

**“MODELING FREQUENCY DIVISION MULTIPLEXING/DE-
MULTIPLEXING”**

LAB # 09



FALL 2023

DCSE Digital Signals Processing Lab

Submitted By: Esha Rizwan

Registration No: 21PWCSE2010

Section: C

“On my honor, as a student of University of Engineering and Technology, I have neither given not received unauthorized assistance on this academic work.”

Submitted To:

Sir Yasir Saleem

January 8th, 2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objectives:

- To understand and learn the concept of multiplexing.
- To understand the concept of de-multiplexing.
- To learn to use MATLAB for multiplexing and de-multiplexing.

MULTIPLEXING:

Multiplexing is a technique used in telecommunications and data transmission to combine multiple signals into a single signal that can be transmitted over a shared medium, such as a cable or a communication channel, and then separated back into individual signals at the receiving end.

Types of multiplexing are:

- Time Division Multiplexing (TDM)
- Frequency Division Multiplexing (FDM)
- Wavelength Division Multiplexing (WDM)
- Code Division Multiplexing (CDM)

Multiplexing significantly increases the efficiency of communication channels by allowing multiple signals to share the same medium, reducing costs and maximizing bandwidth utilization.

DE-MULTIPLEXING:

De-multiplexing is the process of separating a combined signal, which contains multiple individual signals that were multiplexed together, back into its original constituent signals at the receiving end.

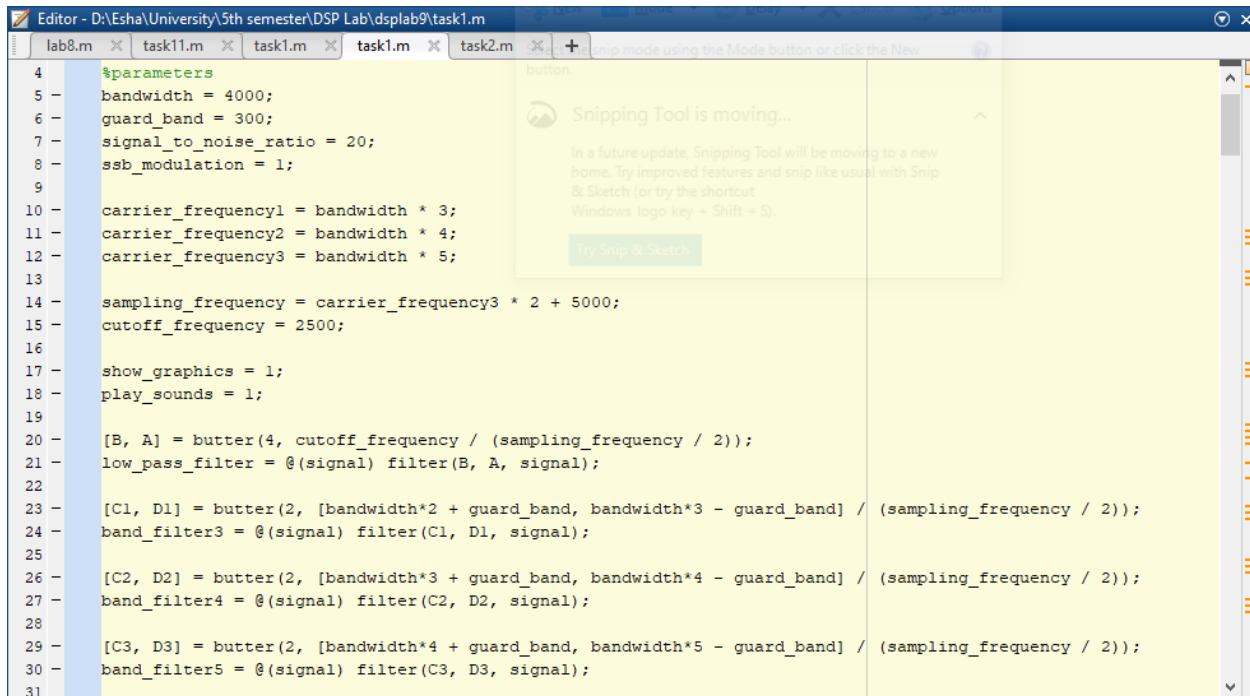
When multiple signals are multiplexed together for transmission over a shared medium, such as in techniques like Time Division Multiplexing (TDM), Frequency Division Multiplexing (FDM), or others, they are combined into a single signal for transmission. At the receiving end, the de-multiplexing process is employed to separate and extract each individual signal from the combined signal.

LAB TASKS:

Implement the following steps in MATLAB to multiplex three input voice signals at the transmitter end and de-multiplex and play them back at the Receiver end. Add random noise to the signal while propagating via the channel.

SETTING UP THE PARAMETERS.

