Table of Contents

Table of Contents **1**

List of Figures **2**

Design Prompt **3**

Introduction **4**

Generation of White Noise Signal **5**

Input Bandpass Filter **7**

Envelope Detector Circuit **8**

Output Bandpass Filter **9**

Simulation Results **12**

Baseband Audio Signal 12

Up-sampled Baseband Audio Signal 13

Amplitude Modulated Signal 14

Bandpass Filter 14

Diode Output 15

Envelope Detector Output 16

Demodulator Output 17

Conclusion **17**

List of Figures **19**

Circuit Schematic 20

Appendix **19**

Circuit Schematic 20

MATLAB Code 21

Datasheets 32

List of Figures

Figure 1 – Envelope Detector Block Diagram 4

Figure 2 – White Noise Generation Code Snippet 5

Figure 3.1 – White Noise Theoretical and Empirical Probability Distributions 5

Figure 3.2 – White Noise Signal plotted in the time domain 6

Figure 4 – AM3235 Frequency Response 7

Figure 5 – Envelope Detector Schematic 9

Figure 6.1 – OPAx134 Low-pass Filter Schematic 9

Figure 6.2 – OPAx134 Low-pass Filter Detailed Design Procedure 10

Figure 6.3 – OPAx134 Low-pass Frequency Response 10

Figure 7 – OPAx134 Bandpass filter Schematic 11

Figure 8.1 – Unmodulated Baseband Music Signal in the Time Domain and Frequency Domain (44KHz Sampling Rate) 12

Figure 8.2 – Unmodulated Baseband Music Signal in the Time Domain and Frequency Domain (4.41MHz Sampling Rate) 13

Figure 9.1 – Amplitude Modulated Music Signal Before Corruption by Additive White Noise (1MHz Carrier Frequency) 14

Figure 9.2 – Amplitude Modulated Music Signal After Corruption by Additive White Noise (1MHz Carrier Frequency) 14

Figure 9.3 – Output of AM3235 Bandpass Filter 15

Figure 10.1 – Diode Output Waveform 16

Figure 10.2 – Envelope Detector Output Waveform 17

Figure 11 – Output of OP2134 and Final Output of Demodulator– Envelope Detector Output Waveform 18

Figure 12.1 – Unmodulated Baseband Signal Comparison with Demodulator Output 19

Figure 12.2 –Demodulator Output compared with Unmodulated Baseband Signal 19

Circuit Schematic 21

Design Problem 2: Envelope Detector

Design an envelope detector for demodulating an amplitude modulated signal arriving at the receiver. The signal arriving at the input of the demodulator is the amplitude modulated signal generated in Design Project #1. The signal arriving at the receiver is corrupted by additive white noise. Propagation through the communication channel can be modeled as a low pass filter.

You must model the various electrical components, such as the diode, and process the signal in the time domain. You should display the signal at the various stages of processing. Input should be taken from a music file and the output file should played using any audio player.

Due date: Dec 5.

NB: Program using MATLAB ONLY: NO TOOLBOXES OR SIMULINK

The deliverable is a written report, which should contain a description of the design, complete circuit schematic, analytic description, circuit simulation results that confirm the required objectives.

Introduction

Our approach to the design problem involves using random number generation to simulate a white noise signal that can be added to the received amplitude modulated signal that was generated in the previous design problem. Additive white noise has a uniform PSD at all frequencies, therefore, a bandpass filter at the input stage of the demodulator can be used to filter out any noise that is outside of the frequency band of interest. A peak detector circuit can then track the envelope of the filtered signal resulting an approximation of the modulating signal. A band-pass filter at the output stage can then be used to smooth the peak detector output and ac-couple the music signal to the output. Finally, an amplifier can be added to the output stage to boost the signal-to-noise ratio of the demodulated signal. If a bandpass filter is built around the amplifier, the amplifier and filter can be lumped into the same module resulting in an active bandpass filter at the output stage. A functional block diagram of our design is shown below in figure 1.

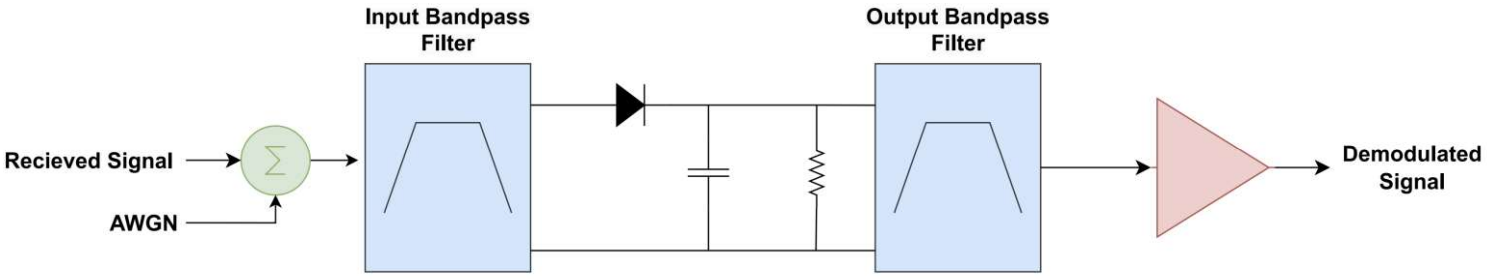


Figure 1 – Envelope Detector Block Diagram

Generation of a White Noise Signal

In order to generate a white noise signal, we use the *randn()* function in MATLAB. This function returns a random scalar drawn from a standard normal distribution. This scalar can be normalized for a given standard deviation and mean by multiplying the result by the standard deviation and adding the mean. A snippet of the code used to generate a white noise signal is shown in figure 2.

```
% Calculate the total number of samples
num_samples = round(fs * duration);

% Generate white noise with specified mean and standard deviation
white_noise = std * randn(1, num_samples) + mean;
```

Figure 2 – White Noise Generation Code Snippet

We created a histogram plot of the amplitude of the white noise signal generated by the code snippet in figure 2 by counting the number of times a given amplitude value occurs and dividing the result by the total number of samples. Figure 3.1 shows an empirical distribution of our white noise signal that is plotted alongside the theoretical distribution for white noise.

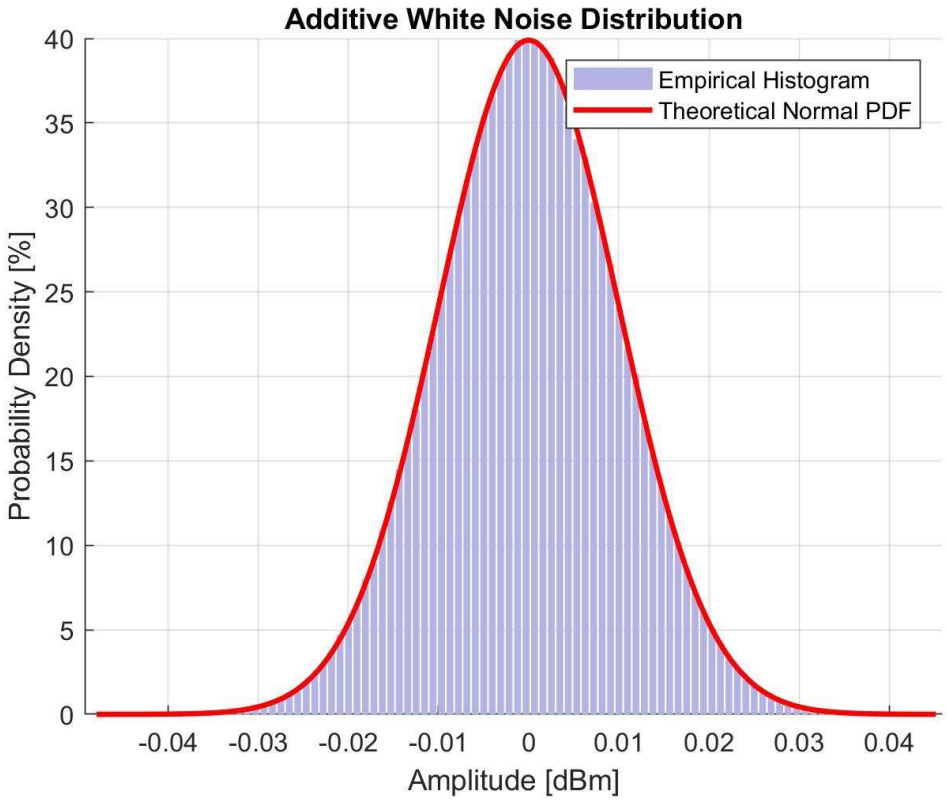


Figure 3.1 – White Noise Theoretical and Empirical Probability Distributions

Figure 3.2 shows a simulated realization of a white noise signal that is generated using the code snippet in Figure 2.

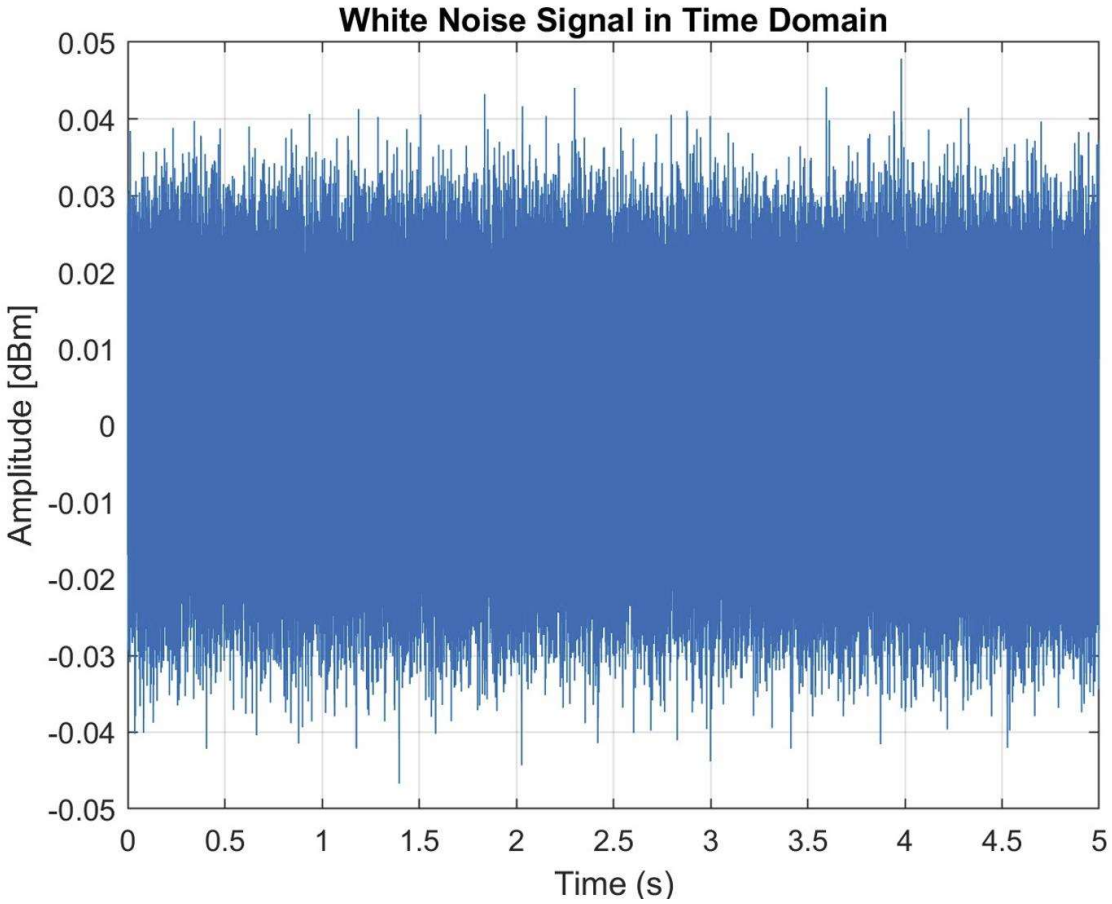


Figure 3.2 – White Noise Signal plotted in the time domain

The simulation results shown in Figure 3a and Figure 3b demonstrate that we can accurately simulate an additive white noise signal that we can use to simulate corruption of the received amplitude modulated signal.

