Cairo University - Faculty Of Engineering
Computer Engineering Department
Communication Engineering - Fall 2024

Computer Engineering Project

Frequency Division Multiplexing using SSB

Name	ID
Asmaa Mohamed	9220154
Shehab Khaled	9220393

Presented to

Dr. Michael Melek
Eng. Mohammed Khaled

Contents

Chapter 1

Problem Description

1.1 Overview

The objective of this task is to process and analyze audio signals using various signal processing techniques. The main steps are as follows:

- Record three audio samples using a suitable sampling frequency.
- Apply a Low-Pass Filter (LPF) to limit the frequency of the samples.
- Use Single Sideband (SSB) modulation to modulate the samples onto different carrier frequencies while satisfying the Nyquist theorem.
- Perform SSB demodulation to recover the original samples.

1.2 Objective

The goal is to validate the process of recording, filtering, modulating, and demodulating audio signals using practical and theoretical signal processing principles.

Chapter 2

Recording Test Samples

2.1 Recording

For this task, we recorded three audio samples, each with a duration of 10 seconds. The chosen sampling rate was 48,000 Hz due to its advantages:

- It allows for a wide range of carrier frequencies that satisfy the Nyquist theorem.
- It is a standard sampling frequency in many audio and communication applications.
- It satisfies the efficient storage and bandwidth requirements.

These test samples serve as the basis for the subsequent filtering, modulation, and demodulation processes.

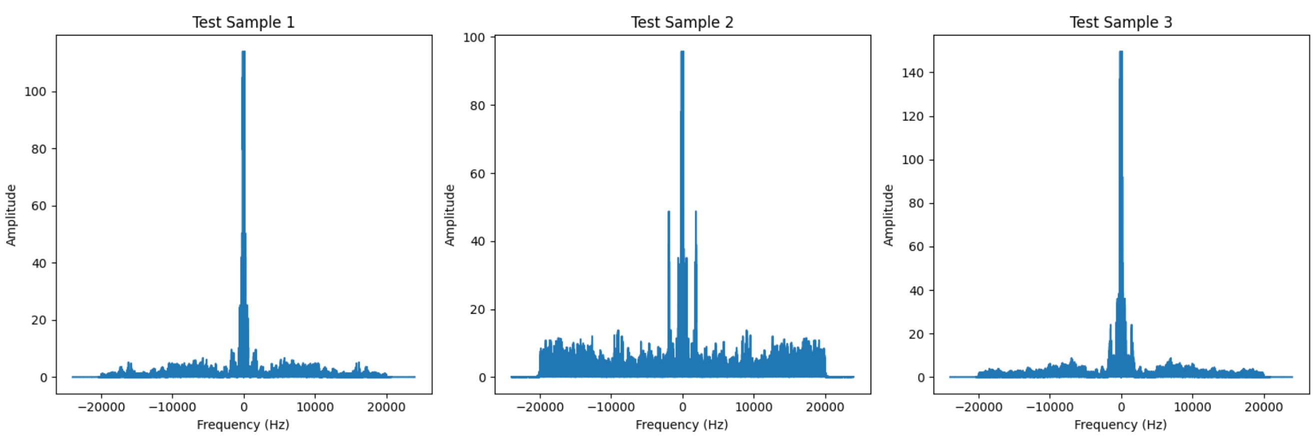

Figure 2.1: Input Signals

Chapter 3

Filtering Test Samples

3.1 Filtering

For this task, I applied a low pass filter (LPF) to the recorded audio samples to limit their frequency content. The goal was to remove high-frequency noise while retaining the main components of the audio signals. I considered two types of filters for this operation:

- Chebyshev Type I Filter (Cheby1)
- Butterworth Filter (Butter)

I chose the **Chebyshev Type I filter** because of its sharper roll-off compared to the Butterworth filter. However, the Chebyshev filter introduces a ripple in the pass-band, which can be controlled by adjusting the filter parameters.

3.2 Choosing the Cutoff Frequency

I tested different cutoff frequencies, as follows:

- Freq = 4 kHz
- Freq = 3.75 kHz
- Freq = 3.5 kHz

I found that the results for clarity and detail were similar across all tested cutoff frequencies. Ultimately, I settled on **3500 Hz** as the cutoff frequency.

