Cairo University   Faculty Of Engineering

Computer Engineering Departement
Communication Engineering Course
Dr Magdy Mohamed
Dr Michael Melek

SSB_FDM_System

20th December 2024

OVERVIEW

The project follows a series of steps, starting with the recording of speech signals at a sampling frequency of 48 kHz. These signals are then filtered using a low-pass filter with different cutoff frequencies to determine the most suitable value for maintaining audio quality. Afterward, SSB modulation is applied, followed by the demodulation process. The magnitude spectra of the signals are plotted at various stages to visualize the effect of filtering, modulation, and demodulation.

SPECIFICATIONS

Stage one (recording audios):

We have chosen a sampling frequency of 16 kHz and the duration is 10 seconds as specified in the project document.
A sampling frequency of 16 kHz was chosen for this project to ensure accurate signal representation and prevent aliasing, as it meets the Nyquist criterion for signals up to 8 kHz. This rate provides a good balance between audio quality and data efficiency, making it suitable for our audio processing needs.

Stage two (filter the recorded audios using LPF):

To ensure that the speech signals do not contain frequency components that are unnecessary for transmission, a low-pass filter (LPF) is designed and applied to each of the three recorded audio files. The LPF is used to limit the frequency range of the speech signals to a suitable bandwidth, which reduces the amount of high-frequency noise and unnecessary information, thus improving the efficiency of modulation and transmission.
We applied different cutoff frequencies (3000 Hz, 4000 Hz, and 5000 Hz) to filter the audio and listened to the results. The 4000 Hz cutoff was chosen because it effectively reduced high-frequency noise while maintaining the clarity and natural quality of the speech. Frequencies beyond 4000 Hz are less critical for speech intelligibility, and the 4000 Hz filter provided a good balance between removing unnecessary content and preserving speech quality. Lower cutoff frequencies, like 3000 Hz, caused the speech to sound muffled, making 5000Hz the best choice.