CODE:



```
4 %parameters
5 bandwidth = 4000;
6 guard_band = 300;
7 signal_to_noise_ratio = 20;
8 ssb_modulation = 1;
9
10 carrier_frequency1 = bandwidth * 3;
11 carrier_frequency2 = bandwidth * 4;
12 carrier_frequency3 = bandwidth * 5;
13
14 sampling_frequency = carrier_frequency3 * 2 + 5000;
15 cutoff_frequency = 2500;
16
17 show_graphics = 1;
18 play_sounds = 1;
19
20 [B, A] = butter(4, cutoff_frequency / (sampling_frequency / 2));
21 low_pass_filter = @(signal) filter(B, A, signal);
22
23 [C1, D1] = butter(2, [bandwidth*2 + guard_band, bandwidth*3 - guard_band] / (sampling_frequency / 2));
24 band_filter3 = @(signal) filter(C1, D1, signal);
25
26 [C2, D2] = butter(2, [bandwidth*3 + guard_band, bandwidth*4 - guard_band] / (sampling_frequency / 2));
27 band_filter4 = @(signal) filter(C2, D2, signal);
28
29 [C3, D3] = butter(2, [bandwidth*4 + guard_band, bandwidth*5 - guard_band] / (sampling_frequency / 2));
30 band_filter5 = @(signal) filter(C3, D3, signal);
31
```

```
31
32 - signal1 = audioread('male1.wav');
33 - lsignal1 = length(signal1);
34
35 - signal2 = audioread('male22.wav');
36 - lsignal2 = length(signal2);
37
38 - signal3 = audioread('femalee.wav');
39 - lsignal3 = length(signal3);
40
41 - beep_sound = audioread('beep.wav');
42 - beep_player = audioplayer(beep_sound, 44100);
43
44 - min_length = min([lsignal1, lsignal2]);
45
46 - time = linspace(0, 5, min_length);
```

THE SIGNALS ARE REPRODUCED AS THEY ARRIVE.

CODE:

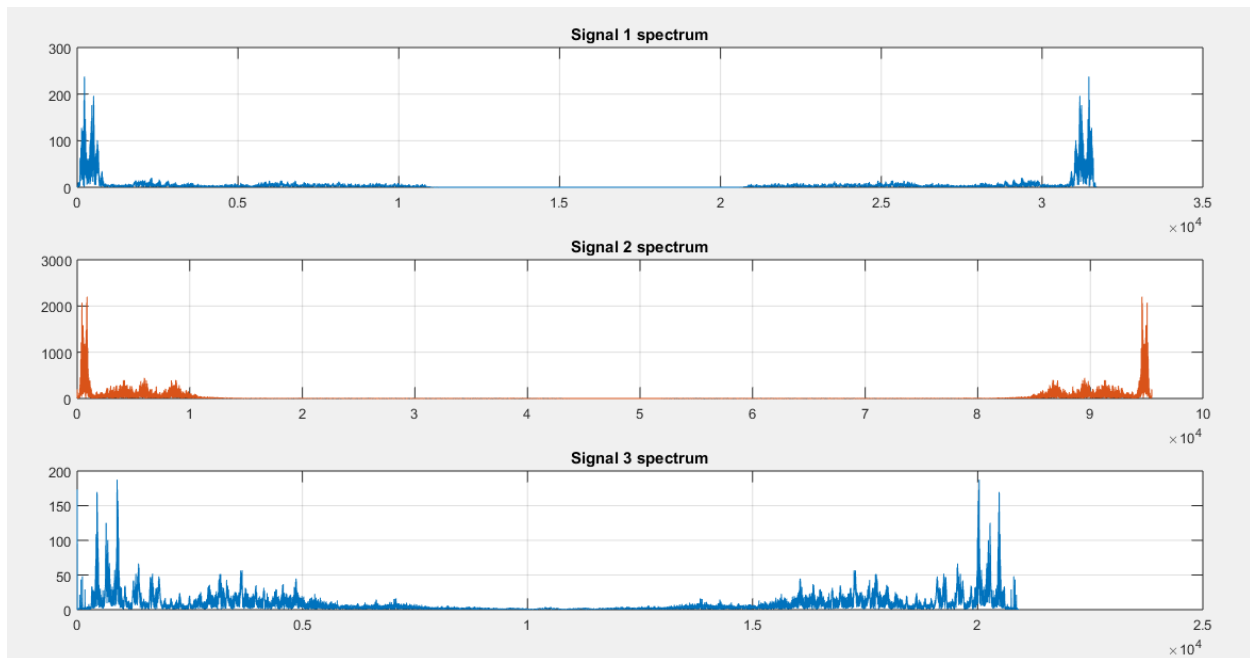
```
48 - disp('STEP - 1, The signals are reproduced as they arrive');
49
50 - if(play_sounds > 0)
51 -     p1 = audioplayer(signal1, 44100);
52 -     playblocking(p1);
53 -     playblocking(beep_player);
54
55 -     p2 = audioplayer(signal2, 44100);
56 -     playblocking(p2);
57 -     playblocking(beep_player);
58
59 -     p3 = audioplayer(signal3, 44100);
60 -     playblocking(p3);
61 - end
```

PLOT THE SPECTRA OF THE SIGNALS AS THEY ARRIVE (USE FFT AND DSP.SPECTRUMANALYZER FOR COMPARISON).

