Signals and Systems

Frequency Division Multiplexing

Project Report

Submitted to:

Dr. Umair Sajid Hashmi

Submitted By:

Group F BEE10A

Muhammad Ismail	258279
Kamran Naveed Syed	266897
Rehman Afzal	218058
Makhdoom Tayyab Altaf	255030
Muhammad Haris Javed	245021

Abstract

The aim of this project is to design a **Frequency Division Multiplexer** in *MATLAB*. The concepts of Modulation, Filters, Fourier analysis and Bandwidth allocation are used to design a working model, the working of which is substantiated by reconstructing four independent signals sharing a single transmission medium. Frequency-division multiplexing serves as a basis for communication in radio and television broadcastings, cable television, communications satellites and broadband DSL modems.

Theory

Frequency Division Multiplexing is a technique by which the total bandwidth available in a communication medium is divided into a series of non-overlapping frequency bands, each of which is used to carry a separate signal. This allows the simultaneous transmittance of multiple signals over a shared communication medium.

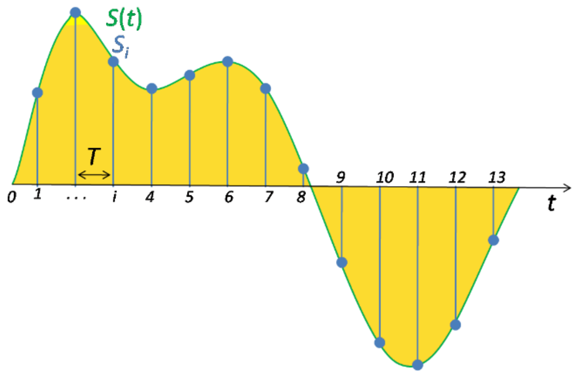
There are some key concepts which should be understood in order to make a Frequency division multiplexer:

→ Sampling
→ Fourier Transform
→ Modulation
→ Digital Filters and Filter Response

These concepts are explained in the following page.

Sampling:

In signal processing, sampling is the reduction of a continuous-time signal to a discrete-time signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal).



The original signal is retrievable from a sequence of samples, up to the Nyquist limit, by passing the sequence of samples through a type of low pass filter called a reconstruction filter. The reconstruction filter is the 3 kHz Low pass filter at the recovery stage of our project.

Discrete Fourier Transform:

Fourier analysis provides several mathematical tools for determining the frequency content of a time-domain signal. It transforms a function of time, *x(t)*, to a function of frequency, *X(ω)*, which helps us to analyze the frequency constituents of a given signal.

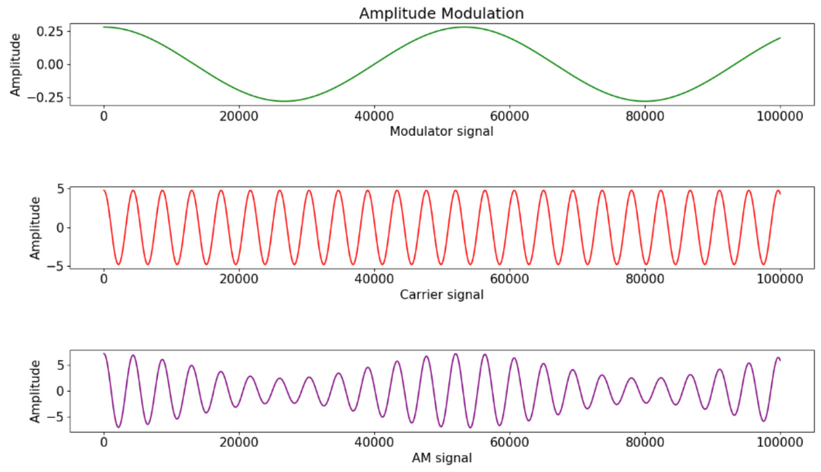
The discrete Fourier transform (DFT) is one of the most important tools in digital signal processing. It deals with a finite discrete-time signal and a finite or discrete number of frequencies, which is useful in our case as audio is analyzed digitally in discrete samples. The DFT is taken using the following formula:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn}$$

We have employed this transform to plot the magnitude of frequency responses of our audio signals in the frequency domain in order to analyze their major frequency components. With this technique, it is easier to observe the outcomes of processing the signals in each step.

Modulation

We have used amplitude modulation in this project. Amplitude modulation is a process by which a signal of frequency fs is transmitted by modulating the amplitude of the signal. This is achieved by imposing our signal on another signal called the carrier signal of frequency fc. In this way the amplitude of the signal varies in line with the instantaneous value of the intensity of the modulation. This means that the radio frequency signal has a representation of the sound wave superimposed in it. The amplitude of the modulated signal is halved.



In FDM, amplitude modulation allows us to shift the frequency of a signal to anywhere on the frequency spectrum according to the frequency of our carrier signal.

Digital Filters and Filter Response

In signal processing, a digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. In our case, they are used to block certain frequencies.

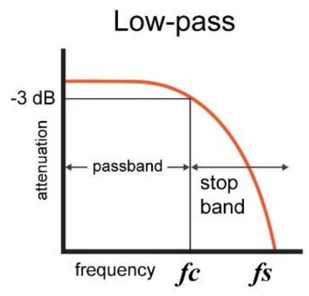
We have used **FIR** type digital filters in our project. These are specific to sampled systems.

