DSP FINAL PROJECT

Design of a MATLAB-Based Audio Equalizer

Abstract

An audio equalizer is a tool used to adjust the amplitude of specific frequency bands within an audio signal. This report presents the development of a digital audio equalizer using MATLAB. The implementation involves designing and applying frequency band filters using both Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filter techniques. The project demonstrates how digital signal processing can be effectively used to enhance or modify audio signals by controlling distinct frequency components.

Zeyad Essam Haridy 8936 G1

Youssef Amr Keshk 8852 G2

1- CODE:

```
% Open a file dialog box to select a .wav audio file
[filename, filepath] = uigetfile('*.wav');
if isequal(filename, 0)
    error('No file selected.');
end
fullFilePath = fullfile(filepath, filename);

% Read the selected audio file
[sig, fs] = audioread(fullFilePath);
fprintf('\nFilter bands at original rate %.1f Hz:\n', fs);

% Calculate length of the audio signal
n = length(sig);

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

% Choose mode
mode = input('1: Standard Mode     2: Custom Mode     ');

while ~isnumeric(mode) || ~isscalar(mode) || ~ismember(mode, [1, 2])
    mode = input('Invalid choice. Enter 1 (Standard) or 2 (Custom): ');
end

if (mode == 1) % Standard Mode
    k = 9; % Set 9 bands
    bands = [0.01, 200, 500, 800, 1200, 3000, 6000, 12000, 16000, 20000]; % Define
band edges

    % Initialize output matrices y (time domain) and Y (frequency domain)
    y = zeros(9, n); % 9 bands, so 9 outputs
    Y = zeros(9, n); % in freq domain

    % Initialize gain array
    gains = zeros(1, 9);

elseif (mode == 2) % Custom Mode
   k = input('Enter number of bands (5-10): ');
    while ~isnumeric(k) || ~isscalar(k) || k < 5 || k > 10 || mod(k,1) ~= 0
        k = input('Invalid. Enter integer between 5-10: ');
    end

    bands = [0.01, zeros(1, k-1)]; % Initialize with 0.01Hz
    for i = 1:k-1
        while true
            bands(i+1) = input(sprintf('Enter band %d frequency (Hz): ', i));
            if bands(i+1) <= bands(i)
                fprintf('Frequency must be > previous band (%.1f Hz)\n', bands(i));
            elseif bands(i+1) >= 20000
                fprintf('Frequency must be < 20kHz\n');
```

```
            else
                break;
            end
        end
    end
end
bands(end+1) = 20000; % Force final band to 20kHz

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

for i = 1 : k
        gains(i) = input(sprintf('Enter gain in db of band %d : ', i));
        gains(i) = 10^(gains(i) / 20); % Convert from dB to linear scale because
digital filters work with linear amplitudes
end

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

% Desired output sampling frequency for playback/saving
desired_rate = input('Enter desired sampling frequeny: ');

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

% Create test signals for filter analysis
step_input = ones(1, fs);
impulse_input = [1 zeros(1 , fs-1)];

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

% Display filter type menu
type = input( sprintf( ['1: FIR Filter (Hamming) ' ...
    '\n2: FIR Filter (Hanning) ' ...
    '\n3: FIR Filter (Blackman) ' ...
    '\n4: IIR Filter (Butterworth) ' ...
    '\n5: IIR Filter (Chebyshev I) ' ...
    '\n6: IIR Filter (Chebyshev II)' ...
    '\n7: Enter filter type (1-6):  ']) ...
    );

while ~isnumeric(type) || ~isscalar(type) || ~ismember(type, 1:6)
    type = input('Invalid. Enter a number from 1 to 6: ');
end


order = input('Enter the order of the filter (press Enter to use default): ');

% Set default values
if isempty(order)
    if ismember(type, [1 2 3])  % FIR filter types
        order = 64;
    elseif ismember(type, [4 5 6])  % IIR filter types
```

```
        order = 4;
    else
        error('Invalid filter type.');
    end
end

% Input Validation
if any(bands >= fs/2)
