Cairo University, Faculty of Engineering
Electronics and Electrical Communications
Department (EECE)

Third Year

First Term

Super-heterodyne Receiver Communication Project report

Name	Sec	ID
Yousef Khaled Omar Mahmoud	4	9220984

Presented by

Instructors: Dr. Ahmed Hesham

Eng. Said Kamel

Abstract:

In this project, we aimed to simulate a multi-stage communication system, focusing on the transmission and reception of audio signals through various stages, including baseband, RF (Radio Frequency), IF (Intermediate Frequency), and AM (Amplitude Modulation). Our system processes audio signals by modulating them into AM DSB-SC (Double Sideband Suppressed Carrier) format.

We implemented a radio simulator device with MATLAB. we focused on emulating the receiver and its filtering process, using software to handle the audio files, and explored different signal processing techniques.

Table of content:

Introduction

The primary goal of this project is to simulate the fundamental components of an analog AM communication system using MATLAB. This involves the creation of an AM modulator to encode audio signals for transmission and a super-heterodyne receiver to recover the transmitted signal. The project uses provided audio signals as input messages, simulating a real-world scenario of radio transmission and reception.

System Design and Implementation:

An AM communication system consists of two main stages:

- **The modulator:** encodes the message signal onto a higher-frequency carrier for efficient wireless transmission.
- **The receiver:** utilizes a super-heterodyne architecture, involving mixing, filtering, and amplification, to recover the original message.

System Implementation:

In *fig[1]*, We describe how we implement the system in MATLAB.[1]

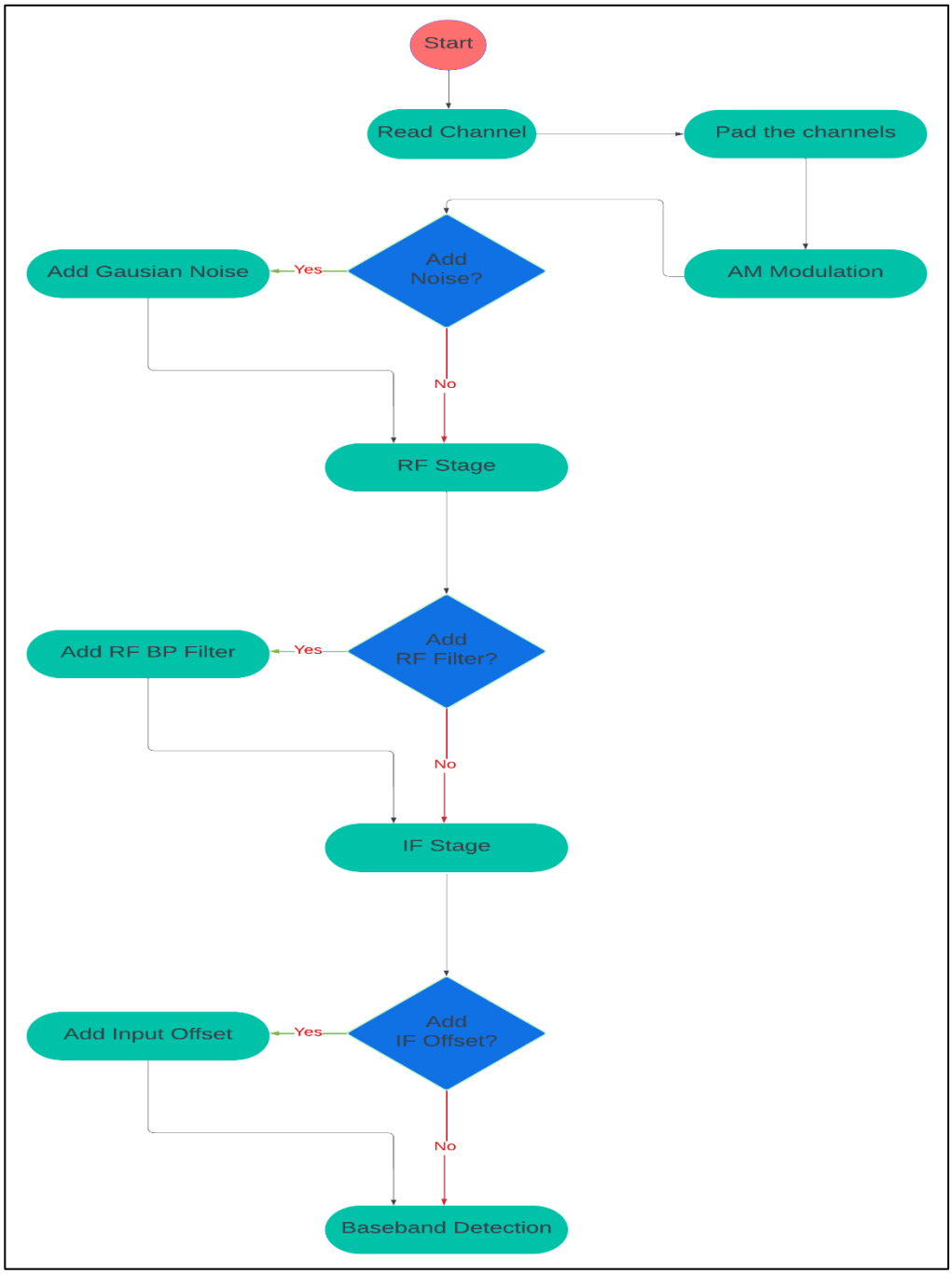

Figure 1 Simulation Flow Chart

System Components:

1. AM Modulator:

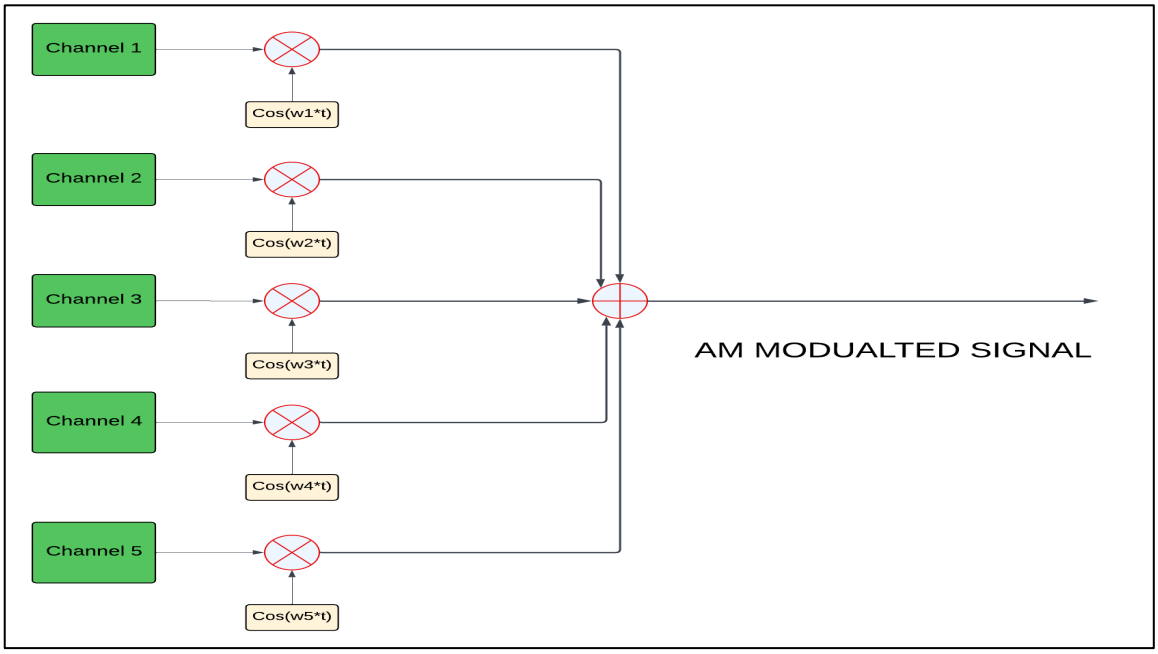

Figure 2 AM Modulation Double Sideband Suppressed Carrier

$$FDM(t) = \sum_{i=0}^{n-1} Mn(t) * Cos(Wn * t)$$

Equation 1 Frequency Division Multiplexing

- *Where: Mn(t) is the n-th channel's message signal, Wn=100+nΔF, with ΔF=50 kHz and n as the channel index (n=0 corresponds to the first signal modulated at 100 kHz)*