We have also employed two types of filter responses in our project:

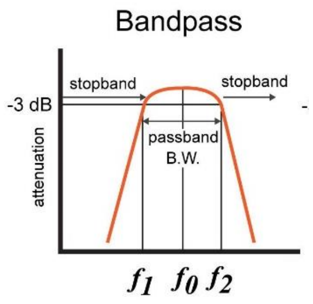
→ Low pass

→ Band pass

Low Pass Filter: Allows only low frequencies to pass through. In FDM this will restrict bandwidth of each signal and stop unwanted high frequencies from overlapping after modulation



Band pass Filter: Allows a specified range of frequencies to pass through. In FDM, these filters allow us to draw out frequency bands in the receiving end



Outline:

The ordered steps to create a Frequency Division Multiplexer we followed are as follows:

- ✓ Take 4 input sound signals from our group members, each of 10s. We plotted the spectrums of these signals in time domain, as well as frequency domain (using *SpectrumPlotter* function)
- ✓ Design a low pass filter for ***3 kHz*** and pass each signal separately through it.
- ✓ Modulate each of the filtered signal by multiplying with a unique cosine function using frequencies ***3 kHz, 9 kHz, 15 kHz and 21 kHz*** respectively.
- ✓ Add the four modulated signals. Plot frequency spectrum of their sum.
- ✓ Design four band-pass filters for ***3 kHz, 9 kHz, 15 kHz and 21 kHz*** using appropriate bandwidth. Pass the sum through each band-pass.
- ✓ Multiply the signals obtained from the bandpass filters with cosine at respective frequencies again to re-center to ***0 Hz*** and pass through Low-pass filter to retrieve original signals.

Task Allocation:

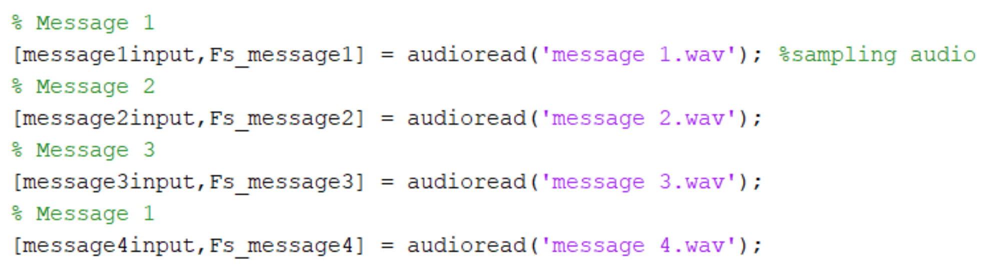
Name	Tasks Assigned
Kamran Naveed Syed	Inputting audio signals, Standardize function
Muhammad Haris Javed	Low Pass Filter Design on FDA Tool and execution of filter
Makhdoom Tayyab Altaf	SpectrumPlotter function and Modulation
Rehman Afzal	Band Pass Filter Design on FDA Tool and execution of filter
Muhammad Ismail	Demodulation, Recovery

Note: The members responsible for each task compiled their working accordingly in the *Theory* and *Procedure* sections of this report.

Procedure and Results:

Inputting Signals and standardization:

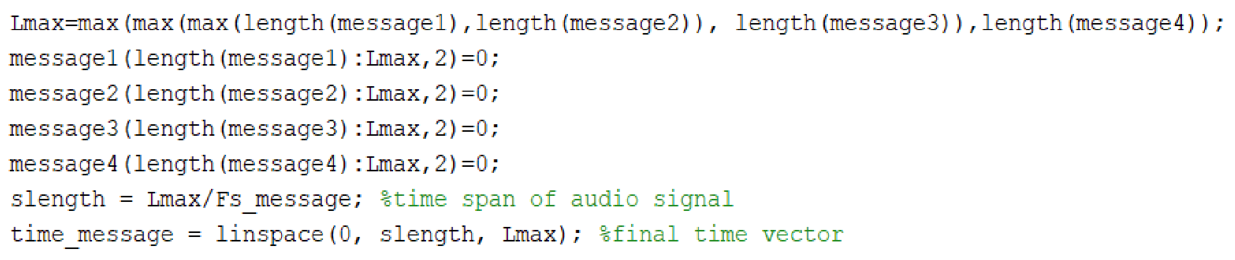
Group members were asked to record their messages for approximately 10s and the ***audioread*** command was used to input them into the code into an array containing sampled data (dubbed *message#*. for each message) and sample rate/frequency (dubbed *Fs_message#*. for each message).



```
% Message 1
[message1input,Fs_message1] = audioread('message 1.wav'); %sampling audio
% Message 2
[message2input,Fs_message2] = audioread('message 2.wav');
% Message 3
[message3input,Fs_message3] = audioread('message 3.wav');
% Message 1
[message4input,Fs_message4] = audioread('message 4.wav');
```

We then set a standard frequency, ***48 kHz***, to meet the Nyquist Criterion for this particular problem and all other sampling rates were set to this rate and messages resampled accordingly using custom-built ***Standardize*** function (details in *Problems Faced* section). It takes the preset standard frequency, sampled data and sampling frequency as arguments.

Next, to set all messages to the same length, the length of the longest message was taken as standard, and every other message was padded with zeros to achieve uniform length. This was done to ensure that all signal vectors had the same size to avoid problems during summation and modulation. A time vector was then created using this standard message length. Subsequently, our Multiplexer is able to take messages of variously differing lengths as input.