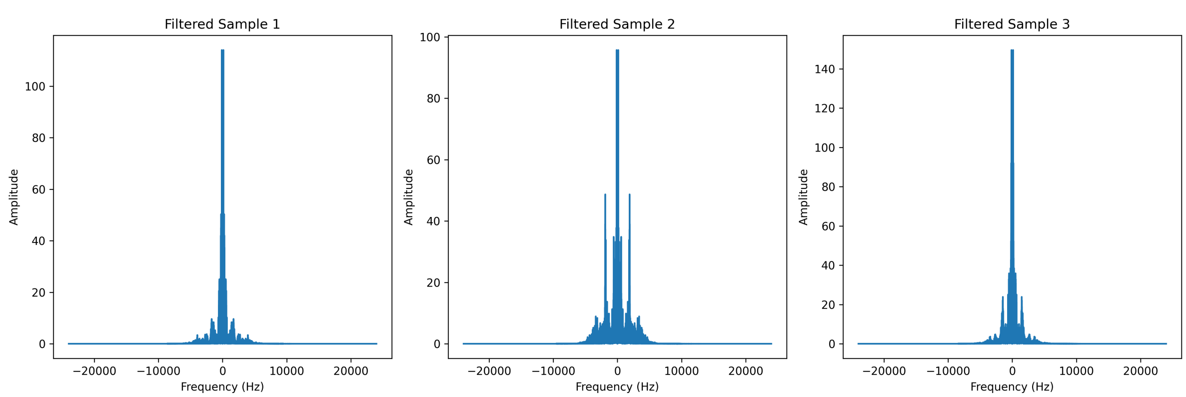

Figure 3.1: Filtered Samples

Chapter 4

Modulation & Frequency Mixing

4.1 SSB Modulation

Each sideband contains complete information of the baseband signal. So, given either sideband the baseband signal can be recovered

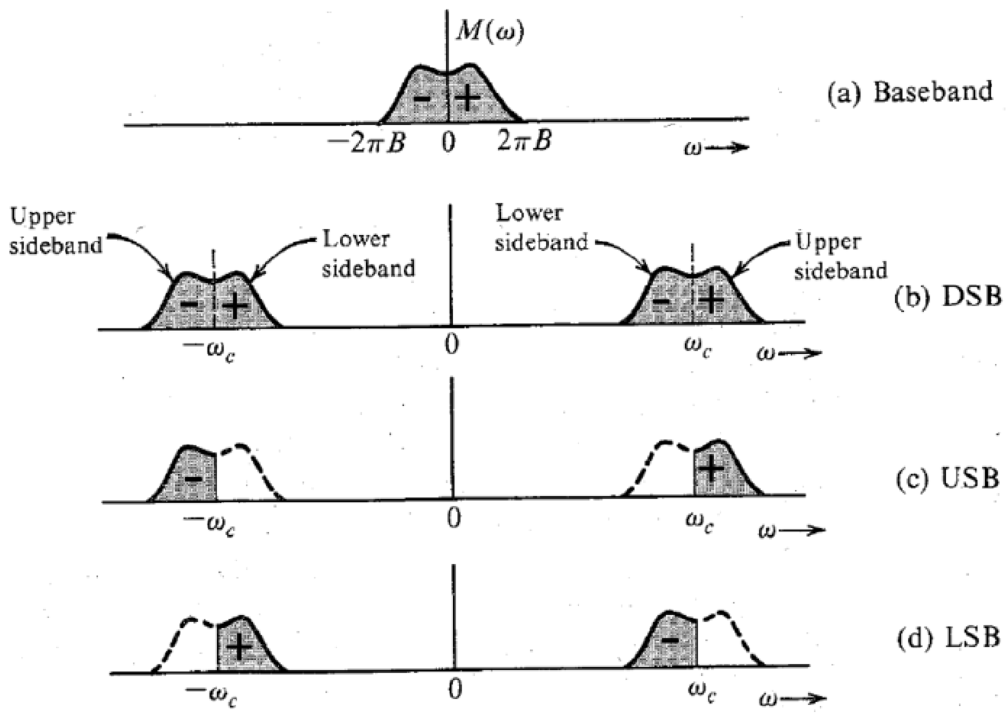

Figure 4.1: SSB Modulators Selective Filtering Method

We perform SSB modulation by first multiplying by the carrier frequency then apply a side-band filter.