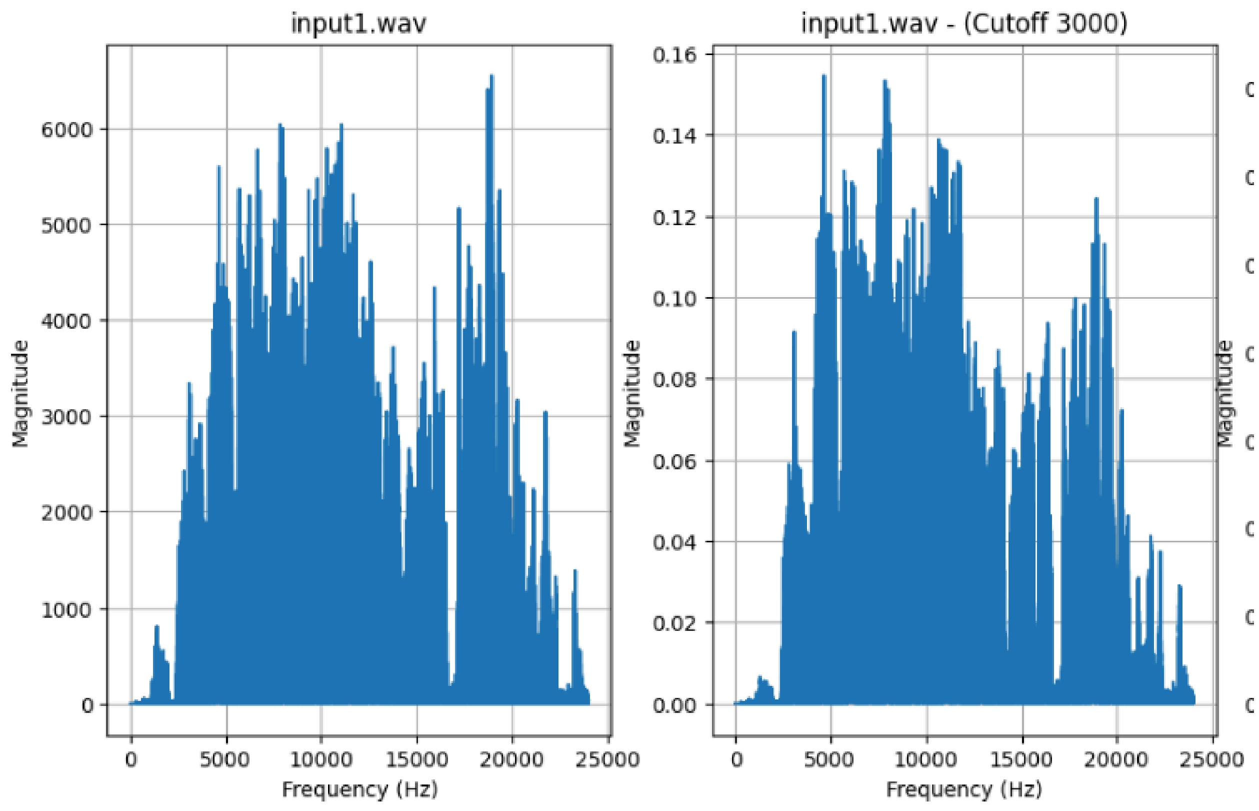

Figure 0: input 1 wav and input 1 after cut-off freq = 4000 hz is applied

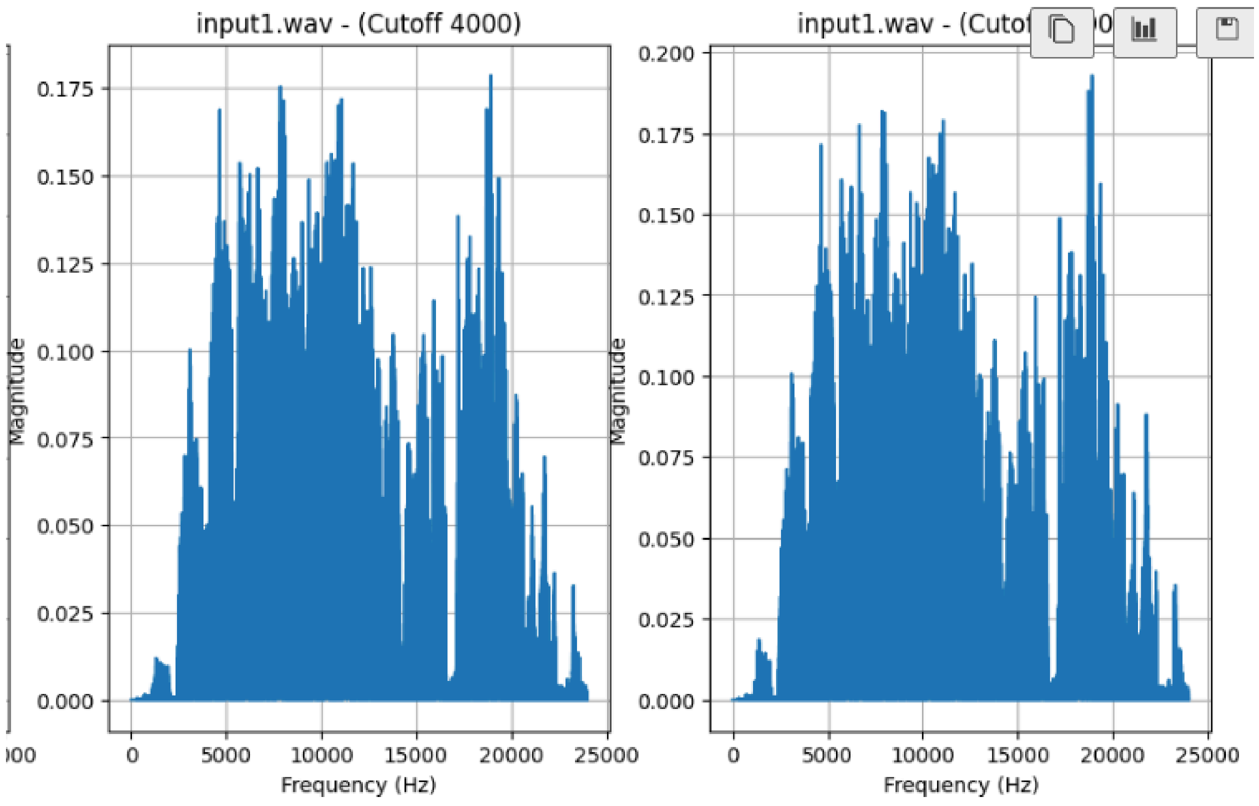

Figure 1: input 1 after cut-off freq = 5000hz is applied and input 1 after cut-off freq = 6000hz is applied