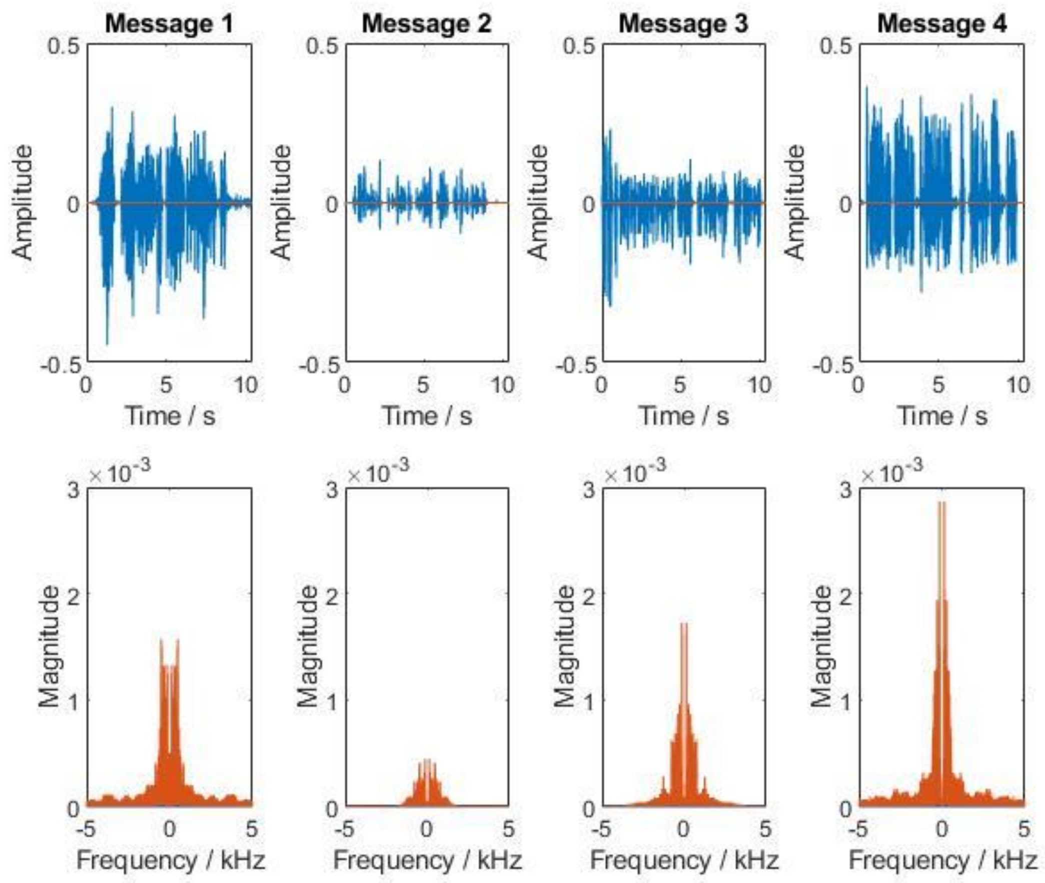


```
Lmax=max(max(max(length(message1),length(message2)), length(message3)),length(message4));
message1(length(message1):Lmax,2)=0;
message2(length(message2):Lmax,2)=0;
message3(length(message3):Lmax,2)=0;
message4(length(message4):Lmax,2)=0;
slength = Lmax/Fs_message; %time span of audio signal
time_message = linspace(0, slength, Lmax); %final time vector
```

A custom-built SpectrumPlotter function was used to represent each signal in frequency domain. It takes the sampled data and sampling frequency as arguments.

```
function SpectrumPlotter(message,frequency)
    samples = length(message); %time domain vector sample number
    Fsub = fftshift(fft(message))/samples;
    Faxis =(-samples/2:samples/2-1)*frequency/(1000*samples); %frequency vector in kHz
    plot(Faxis,abs(Fsub));
    xlabel('Frequency / kHz');
    ylabel('Magnitude');
end
```

The time domain plots, and their respective frequency domain plots below them, are shown:

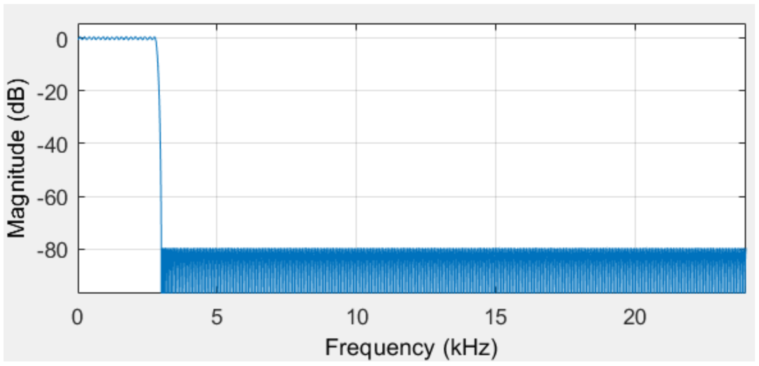


Designing and execution of Low-Pass Filter

To block out the high unwanted frequencies a Low-pass FIR filter of 3 KHz was designed using FDA (Filter and Design Analysis) tool. The parameters were set as follows:

- Cut-off Frequencies, ***Fpass*** = 3Khz and ***Fstop*** = 3.1Khz
- Sampling frequency, ***Fs*** = 48khz
- Stopband weights, ***Wpass*** = 1 and ***Wstop*** = 1
- Minimum filter order

The response of the designed filter is shown below:

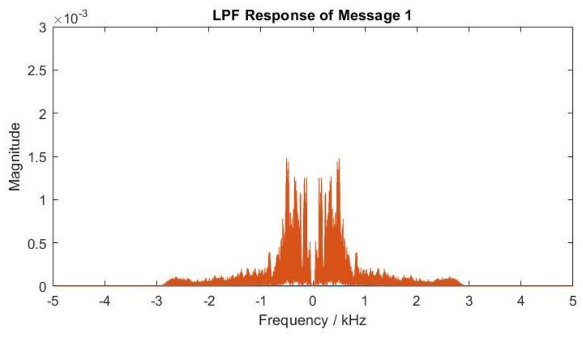


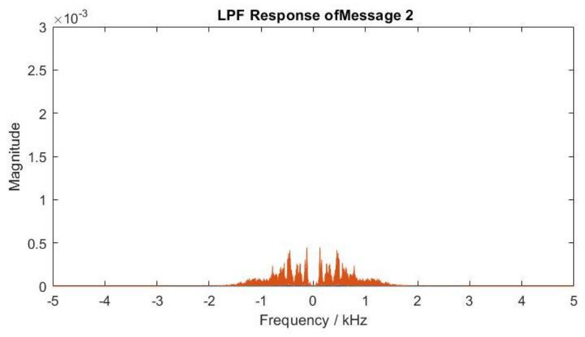
The filter order of the Low pass filter was chosen to be minimum and wasn't set to a particular value. We chose not to calculate required filter order for the filter.