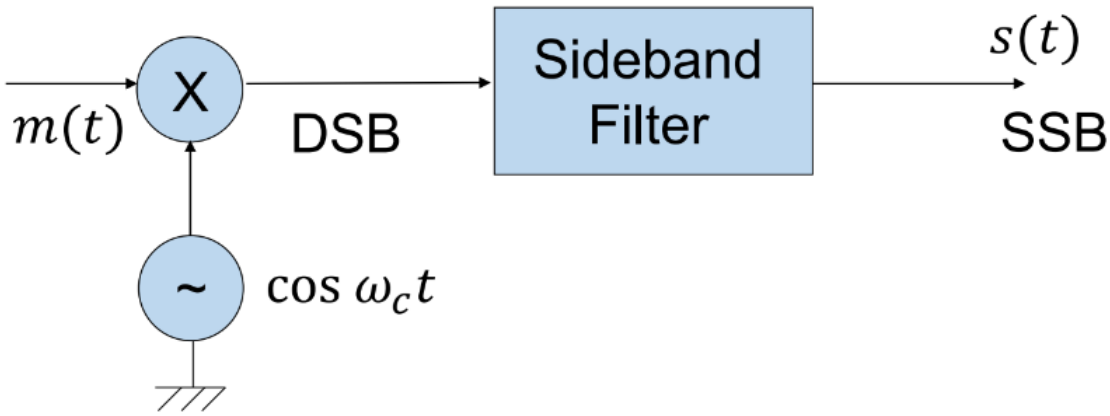

Figure 4.2: Single Side Band Modulation

After modulating the signals, we got these results:

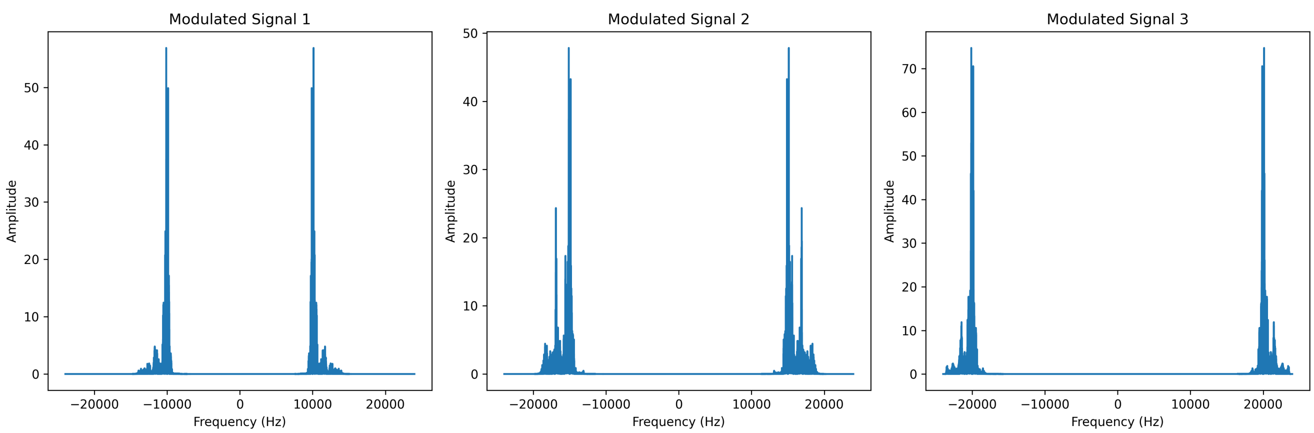

Figure 4.3: Modulated Signals

4.2 Frequency Division Multiplexing

We applied frequency division multiplexing by multiplying each signal by a different carrier frequency and then combining them.

Carrier Frequencies Chosen: **10 kHz, 15 kHz, 20 kHz**. Because

- Each signal is given a bandwidth of 5,000 Hz although each band is filtered at a cufoff frequency of 3500 but we are accounting for practical filter limitations and.
- These carrier frequencies satisfies the Nnyquist condition $f < 0.5 * f_{sampling}$

