect), as the offset frequency creates a low-frequency amplitude modulation.
- **With 1 KHz offset:** The spectrum shifted by 1 KHz. The audio became high-pitched and metallic (similar to a robot voice or ghost), making the speech difficult to understand. This confirms that coherent detection requires precise frequency synchronization.

Figure 11:Output with 0.1KHZ frequency offset

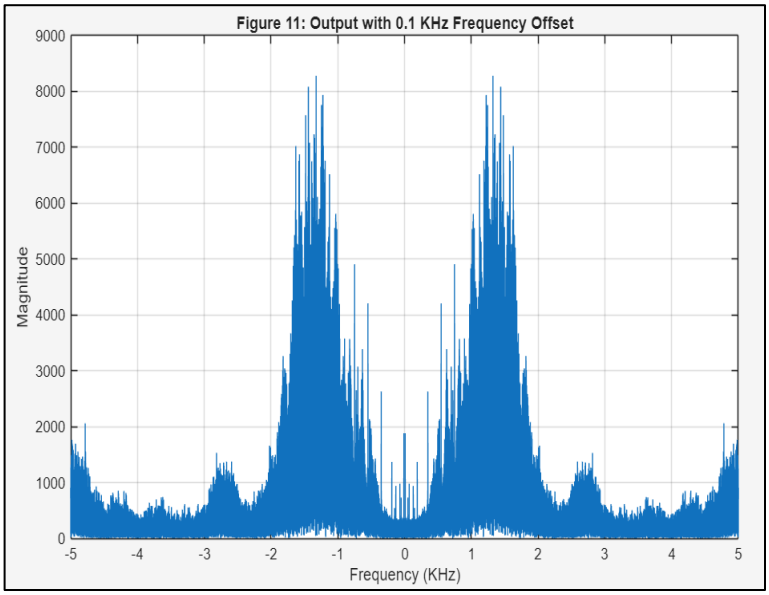
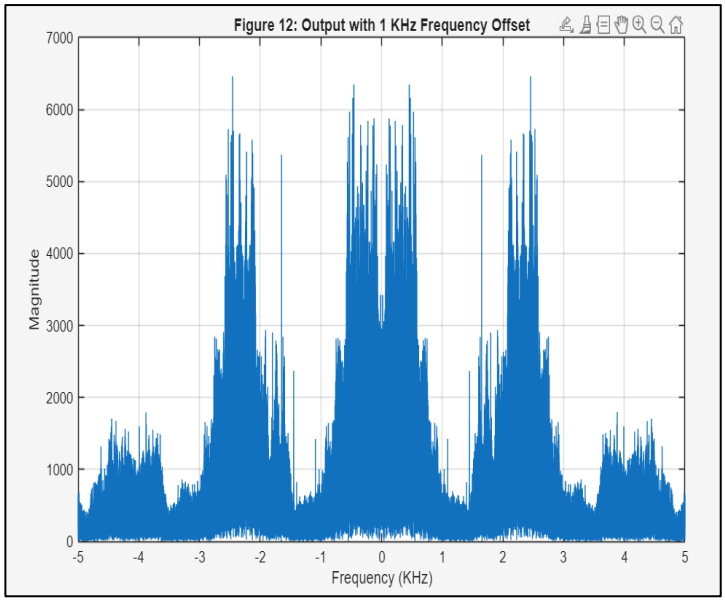


Figure 12:Output with 1KHZ frequency offset



7. ***The code***

```
% ========================================================================
% Project: Super-heterodyne Receiver Simulation (Analog Communications)
% Student Name: Shehab Abo Zaid Abdel Aal Abozaid
% ID: 9230477
% Date: Fall 2025
% ========================================================================

%% Part 1: The Transmitter (5 Signals - Reordered)
clear; clc; close all;

% 1. Define File Names (Order: 9090, Sky, Quran, Russian, BBC)
filenames = {'Short_BBCArabic2.wav', ...
             'Short_FM9090.wav', ...