CODE:

```
62 %  
63 disp('STEP - 2, Plot the spectra of the signals as they arrive');  
64  
65 if(show_graphics > 0)  
66     figure  
67  
68     spectrum1 = abs(fft(signal1));  
69     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Signal 1 spectrum');  
70  
71     spectrum2 = abs(fft(signal2));  
72     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Signal 2 spectrum');  
73  
74     spectrum3 = abs(fft(signal3));  
75     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Signal 3 spectrum');  
76 end
```

PLOT:

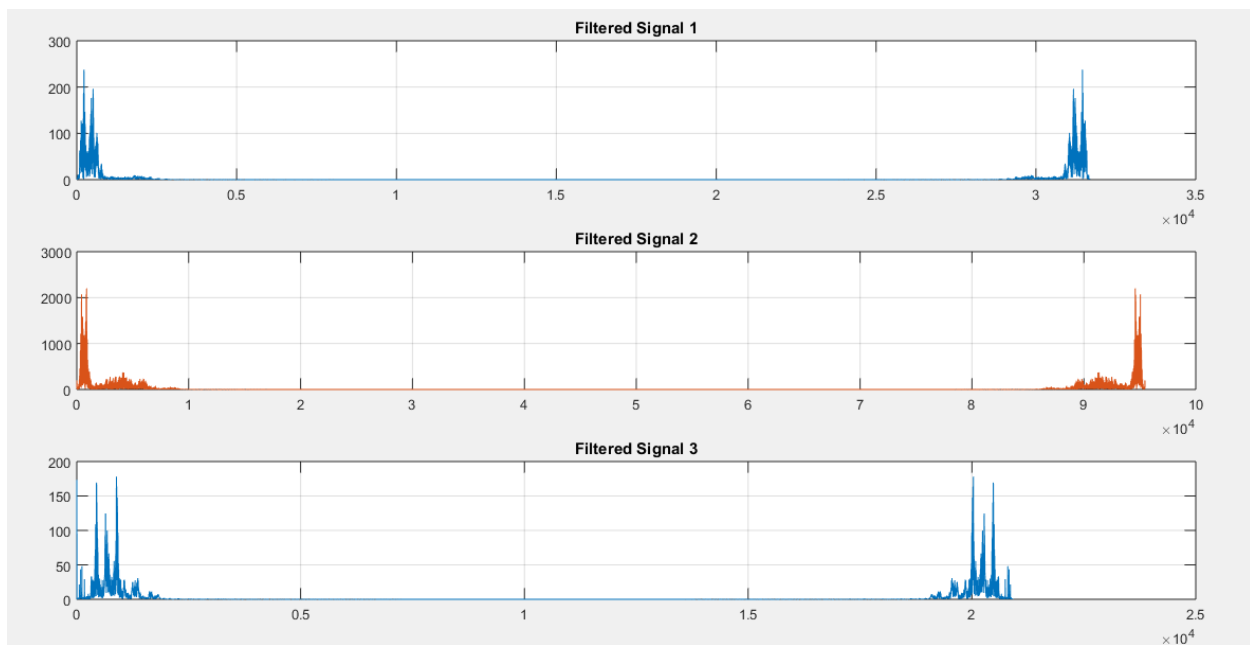


THE SIGNALS ARE PASSED THROUGH A LOW PASS FILTER AND PLOTTED.

CODE:

```
78 - disp('STEP - 3, The signals are passed through a low pass filter and played');
79 -
80 - signal1 = low_pass_filter(signal1);
81 - signal2 = low_pass_filter(signal2);
82 - signal3 = low_pass_filter(signal3);
83 -
84 - if(show_graphics > 0)
85 -     figure
86 -
87 -     spectrum1 = abs(fft(signal1));
88 -     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Filtered Signal 1');
89 -
90 -     spectrum2 = abs(fft(signal2));
91 -     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Filtered Signal 2');
92 -
93 -     spectrum3 = abs(fft(signal3));
94 -     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Filtered Signal 3');
95 - end
```

PLOT:



REPRODUCE THE SIGNALS AFTER PASSING THEM THROUGH THE FILTER.