As shown in *fig[2]*,This project presents a MATLAB simulation of a multi-channel communication system using Frequency Division Multiplexing (FDM). The system processes five audio signals, each encoded using Amplitude Modulation (AM) onto carrier waves with unique frequencies.

The modulated signals are summed to form the FDM signal, defined mathematically in *Equ[1]*, and transmitted over RF channels and processed at the receiver.

2. RF Stage:

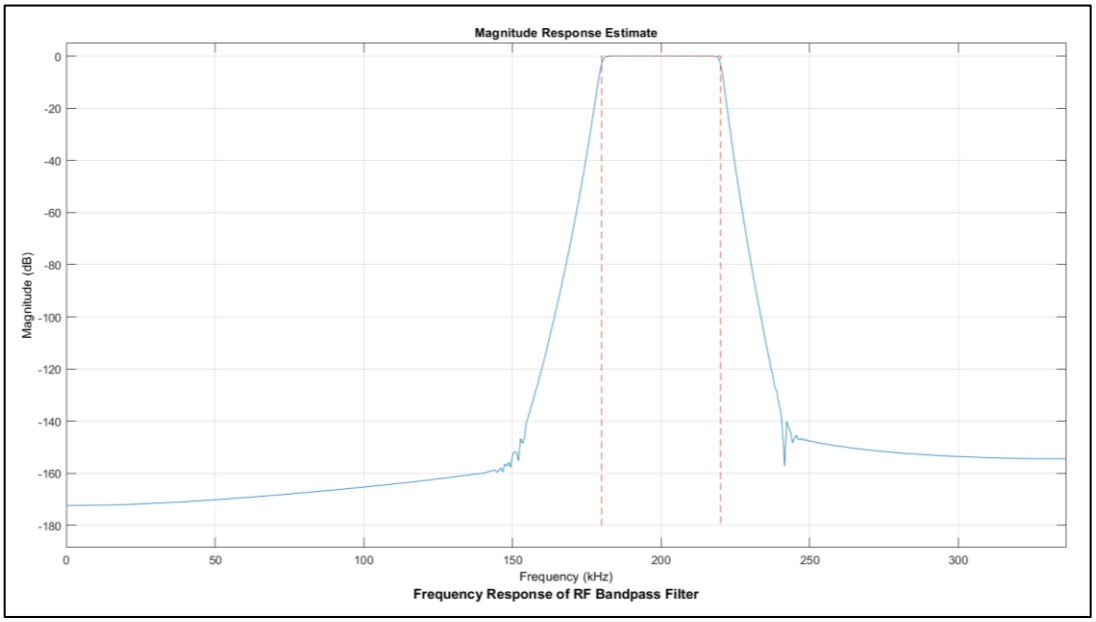

Figure 3 Receiver Filter Frequency Response

As shown in *fig[3]*, RF filter is a band pass filter centered on the desired station's carrier frequency ωn.

- **The purpose of the RF stage**: is to reject the intermediate frequency (IF) image and other unwanted signals outside the passband to extract the desired station.

3. Mixer:

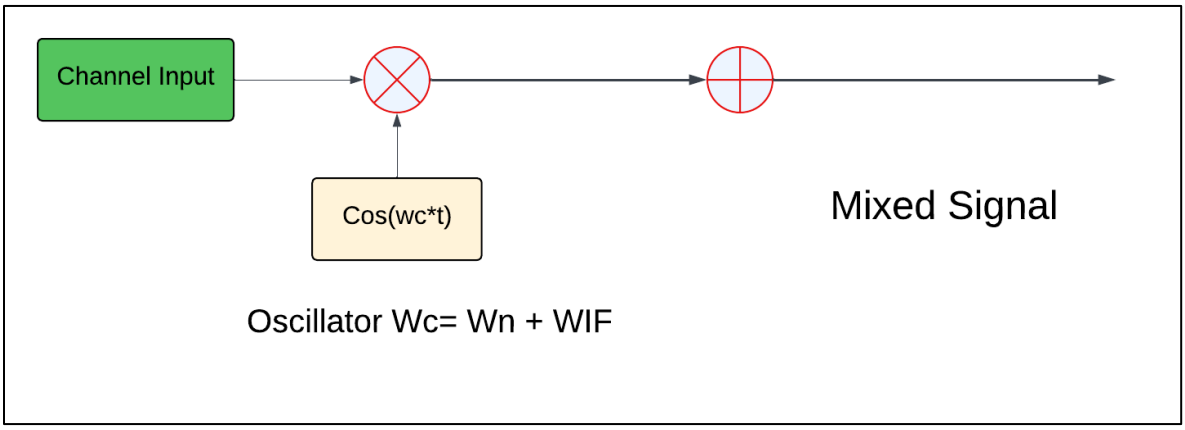

Figure 4 Mixer with Oscillator Block Diagram

As shown in *fig[4]*, The Oscillator generates a carrier signal for mixing, shifting the desired signal to the Intermediate Frequency (IF) band.

$$Wc = Wn + WIF$$

Equation 2 Oscillator Frequency

- *Where: Wc is the oscillator frequency, Wn is the channel frequency and WIF is the intermediate frequency*

According to *Equ[2]*,The oscillator frequency is set to get the desired channel to IF stage.