    error('Band frequencies cannot exceed Nyquist frequency (%.1f Hz)', fs/2);
end

% Create appropriate filter coefficients
for i = 1 : k
    switch type
        case 1
            num = fir1(order, [bands(i) bands(i+1)] / (fs/2), hamming(order + 1));
            den = 1;
        case 2
            num = fir1(order, [bands(i) bands(i+1)] / (fs/2), hanning(order + 1));
            den = 1;
        case 3
            num = fir1(order, [bands(i) bands(i+1)] / (fs/2), blackman(order + 1));
            den = 1;
        case 4
            [num,den] = butter(order, [bands(i) bands(i+1)] / (fs/2));
        case 5
            [num,den] = cheby1(order, 1, [bands(i) bands(i+1)] / (fs/2));
        case 6
            [num,den] = cheby2(order, 40, [bands(i) bands(i+1)] / (fs/2));
    end

    fprintf('Band %d: %.2f-%.2f Hz (normalized: %.4f-%.4f)\n', ...
                i, bands(i), bands(i+1), bands(i)/(fs/2), bands(i+1)/(fs/2));

    % Apply filter to audio signal and scale by gain
    if size(sig, 2) == 2
        signal = mean(sig, 2); % Mix stereo to mono
    else
        signal = sig; % Use as is for mono
    end
    y(i,:) = gains(i) * filter(num, den, signal);

    % Filter test signals to analyze filter characteristics
    step = filter(num, den, step_input);
    impulse = filter(num, den, impulse_input);

    % Compute frequency response of filter
    [H, f] = freqz(num, den, fs/2, fs);

%------------------------------------------------------------------------

    % Filter Visualization
    figure();

    % Magnitude response
```

```
    subplot(3, 2, 1);
    plot(f, abs(H));
    title('Magnitude', 'Color', 'm');
    grid on

    % Phase response
    subplot(3, 2, 2);
    plot(f, angle(H) * 180 / pi);
    title('Phase', 'Color', 'm');
    grid on;

    % Impulse response
    subplot(3, 2, 3);
    plot(impulse);
    title('Impulse Response', 'Color', 'm');
    grid on;

    % Step response
    subplot(3, 2, 4);
    plot(step);
    title('Step Response', 'Color', 'm');
    grid on;

    % Pole-zero plot
    subplot(3, 2, [5 6])
    zplane(num, den);
    title('Poles and Zeros', 'Color', 'm');
    grid on;
end

%------------------------------------------------------------------------
%------------------------------------------------------------------------

% Frequency Domain Analysis
x_axis = (-n/2 : n/2-1) * (fs/n);

for i = 1:k
    Y(i, :) = (1/fs) * fftshift(fft(y(i, :))); % Computes FFT of filtered signal
(shifted for correct frequency display)

    % Display frequency spectrum and time-domain signal
    figure();

    subplot(2, 1, 1);
    stem(x_axis, abs(Y(i, :)))
    subplot(2, 1, 2);
    plot(y(i, :))
end

% Combine all filtered bands by summing their time-domain signals.
filtered_sig = zeros(1, n);
for i = 1 : n
    filtered_sig(i) = sum(y(:, i));
end
```

```
filtered_sig = sum(y, 1); % Sum all bands along rows
filtered_sig = filtered_sig / max(abs(filtered_sig)); % Normalize to prevent clipping

%------------------------------------------------------------------------
%------------------------------------------------------------------------

SIG = (1/fs) * fftshift(fft(sig));
FILTERED_SIG = (1/fs) * fftshift(fft(filtered_sig));

% Show comparison between original and filtered signals in frequency domain
figure();

subplot(2, 1, 1);
stem(x_axis, abs(SIG))
title('Original Signal','Color','m');
xlabel('Frequency (Hz)'); ylabel('Magnitude');

subplot(2, 1, 2)
stem(x_axis, abs(FILTERED_SIG))
title('Filtered Signal','Color','m');
xlabel('Frequency (Hz)'); ylabel('Magnitude');


% Plays the filtered audio at desired sampling rate
if max(abs(filtered_sig)) < 1e-6
    warning('Filtered signal appears to be silent!');
else
    sound(filtered_sig, desired_rate);
end

% Save filtered audio
audiowrite('new_multiplied_by_4.wav', filtered_sig, desired_rate*4); % 4x desired
rate
audiowrite('new_decreased_to_half.wav', filtered_sig, desired_rate/2); % 0.5x desired
rate
```