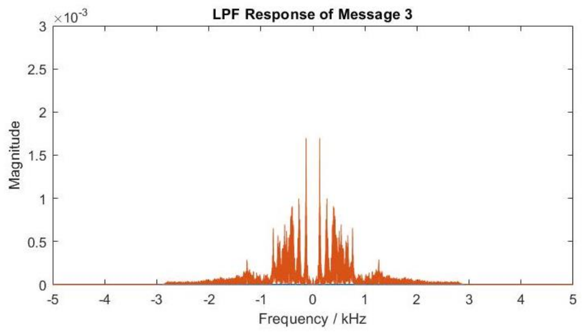
The data was exported as variable *Low_pass_3000* in a ***.mat*** file and loaded into the code. The original messages were then passed through the low pass filter using the ***filter*** command:

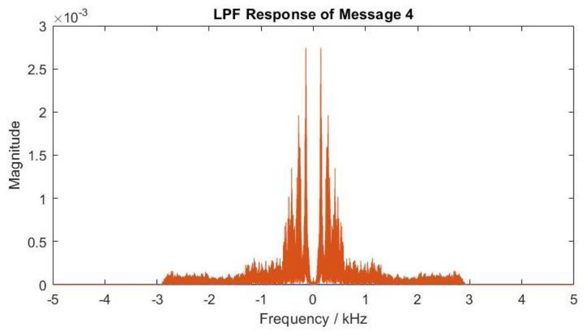
```
load('Low_pass_3000.mat'); %load filter file

lpf_message1 = filter(Low_pass_3000,1,message1); %Passing message1 through LPF
lpf_message2 = filter(Low_pass_3000,1,message2); %Passing message2 through LPF
lpf_message3 = filter(Low_pass_3000,1,message3); %Passing message3 through LPF
lpf_message4 = filter(Low_pass_3000,1,message4); %Passing message4 through LPF
```

The responses of the signals after passing through Low Pass filter are shown:







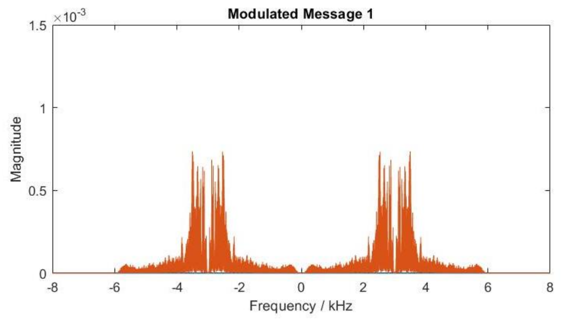


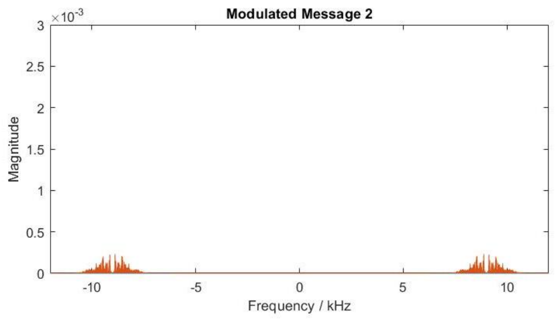
Modulation

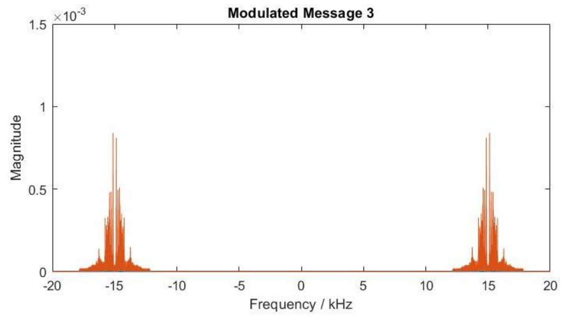
Amplitude Modulation was then achieved by multiplying each message with a cosine at unique frequency. Transpose of these cosines was taken in order to maintain matrix multiplication.

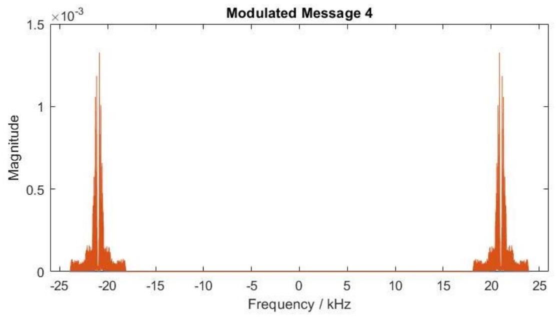
```
%taking 4 unique carriers
carrier1 = (cos(2*pi*3000*time_message)).'; %carrier signal_1 with 3kHz freq
carrier2 = (cos(2*pi*9000*time_message)).';
carrier3 = (cos(2*pi*15000*time_message)).';
carrier4 = (cos(2*pi*21000*time_message)).';
%Modulate each of the filtered signal by multiplying with a unique Cosine
%function using frequencies 3kHz, 9 kHz, 15 kHz and 21 kHz respectively
mod_message1=lpf_message1.*carrier1;
mod_message2=lpf_message2.*carrier2;
mod_message3=lpf_message3.*carrier3;
mod_message4=lpf_message4.*carrier4;
```

The spectrums of modulated signals are shown below:









It should be noted that, after modulation, the amplitude of the respective signals has been halved. This is due to the following mathematical consequence:

$$\text{modulated signal} = \text{signal} \times \text{carrier}$$

$$= \mathbf{sin} 2\pi fst \times \mathbf{cos} 2\pi fct$$

$$= \frac{1}{2}\mathbf{sin}(fc + fs)\pi t + \frac{1}{2}\mathbf{sin}(fc - fs)\pi t$$