CODE:

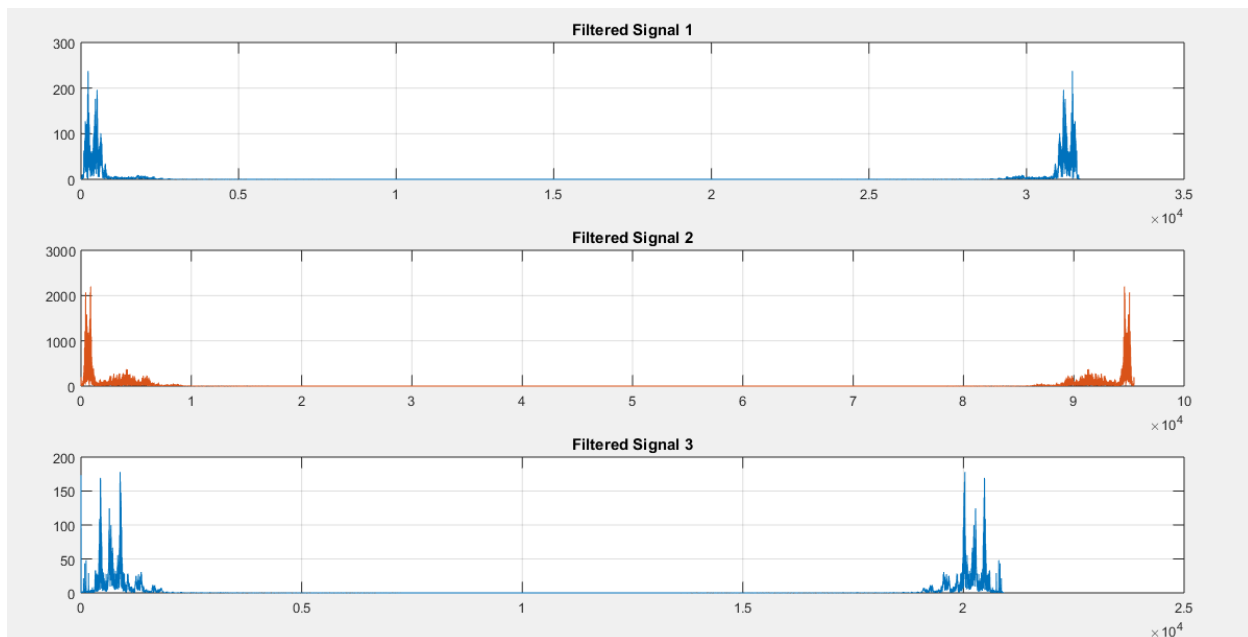
```
96 %%
97 - disp('STEP - 4, Reproduce the signals after passing through the filter');
98
99 - if(play_sounds > 0)
100 -     beep_player = audioplayer(beep_sound, 44100);
101
102 -     p1 = audioplayer(signal1, 44100);
103 -     playblocking(p1);
104 -     playblocking(beep_player);
105
106 -     p2 = audioplayer(signal2, 44100);
107 -     playblocking(p2);
108 -     playblocking(beep_player);
109
110 -     p3 = audioplayer(signal3, 44100);
111 -     playblocking(p3);
112 -     playblocking(beep_player);
113 - end
```

THE SIGNALS ARE MODULATED TO DIFFERENT CARRIERS.

CODE:

```
115 - disp('STEP - 5, The signals are modulated to different carriers');
116
117 - if(ssb_modulation > 0)
118     modulated_signal1 = ssbmod(signal1, carrier_frequency1, sampling_frequency);
119     modulated_signal2 = ssbmod(signal2, carrier_frequency2, sampling_frequency);
120     modulated_signal3 = ssbmod(signal3, carrier_frequency3, sampling_frequency);
121 else
122     modulated_signal1 = ammod(signal1, carrier_frequency1, sampling_frequency);
123     modulated_signal2 = ammod(signal2, carrier_frequency2, sampling_frequency);
124     modulated_signal3 = ammod(signal3, carrier_frequency3, sampling_frequency);
125 end
126
127 - if(show_graphics > 0)
128     figure
129
130     spectrum1 = abs(fft(signal1));
131     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Filtered Signal 1');
132
133     spectrum2 = abs(fft(signal2));
134     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Filtered Signal 2');
135
136     spectrum3 = abs(fft(signal3));
137     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Filtered Signal 3');
138 end
```

PLOT:

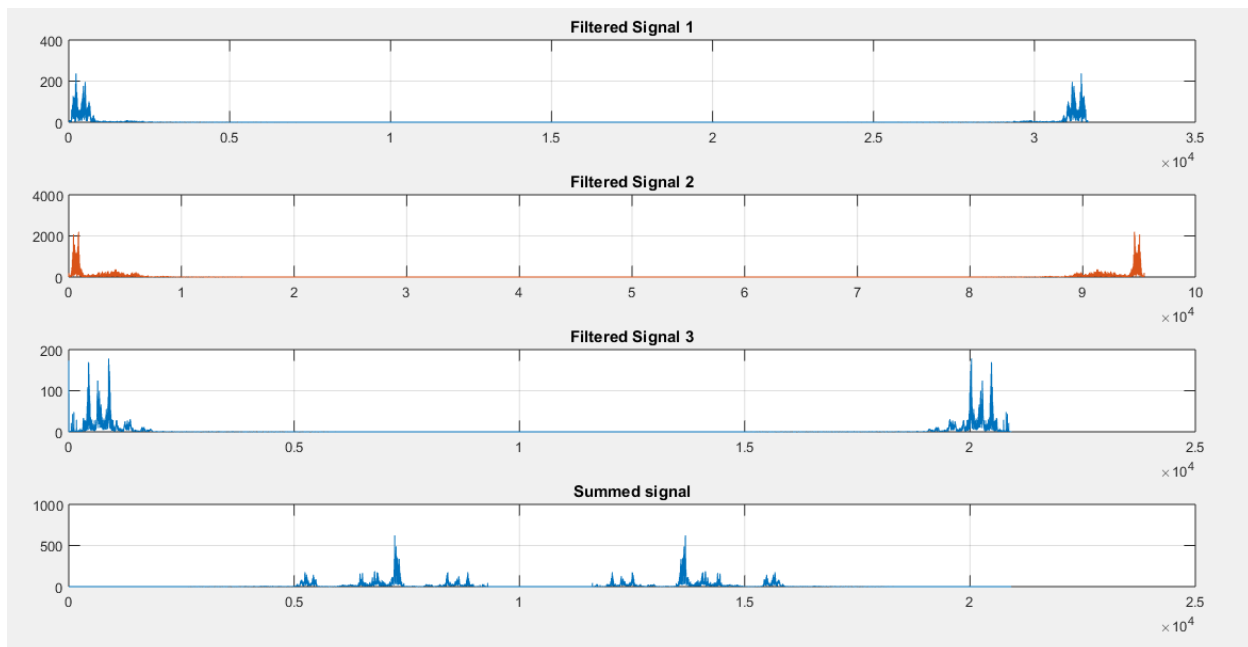


THE MODULATED SIGNALS ARE FILTERED IN THE GIVEN BAND AND ADDED TOGETHER.

CODE:

```
140 - disp('STEP - 6, The modulated signals are filtered in the defined bands and added');
141
142 - filtered_signal1 = band_filter3(modulated_signal1);
143 - filtered_signal2 = band_filter4(modulated_signal2);
144 - filtered_signal3 = band_filter5(modulated_signal3);
145
146 % Find the minimum length among the signals
147 - min_length = min([length(filtered_signal1), length(filtered_signal2), length(filtered_signal3)]);
148
149 % Sum the signals element-wise up to the minimum length
150 - complete_signal = zeros(min_length, 1);
151 - for i = 1:min_length
152 -     complete_signal(i) = filtered_signal1(i) + filtered_signal2(i) + filtered_signal3(i);
153 - end
154
155
156 - if(show_graphics > 0)
157 -     figure
158
159 -     spectrum1 = abs(fft(signal1));
160 -     subplot(4,1,1), plot(spectrum1), grid on, zoom, title('Filtered Signal 1');
161
162 -     spectrum2 = abs(fft(signal2));
163 -     subplot(4,1,2), plot(spectrum2), grid on, zoom, title('Filtered Signal 2');
164
165 -     spectrum3 = abs(fft(signal3));
```

PLOT:

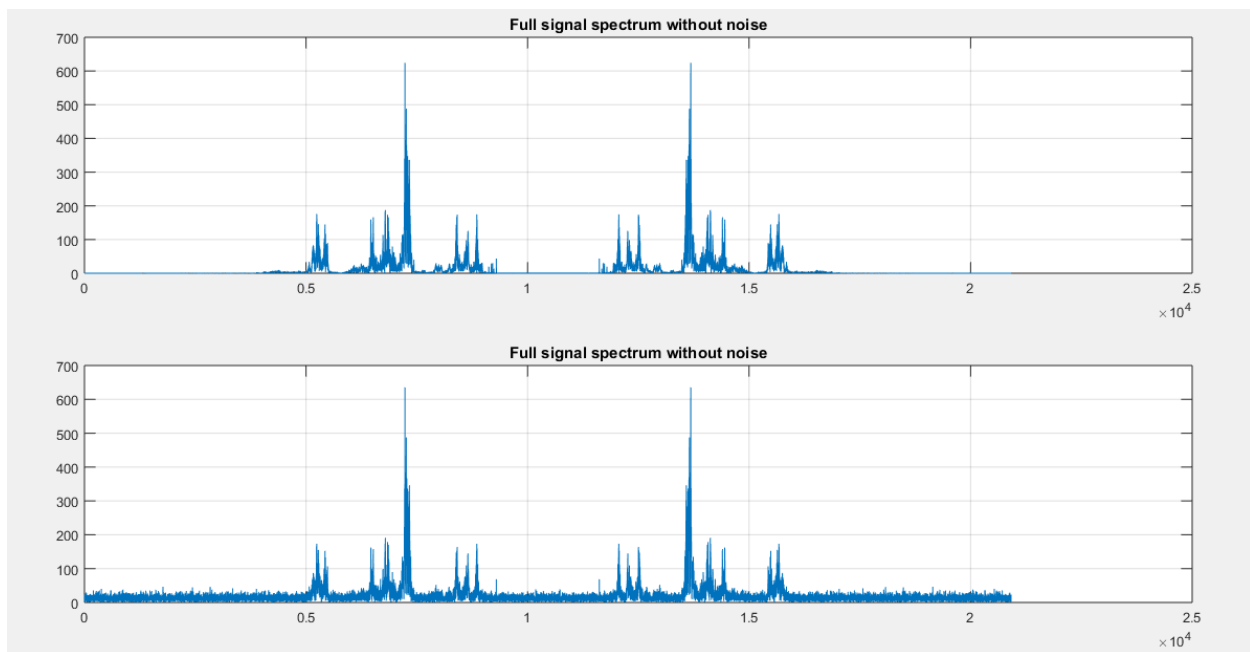


SOME NOISE IS ADDED TO THE TRANSMITTED SIGNAL.

CODE:

```
172 - disp('STEP - 7, Spectrum of transmitted signal before adding noise');
173
174 - if (show_graphics > 0)
175 -     figure
176
177 -     total_spectrum = abs(fft(complete_signal));
178 -     subplot(2,1,1), plot(total_spectrum), grid on, zoom, title('Full signal spectrum without noise');
179 - end
180
181 - complete_signal = awgn(complete_signal, signal_to_noise_ratio);
182
183 - if (show_graphics > 0)
184 -     total_spectrum = abs(fft(complete_signal));
185 -     subplot(2,1,2), plot(total_spectrum), grid on, zoom, title('Full signal spectrum without noise');
186 - end
```

PLOT:

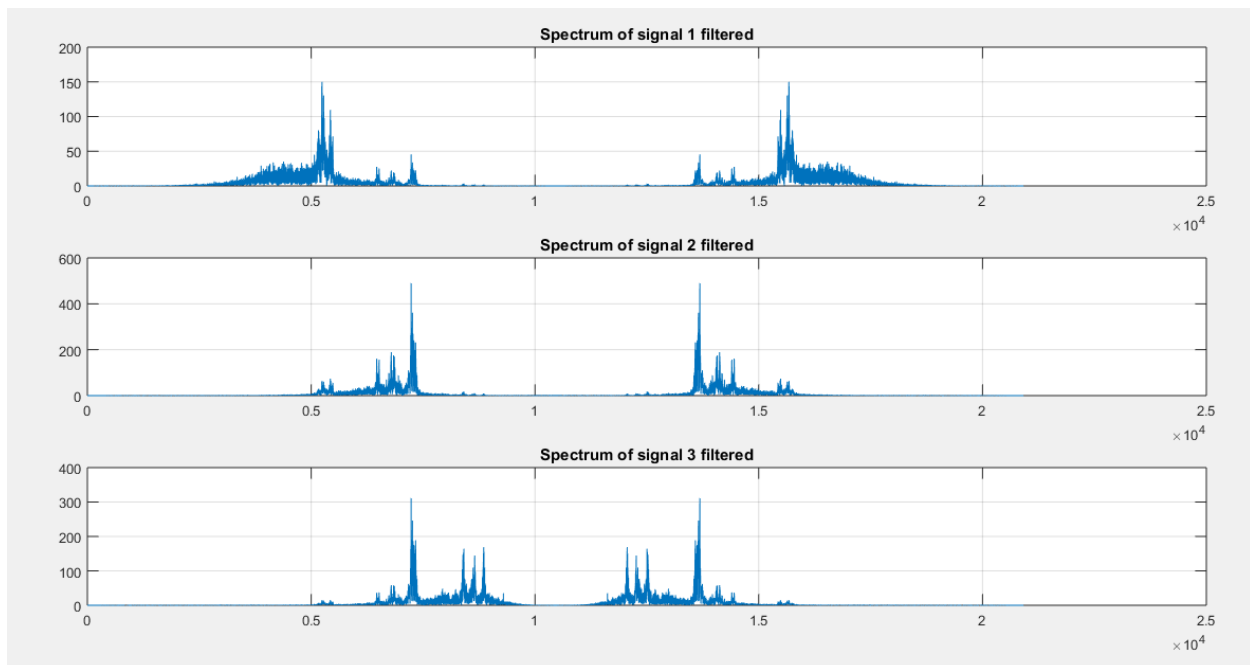


UPON ARRIVAL EACH BAND IS FILTERED.