2- Sample Runs:

2.1 IIR Filter

```
1: Standard Mode     2:Custom Mode     1
Enter gain in db of band 1 : 0
Enter gain in db of band 2 : -3
Enter gain in db of band 3 : 6
Enter gain in db of band 4 : 0
Enter gain in db of band 5 : 3
Enter gain in db of band 6 : -6
Enter gain in db of band 7 : 0
Enter gain in db of band 8 : 3
Enter gain in db of band 9 : -3
Enter desired sampling frequeny: 44100
1: FIR Filter (Hamming)
2: FIR Filter (Hanning)
3: FIR Filter (Blackman)
4: IIR Filter (Butterworth)
5: IIR Filter (Chebyshev I)
6: IIR Filter (Chebyshev II)
4
Enter the order of the filter (press Enter to use default):
Band 1: 0.01-200.00 Hz (normalized: 0.0000-0.0091)
Band 2: 200.00-500.00 Hz (normalized: 0.0091-0.0227)
Band 3: 500.00-800.00 Hz (normalized: 0.0227-0.0363)
Band 4: 800.00-1200.00 Hz (normalized: 0.0363-0.0544)
Band 5: 1200.00-3000.00 Hz (normalized: 0.0544-0.1361)
Band 6: 3000.00-6000.00 Hz (normalized: 0.1361-0.2721)
Band 7: 6000.00-12000.00 Hz (normalized: 0.2721-0.5442)
Band 8: 12000.00-16000.00 Hz (normalized: 0.5442-0.7256)
Band 9: 16000.00-20000.00 Hz (normalized: 0.7256-0.9070)
```