             'Short_QuranPalestine.wav', ...
             'Short_RussianVoice.wav', ...
             'Short_SkyNewsArabia.wav'};

num_signals = length(filenames);
delta_F = 55000; % 55 KHz spacing
start_freq = 100000; % 100 KHz starting carrier

% Initialize variables
sig_cell = cell(1, num_signals);
max_len = 0;
fs_orig = 0;

disp('Reading audio files...');

% 2. Load and Pre-process (Mono conversion + Find Max Length)
for i = 1:num_signals
    [temp_sig, fs_temp] = audioread(filenames{i});

    % Convert to Monophonic if Stereo (Sum columns)
    if size(temp_sig, 2) == 2
        temp_sig = temp_sig(:, 1) + temp_sig(:, 2);
    end

    sig_cell{i} = temp_sig;

    % Track max length for zero padding later
    if length(temp_sig) > max_len
        max_len = length(temp_sig);
    end

    % We assume all files have roughly the same sample rate, taking the first one
    if i == 1
        fs_orig = fs_temp;
    end
end

% 3. Padding and Upsampling
interp_factor = 20; % Increase sampling rate to avoid aliasing
Fs_new = fs_orig * interp_factor;
```

```
% Create Time Vector for the longest signal (upsampled)
L_new = max_len * interp_factor;
t = (0:L_new-1) / Fs_new;

disp('Modulating and Multiplexing signals...');
Tx_Signal = zeros(length(t), 1); % Initialize total FDM signal

for i = 1:num_signals
    % A. Pad with zeros to match max length
    current_sig = sig_cell{i};
    padded_sig = [current_sig; zeros(max_len - length(current_sig), 1)];

    % B. Upsample (Interpolation)
    upsampled_sig = interp(padded_sig, interp_factor);

    % Fix minor length mismatch due to interpolation
    if length(upsampled_sig) > length(t)
        upsampled_sig = upsampled_sig(1:length(t));
    elseif length(upsampled_sig) < length(t)
        upsampled_sig = [upsampled_sig; zeros(length(t)-length(upsampled_sig),1)];
    end

    % C. Calculate Carrier Frequency
    % n = 0 for 1st signal, n=1 for 2nd...
    % Formula: wc = 100K + n*55K
    n = i - 1;
    fc = start_freq + n * delta_F;

    % D. Generate Carrier
    carrier = cos(2*pi*fc*t)';

    % E. DSB-SC Modulation (Mixing)
    mod_sig = upsampled_sig .* carrier;

    % F. Add to total FDM signal (Multiplexing)
    Tx_Signal = Tx_Signal + mod_sig;

    fprintf('Signal %d modulated at %.1f KHz\n', i, fc/1000);
end

% 4. Plotting the Spectrum (Figure 1)
disp('Plotting spectrum in KHz...');
% Define frequency vector in Hz
f_Hz = (-length(Tx_Signal)/2 : length(Tx_Signal)/2 - 1) * (Fs_new /
length(Tx_Signal));
% Convert to KHz for plotting
f_KHz = f_Hz / 1000;

Tx_Spectrum = fftshift(fft(Tx_Signal));
figure;
plot(f_KHz, abs(Tx_Spectrum));
title('Figure 1: Spectrum of the Transmitter Output (FDM Signal)');
xlabel('Frequency (KHz)'); % Label is now KHz
ylabel('Magnitude');
```

```
% Change limits to match KHz (350 KHz instead of 350000 Hz)
xlim([-350 350]);
grid on;

% Save the variable for the next stages
save('Tx_Data.mat', 'Tx_Signal', 'Fs_new', 't', 'delta_F', 'start_freq',