4. IF Stage:

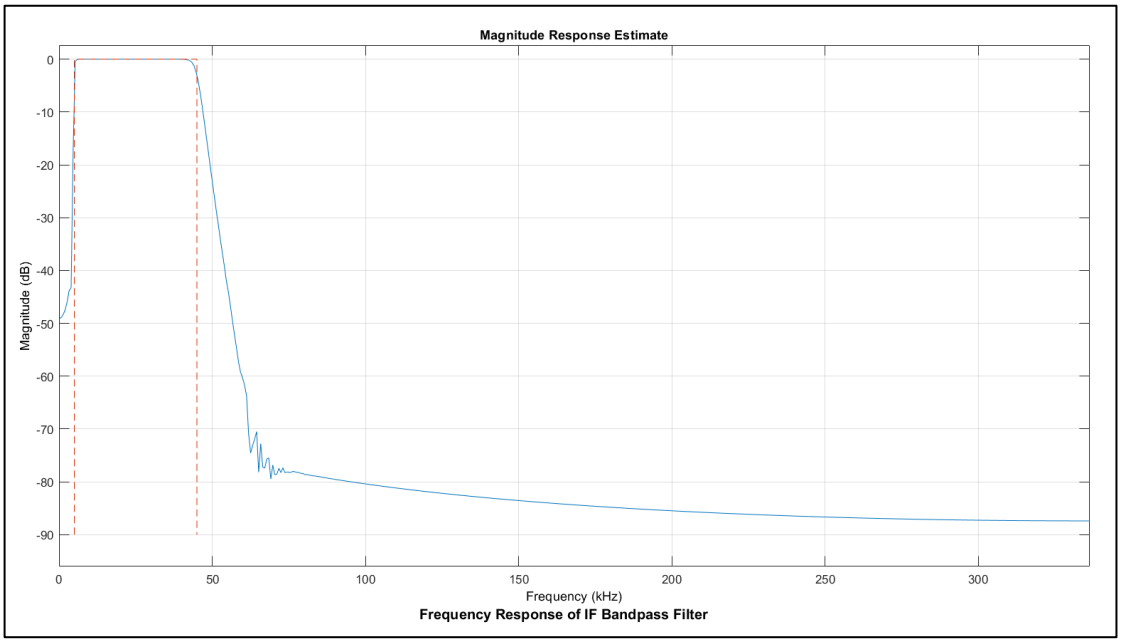

Figure 5 Intermediate Filter Frequency Response

As shown in *fig[5]*, IF filter is a band pass filter centered on intermediate frequency (WIF).

- **The purpose of the IF stage**: filters the IF signal for further processing. By shifting the signal to a fixed WIF, it simplifies the filter design and improves selectivity.

5. Baseband Detection:

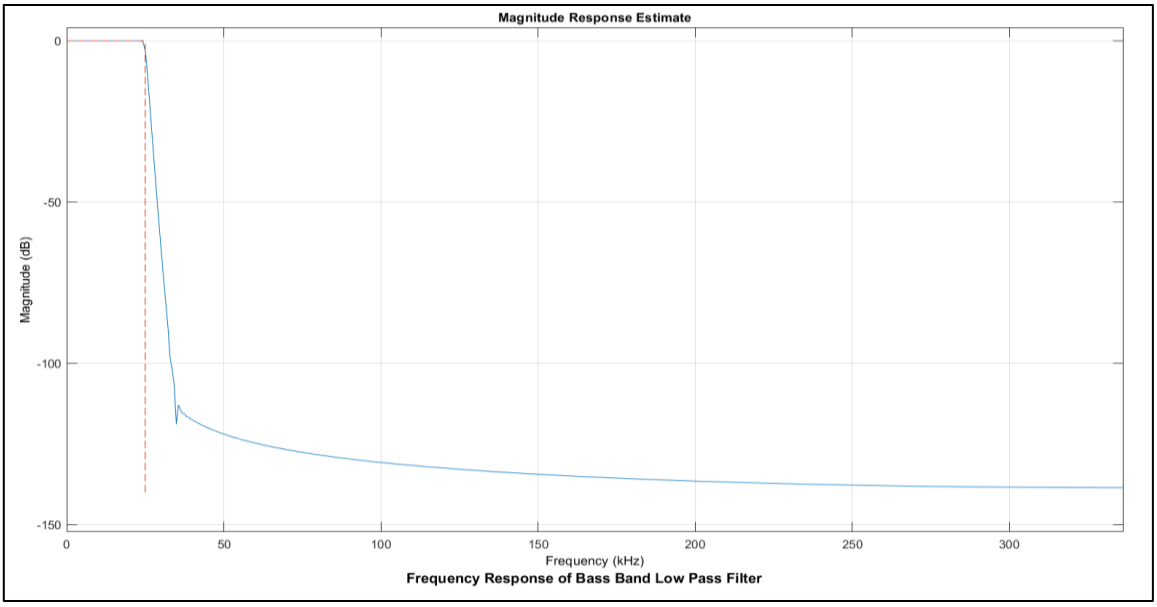

Figure 6 Baseband Filter Frequency Response

As shown in *fig[6]*, The baseband filter a Low-Pass Filter (LPF) that removes high-frequency components after mixing the IF signal with the carrier, centered at a frequency little higher than the channel band width which is 10 kHz.

- **The purpose of the Baseband Detector:** recovering the original message by demodulating the IF signal.

  The IF stage is crucial because it allows the receiver to use fixed components (filters and amplifiers) irrespective of the received carrier frequency.

Simulation Results

In this project, we implemented and analyzed various simulations to evaluate the performance and reliability of our communication system under different conditions. The simulations included the following scenarios:

1. **Normal Simulation:**

The system operates without any external disturbances, providing a baseline for performance analysis.

2. **Noise:**

Additive noise was introduced to simulate real-world interference, demonstrating the system's ability to process and recover the original signal under noisy conditions.

3. **No RF Filter:**

The impact of bypassing the RF stage was analyzed to highlight its role in filtering and isolating the desired signal.

4. **Frequency Offset (200 Hz):**

A small offset was applied to the carrier frequency, simulating minor synchronization errors and testing the system's tolerance.

5. **Frequency Offset (1200 Hz):**

A larger offset was introduced, representing more severe desynchronization, to analyze the limits of the system's error-handling capabilities

Normal Simulation:

We're going to run the simulation without any problems (Ideal Case) and going to choose channel 3.

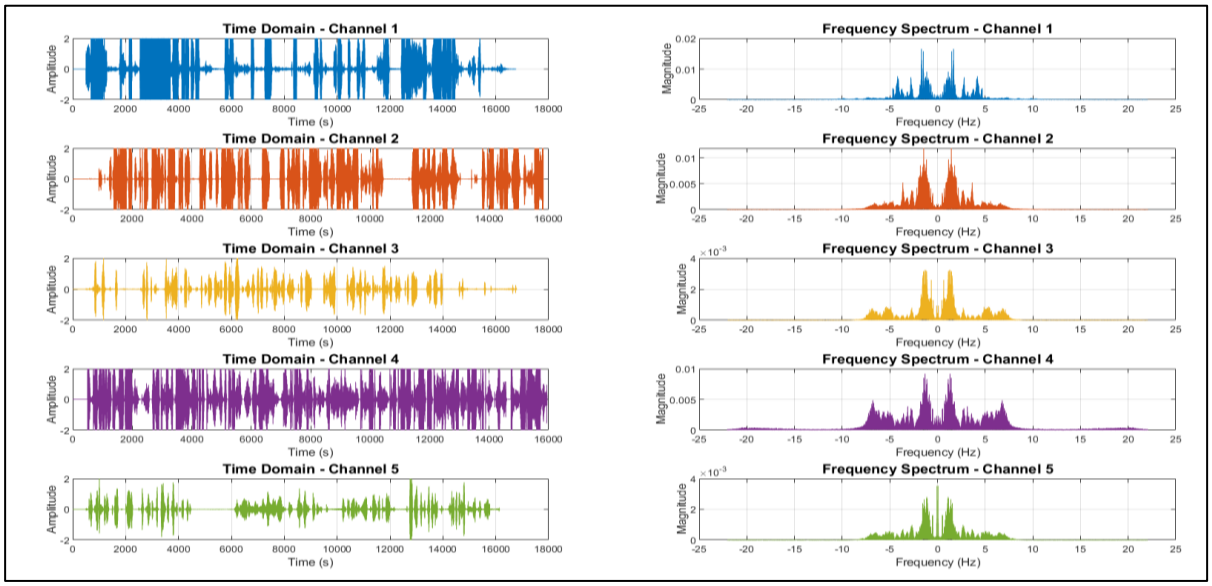

Figure 7 Channels plot in time and frequecny domain

As shown in *fig[7]*, All channels have a Band width approximately equal to 10 kHz.