Figure 2.1: Command window text-based UI

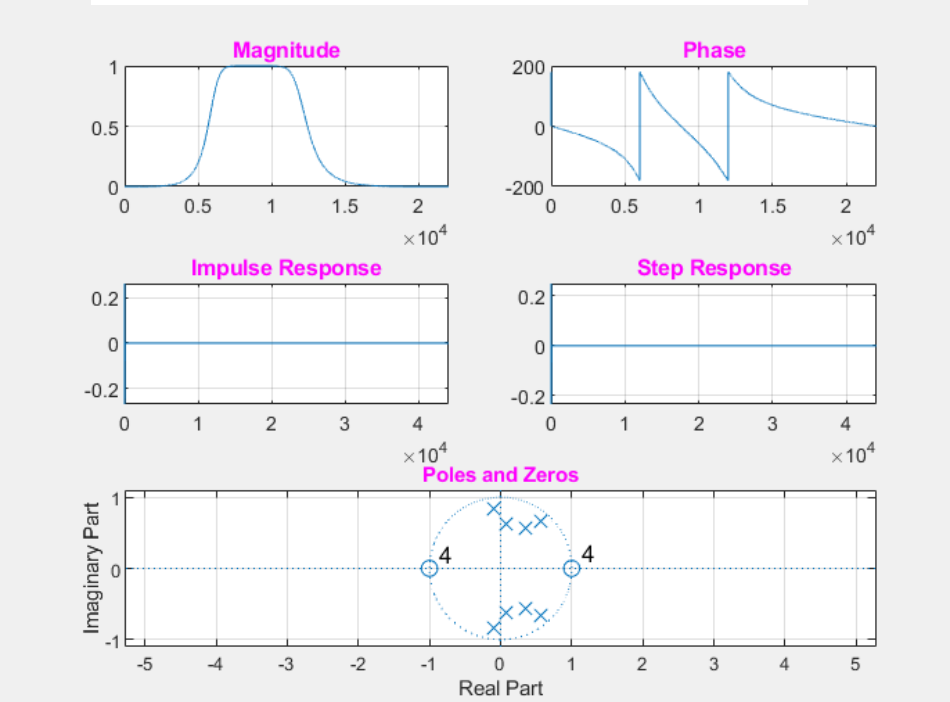

Figure 2.2: Sample of filter responses

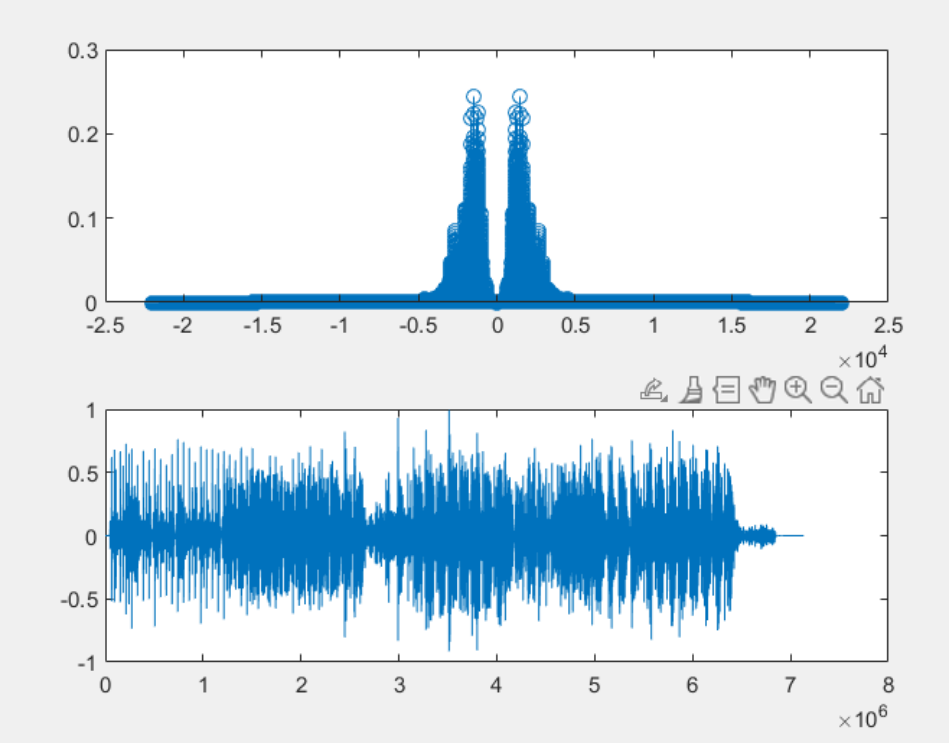

Figure 2.3: Frequency spectrum and time domain signal

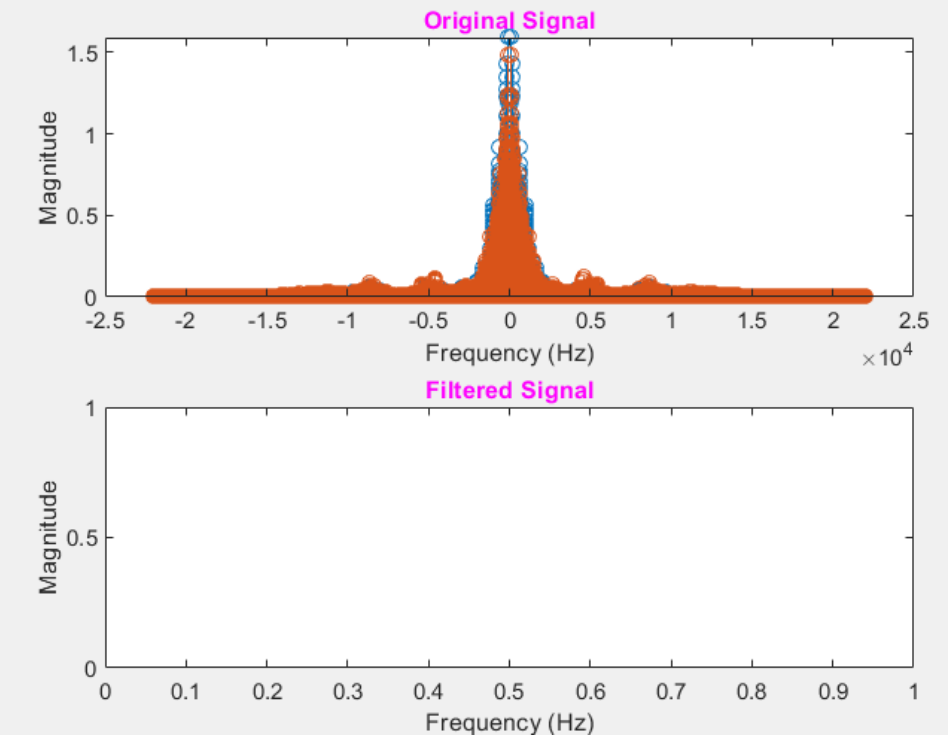

Figure 2.4: Original to filtered signals

2.2 FIR Filter

```
Filter bands at original rate 44100.0 Hz:
1: Standard Mode     2:Custom Mode     2
Enter number of bands between 5 and 10: 5
Enter bands frequency 1 : 500
Enter bands frequency 2 : 2000
Enter bands frequency 3 : 8000
Enter bands frequency 4 : 15000
Enter gain in db of band 1 : 6
Enter gain in db of band 2 : 0
Enter gain in db of band 3 : -6
Enter gain in db of band 4 : 3
Enter gain in db of band 5 : 0
Enter desired sampling frequeny: 22050
1: FIR Filter (Hamming)
2: FIR Filter (Hanning)
3: FIR Filter (Blackman)
4: IIR Filter (Butterworth)
5: IIR Filter (Chebyshev I)
6: IIR Filter (Chebyshev II)
1
Enter the order of the filter (press Enter to use default): 5
Band 1: 0.01-500.00 Hz (normalized: 0.0000-0.0227)
Band 2: 500.00-2000.00 Hz (normalized: 0.0227-0.0907)
Band 3: 2000.00-8000.00 Hz (normalized: 0.0907-0.3628)
Band 4: 8000.00-15000.00 Hz (normalized: 0.3628-0.6803)
Band 5: 15000.00-20000.00 Hz (normalized: 0.6803-0.9070)
```