Input Bandpass Filter

Additive white noise has a uniform PSD at all frequencies, therefore, a bandpass filter at the input stage of the demodulator can be used to filter out most of the noise that is outside of the frequency band of interest. The AMC3235 is an 8 GHz to 12 GHz passive bandpass filter that attenuates all frequencies outside of the X-band of frequencies. Figure 4 shows the frequency response of the AM3235 according to the datasheet. The filter has an attenuation of 2dB inside of the passband and a minimum attenuation of roughly 40dB outside of the passband.

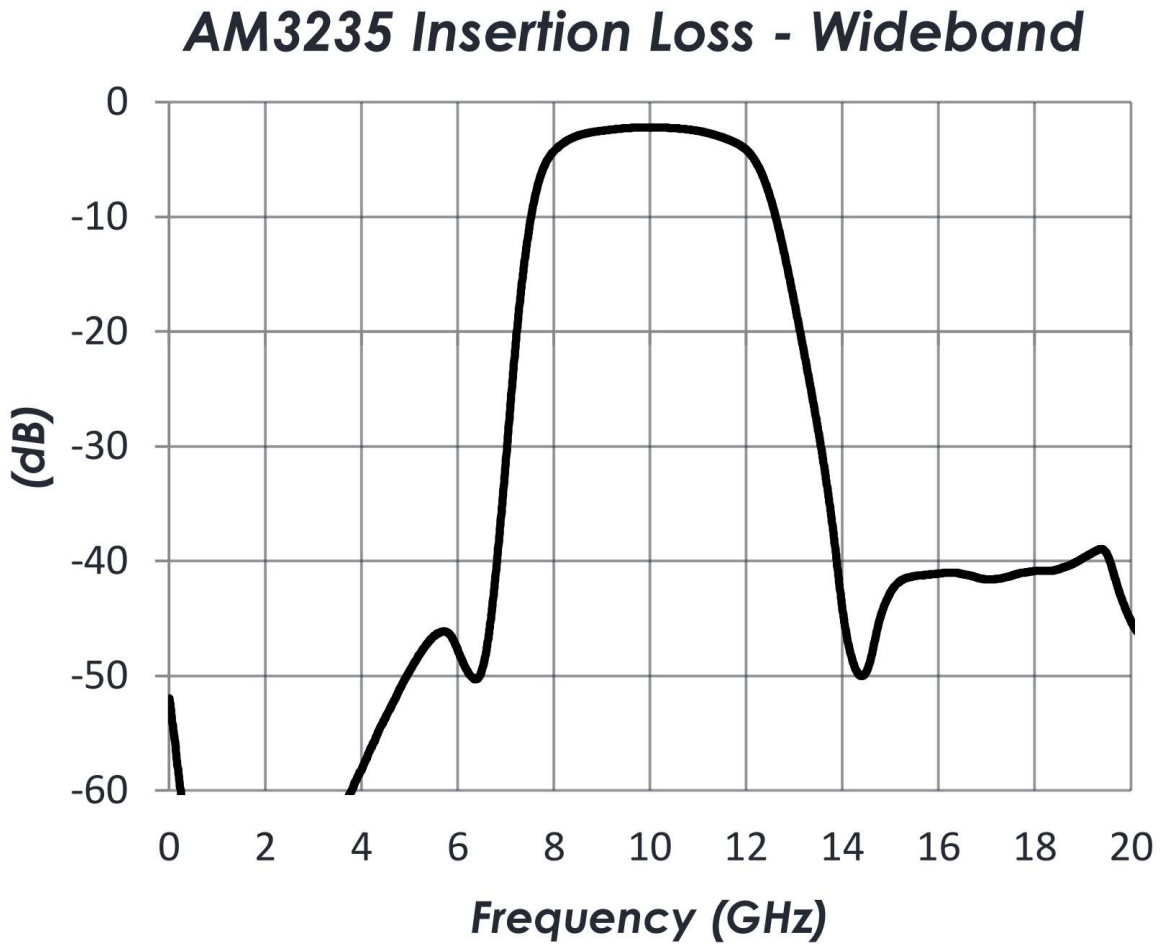


Figure 4 – AM3235 Frequency Response

This bandpass filter at the input stage will help to reduce the amount of white noise outside of the X-band that corrupts the signal. However, the signal will still be corrupted by noise that occurs inside of the X-band.

Envelope Detector Circuit

The output of the envelope detector is intended to track the envelope of the amplitude modulated signal. The circuit consists of diode followed by a parallel RC network which is functionally equivalent to a half-wave rectifier followed by a lowpass filter. The half-wave rectifier (diode) removes any negative portions of the signal resulting in a signal which closely approximates the envelope signal sampled at the carrier frequency. However, instead of the envelope being multiplied by an ideal impulse at each sampling instant, the envelope is multiplied by a positive half-cycle of the carrier sinusoid at each sampling instance. The envelope signal has an audio bandwidth, and the carrier signal is 10 GHz, therefore, each half-cycle is narrow enough that we can approximate the half-wave rectifier output as the envelope signal sampled at the carrier frequency. In the frequency domain, this is equivalent to the base-band envelope spectrum replicated at integer multiples of the carrier frequency. To ensure proper envelope detection, the RC value should satisfy the condition:

$$B < \frac{1}{2\pi RC} \ll f_c \qquad (\textbf{Equation 1.1})$$

where B is the bandwidth of the envelope signal in Hz, and f_c is the carrier frequency in Hz. This ensures that the RC circuit effectively filters out the harmonics of the envelope signal, leaving only the desired envelope of the amplitude-modulated signal. The modulated audio signal has a bandwidth of 44.1kHz centered at 10 GHz, therefore, we have chosen to use a 10Ω resistor and a $100nF$ capacitor resulting in a filter cutoff frequency of:

$$f_c = \frac{1}{2\pi(10\Omega)(100nF)} \cong 160kHz \qquad (\textbf{Equation 1.2})$$

This filter cut-off is sufficient to separate the high frequency components occurring at integer multiples of the carrier signal from the base-band envelope signal which we desire. The envelope detector circuit is shown below in figure 5.

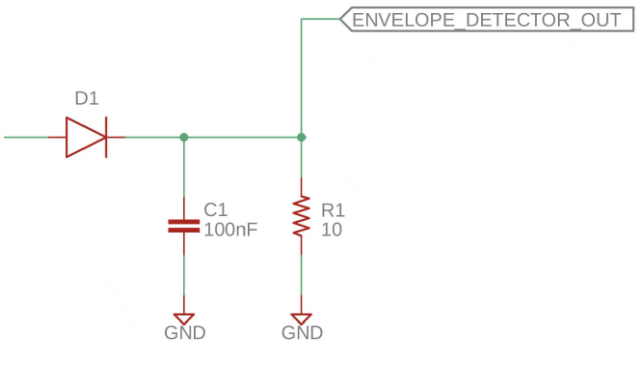


Figure 5 – Envelope Detector Schematic

Output Bandpass Filter

The audible frequency range typically ranges from 20 Hz to 20kHz, therefore, we would like to isolate this frequency band at the output stage. By using an active bandpass filter at the output stage, we can amplify the frequencies in the signal's bandwidth and attenuate any frequencies outside of the signal bandwidth, thus giving us a higher signal-to-noise ratio at the demodulator output. In order to achieve this, we will design the active bandpass filter using the OPA2134, a low-noise, ultra-low distortion operational amplifier specifically designed for audio applications. Figure 6.1 shows a 2nd-order active low-pass filter topology for the OPA2134.

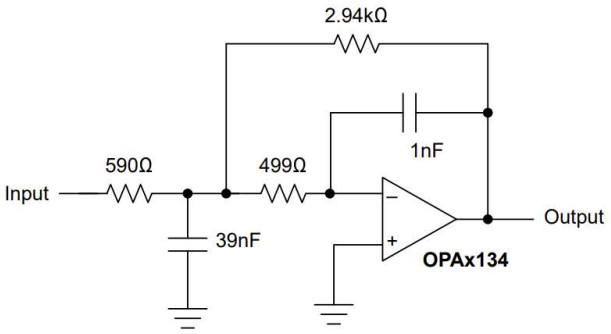


Figure 7-1. OPA134 2nd-Order, 30kHz, Low-Pass Filter Schematic

Figure 6.1 – OPAx134 Low-pass Filter Schematic

This schematic is provided on page 17 of the datasheet for the OPA2134. Additionally, figure 6.2 shows a snippet from the same page of the datasheet detailing the design procedure for this circuit topology. Figure 6.3 shows the frequency response of this filter according to page 18 of the datasheet.