'filenames');
disp('Part 1 Complete. Data saved to Tx_Data.mat');


%% Part 2: The RF Stage
% Goal: Tune to the first station (100 KHz) and mix to IF (27.5 KHz)
disp('-----------------------------------');
disp('Starting RF Stage...');

% --- 1. RF Filter Design (Band-Pass Filter) ---
% We want to tune to the first carrier: 100 KHz
Fc_RF = 100000;  % Center Frequency
BW_RF = 20000;   % Bandwidth (20 KHz)

% Design a Butterworth BPF (4th Order)
Wn_RF = [(Fc_RF - BW_RF/2)/(Fs_new/2), (Fc_RF + BW_RF/2)/(Fs_new/2)];
[b_rf, a_rf] = butter(4, Wn_RF, 'bandpass');

% Apply the filter to the Tx_Signal
RF_Output = filter(b_rf, a_rf, Tx_Signal);

% --- 2. Plot RF Output (Figure 2) ---
disp('Plotting RF Output...');
RF_Spectrum = fftshift(fft(RF_Output));

% Recalculate Frequency Vector in KHz
f_Hz = (-length(RF_Output)/2 : length(RF_Output)/2 - 1) * (Fs_new /
length(RF_Output));
f_KHz = f_Hz / 1000;

figure;
plot(f_KHz, abs(RF_Spectrum));
title('Figure 2: Output of the RF Filter (Tuned to 100 KHz)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-350 350]); % View full range to show other stations are gone
grid on;

% --- 3. The Mixer (RF to IF) ---
% Requirement: Oscillator = w_c + w_IF
IF_Freq = 27500;           % IF = 27.5 KHz (User Specified)
LO_Freq = Fc_RF + IF_Freq; % 100 + 27.5 = 127.5 KHz
fprintf('Local Oscillator Frequency: %.1f KHz\n', LO_Freq/1000);

% Generate Oscillator
oscillator = cos(2*pi*LO_Freq*t)';

% Mix (Multiply)
Mixer_Output = RF_Output .* oscillator;
```

```
% --- 4. Plot Mixer Output (Figure 3) ---
disp('Plotting Mixer Output...');
Mixer_Spectrum = fftshift(fft(Mixer_Output));
figure;
plot(f_KHz, abs(Mixer_Spectrum));
title('Figure 3: Output of the Mixer');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-350 350]);
grid on;
disp('Part 2 Complete.');


%% Part 3: The IF Stage
% Goal: Filter the signal at the fixed Intermediate Frequency (27.5 KHz).
disp('-----------------------------------');
disp('Starting IF Stage...');

% --- 1. IF Filter Design ---
% This filter is FIXED at 27.5 KHz. It does not change.
Fc_IF = 27500;   % Center Frequency =5 27.5 KHz
BW_IF = 20000;   % Bandwidth = 20 KHz

% Design Butterworth Band-Pass Filter (4th Order)
Wn_IF = [(Fc_IF - BW_IF/2)/(Fs_new/2), (Fc_IF + BW_IF/2)/(Fs_new/2)];
[b_if, a_if] = butter(4, Wn_IF, 'bandpass');

% --- 2. Apply the Filter ---
% Filter the output of the Mixer from the previous step
IF_Output = filter(b_if, a_if, Mixer_Output);

% --- 3. Plot IF Output (Figure 4) ---
disp('Plotting IF Output...');
IF_Spectrum = fftshift(fft(IF_Output));
figure;
plot(f_KHz, abs(IF_Spectrum));
title('Figure 4: Output of the IF Filter (Centered at 27.5 KHz)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-100 100]); % Zoom in to see the clean signal at 27.5 KHz
grid on;
disp('Part 3 Complete.');


%% Part 4: The Baseband Demodulator
% Goal: Recover the original audio from the IF signal.