Figure 2.5: Command window text-based UI

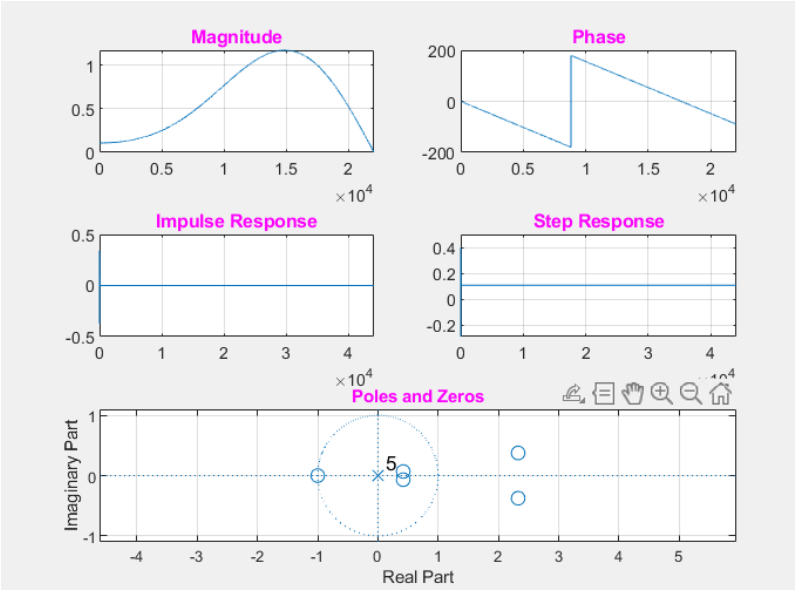

Figure 2.6: Sample of filter responses

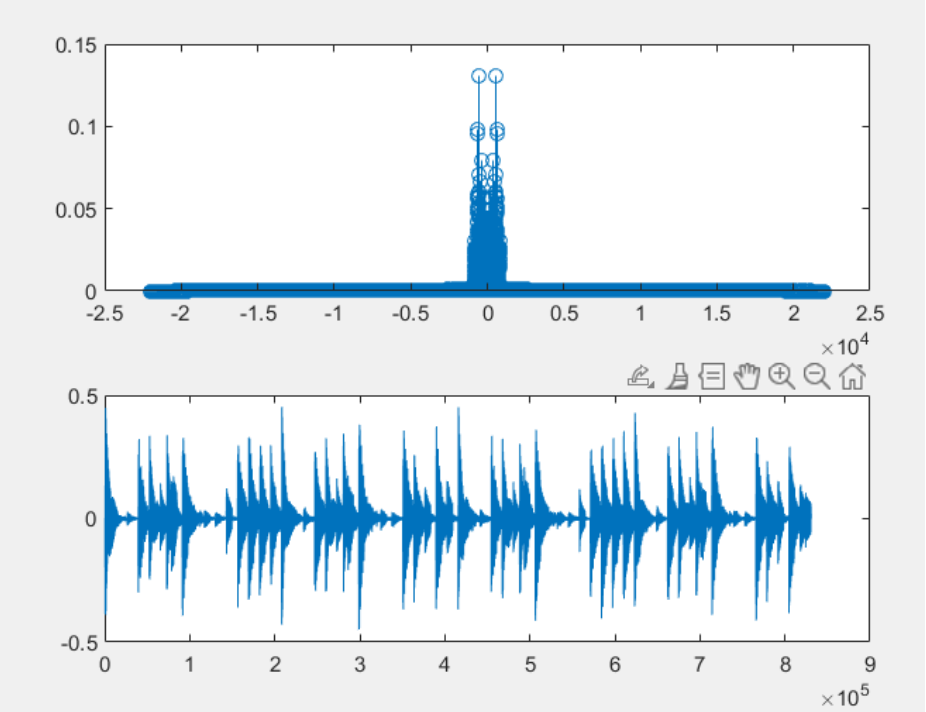

Figure 2.7: Frequency spectrum and time domain signal

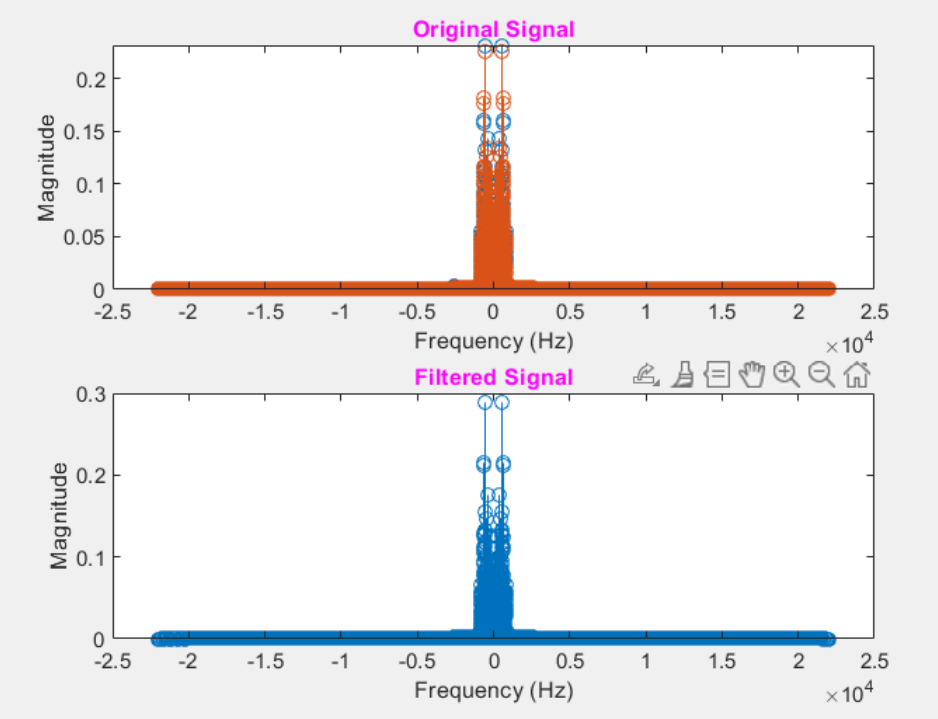

Figure 2.8: Original to filtered signal

3- Analysis of Each Filter and Figures of all Signals in Time and Frequency Domains:

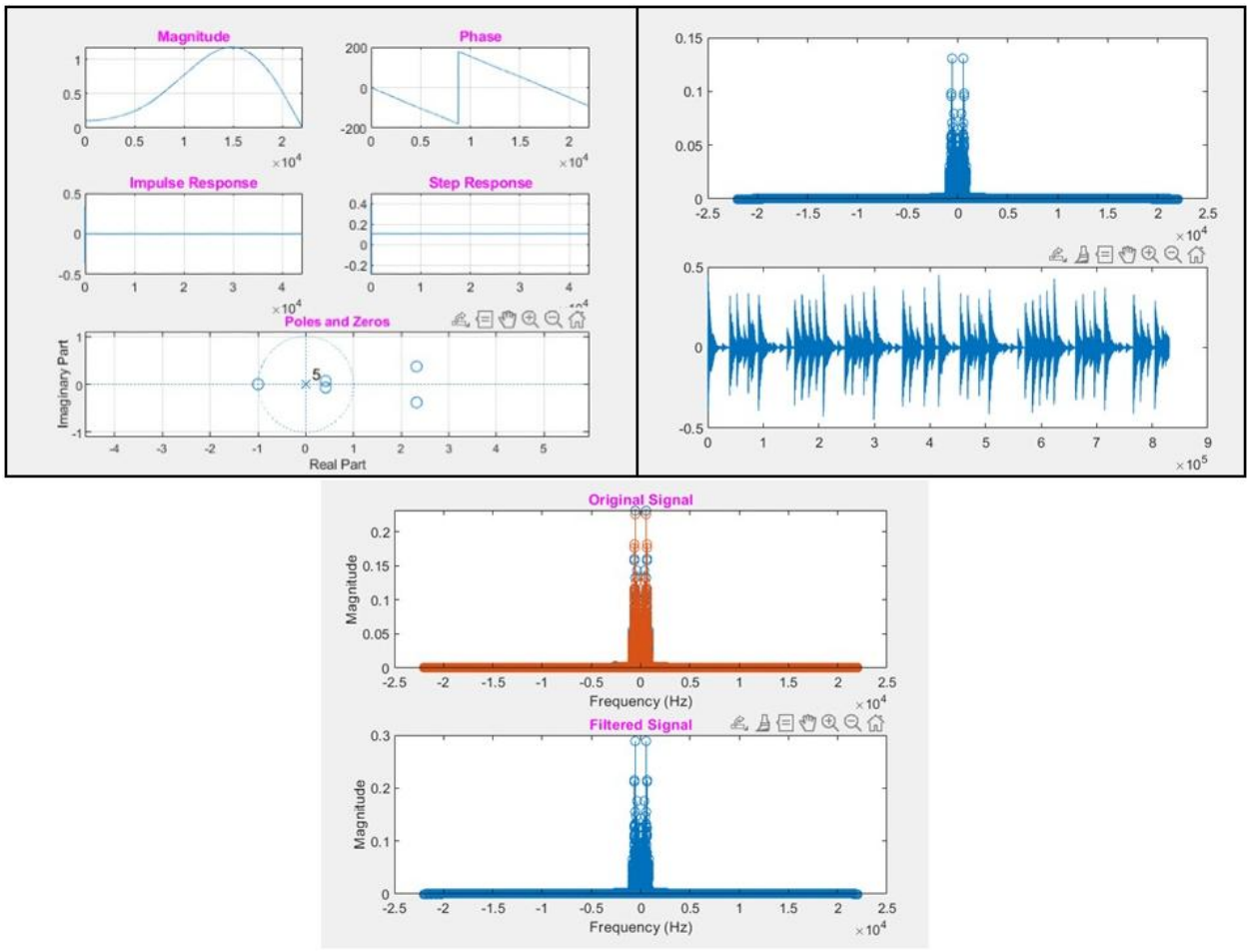

Figure 3.1: Hamming FIR Filter

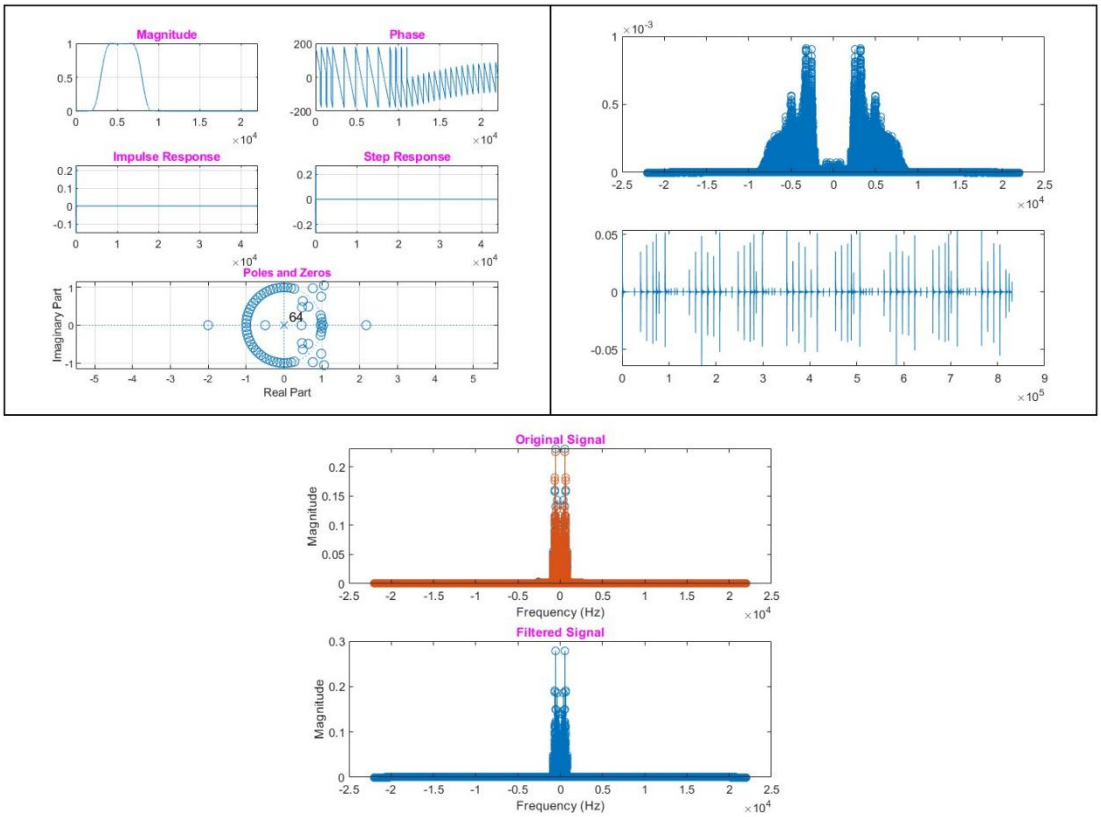

Figure 3.2: Hanning FIR Filter