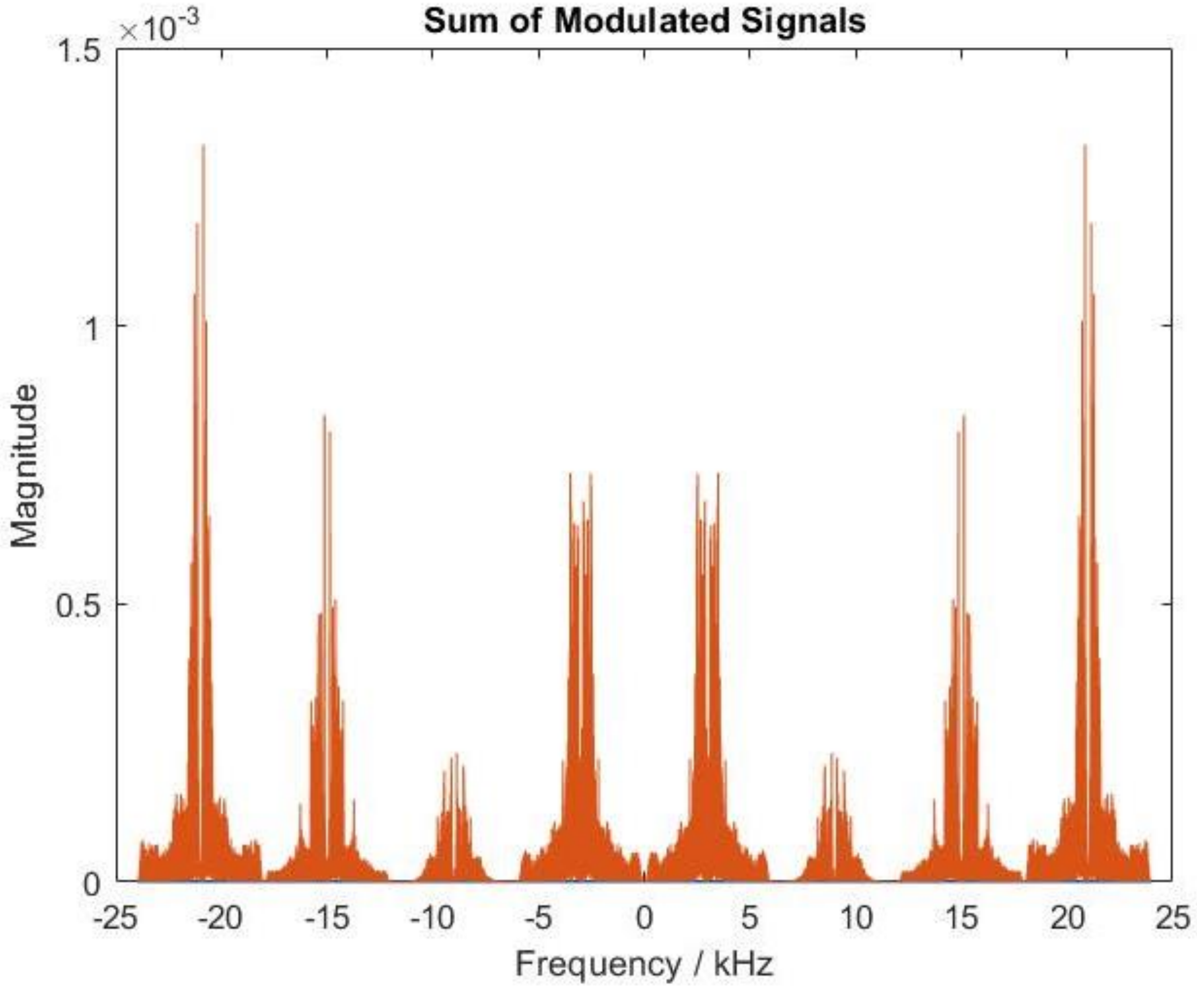
This halving of amplitude should be remembered during the recovery phase. The step of modulation also helps us to shift the frequencies of the signals to where we want on the spectrum, according to the following mathematical consequence:

Using the Fourier transform of sine: $\mathbf{sin}(2\pi fxt) \leftrightarrow \frac{1}{2}\delta(f - fx) - \frac{1}{2}\delta(f + fx)$, the corresponding frequency domain representation in hertz is thus:

$$F(jf) = \frac{1}{4}\, j\delta(f - (fc + fs)) - \frac{1}{4} j\delta(f + (fc + fs)) + \frac{1}{4} j\delta(f - (fc - fs)) - \frac{1}{4} j\delta(f + (fc - fs))$$

The modulated signals were then added, thus creating a single channel of transmission. This was the last stage in the transmission section of FDM.

The frequency spectrum of the sum was plotted is shown below:



Designing and execution of Band-Pass Filter

To obtain individual signals from the single channel of transmission, four Band-pass filters of 3 KHz bandwidth were designed in ***FDA tool*** and the sum passed through each of them. Different cut-off frequencies were used for each filter. The following parameters were common to all the filters:

- Equiripple FIR
- Minimum Filter Order
- Density Factor = 20
- Fs = 48000 Hz
- Apass = 1dB
- Astop2 = 80dB

The distinguished parameters of the band pass filters are as follows:

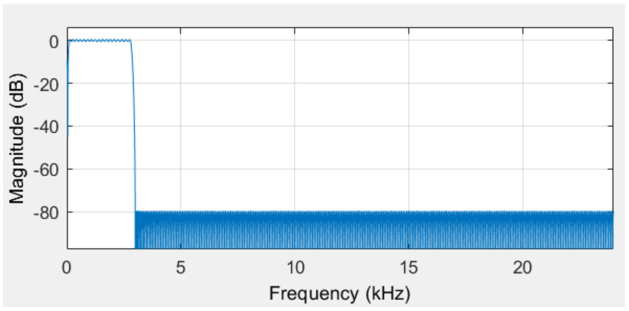
➢ **Filter for 3kHz:**
→ Astop1 = 10dB
→ Fstop1 = 1Hz
→ Fpass1 = 100Hz
→ Fpass2 = 2800Hz
→ Fstop2 = 3000Hz

➢ **Filter for 9kHz:**
→ Astop1 = 80dB
→ Fstop1 = 6kHz
→ Fpass1 = 6.2Hz
→ Fpass2 = 8.8Hz
→ Fstop2 = 9kHz

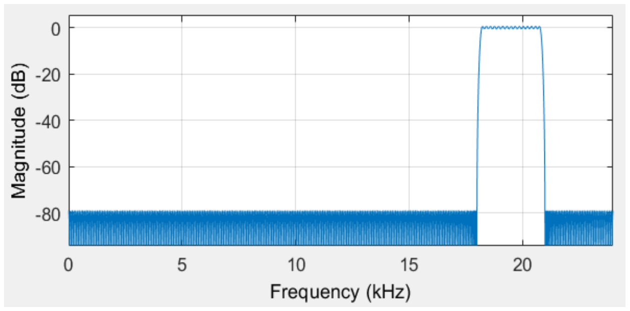
➢ **Filter for 12kHz:**
→ Astop1 = 80dB
→ Fstop1 = 12kHz
→ Fpass1 = 12.2kHz
→ Fpass2 = 14.8kHz
→ Fstop2 = 15kHz