CODE:

```
188 - disp('STEP - 8, Upon arrival each band is filtered');
189
190 - demod_signal1 = band_filter3(complete_signal);
191 - demod_signal2 = band_filter4(complete_signal);
192 - demod_signal3 = band_filter5(complete_signal);
193
194 - if(show_graphics > 0)
195 -     figure
196
197 -     spectrum1 = abs(fft(demod_signal1));
198 -     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Spectrum of signal 1 filtered');
199
200 -     spectrum2 = abs(fft(demod_signal2));
201 -     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Spectrum of signal 2 filtered');
202
203 -     spectrum3 = abs(fft(demod_signal3));
204 -     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Spectrum of signal 3 filtered');
205 - end
```

PLOT:

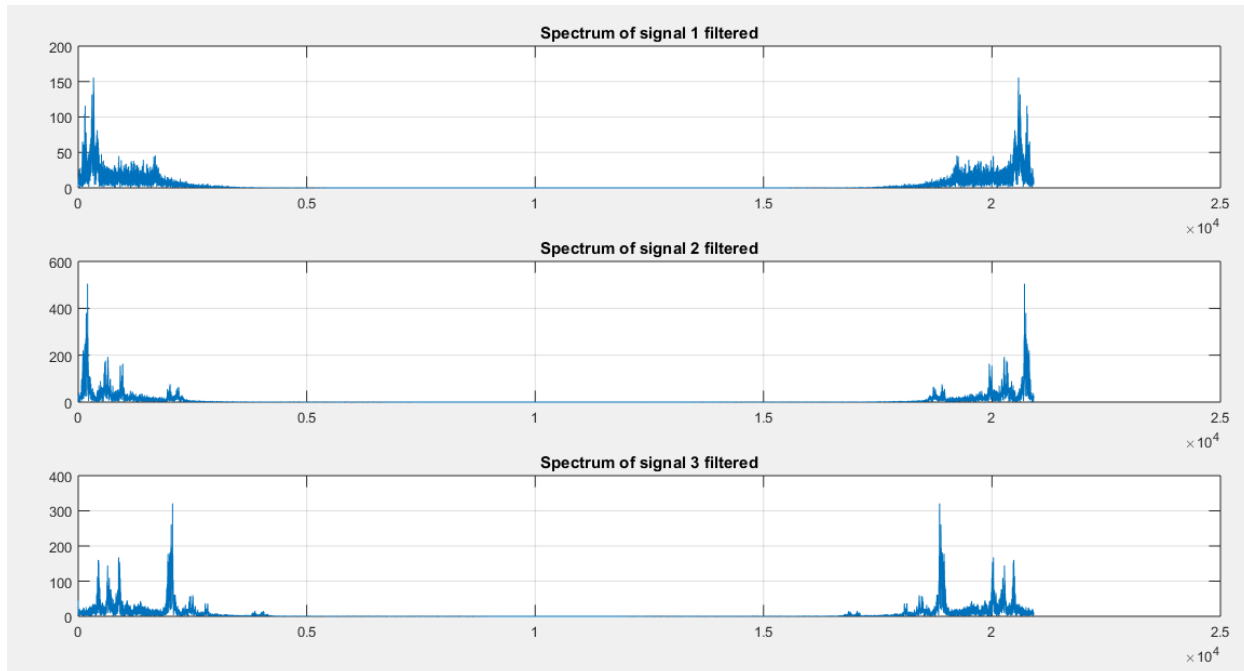


EACH RECOVERED BAND IS DEMODULATED TO RETURN THE SIGNAL AT THE INDICATED FREQUENCY.

CODE:

```
207 - disp('STEP - 9, Each recovered band is demodulated to return the signal to the baseband frequency');
208
209 - if(ssb_modulation > 0)
210 -     demod_signal1 = ssbmod(demod_signal1, carrier_frequency1, sampling_frequency);
211 -     demod_signal2 = ssbmod(demod_signal2, carrier_frequency2, sampling_frequency);
212 -     demod_signal3 = ssbmod(demod_signal3, carrier_frequency3, sampling_frequency);
213 - else
214 -     demod_signal1 = ammod(demod_signal1, carrier_frequency1, sampling_frequency);
215 -     demod_signal2 = ammod(demod_signal2, carrier_frequency2, sampling_frequency);
216 -     demod_signal3 = ammod(demod_signal3, carrier_frequency3, sampling_frequency);
217 - end
218
219 - if(show_graphics > 0)
220 -     figure
221
222 -     spectrum1 = abs(fft(demod_signal1));
223 -     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Spectrum of signal 1 filtered');
224
225 -     spectrum2 = abs(fft(demod_signal2));
226 -     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Spectrum of signal 2 filtered');
227
228 -     spectrum3 = abs(fft(demod_signal3));
229 -     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Spectrum of signal 3 filtered');
230 - end
```