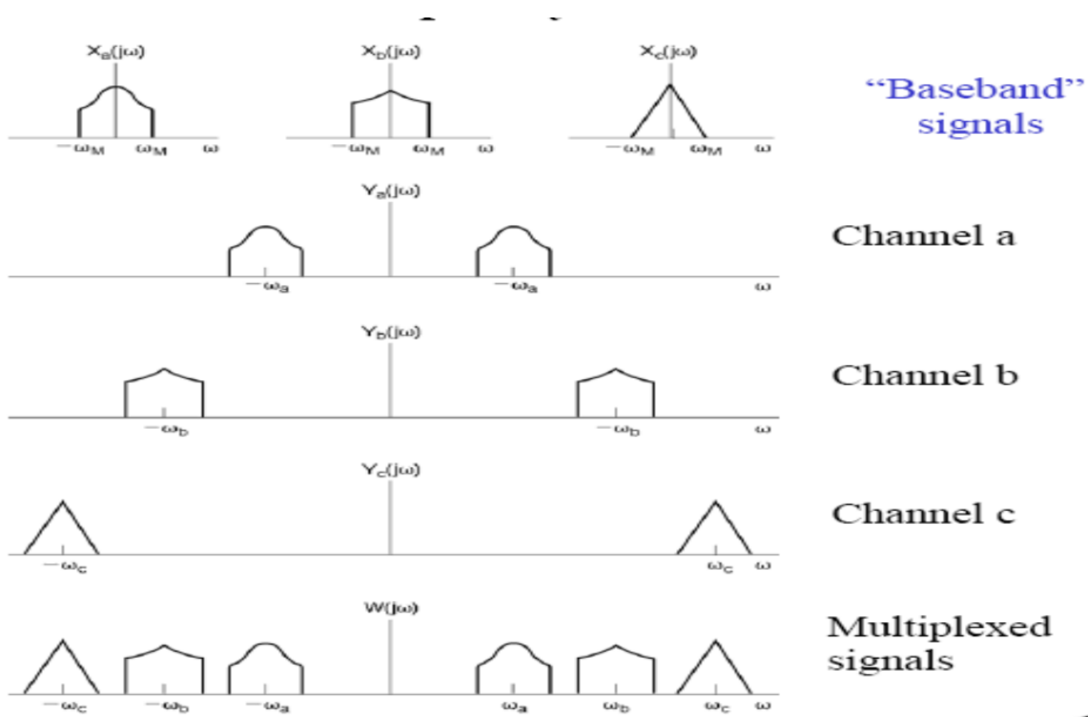

Figure 4.4: Frequency Division Multiplexing

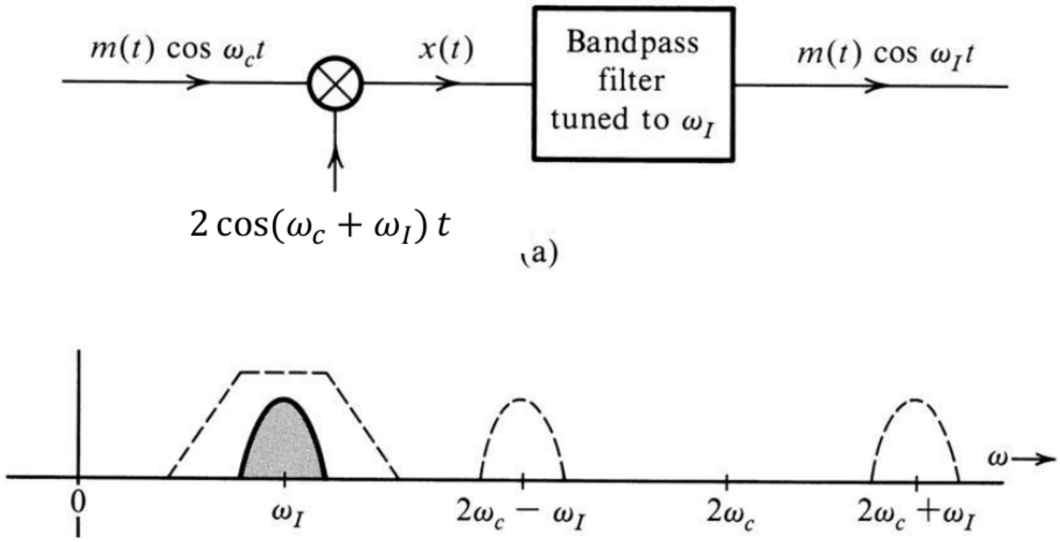

Figure 4.5: Frequency Division Multiplexing

Here is the combined signal after applying FDM:

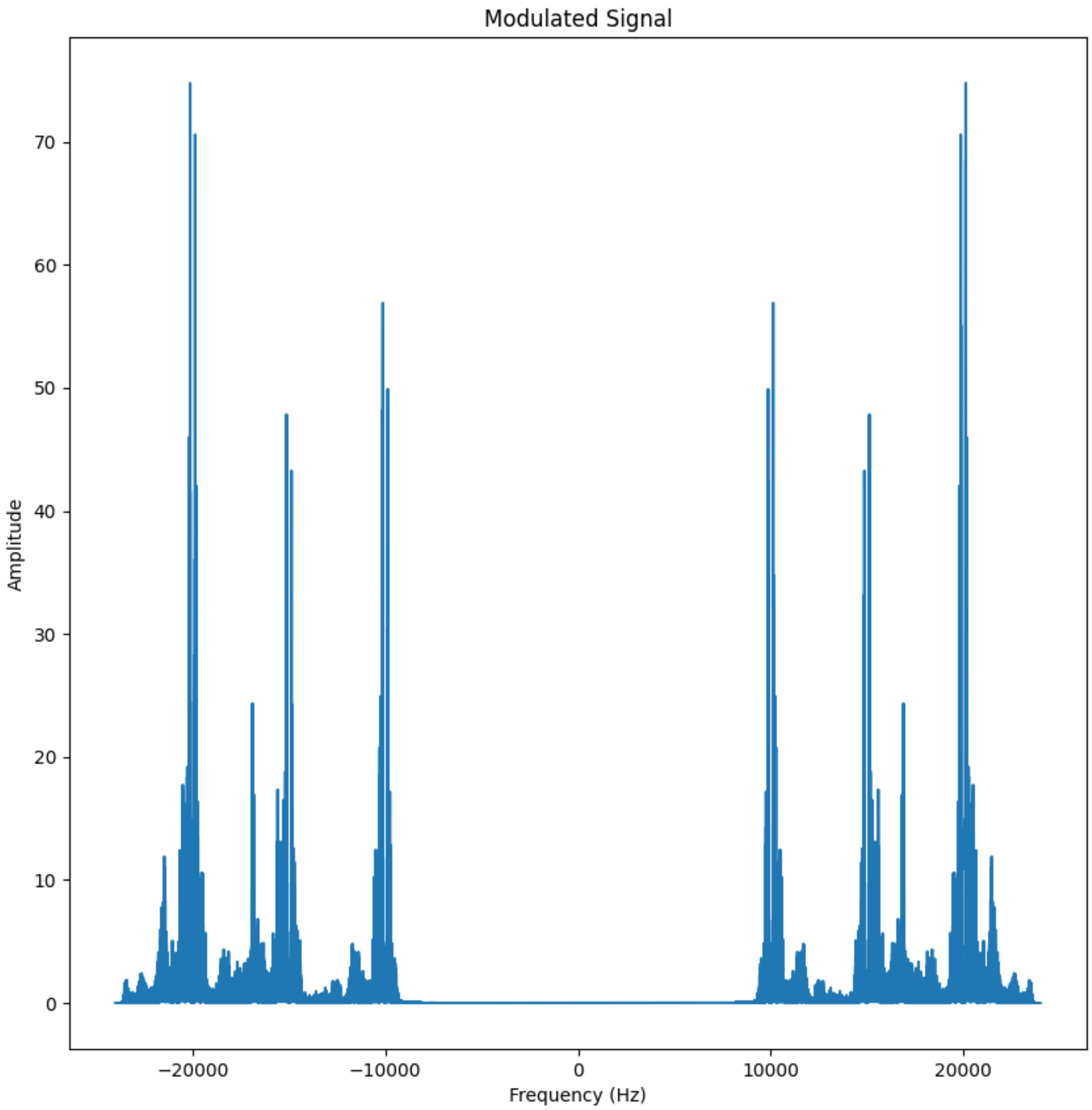

Figure 4.6: Result Signal

Chapter 5

Demodulation

Before demodulation, we first apply band-bass filter to select the signal we want to demodulate then apply demodulation.

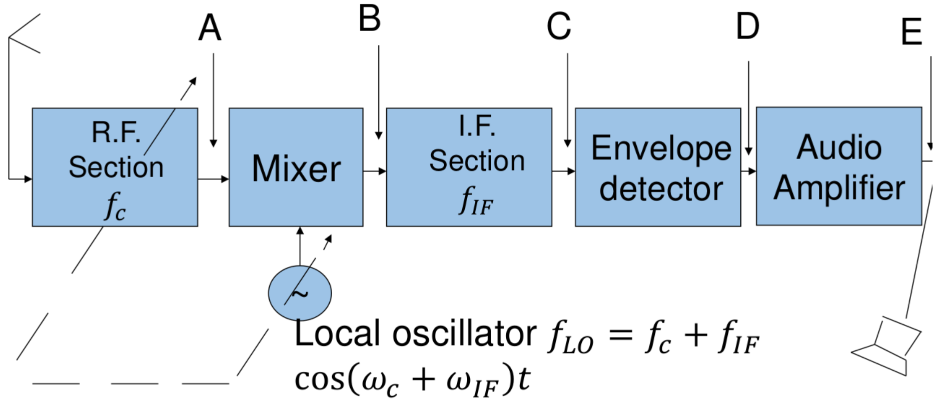

Figure 5.1: Superheterodyne receiver

We used SSB-SC in modulation which can be demodulated using synchronous demodulator as in DSB-SC.

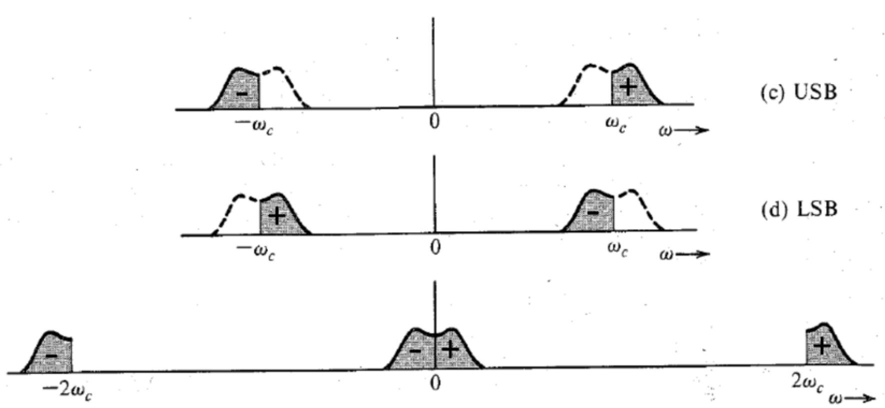

Figure 5.2: SSB Demodulation

Final result of demodulation:

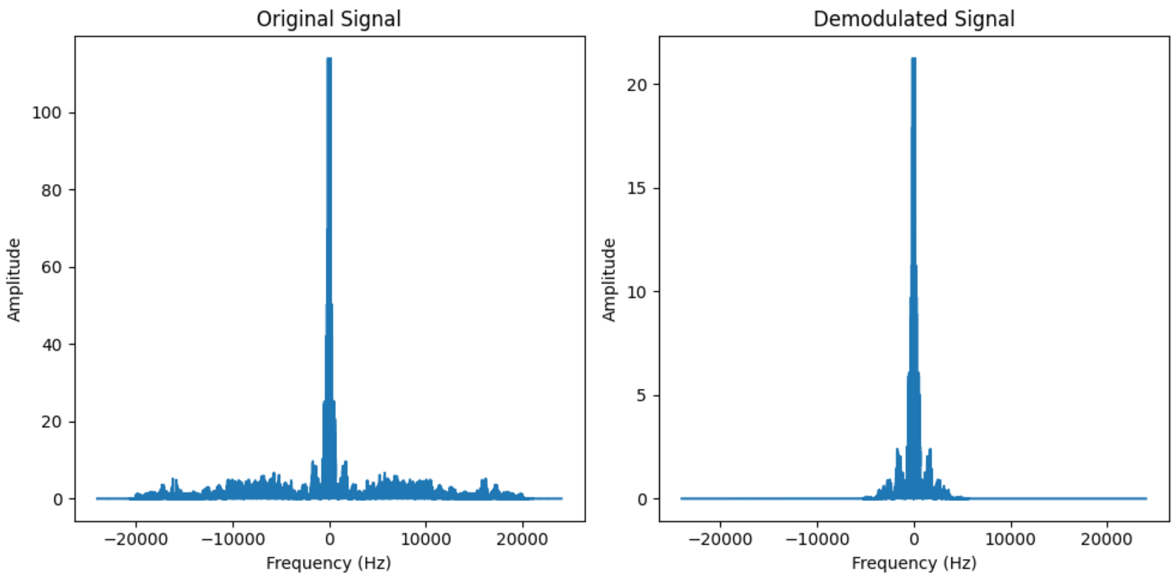

Figure 5.3: Input 1 Final Results

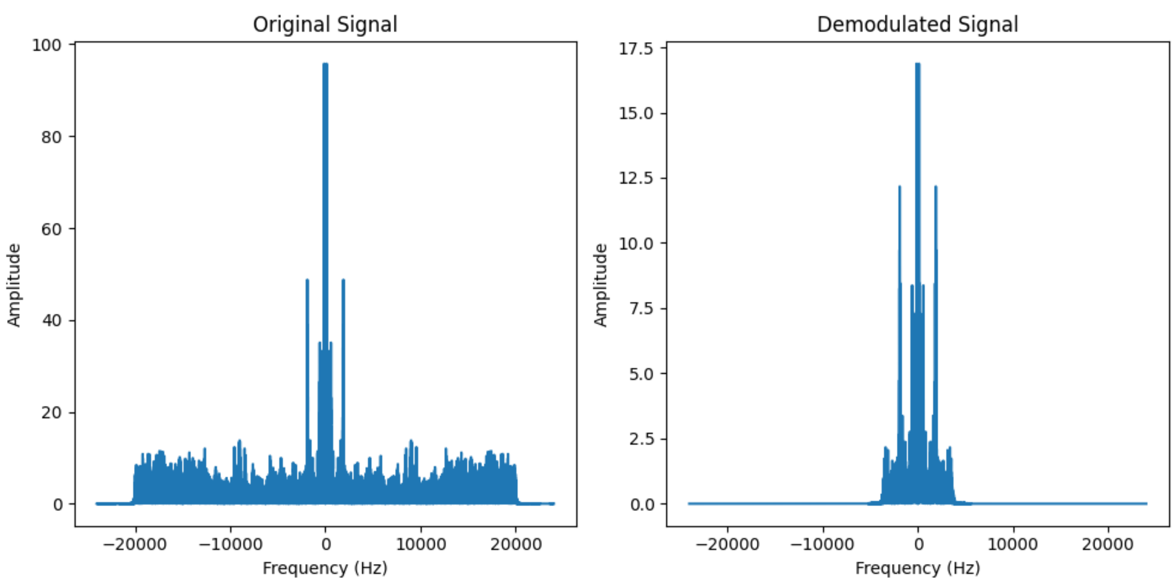

Figure 5.4: Input 2 Final Results

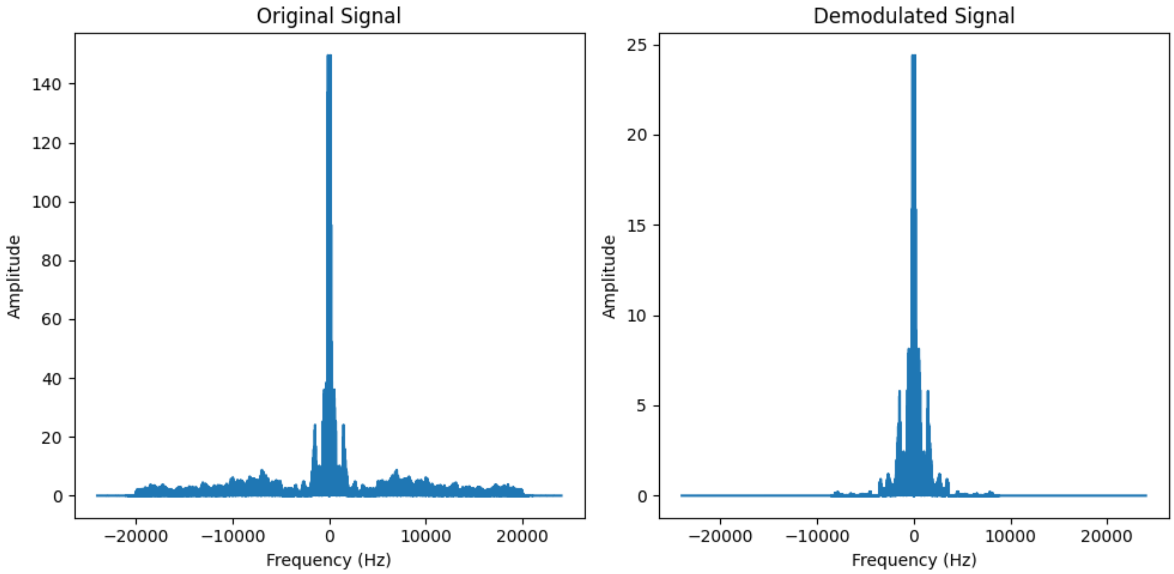

Figure 5.5: Input 3 Final Results

Appendix A

Appendix

A.1 Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
from scipy.signal import butter, cheby1, filtfilt, lfilter, ellip

def plotSignal(data, title="Waveform"):
    plt.figure()
    plt.plot(data)
    plt.xlabel('Sample Index')
    plt.ylabel('Amplitude')
    plt.title('Waveform of Test Audio')
    plt.show()

def plotSignal_Freq(data, fs, title=''):
    n = len(data)
    fft_data = np.fft.fft(data)
    fft_magnitude = np.abs(fft_data) / n
    frequencies = np.fft.fftfreq(n, d=1/fs)
    fft_magnitude_shifted = np.fft.fftshift(fft_magnitude)
    frequencies_shifted = np.fft.fftshift(frequencies)
    plt.plot(frequencies_shifted, fft_magnitude_shifted)
    plt.title(title)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude')
    plt.show()

# Low Pass Filters
def lowpass_filter_ellip(data, cutoff, fs, order=5, rp=0.1, rs=60):
    filter = ellip(order, rp, rs, cutoff, btype='lowpass', fs=fs, output='ba')
    return lfilter(*filter, data)

def lowpass_filter_butter(data, cutoff, fs, order=5):
    filter = butter(order, cutoff, btype='lowpass', fs=fs, output='ba')
    return filtfilt(*filter, data)

def lowpass_filter_ideal(data, cutoff, fs):
    n = len(data)
    fft_data = np.fft.fft(data)
    frequencies = np.fft.fftfreq(n, d=1/fs)
    fft_data[np.abs(frequencies) > cutoff] = 0
    return np.real(np.fft.ifft(fft_data))

def lowpass_filter_cheby1(data, cutoff, fs, order=5, rp=0.1):
    filter = cheby1(order, rp, cutoff, btype='lowpass', fs=fs, output='ba')
    return lfilter(*filter, data)

# Band Pass Filters
def bandpass_filter_ellip(data, begin, end, fs, order=5, rp=0.1, rs=60):
    filter = ellip(order, rp, rs, [begin, end], btype='bandpass', fs=fs, output='ba')
```