7.2.2 Detailed Design Procedure

The infinite-gain multiple-feedback circuit for a low-pass network function is shown in Figure 7-1. The voltage transfer function is:

$$\frac{Output}{Input}(s) = \frac{\frac{-1}{R_1 R_3 C_2 C_5}}{s^2 + s\left(\frac{1}{C_2}\right)\left(\frac{1}{R_1} + \frac{1}{R_3} + \frac{1}{R_4}\right) + \left(\frac{1}{R_3 R_4 C_2 C_5}\right)} \quad (2)$$

This circuit produces a signal inversion. For this circuit, the gain at DC and the low-pass cutoff frequency are calculated using Equation 3 and Equation 4.

$$Gain = \frac{R_4}{R_1} \quad (3)$$

$$f_C = \frac{1}{2\pi}\sqrt{\frac{1}{R_3 R_4 C_2 C_5}} \quad (4)$$

Figure 6.2 – OPAx134 Low-pass Filter Detailed Design Procedure

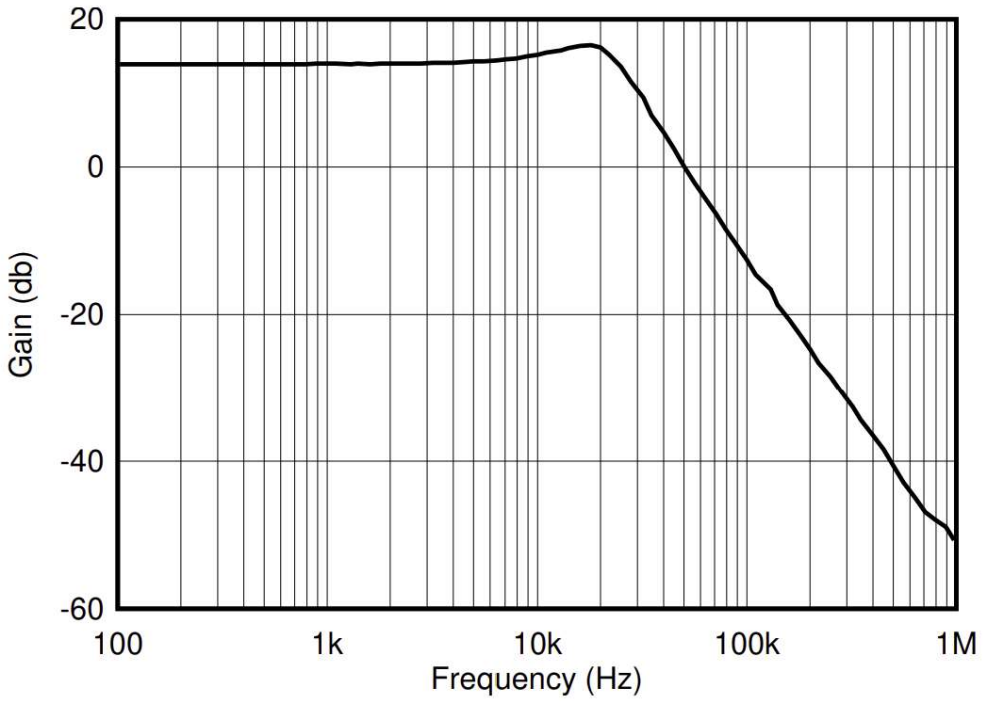


Figure 7-2. OPA134 2nd-Order, 30kHz, Low-Pass Filter Response

Figure 6.3 – OPAx134 Low-pass Frequency Response

The low-pass filter can easily be turned into a band-pass filter by adding a dc-blocking capacitor in series at the input to the filter. A schematic of an active bandpass filter using the OPA2134 is shown below in figure 7.

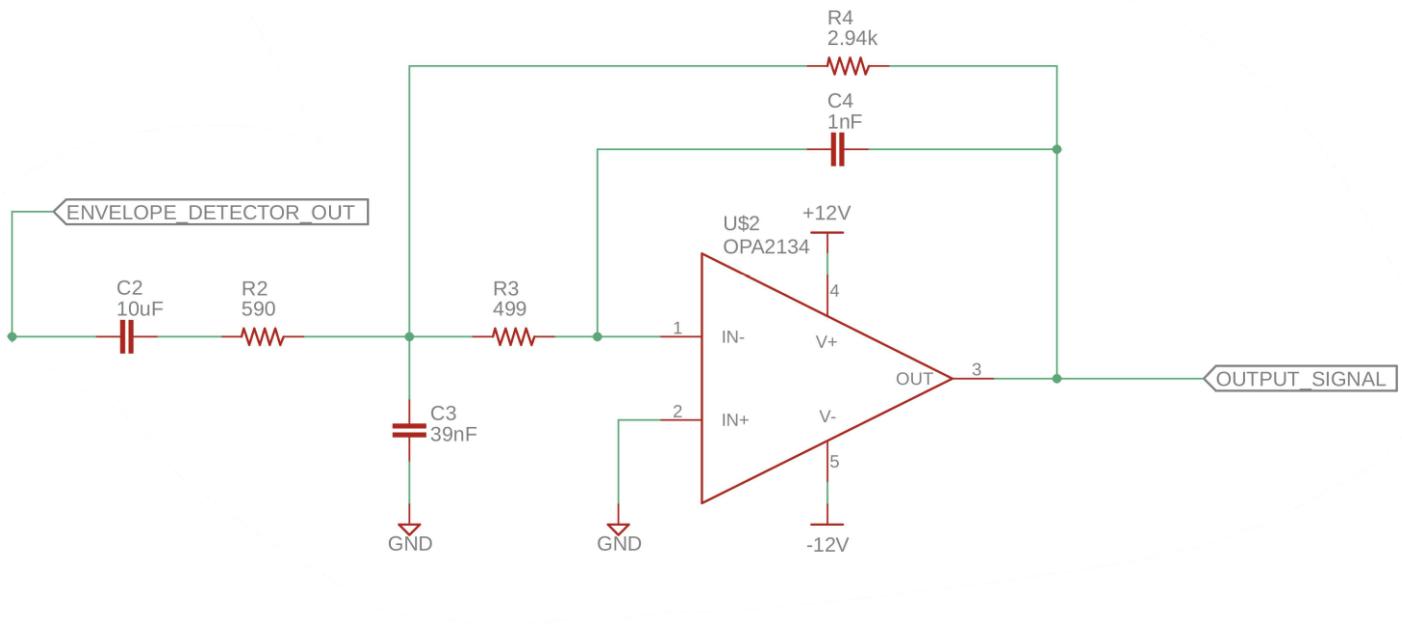


Figure 7 – OPAx134 Bandpass filter Schematic

Simulation Results

Our simulation begins by reading a 25ms audio clip from our music signal. The unmodulated baseband audio signal and its spectrum are shown in figure 8.1. The signal has most of its spectral energy between 300Hz and 2kHz with a strong spike at 679Hz.

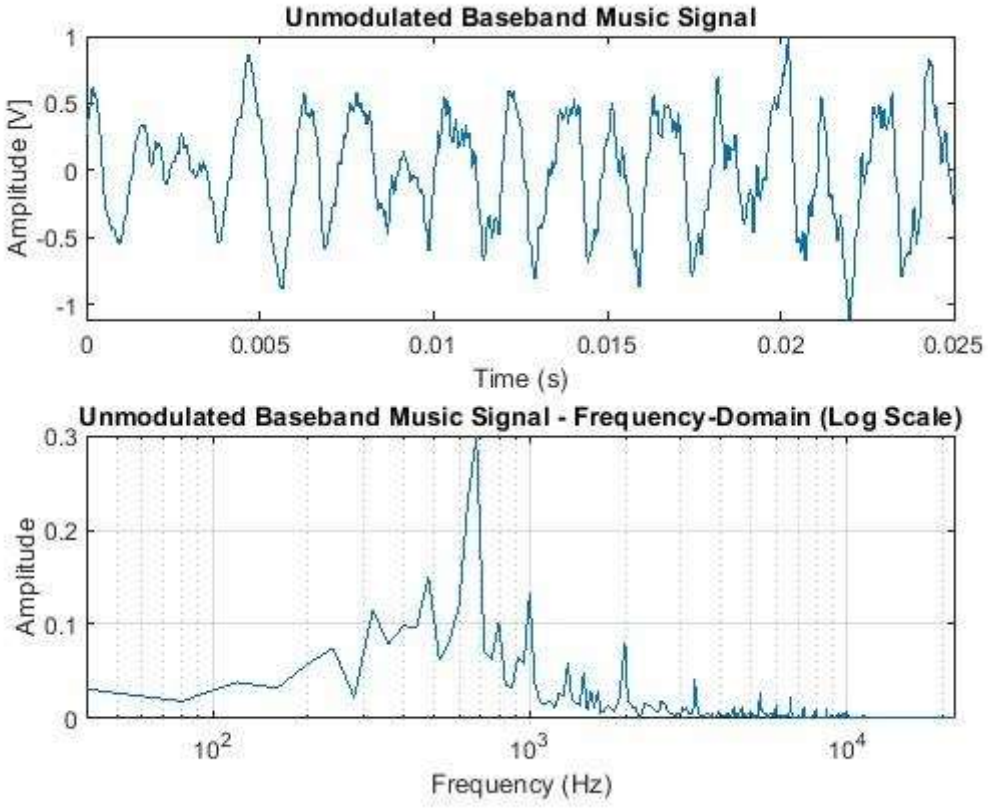


Figure 8.1 – Unmodulated Baseband Music Signal in the Time Domain and Frequency Domain (44KHz Sampling Rate)

This signal has an audio sample rate of 44.1kHz, therefore, it must be up-sampled before modulation so that the sample rate is at least twice the carrier frequency. Due to computational and memory limitations, we could not use the 10GHz carrier from the first design problem as the up-sampling process causes the computer to freeze. Therefore, we have decided to use a 1MHz carrier in our simulation because it is much larger than the audio bandwidth of roughly 20kHz. Since a 1MHz carrier frequency is much larger than the modulating bandwidth, all of the properties of amplitude modulation will remain true. In order to increase the sample rate, we must insert zeroes in between the 44.1kHz sample signal and then pass this signal through a low-pass filter by convolving this signal with a Sinc function. The lowpass filter interpolates between the samples and gives us the same signal but with a higher sample rate. The signal is shown in figure 8.2 is the music signal up-sampled by a factor of 100, yielding a new sampling rate of 4.41MHz.

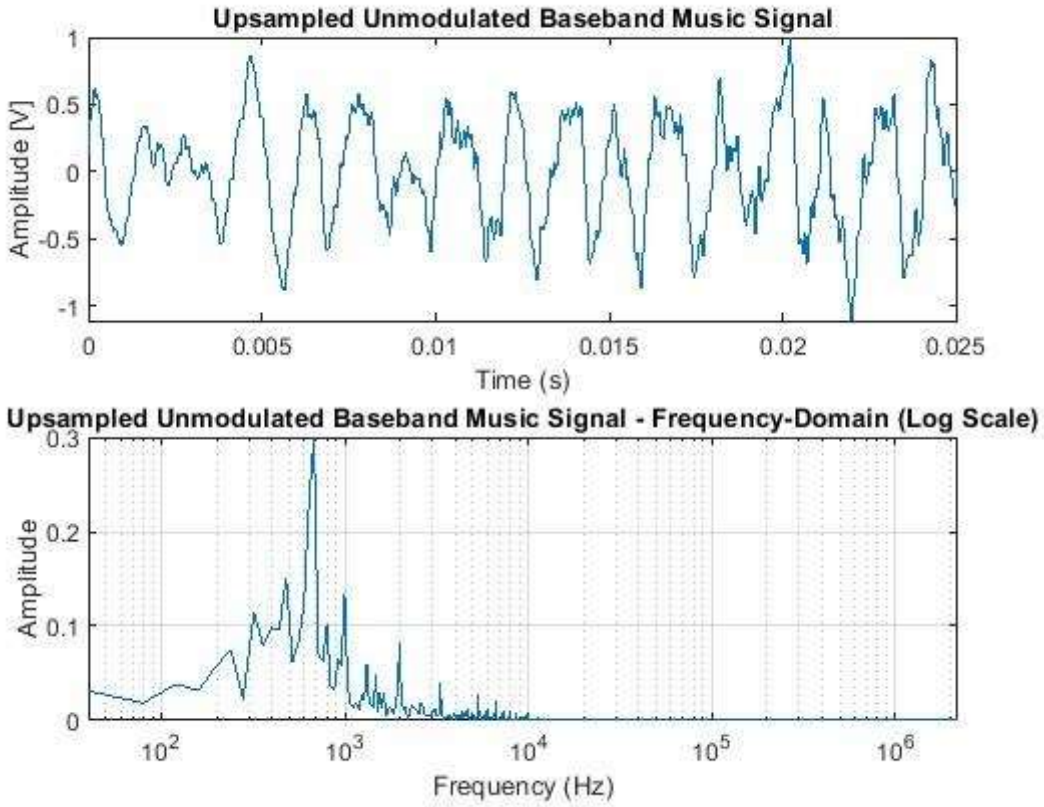


Figure 8.2 – Unmodulated Baseband Music Signal in the Time Domain and Frequency Domain (4.41MHz Sampling Rate)

The baseband signal is then modulated by a 1MHz carrier. Note that an additional carrier frequency component is added for adequate envelope detection, therefore, the signals is figures 9.1 and 9.2 are not DSB-SC signals. The amplitude modulated signal before being corrupted by additive white noise is shown in figure 9.1. The amplitude modulated signal after being corrupted by additive white noise is shown in figure 9.2.