Their length wasn't equal so we need to pad them

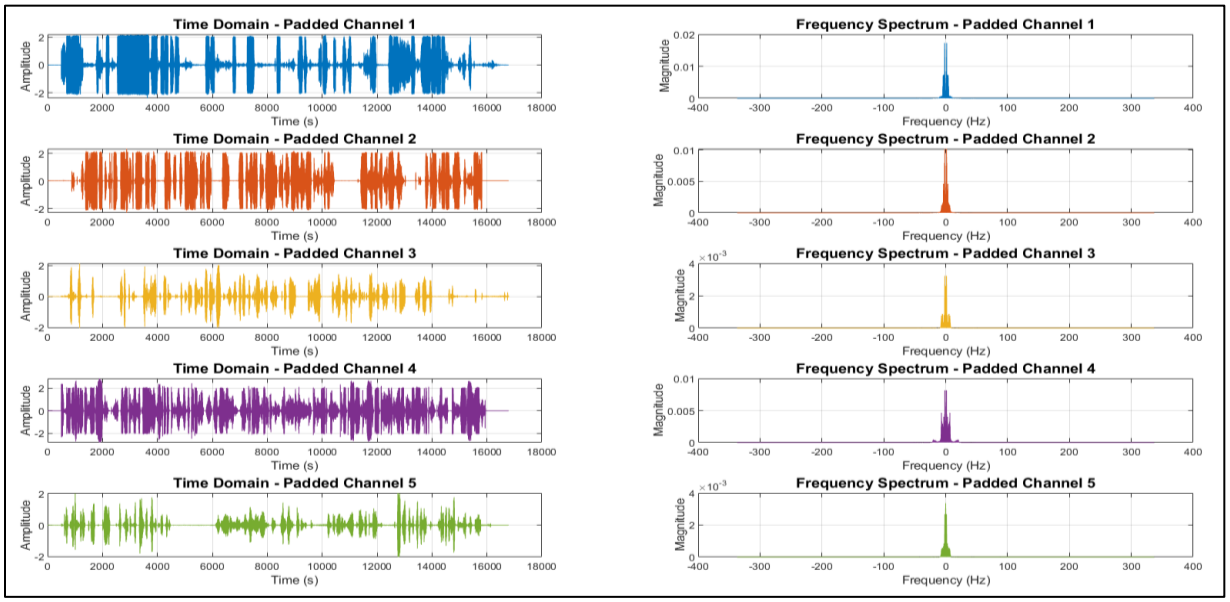

Figure 8 Padded Channels

As shown in *fig[8]*, The channels are now equal in length, so they are padded.

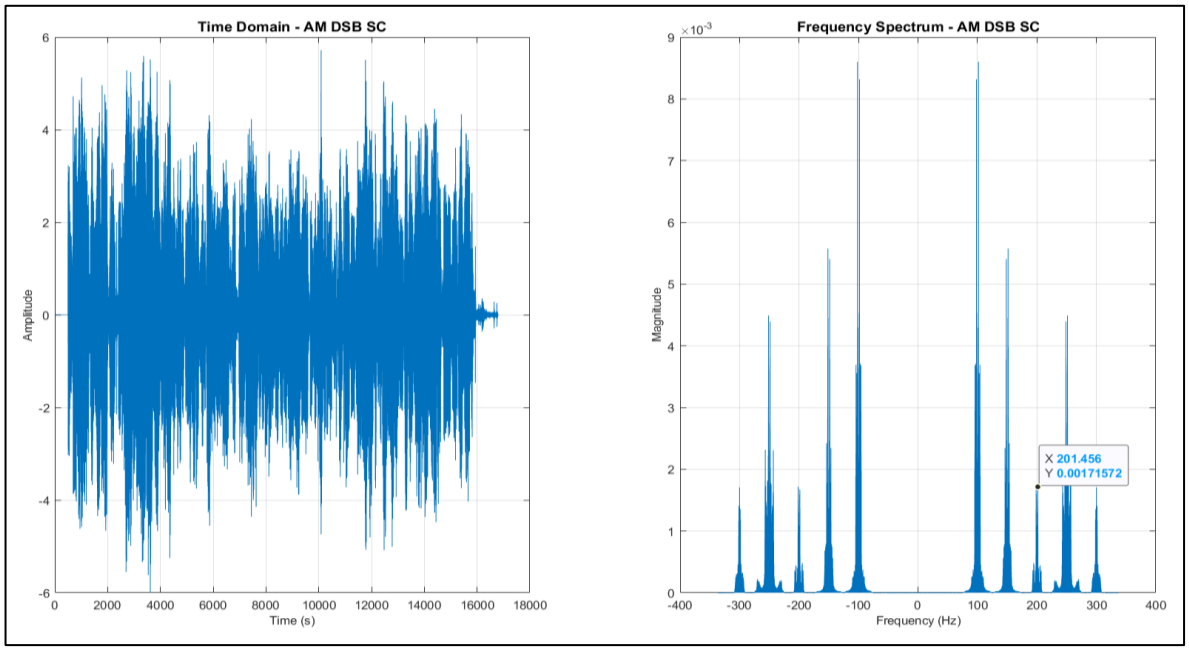

Figure 9 frequency division multiplexing

As shown in *fig[9]*, We used AM DSB-SC modulation where the carrier and message signal are multiplied. The channel 3 that's desired is at frequency equals to 200 kHz using *Equ[1]*.

We sum each channel modulated, as shown in *Equ[1]*, to transmit it into free space.