```
    return lfilter(*filter, data)

def bandpass_filter_butter(data, begin, end, fs, order=5):
    filter = butter(order, [begin, end], btype='bandpass', fs=fs, output='ba')
    return lfilter(*filter, data)

def bandpass_filter_ideal(data, begin, end, fs):
    n = len(data)
    fft_data = np.fft.fft(data)
    frequencies = np.fft.fftfreq(n, d=1/fs)
    fft_data[(np.abs(frequencies) > end) | (np.abs(frequencies) < begin)] = 0
    return np.real(np.fft.ifft(fft_data))

def bandpass_filter_cheby1(data, begin, end, fs, order=5, rp=0.1):
    filter = cheby1(order, rp, [begin, end], btype='bandpass', fs=fs, output='ba')
    return lfilter(*filter, data)

# Generic Filters
def lowpass_filter(data, cutoff, fs, filter_type='butter'):
    if filter_type == 'butter':
        return lowpass_filter_butter(data, cutoff, fs)
    elif filter_type == 'ideal':
        return lowpass_filter_ideal(data, cutoff, fs)
    elif filter_type == 'cheby1':
        return lowpass_filter_cheby1(data, cutoff, fs)
    elif filter_type == 'ellip':
        return lowpass_filter_ellip(data, cutoff, fs)
    else:
        raise ValueError('Invalid filter type')

def bandpass_filter(data, begin, end, fs, filter_type='butter'):
    if filter_type == 'butter':
        return bandpass_filter_butter(data, begin, end, fs)
    elif filter_type == 'ideal':
        return bandpass_filter_ideal(data, begin, end, fs)
    elif filter_type == 'cheby1':
        return bandpass_filter_cheby1(data, begin, end, fs)
    elif filter_type == 'ellip':
        return bandpass_filter_ellip(data, begin, end, fs)
    else:
        raise ValueError('Invalid filter type')

# Carrier Generation
def generate_carrier(fc, fs, duration):
    t = np.arange(0, duration, 1/fs)
    return np.cos(2*np.pi*fc*t)

# Loading Tests
def loadTests(files, FS, filter_type):
    data = []
    for file in files:
        fs, d = read(file)
        ten_seconds = 10 * fs

        if fs != FS:
            raise ValueError("Sampling rate of file does not match the desired sampling rate")

        d = d[:ten_seconds,0]
        d = lowpass_filter(d, 3500, fs, filter_type)
        data.append(d)

    return data

# SSB Modulation and Demodulation
def modulate(data, fc, fs, maxfreq, filter_type):
    carrier = generate_carrier(fc, fs, len(data)/fs)
    modulated = data * carrier
    return bandpass_filter(modulated, fc, fc + maxfreq, fs, filter_type)

def FDM(signals, frequencies, fs, filter_type):
    modulated = np.zeros(len(signals[0]))
```

```python
    for i in range(len(signals)):
        modulated_signal = modulate(signals[i], frequencies[i], fs, 3500, filter_type)
        modulated += modulated_signal
    return modulated

def demodulate(data, fc, fs, maxfreq, filter_type):
    filtered_data = bandpass_filter(data, fc, fc + maxfreq, fs, filter_type)
    carrier = generate_carrier(fc, fs, len(data)/fs)
    demodulated = filtered_data * carrier
    return lowpass_filter(demodulated, maxfreq, fs, filter_type)

# Example usage
if __name__ == "__main__":
    FS = 48000
    filter_type = 'cheby1'

    # Load and process input files
    files = ['inputs/input1.wav', 'inputs/input2.wav', 'inputs/input3.wav']
    signals = loadTests(files, FS, filter_type)

    # Modulation
    frequencies = [10000, 15000, 20000]
    modulated = FDM(signals, frequencies, FS, filter_type)

    # Plot modulated signal
    plt.figure(figsize=(10,10))
    plotSignal_Freq(modulated, FS, 'Modulated Signal')
    plt.savefig('outputs/modulated_signal.png')


    GAIN = 4
    # Demodulation
    signal1 = GAIN * demodulate(modulated, frequencies[0], FS, 3500, filter_type)
    signal2 = GAIN * demodulate(modulated, frequencies[1], FS, 3500, filter_type)
    signal3 = GAIN * demodulate(modulated, frequencies[2], FS, 3500, filter_type)

    plotSignal_Freq(signal1, FS, 'Demodulated Signal 1')
    plotSignal_Freq(signal2, FS, 'Demodulated Signal 2')
    plotSignal_Freq(signal3, FS, 'Demodulated Signal 3')

    # Save output files
    write('outputs/output1.wav', FS, signal1.astype(np.int16))
    write('outputs/output2.wav', FS, signal2.astype(np.int16))
    write('outputs/output3.wav', FS, signal3.astype(np.int16))
```

Listing A.1: project code