PLOT:

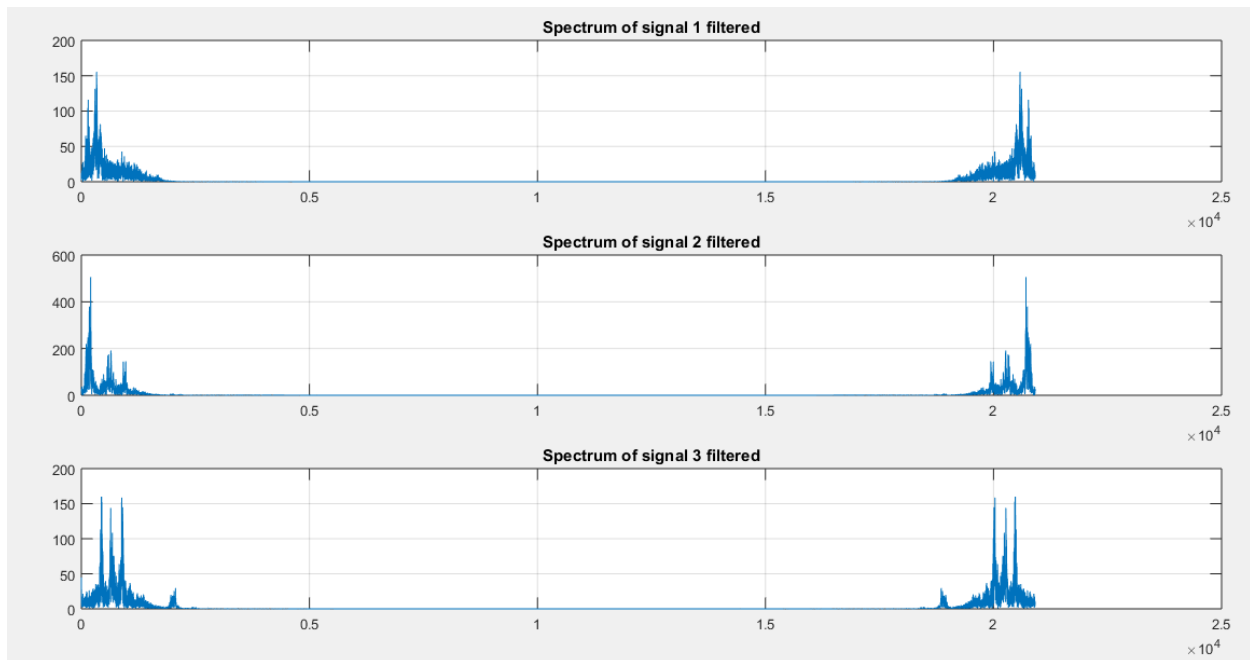


THE RECOVERED SIGNAL IS PASSED THROUGH A LOW PASS FILTER.

CODE:

```
232 - disp('STEP - 10, The recovered signal is passed through a low pass filter');
233
234 - demod_signal1 = low_pass_filter(demod_signal1);
235 - demod_signal2 = low_pass_filter(demod_signal2);
236 - demod_signal3 = low_pass_filter(demod_signal3);
237
238 - if(show_graphics > 0)
239 -     figure
240
241 -     spectrum1 = abs(fft(demod_signal1));
242 -     subplot(3,1,1), plot(spectrum1), grid on, zoom, title('Spectrum of signal 1 filtered');
243
244 -     spectrum2 = abs(fft(demod_signal2));
245 -     subplot(3,1,2), plot(spectrum2), grid on, zoom, title('Spectrum of signal 2 filtered');
246
247 -     spectrum3 = abs(fft(demod_signal3));
248 -     subplot(3,1,3), plot(spectrum3), grid on, zoom, title('Spectrum of signal 3 filtered');
249 - end
```

PLOT:



SIGNAL REPRODUCED AFTER TRANSMISSION.

CODE:

```
250 %%  
251 - disp('STEP - 11, Play the reproduced signal after the transmission');  
252  
253 - player4 = audioplayer(demod_signal1, 44100);  
254 - playblocking(player4);  
255 - playblocking(beep_player);  
256  
257 - player5 = audioplayer(demod_signal2, 44100);  
258 - playblocking(player5);  
259 - playblocking(beep_player);  
260  
261 - player6 = audioplayer(demod_signal3, 44100);  
262 - playblocking(player6);
```