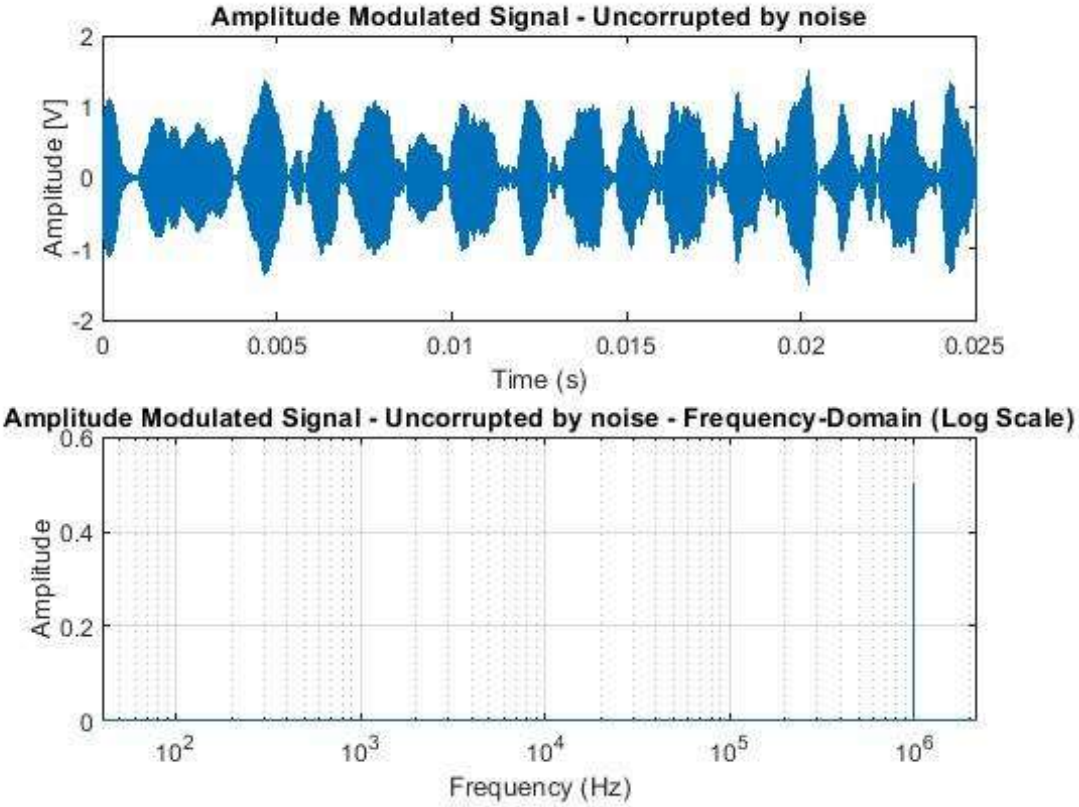


Figure 9.1 – Amplitude Modulated Music Signal Before Corruption by Additive White Noise (1MHz Carrier Frequency)

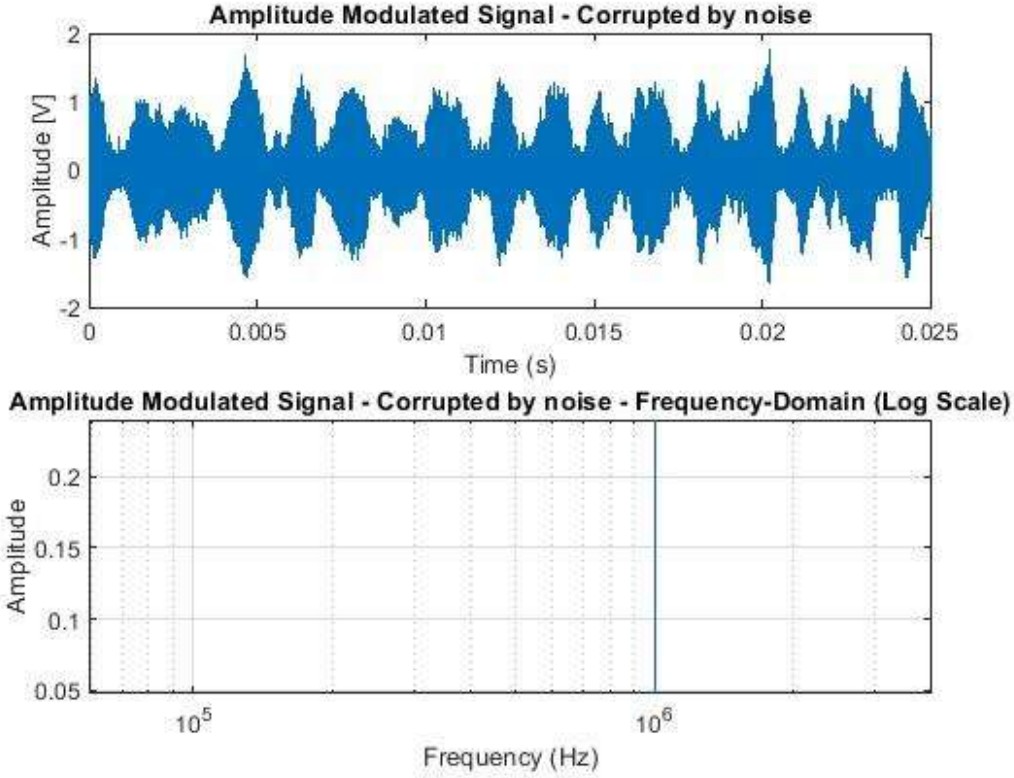


Figure 9.2 – Amplitude Modulated Music Signal After Corruption by Additive White Noise (1MHz Carrier Frequency)

Our design includes a fron-end noise filter at the input of the demodulator. The AM3235 has a center frequency of 10GHz and a bandwidth of 4GHz. Since we are using a 1MHz carrier frequency, we have decided to use a bandpass filter with a 1MHz center frequency and a 400kHz bandwidth so the ratio between center frequency and bandwidth is 0.4 in both cases. Due to not being able to use toolboxes, we have decided to simulate an ideal bandpass filter by taking a Fast-Fourier Transform (FFT) of the input signal and manually setting all frequency components outside of the passband to zero. Since we are merely aiming to verify that the presence of a bandpass filter at the input stage will reduce the amount of noise that corrupts the message signal, we can use an ideal filter as a valid approximation. The output of the AM3235 bandpass filter is shown in figure 9.3.

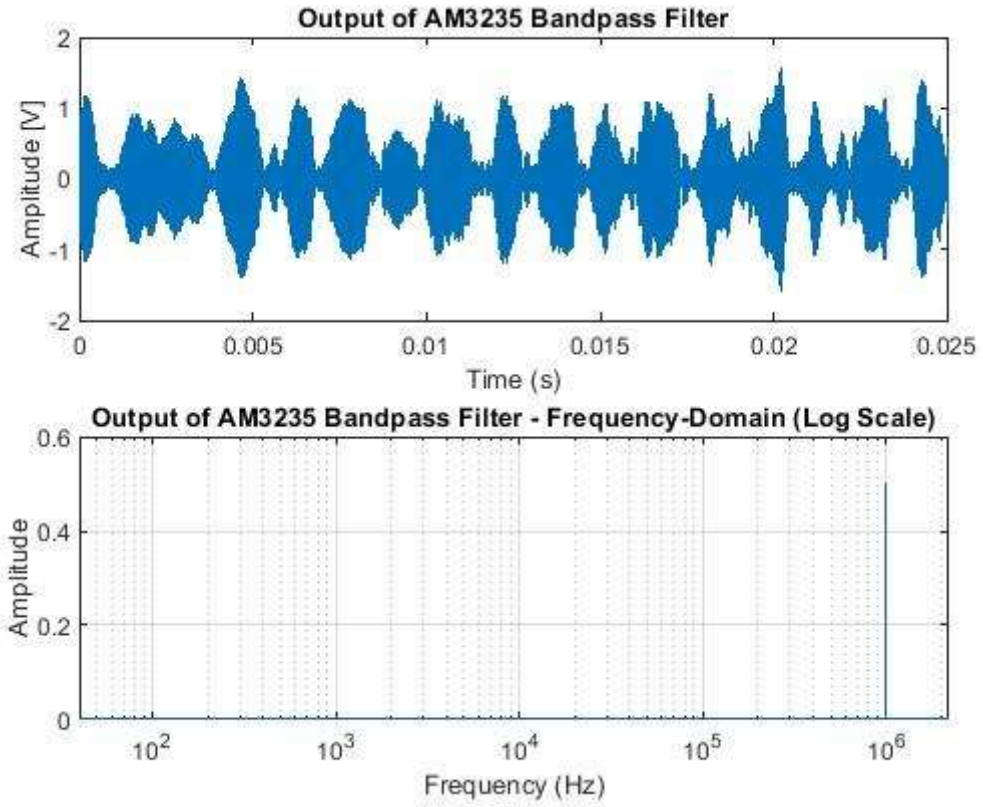


Figure 9.3 – Output of AM3235 Bandpass Filter

The output of the bandpass filter is sent to the envelope detector. We reasoned that the switching action of the diode would act as a half-wave rectifier that samples the envelope of the amplitude modulated (AM) signal. Note that the envelope of the AM signal has a dc component whereas the original music signal was purely ac. The output waveform of the diode is shown in figure 10.1. The diode creates a baseband reconstruction of the AM signal in addition to other AM terms at harmonics of the carrier frequency.