➢ **Filter 15kHz:**
→ Astop1 = 80dB
→ Fstop1 = 18kHz
→ Fpass1 = 18.2kHz
→ Fpass2 = 20.8kHz
→ Fstop2 = 21kHz

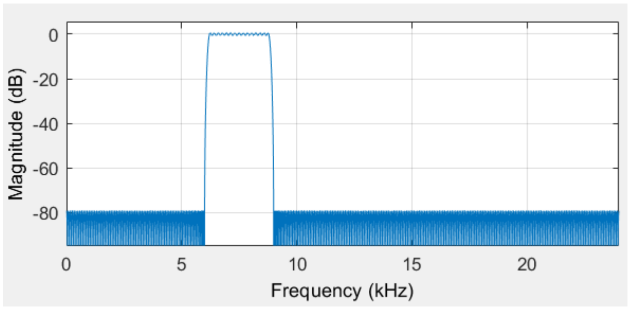
The responses of each filter are below:



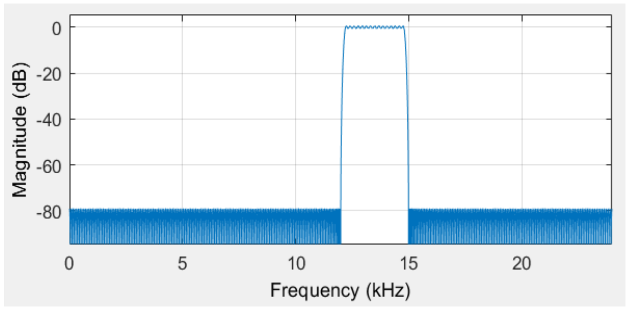
BPF for 3 kHz



BPF for 15 kHz



BPF for 9 kHz

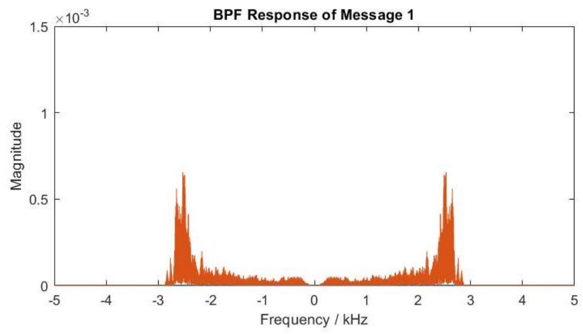


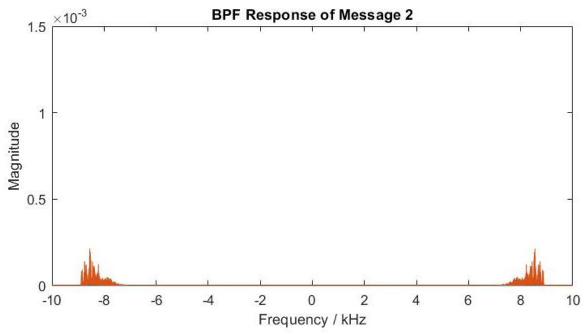
BPF for 21 kHz

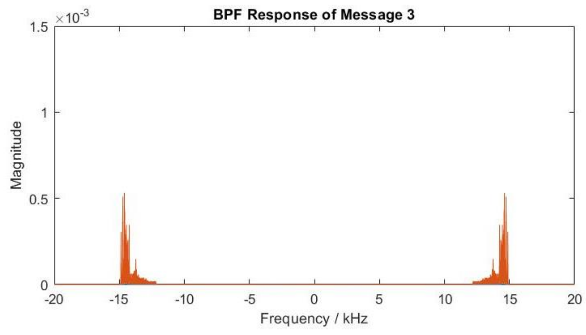
The data for each filter was exported as variable *Band_pass_# in* a ***.mat*** file and loaded into the code. The sum representing the single channel was then passed through each of them.

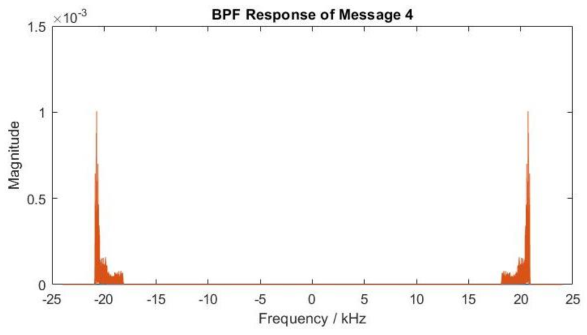
Band pass filters helped us to isolate bands of frequency according to our chosen bandwidth, which was 3 kHz single sided (details explained in the *Problems Faced* section). This is the first part of the recovery phase, since we have obtained one signal from each band pass filter.

The frequency response of the signals after passing them through Band pass filters are shown below:









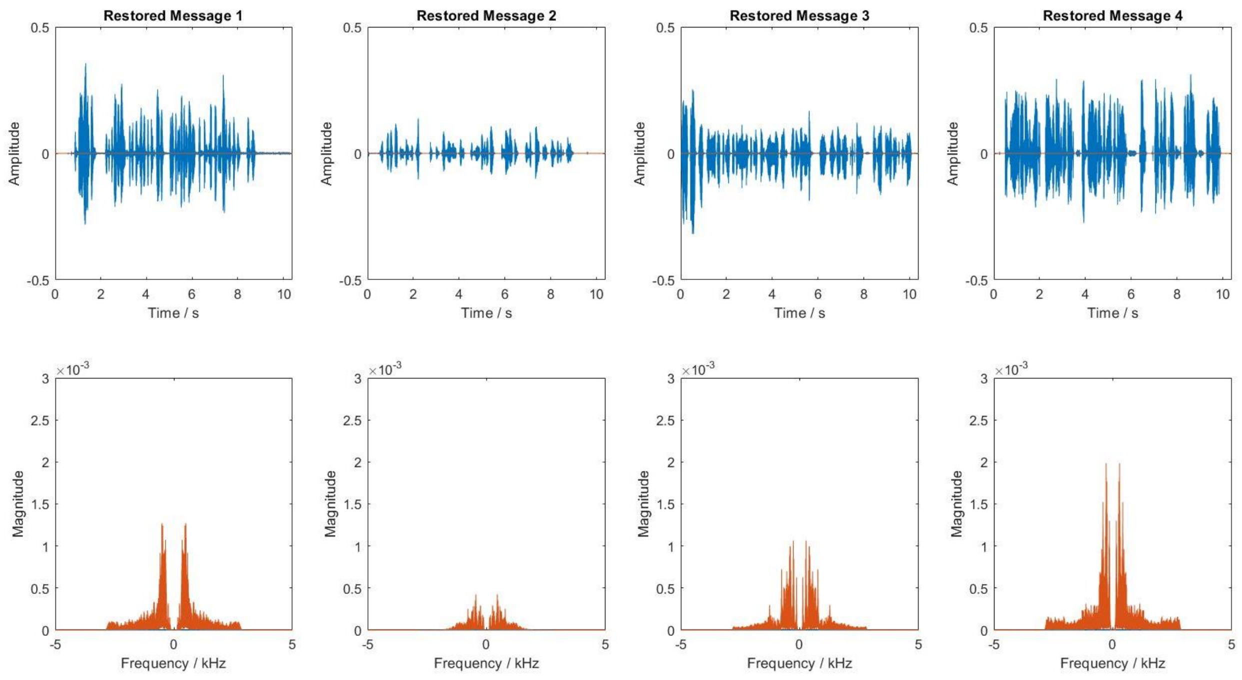
Demodulation and Recovery

The signals were then multiplied again with respective carrier signals at frequencies of ***3 kHz, 9 kHz, 15 kHz and 21 kHz*** to re-center them to 0Hz. The re-centered signals were fed through the ***3 KHz*** Low pass filter to block unwanted frequencies. Since the overall amplitude was halved each time we multiplied with the carrier signals, we had to multiply each signal by 4 to obtain the original waveforms.

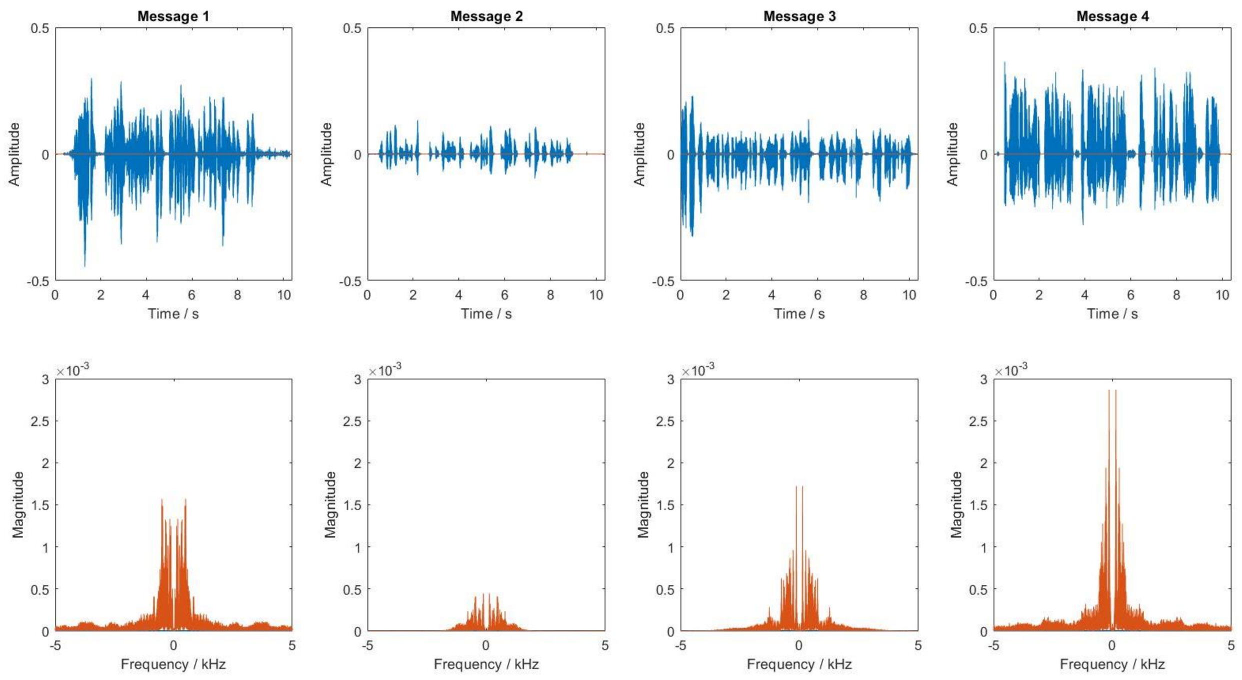
```
restored_message1 = 4*filter(Low_pass,1,recov_message1);
restored_message2 = 4*filter(Low_pass,1,recov_message2);
restored_message3 = 4*filter(Low_pass,1,recov_message3);
restored_message4 = 4*filter(Low_pass,1,recov_message4);
```

Thus, our final stage of the recovery phase is complete and the four original signals have been restored.

The time and frequency plots of the recovered signals ae shown below. It can be seen that the recovered and original signals are approximately the same.



Original for comparison:



Problems Faced:

- The biggest issue faced was in terms of incompatibility between 3 factors governing each signal:

1. Audio format - mpeg, wav, ogg, m4a, etc.
2. Sampling frequency – 22050Hz (Tape), 44100 Hz (CD), 48000 Hz (DVD), etc.
3. Audio channels – Mono (1.0) or Stereo (2.0)

Solution:

The two options we were facing were

Option 1: To standardize messages in an online converter

Option 2: To standardize messages in the code itself

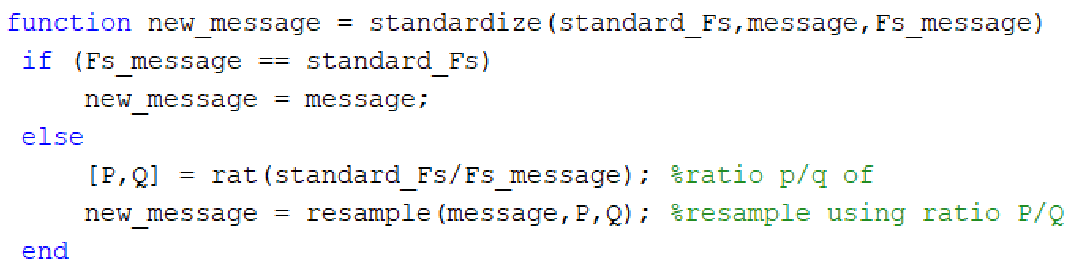
After much debate, we ultimately chose option 2 in order to put the burden on the code instead of the user.