disp('-----------------------------------');
disp('Starting Baseband Detection...');

% --- 1. Baseband Mixing (Coherent Detection) ---
% We mix the IF output with a carrier at exactly the IF frequency (27.5 KHz)
% Formula: Output = IF_Signal * cos(w_IF * t)
w_IF = 2 * pi * IF_Freq; % 2 * pi * 27500
demod_carrier = cos(w_IF * t)';
```

```
% Mix
Baseband_Mixer_Output = IF_Output .* demod_carrier;

% --- 2. Plot Mixer Output (Figure 5) ---
disp('Plotting Baseband Mixer Output...');
BB_Mixer_Spectrum = fftshift(fft(Baseband_Mixer_Output));
figure;
plot(f_KHz, abs(BB_Mixer_Spectrum));
title('Figure 5: Output of the Baseband Mixer (Before LPF)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-100 100]); % You will see audio at 0 KHz and a copy at 55 KHz
grid on;

% --- 3. Low Pass Filter (LPF) Design ---
% We need to keep only the audio (approx 0 to 10 KHz) and reject the 55 KHz part.
Fc_LPF = 10000; % Cutoff frequency 10 KHz (Audio is typically < 10K)

% Design Butterworth Low Pass Filter (4th Order)
Wn_LPF = Fc_LPF / (Fs_new/2);
[b_lpf, a_lpf] = butter(4, Wn_LPF, 'low');

% Apply LPF
Final_Output = filter(b_lpf, a_lpf, Baseband_Mixer_Output);

% --- 4. Plot Final Output (Figure 6) ---
disp('Plotting Final Output...');
Final_Spectrum = fftshift(fft(Final_Output));
figure;
plot(f_KHz, abs(Final_Spectrum));
title('Figure 6: Output of the LPF (Recovered Audio)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-20 20]); % Zoom in close to 0 to see the audio spectrum
grid on;

% --- 5. Prepare the Sound (But wait for command to play) ---
disp('Demodulation Complete.');
% Downsample back to original rate to hear it correctly
Sound_Final = resample(Final_Output, 1, interp_factor);
% Normalize volume to avoid clipping/distortion
Sound_Final = Sound_Final / max(abs(Sound_Final));

% NOTE: Auto-playback is disabled here. See Section 7 below.
% sound(Sound_Final, fs_orig);


%% Part 5: Performance Evaluation without RF Stage
% Goal: Demonstrate "Image Frequency" interference.
% We skip the RF filter and go straight to the Mixer.
disp('-----------------------------------');
disp('Starting Part 5 (No RF Filter)...');

% 1. The Mixer (Directly on Tx_Signal)
```

```
% We use the SAME oscillator as before (tuned for 100 KHz station)
% LO Frequency = 127.5 KHz
oscillator = cos(2*pi*LO_Freq*t)';

% MIXING directly without filtering!
Mixer_NoRF = Tx_Signal .* oscillator;

% --- Figure 7: Output of the RF mixer (No RF Filter) ---
Mixer_NoRF_Spectrum = fftshift(fft(Mixer_NoRF));
figure;
plot(f_KHz, abs(Mixer_NoRF_Spectrum));
title('Figure 7: Output of the RF Mixer (No RF Filter)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-350 350]);
grid on;

% 2. The IF Filter (Same as before)
% Filter centered at 27.5 KHz
IF_Output_NoRF = filter(b_if, a_if, Mixer_NoRF);

% --- Figure 8: Output of the IF filter (No RF Filter) ---
IF_NoRF_Spectrum = fftshift(fft(IF_Output_NoRF));
figure;
plot(f_KHz, abs(IF_NoRF_Spectrum));
title('Figure 8: Output of the IF Filter (No RF Filter)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-100 100]);
grid on;

% 3. Baseband Detection (Same as before)
% Mix with 27.5 KHz carrier
Baseband_Mixer_NoRF = IF_Output_NoRF .* demod_carrier;

% --- Figure 9: Output of the IF mixer before LPF (No RF Filter) ---
BB_Mixer_NoRF_Spectrum = fftshift(fft(Baseband_Mixer_NoRF));
figure;
plot(f_KHz, abs(BB_Mixer_NoRF_Spectrum));
title('Figure 9: Baseband Mixer Output (No RF Filter)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-100 100]);
grid on;

% 4. Low Pass Filter (Same as before)
Final_Output_NoRF = filter(b_lpf, a_lpf, Baseband_Mixer_NoRF);

% --- Figure 10: Output of the LPF (No RF Filter) ---
Final_NoRF_Spectrum = fftshift(fft(Final_Output_NoRF));
figure;
plot(f_KHz, abs(Final_NoRF_Spectrum));
title('Figure 10: Output of the LPF (No RF Filter)');
xlabel('Frequency (KHz)');
ylabel('Magnitude');
```

```
xlim([-20 20]);
grid on;

% 5. Prepare the Distorted Sound
disp('No-RF Simulation Complete.');
Sound_NoRF = resample(Final_Output_NoRF, 1, interp_factor);
Sound_NoRF = Sound_NoRF / max(abs(Sound_NoRF));

% NOTE: Auto-playback disabled.