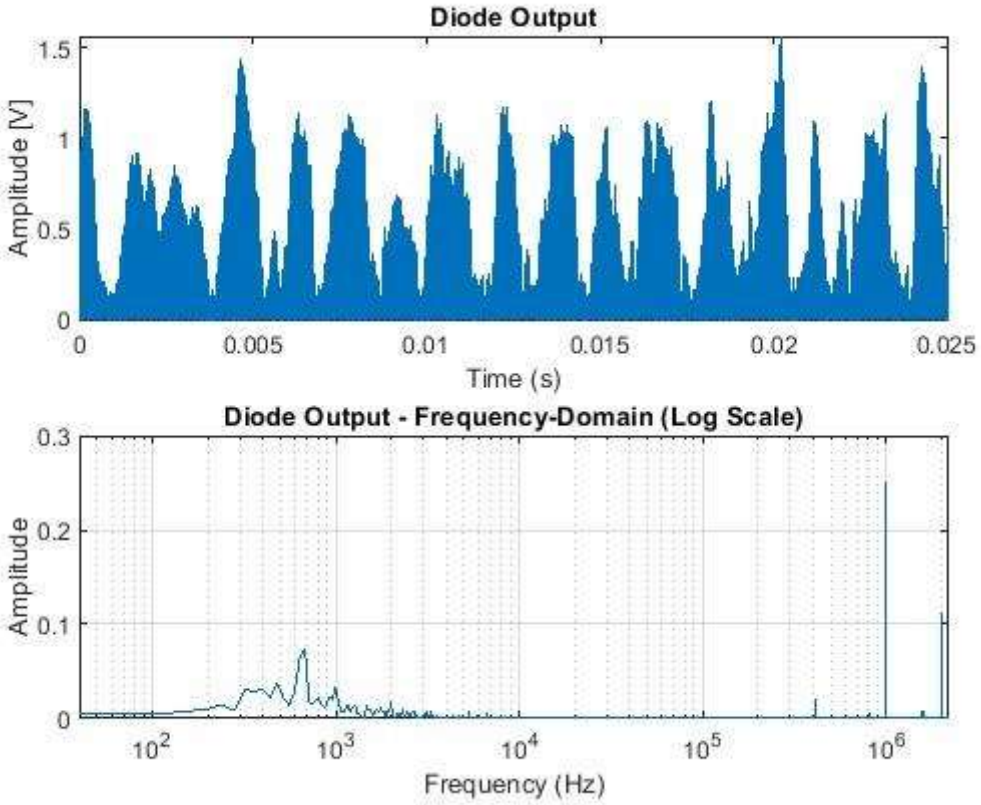


Figure 10.1 – Diode Output Waveform

The envelope detector is not complete yet. We must not simulate the parallel RC filter that is connected to the diode output. This RC filter will filter out the higher frequency AM terms and leave behind a good approximation of the envelope signal. The output waveform of the envelope detector circuit is shown in figure 10.2.

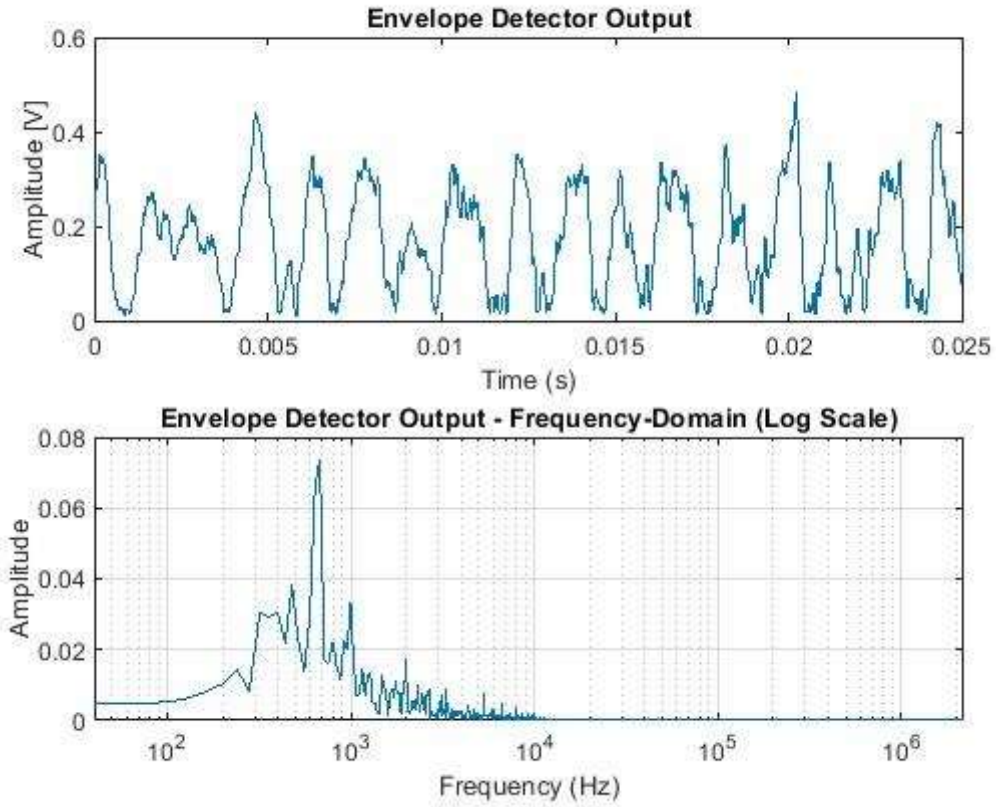


Figure 10.2 – Envelope Detector Output Waveform

The envelope signal passes through the final stage which is the OP2134 active bandpass filter. T component values in figure 7 were obtained from the design procedure outlined in figure 6.2. The filter in figure 7 has a cutoff frequency of roughly 47kHz and a gain of 5. Again, we are not looking to simulate the exact filter response, but we are more interested in the general effect that a filter will have on the envelope signal. For this reason, we are simulating an ideal lowpass filter with a cutoff frequency of 47kHz and a gain of 5. The filter is simulated by taking an FFT of the input signal and manually setting all values outside of the passband equal to zero. An inverse FFT is then used to recover the time domain signal. The dc blocking capacitor is simulated by subtracting the average value of the signal so that it has no dc component. The output waveform of the OP2134 is shown in figure 11. This is the final output of the envelope detector demodulator.

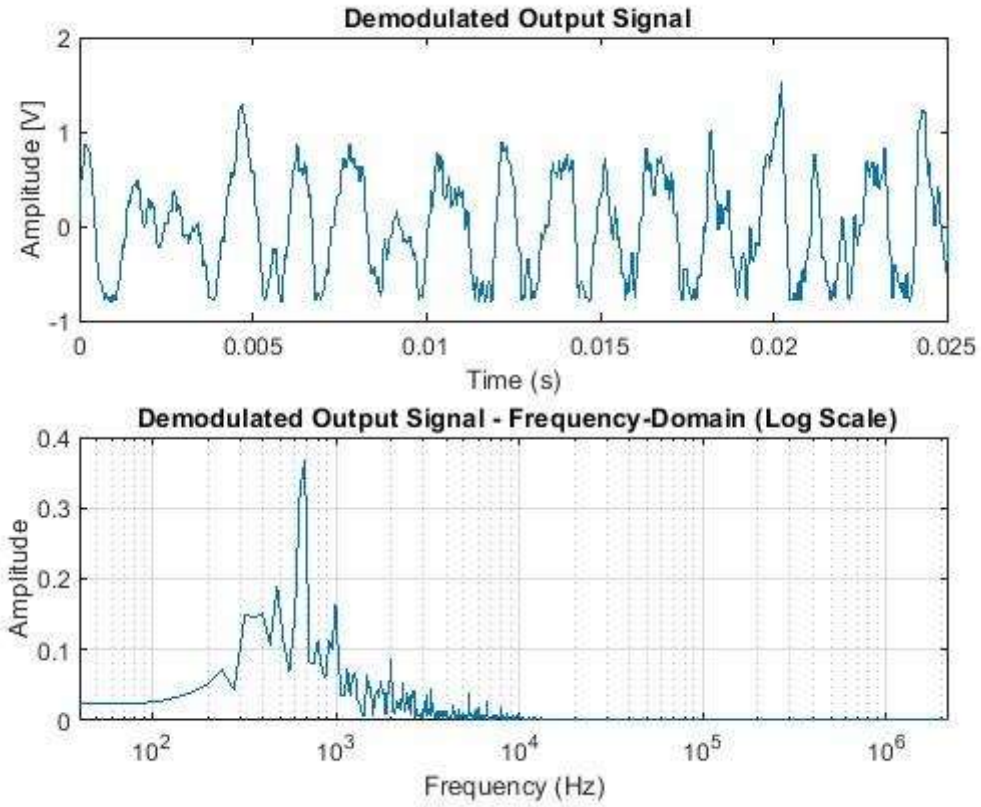


Figure 11 – Output of OP2134 and Final Output of Demodulator

Conclusion

The original audio clip read from the music file is nearly indistinguishable from the demodulated signal. Therefore, we have successfully recovered the baseband message signal from the passband amplitude modulated signal using envelope detection. Figures 12.1 and 12.2 show the unmodulated baseband audio clip and the demodulator output. Both signals have strong components in the frequency domain at 679Hz, 319 Hz, 479 Hz, 998 Hz, and 1997 Hz. Since the output signal is identical to the baseband signal, we can say that the demodulation is distortion less.