Consequently, our Multiplexer can take both mono and stereo messages with different sampling frequencies, only the format has to be the same.

To maintain uniformity, all time vectors, carrier signals, filters and modulated signals must have the same sampling frequency. In order to standardize our messages to the same sampling frequency we used a custom-built function called ***standardize***. It consists of two parts:

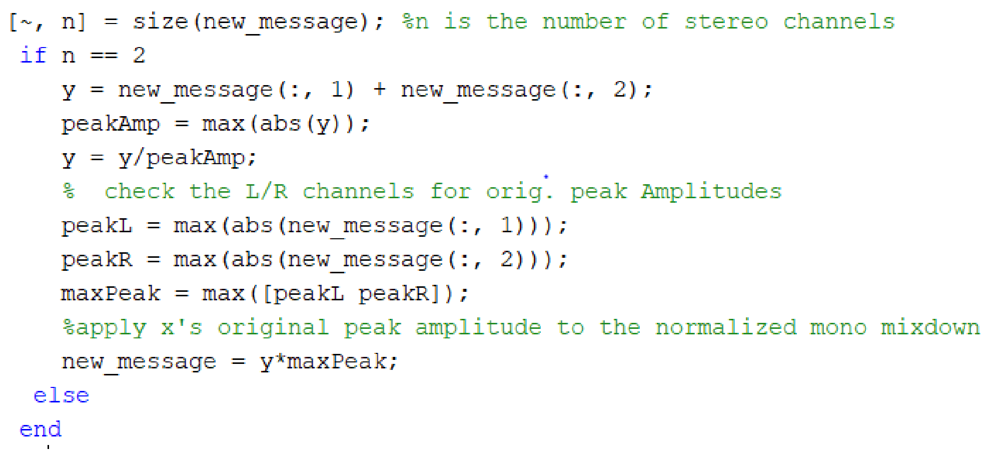
- **Part 1** standardizes the sampling frequency. It achieves so by taking a standard frequency as argument. To meet the *Nyquist* Criterion, f_s must be more than double the max frequency to avoid problem of

aliasing. Our fourth signal band is located at 21 kHz and has a bandwidth of ≈ 3 kHz. The maximum frequency of this signal after modulation is ≈ 24 kHz. Thus, to meet the *Nyquist Criterion*, $f_s > 2 \times 24$ kHz or $f_s > 48$ kHz. Thus, we chose a standard sampling frequency of ***48 kHz***. The function achieves this standardization by calculating the ratio P/Q by which each message's sampling frequency differs from the standard. It then resamples the message according to this ratio.



```
function new_message = standardize(standard_Fs,message,Fs_message)
 if (Fs_message == standard_Fs)
     new_message = message;
 else
     [P,Q] = rat(standard_Fs/Fs_message); %ratio p/q of
     new_message = resample(message,P,Q); %resample using ratio P/Q
 end
```

- **Part 2** standardizes the number of channels to ***mono*** as default. It first checks whether the incoming message is stereo or mono. If it is stereo, it takes the sum of the two channels and normalizes this sum according to the absolute value of the maximum peak in either channel. This process is called Audio Normalization



```
[~, n] = size(new_message); %n is the number of stereo channels
 if n == 2
    y = new_message(:, 1) + new_message(:, 2);
    peakAmp = max(abs(y));
    y = y/peakAmp;
    %  check the L/R channels for orig. peak Amplitudes
    peakL = max(abs(new_message(:, 1)));
    peakR = max(abs(new_message(:, 2)));
    maxPeak = max([peakL peakR]);
    %apply x's original peak amplitude to the normalized mono mixdown
    new_message = y*maxPeak;
  else
 end
```

- The bandwidth of the Band Pass filters needed consideration since a 6 kHz bandwidth cause double sided bands and overlapping in the recovered signals after demodulation.

  The filter orders and ***Apass-Astop*** ratio were also to be considered. Higher values of these parameters give accurate cut-off frequencies and attenuation in stop band, but increases the time and hardware cost of the filter. Thus a compromise had to be reached.

Solution

The bandwidth was set to 3 kHz to obtain single sided bands. No overlapping was observed in the recovered signals, and they were similar to the original signals.

The filter orders were set to the default minimum value and compromise values of Apass-Astop were used. Consequently, our filters are not perfectly ideal and some undesirable frequencies leaked through

- We also faced problems of co-ordination during this project, because of the ongoing pandemic situation. It was difficult to explain the workings of a section of code to the rest of the members, and the solving of problems and explaining of concepts is difficult when the members of a team are not in the same room.

Solution

We divided the tasks in order of the steps to be taken, and each member was requested to comment his code meaningfully and provide a voice recording to the rest of the group members briefly explaining what their code accomplished. Any given task was initiated only when the previous member had done and submitted his task. MS Teams was also used to hold meetings after every couple of days to ensure there were no problems or misunderstandings. In this way, we ensured a smooth workflow for the entire duration of the project.

Conclusion

The results of our project are consistent. From the plot provided in the *Procedure and Results* section, it is evident that both the time and frequency spectrums of the original and recovered signals are similar, with some small discrepancies. As explained in the *Problems Faced* section, these discrepancies arise due to the use of non-ideal filters.

The choice of bandwidth for the Band pass filters may also not be the most efficient implementation, but since the discrepancies observed are not even noticeable when the recovered audio is heard, we can conclude that our model of FDM was successful in the transmittance and recovery of multiple signals over a single channel with adequate precision.