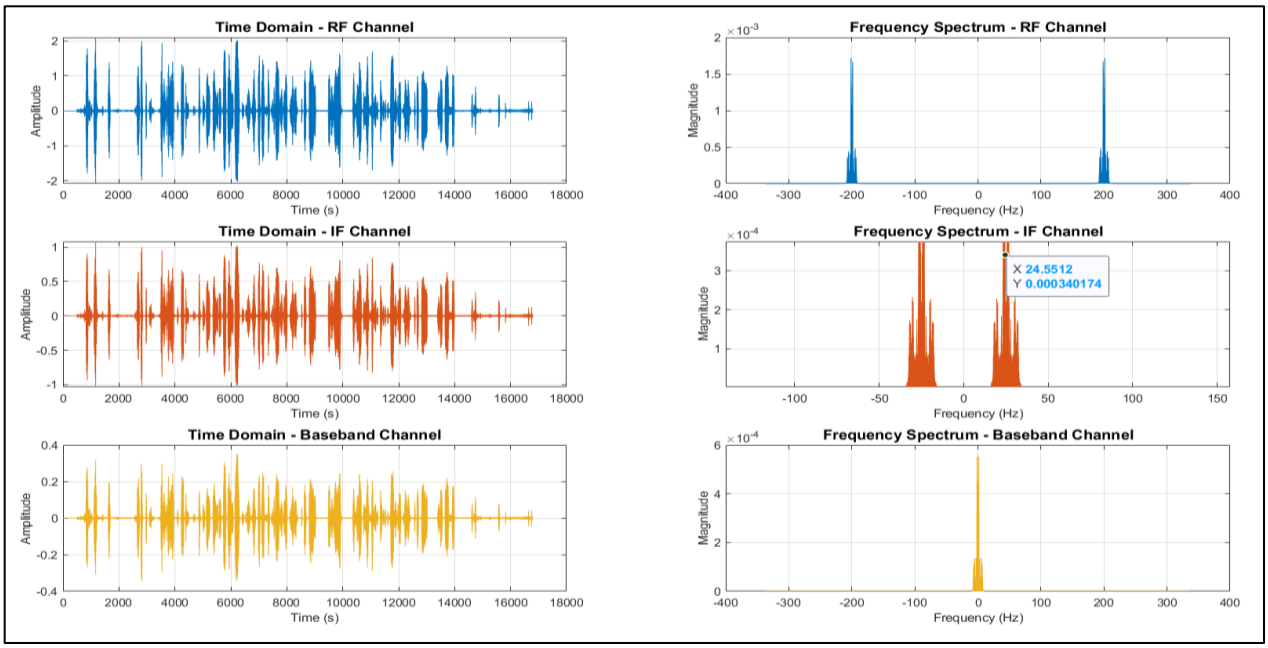

Figure 10 Receiver Signals

As shown *fig[10]*, We first filter the signals using the RF filter in *fig[3]*.

After that we use the mixer with oscillator in *fig[4]* to get the signal to IF frequency using the *Equ[2]* then filter it using the IF filter *fig[5]*.

The baseband detection is the last stage to retreat the orignal message using the mixer with oscillator frequency at WIF.

The received message doesn't have any problems as it's **Ideal Case.**

We listened to it and it was identical to the orignal message.

Noise Simulation:

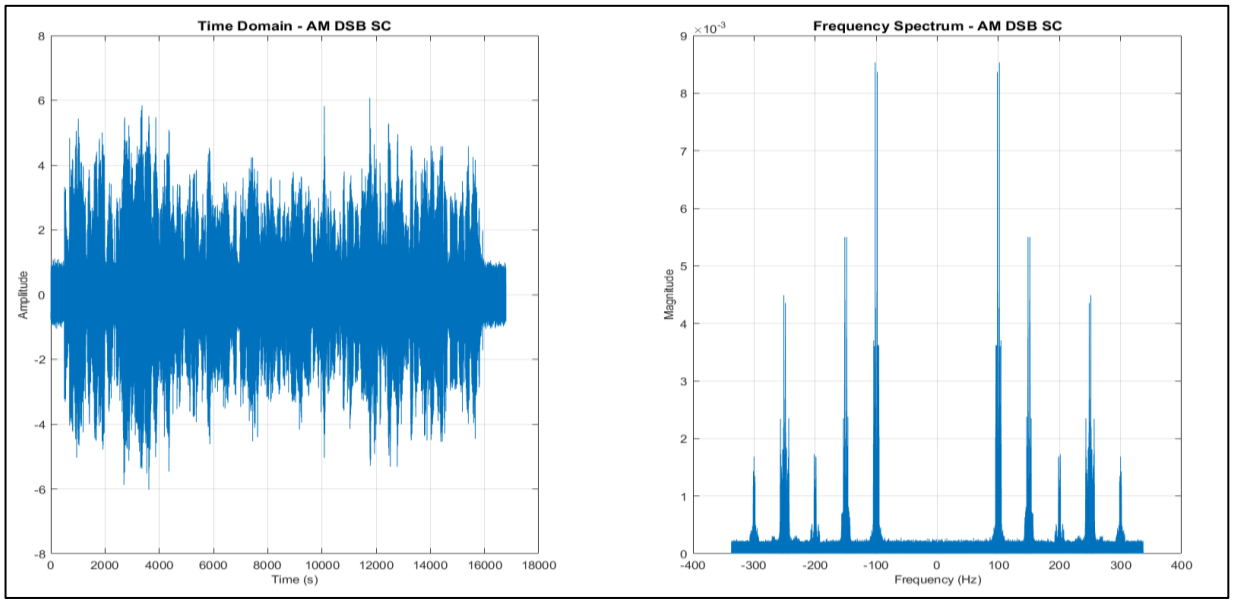

Figure 11 frequency division multiplexing with noise

As shown in *fig[11]*, We added the gaussian noise to the AM Modulation mixer with SNR=10.

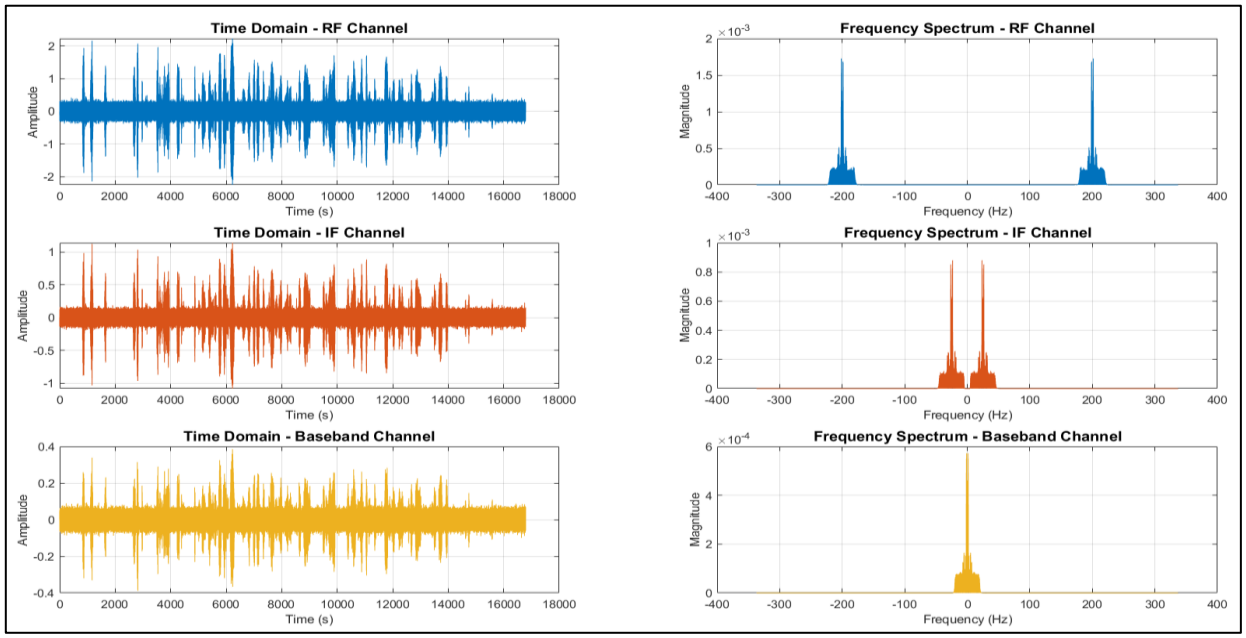

Figure 12 Receiver Signals with Noise

As shown in *fig[12]*, The receiver did its job and got the desired channel but it didn't filter the noise.

So listening to the desired channel we couldn't understand the message because it was too noisy.

But if the SNR was about 80-100 dB It's understandable.