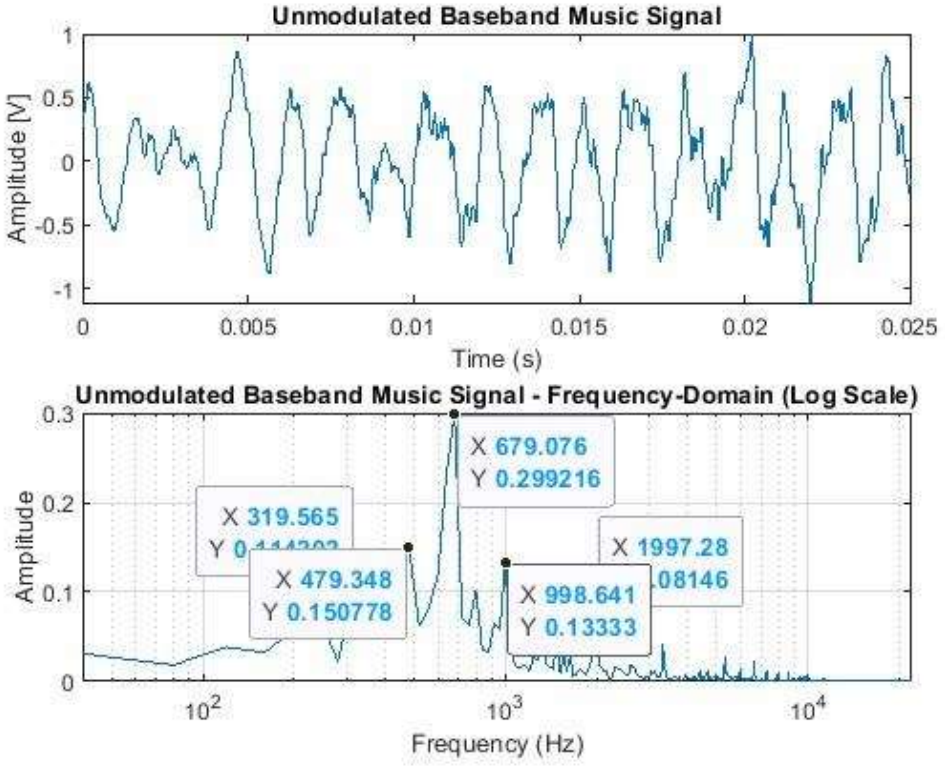


Figure 12.1 – Unmodulated Baseband Signal Comparison with Demodulator Output

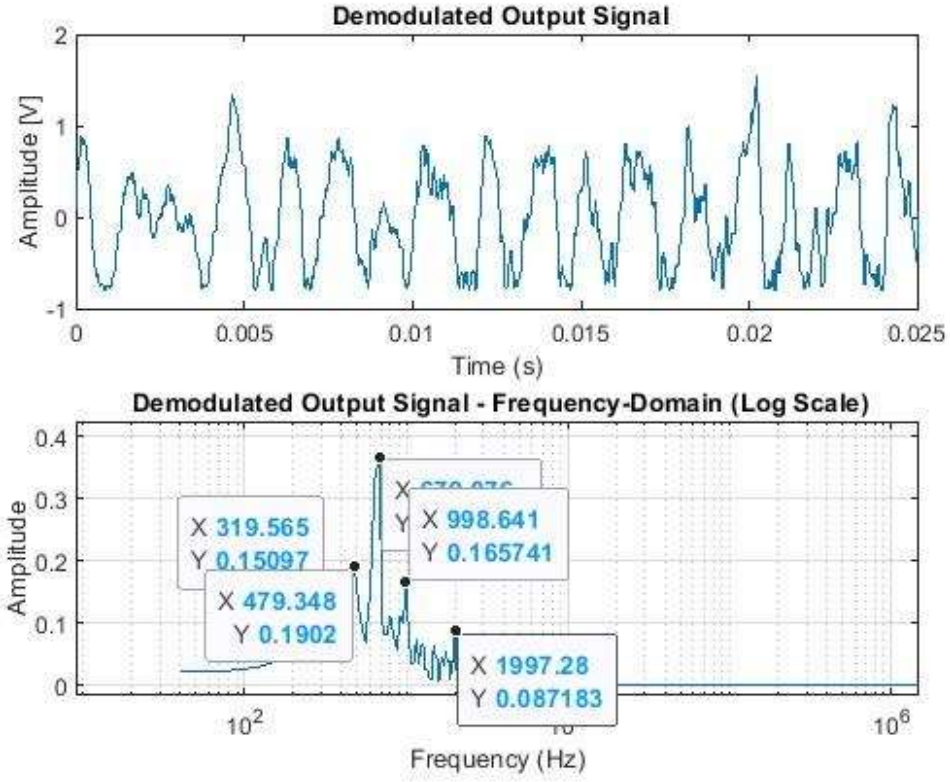
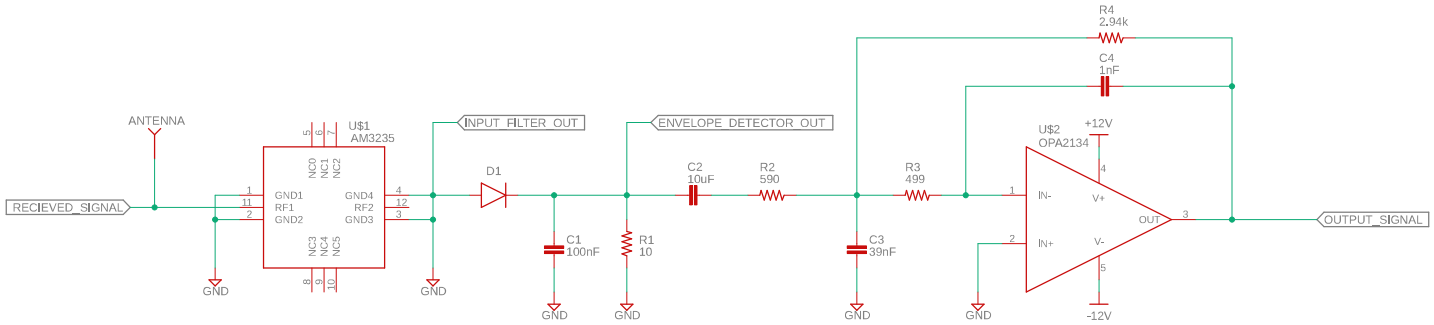


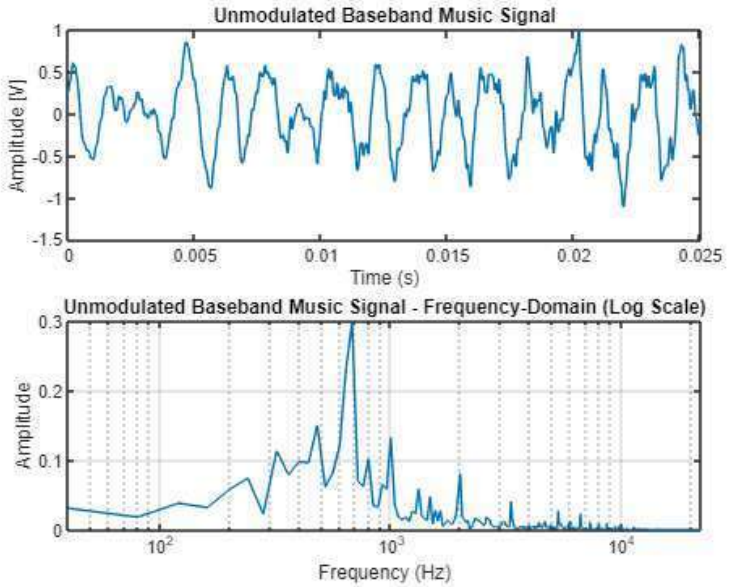
Figure 12.1 –Demodulator Output compared with Unmodulated Baseband Signa

Appendix

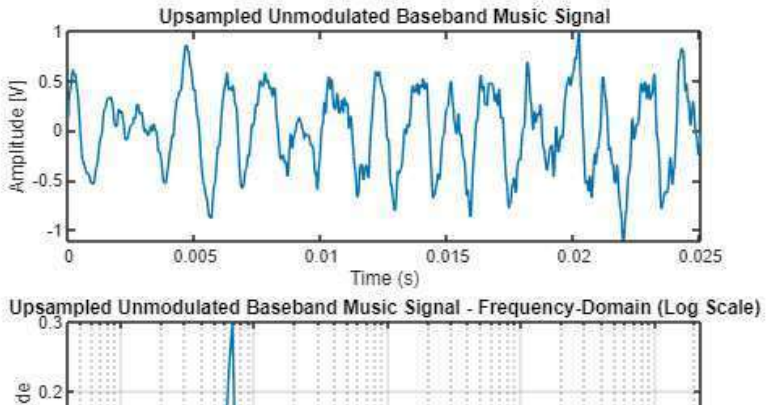


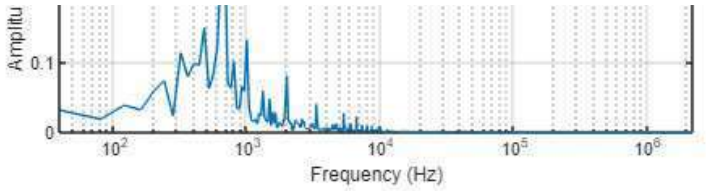
TITLE: ENVELOPE DETECTOR v1	
Document Number: Omar Mohamed & Josephine Elumeze	REV:
Date: 12/4/2024 6:53 AM	Sheet: 1/1

```
% This script simulates demodulation of a tone-modulated AM signal
% using an envelope detector circuit

file_name = 'E:\tearstream-alley-262939.mp3';

% Parameters
f_carrier = 1000000;            % 1MHz Carrier Frequency
t_start = 55;                   % start time in seconds of snippet from music file
t_end = 55.025;                 % end time in seconds of snippet from music file
duration = t_end - t_start;     % 25ms Duration

% Generate music snippet from file
[fs, am_signal] = generate_am_signal(file_name,t_start, t_end);
plot_fft(am_signal, fs, 'Unmodulated Baseband Music Signal');
```




```
% Upsample by a factor of 250
% Multiplply by 50 because upsampling attenuates the signal
upsample_factor = 100;
am_signal = 100*upsample_signal(am_signal, upsample_factor);
% time vector for upsampled signal
fs = fs * upsample_factor;
t = 0:1/fs:(length(am_signal)-1)/fs;

% Moving average filter to smooth upsampler output. Moving average is
% convolution with a rectangular pulse
averaging_filter = ones(1, 100) / 100;
am_signal = conv(am_signal, averaging_filter, 'same');

plot_fft(am_signal, fs, 'Upsampled Unmodulated Baseband Music Signal');
```

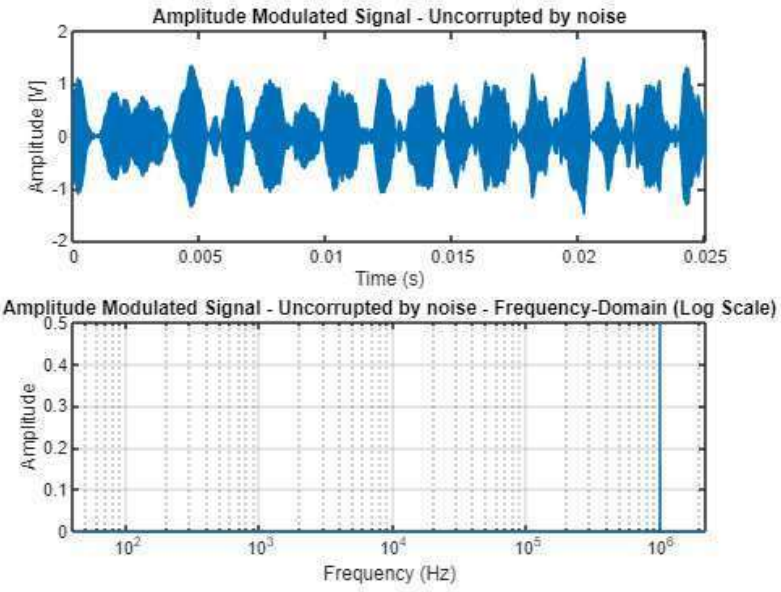




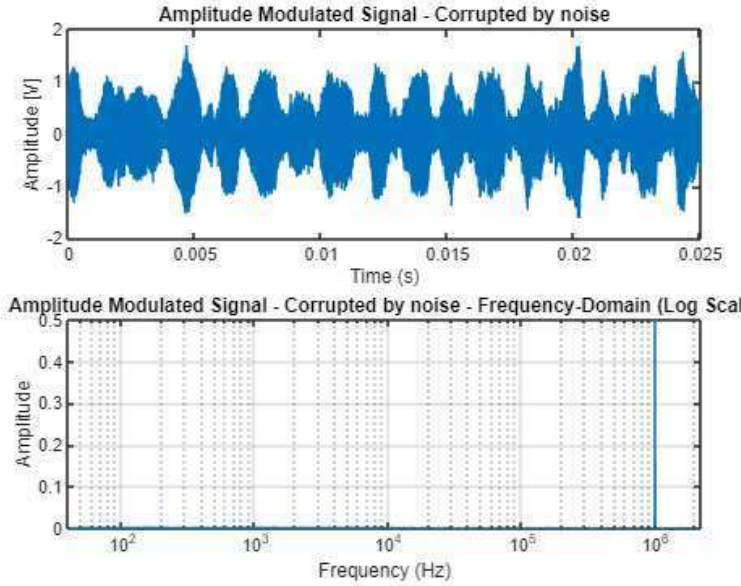

```
% Modulate with Carrier signal
carrier = cos(2*pi*f_carrier*t);
am_signal = (0.5 + am_signal) .* carrier;

% White noise parameters
noise_std = 0.1;                    % Standard deviation of the noise
noise_mean = 0;                     % Mean of the noise

% Envelope Detector RC filter Parameters
R = 10;
C = 1000e-9;

% Generate white noise
white_noise = generate_white_noise(fs, (1/fs)*length(am_signal), noise_std, noise_mean);
am_signal = am_signal + white_noise;

% Simulate AM3235 bandpass filter. Note this is just a generic ideal bandpass filter and
% not the exact response of the AM3235
filterd_signal = bandpass_filter(am_signal, fs, f_carrier, 0.4*f_carrier);

% Simulated the signal passing through the diode in envelope detector
rectified_signal = max(filterd_signal, 0); % Half-wave rectified signal

% Simulate the signal passing through the parallel RC circuit in the
% envelope detector
envelope_detector_signal = envelope_detector(rectified_signal, fs, R, C);

% Simulate the signal passing through the OP2134 active bandpass filter
output_signal = OP2134_filter(envelope_detector_signal, fs)
```

```
output_signal = 1×110400
    0.0277    0.0327    0.0376    0.0426    0.0475    0.0525    0.0575    0.0625    0.0675    0.0725    0.0776    0.0826
```