---

Stage three and four (performing SSB modulation):

We perform Single Sideband (SSB) modulation to combine the three audio signals into a Frequency Division Multiplexing (FDM) system. Appropriate carrier frequencies are selected for each signal to prevent overlap and ensure efficient transmission, based on the bandwidth of the audio signals. Each filtered audio signal is then multiplied by a cosine wave at the corresponding carrier frequency, modulating the audio onto the carrier. A bandpass filter isolates the upper sideband (LSB) of the modulated signal, removing the upper sideband while retaining the necessary information. Finally, the magnitude spectrum of the modulated signals is plotted before and after filtering to confirm proper isolation of the lower sideband.

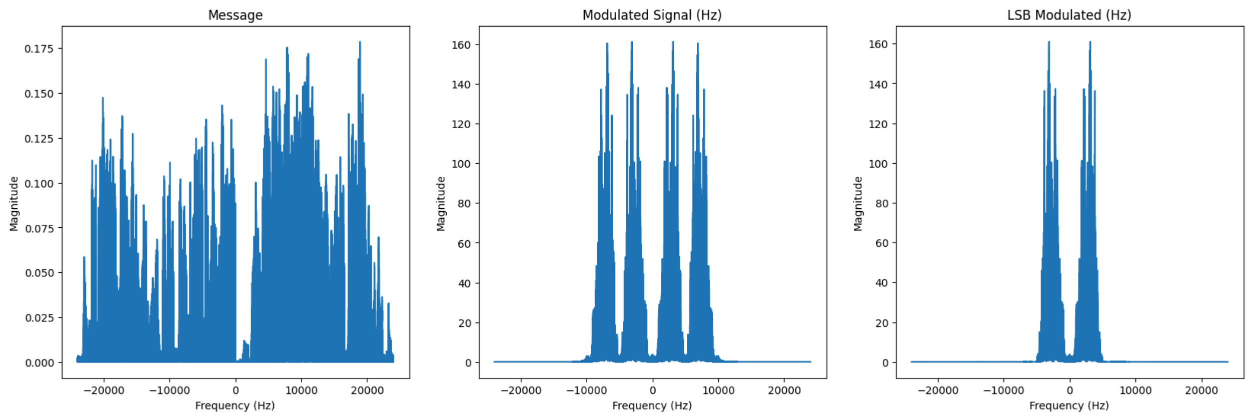

Figure 2: input 1 after modulation and after applying the LPF to extract the LSB

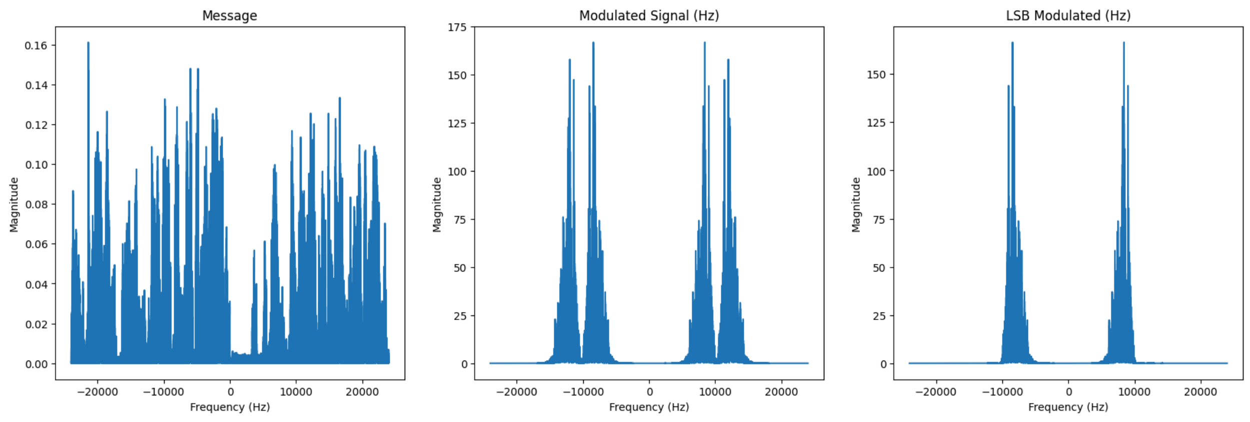

Figure 3: input 2 after modulation and after applying the LPF to extract the LSB

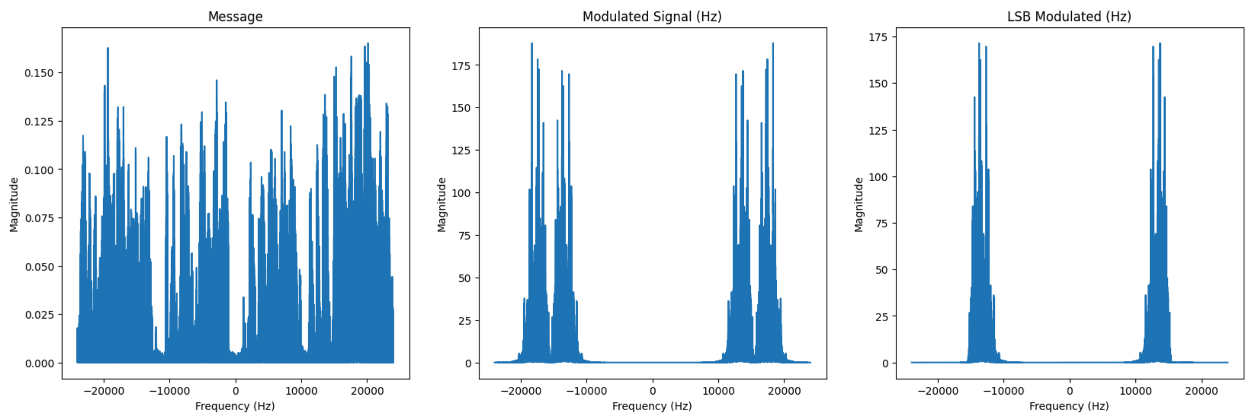

Figure 4: input 3 after modulation and after applying the LPF to extract the LSB

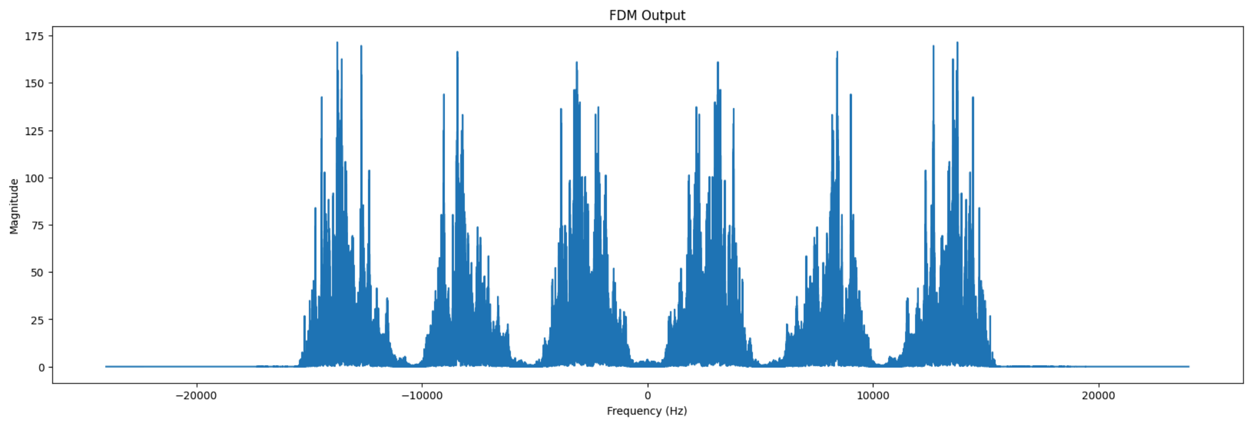

Figure 5: final FDM output

Stage five (performing SSB demodulation):