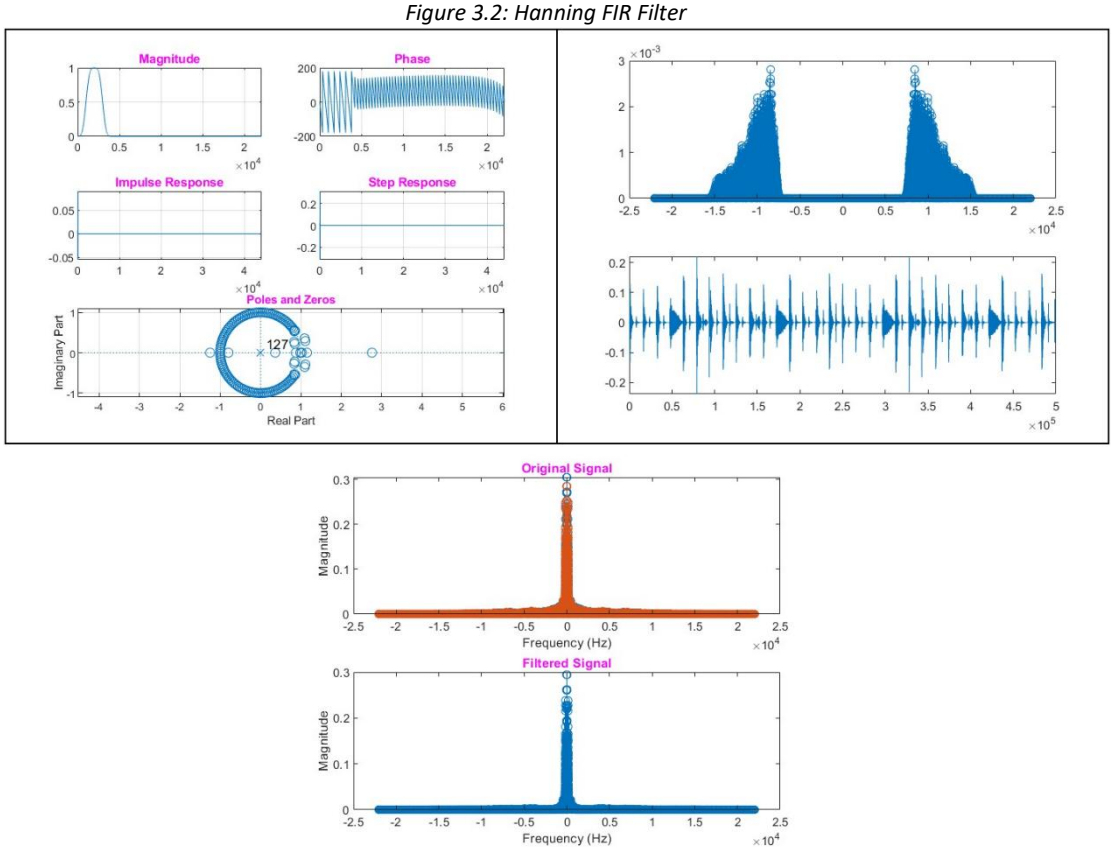

Figure 3.3: Blackman FIR Filter

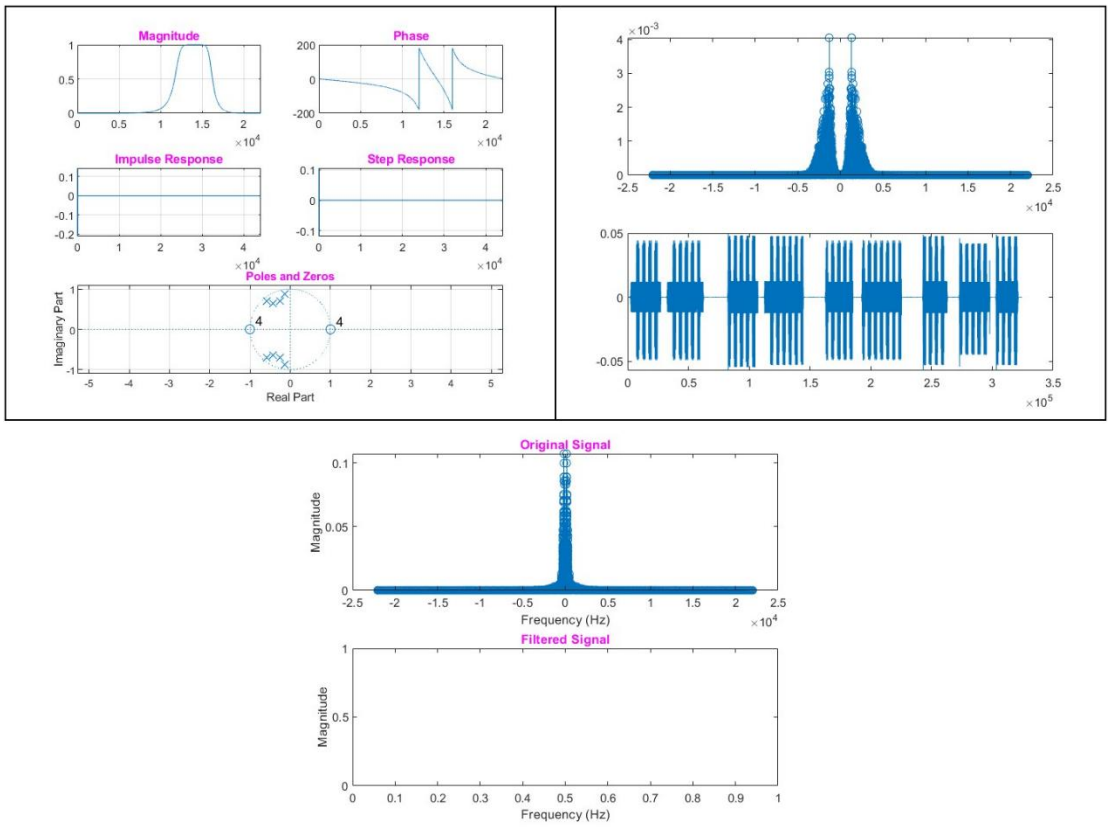

Figure 3.4: Butterworth IIR Filter