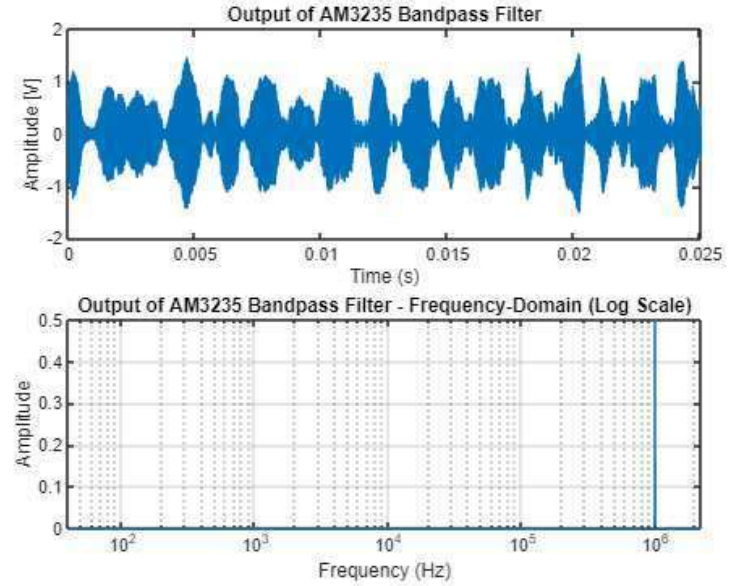
```
plot_fft(am_signal - white_noise, fs, 'Amplitude Modulated Signal - Uncorrupted by noise');
```



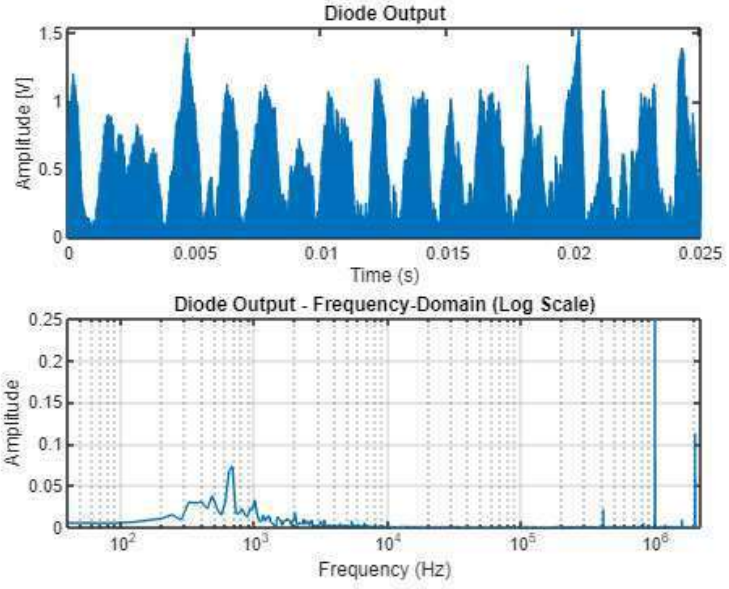

```
plot_fft(am_signal, fs, 'Amplitude Modulated Signal - Corrupted by noise');
```



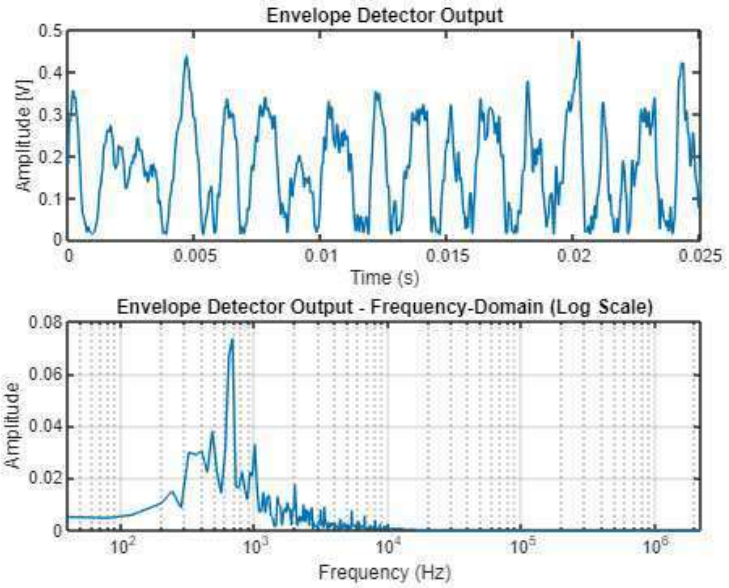

```
plot_fft(filterd_signal, fs, 'Output of AM3235 Bandpass Filter');
```



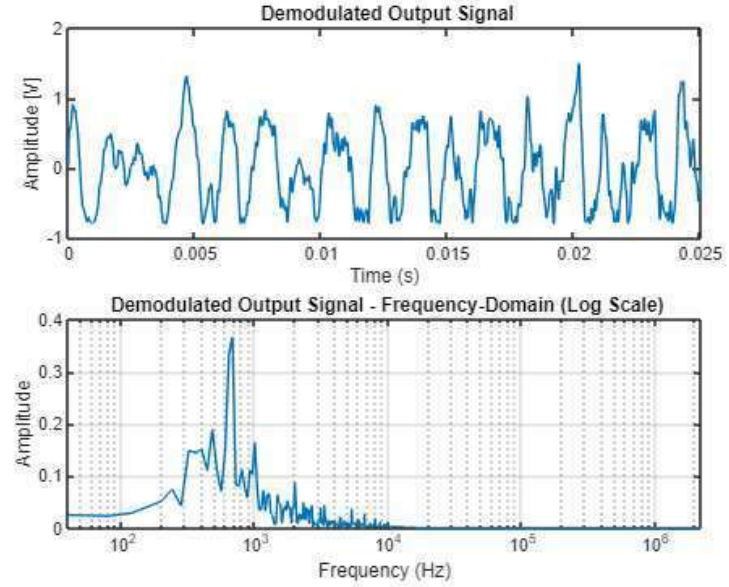

```
plot_fft(rectified_signal, fs, 'Diode Output');
```



```
plot_fft(envelope_detector_signal, fs, 'Envelope Detector Output');
```




```
plot_fft(output_signal, fs,' Demodulated Output Signal');
```




```
% play the demodulated audio clip
sound(output_signal,fs);
```

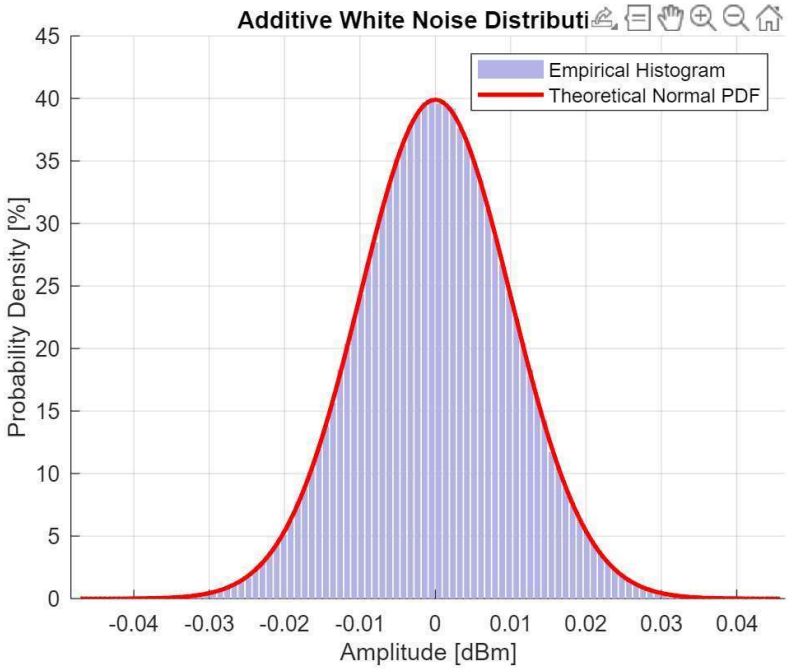
```
function output_signal = envelope_detector(input_signal, fs, R, C)
    % This function simulates a signal passing through an envelope detector

    % Half-wave rectifier, clip negative portions of the signal
    rectified_signal = max(input_signal, 0);

    % Parallel RC filter cut-off
    fc = 1 / (2 * pi * R * C);

    % Length of input signal, i.e, number of samples
    N = length(rectified_signal);

    % Compute FFT of input signal
    X = fft(rectified_signal);

    % Frequency vector
    f = (0:N-1) * (fs / N);
    f(f > fs/2) = f(f > fs/2) - fs;  % Adjust for negative frequencies

    % Ideal Lowpass Filter in Frequency Domain. Magnitude is 1 inside of
    % the passband, f < f_c and magnitude is 0 outside of the
    % passband, f > f_c
    H = abs(f) <= fc;

    % Apply Filter by Multiplication in Frequency Domain
    X_filtered = X .* H;

    % Compute Inverse FFT to return to Time Domain
    output_signal = real(ifft(X_filtered));
end

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

function output_signal = OP2134_filter(input_signal, fs)
    % This function simulates a signal passing through the OP2134 active
    % filter

    % Circuit Parameters
    R1 = 590;
    R3 = 499;
    R4 = 2940;
    C2 = 39*1e-9;
    C5 = 1*1e-9;

    fc = 2 * pi * sqrt((R1) * (R3) * (C2) * (C5));
    fc = 1/ fc;

    gain = R4/R1;

    % Remove the DC offset, simulate dc blocking capacitor
    input_signal = input_signal - mean(input_signal);

    % Length of input signal, i.e, number of samples
    N = length(input_signal);

    % Compute FFT of input signal
    X = fft(input_signal);

    % Frequency vector
    f = (0:N-1) * (fs / N);
    f(f > fs/2) = f(f > fs/2) - fs;  % Adjust for negative frequencies

    % Ideal Lowpass Filter in Frequency Domain. Magnitude is gain inside
    % ofthe passband, f < f_c and magnitude is 0 outside of the
    % passband, f > f c
```

```
    H = abs(f) <= fc;
    H = H * gain;

    % Apply Filter by Multiplication in Frequency Domain
    X_filtered = X .* H;

    % Compute Inverse FFT to return to Time Domain
    output_signal = real(ifft(X_filtered));
end

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

function white_noise = generate_white_noise(fs, duration, std, mean)
    % This function generates an additive white noise signal with a given
    % duration in seconds and sampling rate. The mean and std can be specified
    % in the input parameters.

    % Calculate the total number of samples
    num_samples = round(fs * duration);

    % Generate white noise with specified mean and standard deviation
    white_noise = (std * randn(1, num_samples) + mean);
end

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

function output_signal = bandpass_filter(input_signal, fs, fc, bw)
    % This function simulates the signal passing through the input bandpass
    % filter

    % Filter specifications
    f_low = fc - 0.5*bw;        % Lower cutoff frequency (Hz)
    f_high = fc + 0.5*bw;       % Upper cutoff frequency (Hz)

    % Length of input signal, i.e, number of samples
    N = length(input_signal);

    % Compute FFT of input signal
    X = fft(input_signal);

    % Frequency vector
    f = (0:N-1) * (fs / N);
    f(f > fs/2) = f(f > fs/2) - fs;  % Adjust for negative frequencies

    % Ideal Bandpass Filter in Frequency Domain. Magnitude is 1 inside of
    % the passband, f_low < f < f-high and magnitude is 0 outside of the
    % passband, f < f_low or f > f_high
    H = (abs(f) >= f_low) & (abs(f) <= f_high);

    % Apply Filter by Multiplication in Frequency Domain
    X_filtered = X .* H;

    % Compute Inverse FFT to return to Time Domain
    output_signal = real(ifft(X_filtered));
end

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

function output_signal = lowpass_filter(input_signal, fs, fc)
    % This function simulates the signal passing through a lowpass
    % filter
```