We perform Single Sideband (SSB) demodulation to recover the original audio signals. Each modulated signal is multiplied by a cosine wave of the carrier frequency used during modulation. This process effectively shifts the signal back to baseband. A low-pass filter is then applied to remove any high-frequency components, allowing us to recover the original speech signals. The demodulated signals are saved as .wav files and their magnitude spectra are plotted to verify that the original audio content has been successfully restored. This step ensures that the modulated signals are correctly demodulated, completing the communication process.

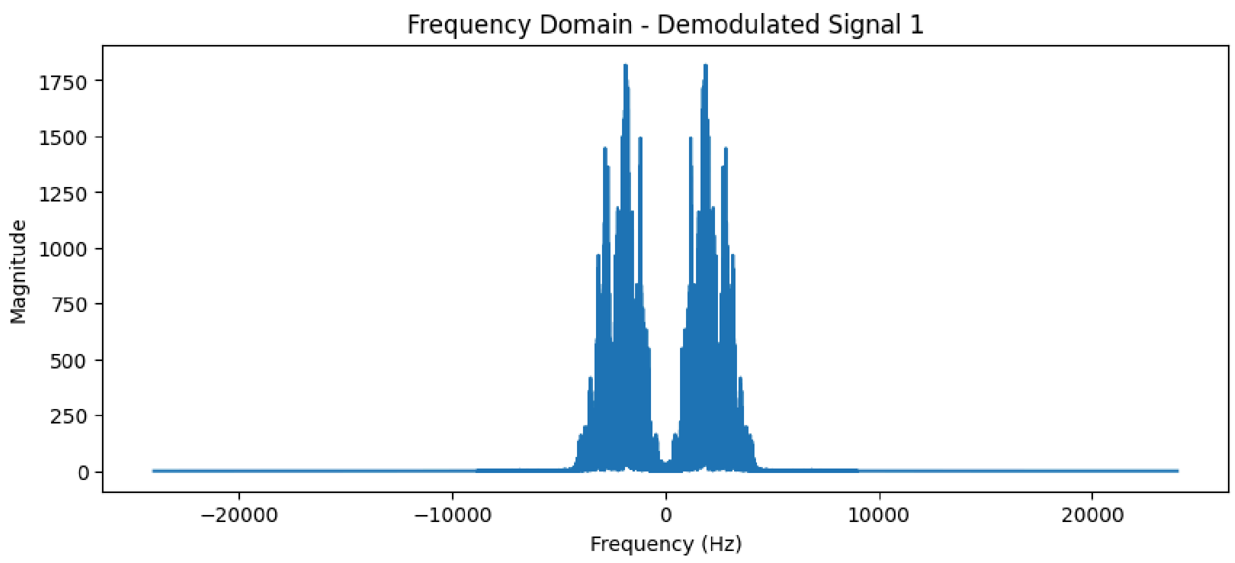

Figure 6: demodulated signal 1

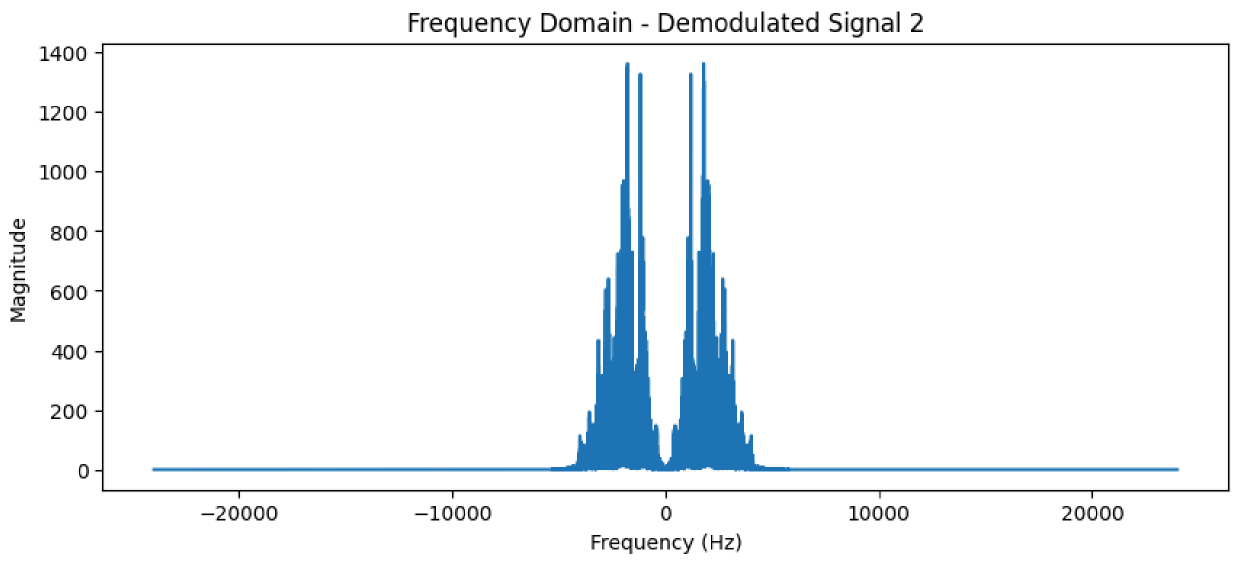

Figure 7: demodulated signal 2

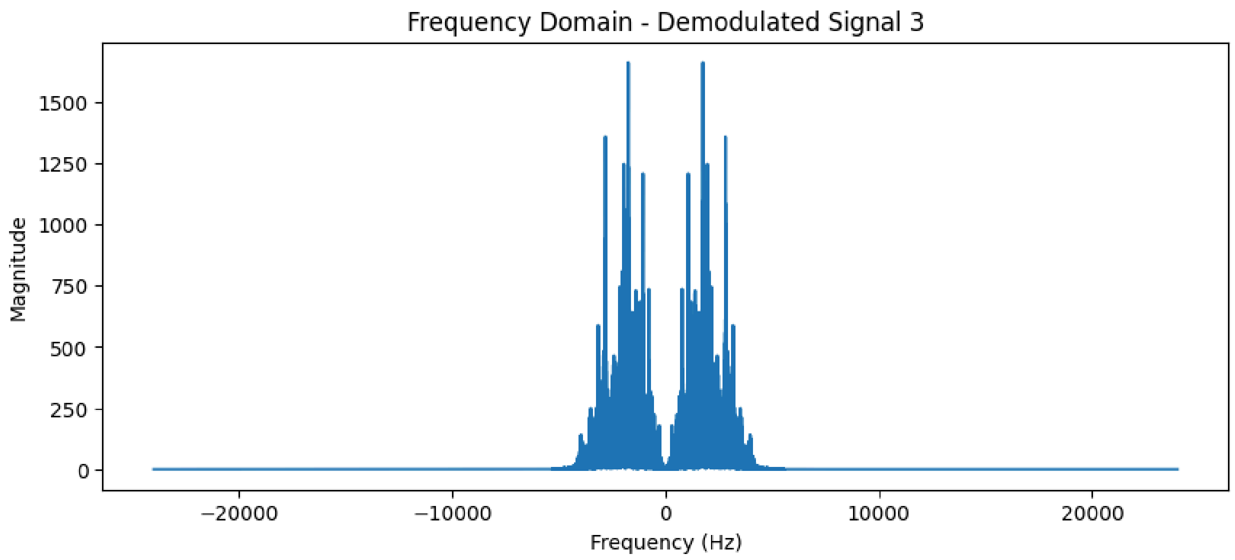

Figure 8: demodulated signal 3

Conclusion

We successfully implemented an SSB modulation and demodulation system for three speech signals using frequency-division multiplexing. Through careful filtering and modulation, we ensured minimal signal distortion and the demodulated signals were accurately restored. This project demonstrated the practical application of SSB in communications, providing a clear understanding of both the theory and implementation of the process.