% sound(Sound_NoRF, fs_orig);


%% Part 6: Frequency Offset Simulation
% Goal: Simulate a frequency offset.
disp('-----------------------------------');
disp('Starting Part 6: Offset Simulation...');

% 1. Define the Offset
offset = 100; % Set to 100 Hz

% 2. Create the mis-tuned oscillator
% The receiver thinks it's 27.5 KHz, but we add the error.
w_demod_offset = 2 * pi * (IF_Freq + offset);
demod_carrier_offset = cos(w_demod_offset * t)';

% 3. Mix
Mixer_Offset = IF_Output .* demod_carrier_offset;

% 4. Filter with the same LPF from Part 4
Output_Offset = filter(b_lpf, a_lpf, Mixer_Offset);

% 5. Plot Spectrum (Zoomed in very close)
Spec_Offset = fftshift(fft(Output_Offset));
figure;
plot(f_KHz, abs(Spec_Offset));
title(['Figure 11: Output with ' num2str(offset/1000) ' KHz Frequency Offset']);
xlabel('Frequency (KHz)');
ylabel('Magnitude');
xlim([-5 5]); % Zoom in close to see the small shift
grid on;

% 6. Prepare Sound
disp('Offset Simulation Complete.');
Sound_Offset = resample(Output_Offset, 1, interp_factor);
Sound_Offset = Sound_Offset / max(abs(Sound_Offset));

% NOTE: Auto-playback disabled.
% sound(Sound_Offset, fs_orig);


%% Part 7: Interactive Playback Menu
% =========================================================
% This section controls the playback based on user input
% =========================================================
disp(' ');
```

```
disp('================================================');
disp('        SIMULATION COMPLETE - AUDIO MENU          ');
disp('================================================');

% 1. Play Clean Audio immediately (First time)
disp('>> Automatically playing Clean Signal (Part 4)...');
sound(Sound_Final, fs_orig);
pause(length(Sound_Final)/fs_orig + 0.5); % Wait for audio to finish

% 2. Loop for User Commands
while true
    fprintf('\n----------------------------------------\n');
    fprintf(' Choose a sound to play:\n');
    fprintf('  [4] Replay Clean Signal (Part 4)\n');
    fprintf('  [5] Play Interference Signal (Part 5)\n');
    fprintf('  [6] Play Offset Signal (Part 6)\n');
    fprintf('  [0] Exit\n');
    fprintf('----------------------------------------\n');

    choice = input('Enter choice (4, 5, 6, or 0): ');

    if isempty(choice)
        choice = -1; % Handle empty enter key
    end

    if choice == 0
        disp('Exiting...');
        break;

    elseif choice == 4
        disp('>> Playing Clean Signal...');
        sound(Sound_Final, fs_orig);

    elseif choice == 5
        disp('>> Playing Image Interference (No RF)...');
        sound(Sound_NoRF, fs_orig);

    elseif choice == 6
        disp(['>> Playing Frequency Offset (' num2str(offset) ' Hz)...']);
        sound(Sound_Offset, fs_orig);

    else
        disp('Invalid selection. Please type 4, 5, 6, or 0.');
    end
end
```