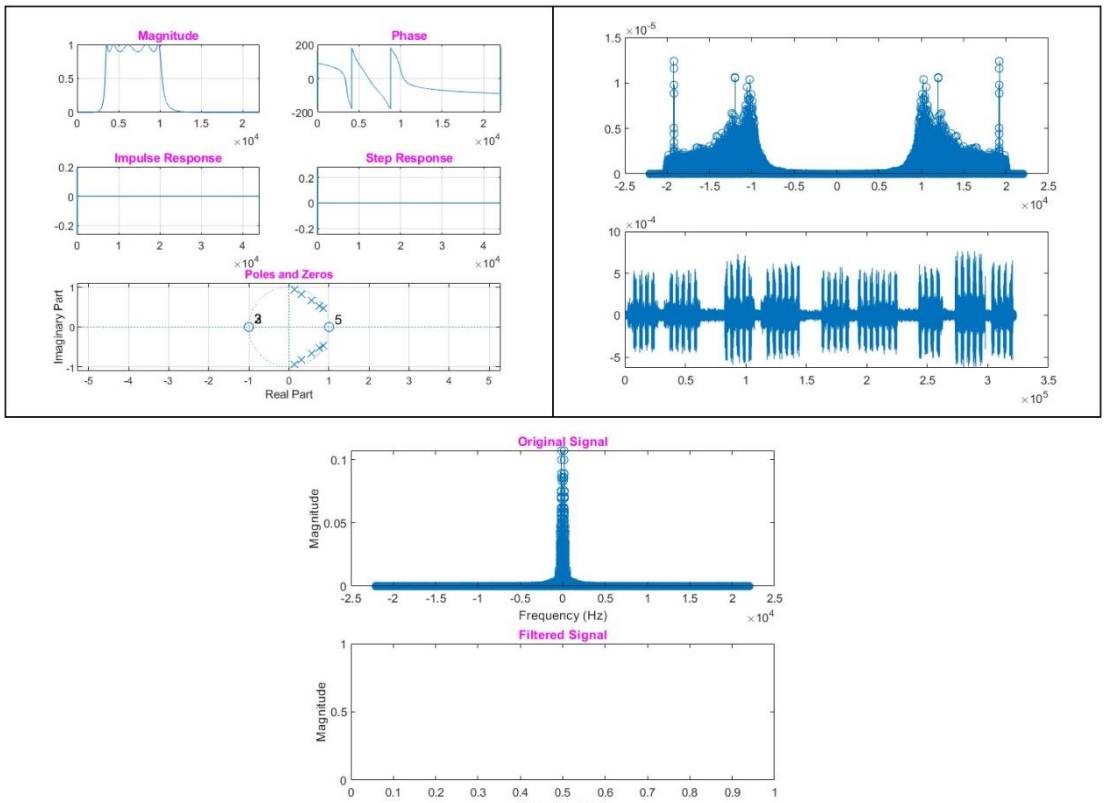

Figure 3.5: Chebyshev-I IIR Filter

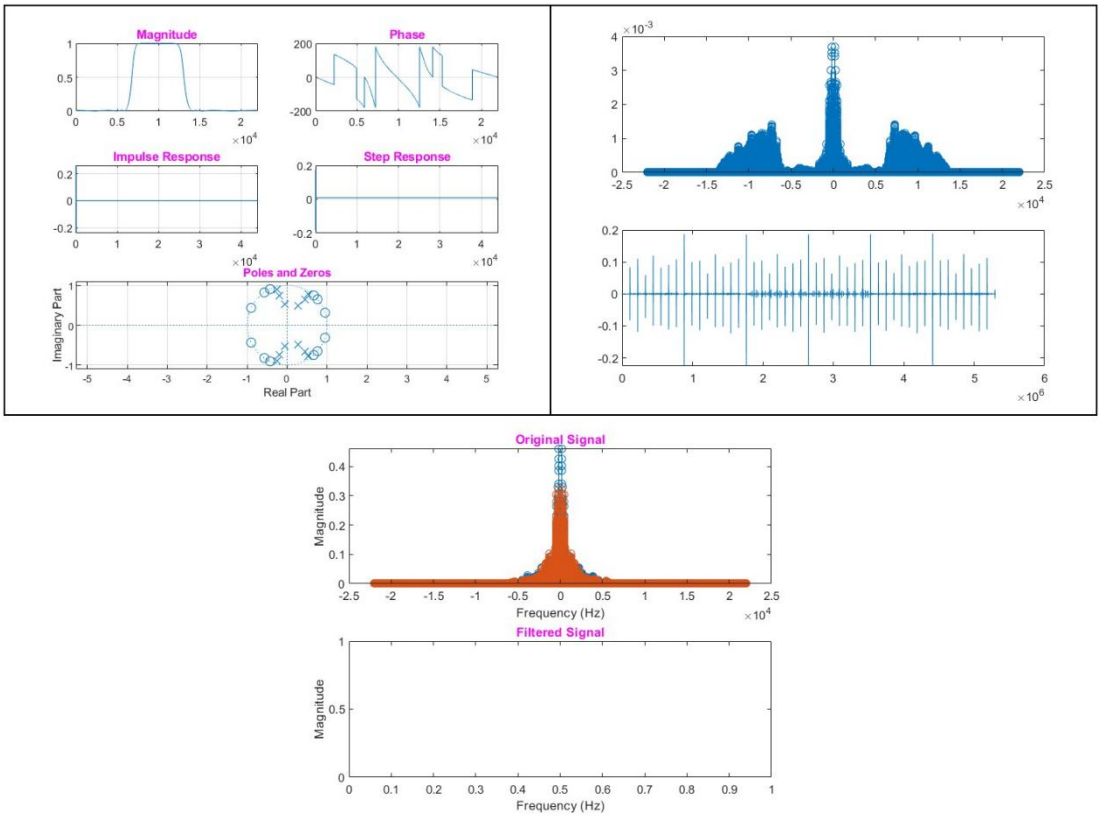

Figure 3.6: Chebyshev -II IIR Filter