Code

```
def record_audio(filename, duration, Fs):
    print("Recording...")

    audio = sd.rec(int(duration * Fs), samplerate=Fs, channels=1,
dtype='float32')

    sd.wait()  # Wait until recording is finished

    wv.write(filename, rate = Fs, data = (audio * 32767).astype(np.int16))  #
Save as WAV

    print(f"Recording saved to {filename}")
```

This function records audio for a specified duration and sample rate (Fs) using sound device. The recorded audio is then saved as a .wav file using wavio.write.

```
def design_lpf(cutoff, Fs, order=4):
    nyquist = Fs / 2
    normal_cutoff = cutoff / nyquist
    b, a = signal.butter(order, normal_cutoff, btype='low', analog=False)
    return b, a
```

This function designs a low-pass filter with a specified cutoff frequency using the scipy.signal.butter method. The cutoff is normalized by the Nyquist frequency.

```python
def plot_magnitude_spectrum(filtered_audio, audio, Fs, cutoff_frequencies,
title):
    N = len(audio)
    freqs = fftfreq(N, d=1/Fs)
    magnitude = np.abs(fft(audio))

    plt.figure(figsize=(20, 6))
    plt.subplot(1, 4, 1)

    plt.plot(freqs[:N//2], magnitude[:N//2])
    plt.title(title)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')
    plt.grid()
```

This function computes and plots the magnitude spectrum of the audio signal using the Fast Fourier Transform (FFT). It plots the frequency vs. magnitude, which is useful for visualizing the frequency content of the signal.

```python
def bandpass_filter(data, lowcut, highcut, fs, order=5):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = signal.butter(order, [low, high], btype='band')
    y = signal.filtfilt(b, a, data)
    return y
```

This function designs and applies a bandpass filter to the input signal using scipy.signal.butter. It retains frequencies within the range.

```
for index, filtered_audio in enumerate(filtered_audios):
    # initialize the carrier
    Fc = (2 + 1.5 * (index + 1)) * bandwidth
    Wc = 2 * np.pi * Fc
    carrier = np.cos(Wc * t)

    arr1 = np.array(carrier)
    arr2 = np.array(filtered_audio.reshape(-1))

    modulated_signal = (arr1) * (arr2)

    modulated_signals.append(modulated_signal)
    modulated_signal_freq = np.fft.fftshift(np.fft.fft(modulated_signal))  ##
DSB


    ## Now we must get the single sideband
    filter_order = 500
    cutoff_freq1 = 0.1
    cutoff_freq2 = Fc
    sample_rate = Fs


    # Design the bandpass filter
    nyquist_rate = sample_rate / 2.0
    low = cutoff_freq1 / nyquist_rate
    high = cutoff_freq2 / nyquist_rate
    coefficients = signal.firwin(filter_order + 1, [low, high],
pass_zero=False)

    # Apply the filter to signal
    ssb_signal = signal.lfilter(coefficients, 1.0, modulated_signal)
    ssb_signal_freq = np.fft.fftshift(np.fft.fft(ssb_signal))


    final_signal = final_signal + ssb_signal
```

This snippet generates the SSB modulated signal by first calculating the carrier frequency (Fc) based on the bandwidth and the index of the audio signal. The carrier is a cosine wave, and the modulated signal is obtained by multiplying the carrier with the filtered audio signal. This results in the amplitude modulation of the audio signal, which forms the core of the SSB modulation process.

```
def ssb_demodulator(modulated_signal, carrier_freq, fs, bandwidth):
    t = np.arange(len(modulated_signal)) / fs
    demodulated_signal = modulated_signal * np.cos(2 * np.pi * carrier_freq *
t)

    b, a = signal.butter(4, bandwidth / (fs / 2), btype='low')  # Low-pass
filter to recover baseband
    demodulated_signal = signal.filtfilt(b, a, demodulated_signal)

    # multiply the signal by 4 to retain the signal magnitude
    demodulated_signal = demodulated_signal * 4

    return demodulated_signal
```

This function demodulates a Single Sideband (SSB) modulated signal by multiplying it with a cosine carrier wave. It then applies a low-pass filter to recover the baseband signal. The output is scaled by 4 to preserve the signal magnitude.