NO RF Filter:

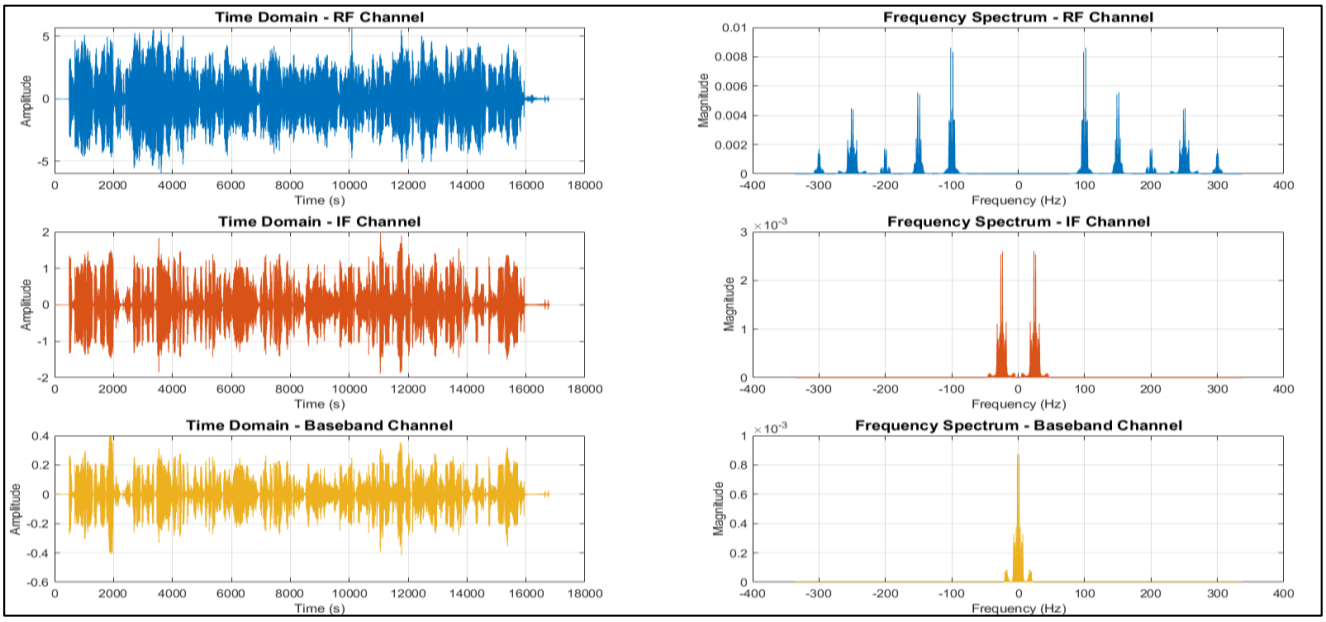

Figure 13 Receiver Signals without RF Filter

As shown in *fig[13]*, The receiver no longer isolates the desired signal from the spectrum. Instead, multiple signals, including unwanted neighboring channels, and IF image of the desired channel in the frequency domain.

The result is a mix of multiple modulated signals entering the demodulator, making it impossible to correctly reconstruct the original message.

So when listening to the bassband message we're hearing two channels at the same time.

Frequency Offset (200 Hz):

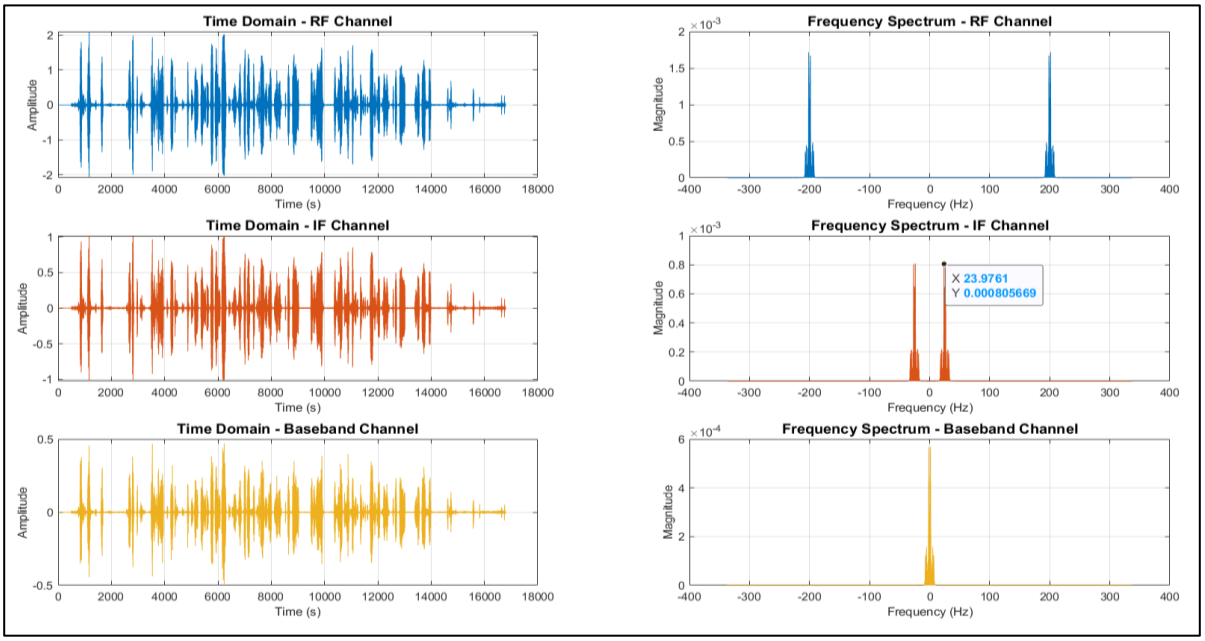

Figure 14 Receiver Signals with Offset = 200 Hz

As shown in *fig[13]*, The RF mixer has offset = 200Hz causing a slight shift in the carrier frequency during modulation and demodulation. This offset results in a minor distortion in the reconstructed signal at the receiver.

While the original message remains somewhat understandable, the quality is reduced due to the frequency mismatch.

Frequency Offset (1200 Hz):