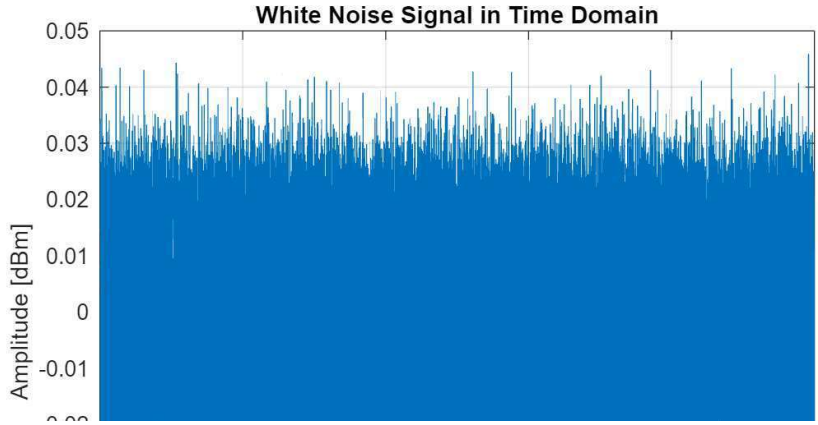
```
    % Length of input signal, i.e, number of samples
    N = length(input_signal);

    % Compute FFT of input signal
    X = fft(input_signal);

    % Frequency vector
    f = (0:N-1) * (fs / N);
    f(f > fs/2) = f(f > fs/2) - fs;  % Adjust for negative frequencies

    % Ideal Lowpass Filter in Frequency Domain. Magnitude is 1 inside of
    % the passband, f < f_c and magnitude is 0 outside of the
    % passband, f > f_c
    H = abs(f) <= fc;

    % Apply Filter by Multiplication in Frequency Domain
    X_filtered = X .* H';

    % Compute Inverse FFT to return to Time Domain
    output_signal = real(ifft(X_filtered));
end

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

function y_upsamped_signal = upsample_(input_signal, upsample_factor)
    % Insert zeros between original samples
    N = length(input_signal);
    y_upsampled = zeros(1, upsample_factor * N);
    y_upsampled(1:upsample_factor:end) = input_signal;

    % Pass signal through a lowpass filter -> convolve with Sinc

    filter_length = 51;  % Length of the filter (odd number for symmetry)
    cutoff = 1 / upsample_factor;  % Normalized cutoff frequency
    n = -(filter_length-1)/2:(filter_length-1)/2;  % Time indices
    h = sinc(cutoff * n) .* hann(filter_length)';  % Sinc filter with Hann window
    h = h / sum(h);  % Normalize filter coefficients

    % Step 3: Apply the filter to smooth the signal
    y_upsampled = conv(y_upsampled, h, 'same');
end

function [fs, am_signal] = generate_am_signal(file_name, t_start, t_end)

    % read audio signal from the file and only take one channel of stereo
    [stereo, fs] = audioread(file_name);
    mt = stereo(:, 2);

    % take a short clip of the signal
    idx_start = max(1, round(fs * t_start));  % Ensure at least 1
    idx_end = min(length(mt), round(fs * t_end));  % Ensure within bounds
    mt = mt(idx_start:idx_end);  % Take the clip

    % make sure amplitude is atleast 1
    max_value = max(mt);
    mt = (mt / max_value);

    % Recieved Power arriving at the input of the demodulator
    am_signal = mt;
end

%-------------------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------------------
```

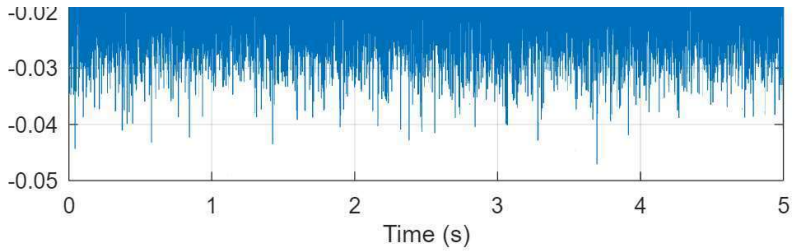
```
function plot_fft(x, fs, plot_title)
    % Signal length
    N = length(x);

    % Compute FFT
    X = fft(x);                          % Perform FFT
    amplitude = abs(X) / N;              % Compute magnitude spectrum
    f = (0:N-1) * (fs / N);              % Frequency vector

    % Single-sided spectrum (positive frequencies only)
    f_half = f(1:floor(N/2)+1);          % Positive frequencies
    amplitude_half = 2 * amplitude(1:floor(N/2)+1); % Single-sided amplitude

    % Plot Time-Domain Signal
    t = (0:N-1) / fs;                    % Time vector
    figure;
    subplot(2, 1, 1);
    plot(t, x);
    title(plot_title);
    xlabel('Time (s)');
    ylabel('Amplitude [V]');
    xlim([0 , N/fs]);

    % Plot Frequency-Domain Signal with Logarithmic X-Axis
    subplot(2, 1, 2);
    semilogx(f_half, amplitude_half);
    title([plot_title, ' - Frequency-Domain (Log Scale)']);    xlabel('Frequency (Hz)');
    ylabel('Amplitude');
    xlim([min(f_half(f_half > 0)) fs/2]); % Exclude 0 Hz and set range to Nyquist frequency
    grid on;
end

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

function y_upsampled = upsample_signal(input_signal, upsample_factor)

    % Insert zeros between original samples
    N = length(input_signal);
    y_upsampled = zeros(1, upsample_factor * N);
    y_upsampled(1:upsample_factor:end) = input_signal;

    % Pass the signal through a lowpass filter to interpolate inbetween the
    % samples
    % Use custom sinc function to create the lowpass filter in the time domain
    sinc_length = 70;
    cutoff = 1 / upsample_factor;  % Normalized cutoff frequency

    n = -(sinc_length-1)/2:(sinc_length-1)/2;  % Time indices for h(t)

    h = sinc_func(cutoff * n);  % Sinc function, low-pass filter impulse response
    h = h / sum(h);  % Normalize filter coefficients to 1

    % convolve the input x(t) with the lowpass filter impulse response h(t)
    y_upsampled = conv(y_upsampled, h, 'same');
end

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

function y = sinc_func(x)
    % Generate a Sinc function
    y = ones(size(x));  % Default value
    idx = x ~= 0;       % Indices where x is not zero
    y(idx) = sin(pi * x(idx)) ./ (pi * x(idx));  % Compute sinc only where x ~= 0
end
```

```
% This script is meant to provide a simulation of generating a white noise
% signal that can be added to the recieved signal at the input of the
% demodulator

% Parameters
fs = 100000;          % Sampling frequency in Hz
duration = 5;       % Duration in seconds
noise_std = 0.01;   % Standard deviation of the noise
noise_mean = 0;     % Mean of the noise
nbins = 100;         % Number of histogram bins

% Generate white noise
white_noise = generate_white_noise(fs, duration, noise_std, noise_mean);

% Define x range for PDF
x_range = linspace(min(white_noise), max(white_noise), 1000);

% Compute the probability density function (PDF) of the normal distribution
theoretical_pdf = (1 / (noise_std * sqrt(2*pi))) * exp(-0.5 * ((x_range - noise_mean) / noise_std).^2);

% Plot the empirical and theoretical PDF
plot_pdf(white_noise, theoretical_pdf, x_range, nbins);
```




```
% Plot the white noise signal in the time domain
plot_time_domain(white_noise, fs)
```






```
function white_noise = generate_white_noise(fs, duration, std, mean)
    % This function generates an additive white noise signal with a given
    % duration in seconds and sampling rate. The mean and std can be specified
    % in the input parameters.

    % Calculate the total number of samples
    num_samples = round(fs * duration);

    % Generate white noise with specified mean and standard deviation
    white_noise = std * randn(1, num_samples) + mean;
end

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

function plot_pdf(empirical_data, theoretical_data, x_range, num_bins)
    % This function plots an empirical data set for a pdf versus its
    % theoretical pdf

    % Histogram of the data
    [counts, edges] = histcounts(empirical_data, num_bins, 'Normalization', 'pdf');
    bin_centers = (edges(1:end-1) + edges(2:end)) / 2;

    % Plot
    figure;
    hold on;
    bar(bin_centers, counts, 'FaceColor', [0.7 0.7 0.9], 'EdgeColor', 'none'); % Histogram
    plot(x_range, theoretical_data, 'r', 'LineWidth', 2); % Theoretical PDF
    xlabel('Amplitude [dBm]');
    ylabel('Probability Density [%]');
    title('Additive White Noise Distribution');
    legend('Empirical Histogram', 'Theoretical Normal PDF');
    grid on;
    hold off;
end

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

function plot_time_domain(signal, fs)
    % This function plots a signal in the time-domain

    % define time axis
    t = linspace(0, length(signal)/fs, length(signal));

    figure;
    plot(t, signal);
    xlabel('Time (s)');
    ylabel('Amplitude [dBm]');
    title('White Noise Signal in Time Domain');
    grid on;
end

%-----------------------------------------------------------------------------------------------
```