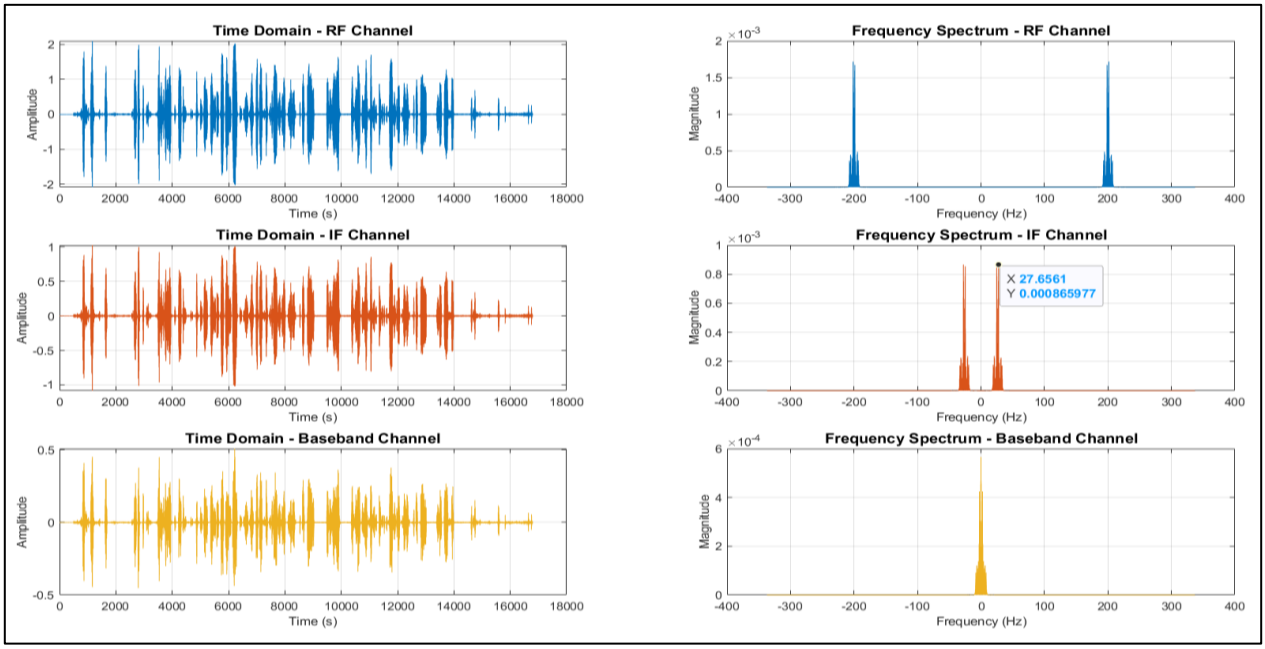

Figure 15 Receiver Signals with Offset = 1200 Hz

As shown in *fig[13]*, The RF mixer has offset = 1200Hz resulting in a severe misalignment of the carrier frequency. This large offset distorts the modulated signal substantially, causing the demodulated message to become heavily corrupted.

Conclusion:

In this project, we successfully designed, simulated, and analyzed a multi-stage AM communication system using MATLAB. By implementing key stages such as AM modulation, RF filtering, mixing, IF filtering, and baseband detection, we created a robust transceiver model capable of transmitting and receiving audio signals. The system was evaluated under various scenarios to understand its strengths and limitations.

Through this project, we can conclude that:[2]

- **Noise and Frequency Offsets:** are two major challenges in AM communication systems. These issues can significantly degrade signal quality and impair message recovery.

- **The RF filter:** is crucial for rejecting the IF image and other unwanted signals outside the passband, ensuring that only the desired channel is isolated and passed to the subsequent stages for accurate demodulation

References:

[1] https://github.com/youefkh05/Super-heterodyne-Receiver

[2] Ziemer, R. E., Tranter, W. H., & Fannin, D. R. (2013). *Digital and Analog Communication Systems*. Pearson, 8th Edition, Chapter 5-2

Appendix:

To have a good data structure to handle for the project we made AudioFile class to make it easier to code

```
classdef AudioFile
    properties
        Filename          % Name of the file
        SamplingFrequency % Sampling frequency of the audio (in kHz)
        AudioData         % Actual audio data
        duration          % File length
        player            % Object to play the sound file
    end
    methods

        function obj = AudioFile(filename)
            % Constructor method to initialize the object
            if nargin > 0
                [audioData, fs] = audioread(filename);

                % Convert to mono if stereo
                if size(audioData, 2) == 2
                    mono_audioData = sum(audioData, 2); % Sum the two channels to get a mono signal
                else
                    mono_audioData = audioData;
                end

                obj.Filename = filename;
                obj.SamplingFrequency = fs;
                obj.AudioData = mono_audioData;
                obj.duration  = length(obj.AudioData) / obj.SamplingFrequency;
                obj.player = audioplayer(obj.AudioData, obj.SamplingFrequency);
                obj.SamplingFrequency = fs/1000;
            end
        end

        function playAudio(obj)
            % Method to play the audio file
            if ~isempty(obj.player)
                play(obj.player);
            else
                disp('No audio data to play.');
            end
        end

        function pauseAudio(obj)
            % Method to pause the audio file
            if ~isempty(obj.player)
                pause(obj.player);
            else
                disp('No audio data to pause.');
            end
```

```
        end

        function stopAudio(obj)
            % Method to stop the audio file
            if ~isempty(obj.player)
                stop(obj.player);
            else
                disp('No audio data to stop.');
            end
        end

        function printAudio(obj)
            % Method to print all the audio file data
            if ~isempty(obj.AudioData)
                disp(['Filename: ', obj.Filename]);
                disp(['Sampling Frequency: ', num2str(obj.SamplingFrequency), ' kHz']);
                disp(['Audio Duration: ', num2str(obj.duration), ' seconds']);
                % Optional: Display first few samples of audio data
                %disp('First 10 samples of Audio Data:');
                %disp(obj.AudioData(1:10));
            else
                disp('No audio data to print.');
            end
        end

        end
end
```

We used functions that we made to make the code more readable. We will show the main functions and the other are in the reference [1].

Main code:

```
clear all;
close all;
clc;


% Add the Functions and Filters folder to the MATLAB path temporarily
addpath('Functions');
addpath('Filters');
load RF_Band_Pass_Filter; % Load predefined RF Band-Pass Filter


% List of channel audio file names
fileNames = [...
    "Ch0_Short_QuranPalestine.wav", ...
    "Ch1_Short_FM9090.wav", ...
    "Ch2_Short_BBCArabic2.wav", ...
    "Ch3_Short_RussianVoice.wav", ...
    "Ch4_Short_SkyNewsArabia.wav"];

ChannelPath = "Channels\";


% Step 1: Load and visualize channel data
channels = read_channels(fileNames, ChannelPath);   % Read audio files into 'channels' structure
print_channels_data(channels);                      % Display details of each channel
plotChannelSpectrum(channels, "Channel", 1);        % Plot spectrum of the channels


% Step 2: Determine max duration, sampling frequency, and length
[maxDuration, maxSamplingFreq, maxLength] = getMaxAudioInfo(channels);


% Display the results
fprintf('Max Duration: %.2f seconds\n', maxDuration);
fprintf('Max Sampling Frequency: %.2f kHz\n', maxSamplingFreq);
fprintf('Max Audio Data Length (number of samples): %d\n', maxLength);



% Step 3: Pad audio files to ensure equal lengths and uniform sampling rate
channels = padAudioFiles(channels, maxLength, maxSamplingFreq);
Total_BW = 7 * plotChannelSpectrum(channels, "Channel", 1);     % Calculate total bandwidth


% Display total bandwidth across all channels
fprintf('Total Bandwidth: %.2f kHz\n', Total_BW);

% Ensure the sampling frequency covers the required bandwidth
if Total_BW>=maxSamplingFreq
    %we gonna resample the audio files
    maxSamplingFreq=Total_BW;
    fprintf('Max Sampling Frequency: %.2f kHz\n', maxSamplingFreq);
    channels = padAudioFiles(channels, maxLength, maxSamplingFreq);

    %check the new max length and frequency
    [maxDuration, maxSamplingFreq, maxLength] = getMaxAudioInfo(channels);
end
```

```
% Save padded audio files
channels=saveChannelsAsWav(channels, "ch_pad", "Channels\Padded");
plotChannelSpectrum(channels, "Padded Channel", 1);         % Visualize padded signals


% Step 4: AM Modulation (DSB-SC)
AM_Modulated_Signal = AM_Modulate_DSB_SC(channels, maxLength, maxSamplingFreq );


% Optional Step: Add Noise
Noise = input('Do you Want to add noise?\n Yes: y  No: anything else \n', 's');
if Noise == "y"
    % Get user input SNR
    SNR_dB = input(["How Much SNR (db) ?\n"]);

    %add the noise
    fprintf('Adding Noise ...\n');
    Fs=ceil(1000*maxSamplingFreq);
    AM_Modulated_Signal = addNoiseToAudio(AM_Modulated_Signal, SNR_dB, Fs);
end


% Save and plot the modulated signal
AM_Modulated_Signal=saveChannelsAsWav(AM_Modulated_Signal, "ch_AM", "Channels\AM");
plotChannelSpectrum(AM_Modulated_Signal, "AM DSB SC", 0);


% Step 5: RF Stage - Select a Channel
% Filter parameters
Fc = 100e3 ;                        % Carrier frequency in Hz
fDelta=50e3;                        % Channel spacing in Hz
Fs=ceil(1000*maxSamplingFreq);      % Sampling frequency in Hz
Channel_BandWidth = 40e3;            % Bandwidth of each channel in Hz


% Signals Parameters
maxChannelNumber = length(channels);


% Filter to get the channel desired
[AM_Modulated_Signal_RF_Filter,ChannelNumber, Channel_Frequency, RF_BPF] = ...
    choose_channel(AM_Modulated_Signal,maxChannelNumber, Fc, fDelta, ...
                Channel_BandWidth,Fs);


% Optional Step: Remove RF Filter
RF_Filter =  input("Do you Want to add RF Filter?\n"+...
"Yes: y  No: anything else \n", 's'); % Specify 's' for string input
if RF_Filter == "y"
    % Visualize the new filter
    fprintf('You selected Channel %d with carrier frequency %.1f kHz.\n', ChannelNumber, Channel_Frequency /
   1e3);

    %Plot the frequency response of RF Bandpass Filter
    plotFilter(RF_BPF, Fs, "Frequency Response of RF Bandpass Filter");
else
    AM_Modulated_Signal_RF_Filter = AM_Modulated_Signal;
end


% Step 6: IF Stage
WIF = 25e3;  % Intermediate Frequency (IF) in Hz


% Optional Step: Add offset to IF
offset = input('Do you Want to add offset?\n Yes: y  No: anything else \n', 's');
if offset == "y"
    % Get user input offset
    offset_frequency = input(["What's your offset (hz) ?\n"]);

    %add the offset
    fprintf('Adding Offset ...\n');
```

```
    IF_Channel = mixer(AM_Modulated_Signal_RF_Filter, Channel_Frequency, WIF + offset_frequency,Fs);
else
    %no offset
    IF_Channel = mixer(AM_Modulated_Signal_RF_Filter, Channel_Frequency, WIF ,Fs);
end


%IF Filter
[IF_Channel_Filtered,IF_BPF]      =   ...
    if_stage(IF_Channel, WIF,Channel_BandWidth,Fs);


%Plot the frequency response of IF Bandpass Filter
plotFilter(IF_BPF, Fs, "Frequency Response of IF Bandpass Filter");


% Step 7: Baseband Stage
[Bass_Band_Channel, Bass_Band_Filter] = baseband_detection(IF_Channel_Filtered, WIF, Fs);


%Plot the frequency response of Bass_Band Low Pass Fileter
plotFilter(Bass_Band_Filter, Fs, "Frequency Response of Bass Band Low Pass Filter");


% Step 8: Save and Visualize Receiver Signals
plotReceiver(AM_Modulated_Signal_RF_Filter, IF_Channel_Filtered, Bass_Band_Channel);

%Save as Wav
AM_Modulated_Signal_RF_Filter=saveChannelsAsWav(AM_Modulated_Signal_RF_Filter, "ch_RF_Filter", "Channels\RF");
IF_Channel_Filtered=saveChannelsAsWav(IF_Channel_Filtered, "IF_Channel", "Channels\IF");
Bass_Band_Channel=saveChannelsAsWav(Bass_Band_Channel, "Bass_Band_Channel", "Channels\Bass_Band");


% Optional Step: Listening to Bass Band
Bass_Band_play = input('Do you Want to listen the received channel?\n Yes: y  No: anything else \n', 's');
if Bass_Band_play  == "y"
    playAudio(Bass_Band_Channel);
else
    %nothing
end


% End of the code
fprintf('Processing complete. All stages executed and signals saved.\n');
```

AM DSC SC MODULATION:

```
function multiplexed_Audio_Signal = AM_Modulate_DSB_SC(channels, Length, sampleFreq)
    % Function to perform AM modulation (DSB-SC) and return multiplexed signal
    % Parameters:
    %   channels - Array of audio signals (cell array of vectors)
    %   sampleFreq - Sampling frequency of the audio signals (scalar)
    % Returns:
    %   multiplexedSignal - Combined FDM signal

    % Parameters for modulation
    baseCarrierFreq = 100; % Base carrier frequency in kHz
    deltaFreq = 50;        % Frequency increment between carriers in kHz

    % Time vector based on the sample frequency and signal length
    %Length = max(cellfun(@length, channels)); % Ensure all signals are the same length
    Ts=1/sampleFreq;    %Sample Time
    t=[0:Ts:Length*Ts-Ts];   %Time vector

    % Initialize the multiplexed signal
    multiplexedSignal = zeros(1, Length);

    % Loop through each channel and modulate it
    for n = 1:length(channels)
        % Get the current channel signal
        signal = channels(n).AudioData;

        % Carrier frequency for this channel
        carrierFreq = baseCarrierFreq + (n-1) * deltaFreq;

        % Generate the carrier signal
        carrier = cos(2 * pi * carrierFreq * t)';

        % Perform DSB-SC modulation
        modulatedSignal = signal .* carrier;

        % Add the modulated signal to the multiplexed signal
        multiplexedSignal = multiplexedSignal + modulatedSignal';
    end

    %same information
    multiplexed_Audio_Signal                    = AudioFile("Channels\Padded\ch_pad_3.wav");
    multiplexed_Audio_Signal.SamplingFrequency = sampleFreq;
    multiplexedSignal=transpose(multiplexedSignal);
    multiplexed_Audio_Signal.AudioData=multiplexedSignal;
end
```

Mixer:

```
function Output_Audio_File = mixer(Input_Audio_File, Wc, WIF,Fs)
    % Function to simulate a mixer
    % Parameters:
    %   InputSignal - The input signal to be mixed
    %   omega_c     - Carrier frequency in radians/sec
    %   omega_IF    - Intermediate frequency (IF) in radians/sec
    %   Fs          - Sampling frequency in Hz
    %
    % Output:
    %   OutputSignal - The mixed output signal

    Output_Audio_File   =   Input_Audio_File;  %to have the same information

    % Time vector based on the input signal length and sampling frequency
    Length=length(Input_Audio_File.AudioData);
    Ts=1/Fs;    %Sample Time
    t=[0:Ts:Length*Ts-Ts];   %Time vector

    % Generate the carrier signal with frequency (omega_c + omega_IF)
    CarrierSignal = cos(2 * pi * (Wc + WIF) * t)';

    % Perform the mixing (multiplication)
    OutputSignal = Input_Audio_File.AudioData .* CarrierSignal;

    % Update the mixed object's properties
    Output_Audio_File.AudioData = OutputSignal;
    Output_Audio_File.player = audioplayer(OutputSignal, Fs); % Update player

end
```