Cairo University

Faculty of Engineering

Electronics and Electrical Communications Engineering Department – Third Year

Analog Communication Project - Implementing a frequency division multiplexing transmitter for two messages and a superheterodyne receiver using MATLAB

Submitted to:

Dr. Ahmad Hesham

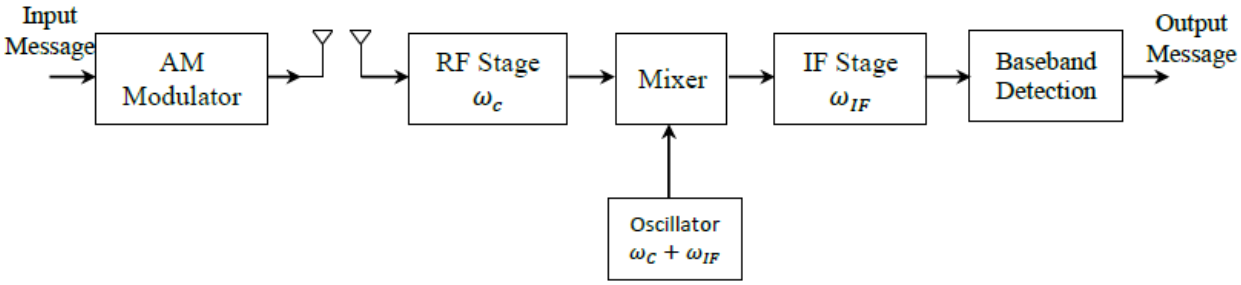
Submitted by:

Abdulrahman Elsayed

Section 3 BN 05

1. Introduction

Figure 1 shows a typical AM transceiver. The input messages are modulated using a frequency division multiplexing transmitter and wirelessly transmitted. The receiver whose type is superheterodyne detects the message using multiple



stages of mixing and filtering which we explain in the following.

Figure 1: An AM modulator and a superheterodyne receiver

1. ***The Transmitter Stage***

Discussion

```
Signals Stage:
First of all we have to read the messages using the audioread() command.
They are stereo (2-channels) signals but our receiver is a monophonic
receiver so we will convert them to monophonic signals. One way of
converting them is to take the average of the two channels and assigning it
to a new signal which will be the monophonic signal. As it is a frequency
division multiplexing transmitter, the input messages have to be added and
therefore they must be in the same length.
So, we pad the short messages with zeros till they have the same length as
the longest one.
Then, we modulate the signals, but before that, we must meet the Nyquist
criteria. The sampling rate must be greater than double the carrier
frequency. Therefore, we increase the sampling frequency Fs, say by 10
times. This is done by increasing the number of samples of the message
signal. We can do that by using the interp() command.
Now the messages are ready, and the carriers are remaining. For each message
we should provide a carrier with its unique frequency that meets the Nyquist
criteria and time period that is equal to the time period of the message
with the same number of samples. We can do that easily by the cos() command.
Modulation Stage:
This is an easy stage in which we multiply every message by its
corresponding carrier to obtain the modulated signals which we multiplex
them and finally transmit them on the wireless channel.
```

The figures

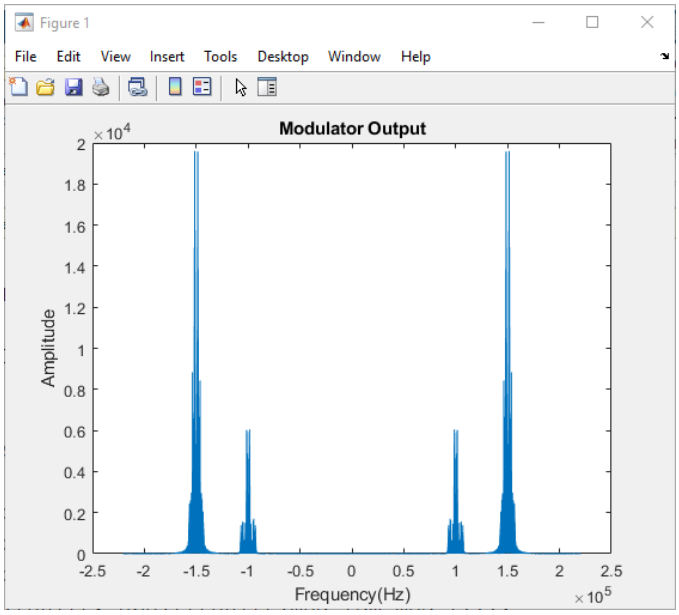


Figure 2: Modulator Output

2. ***The RF Stage***

Discussion

Bandpass filter at RF stage:
At the receiver, we receive the frequency division multiplexing signal which contain more than one message so we have to pass it to a bandpass filter to choose the message we want according to its carrier frequency. Ideally, all we need is one sharp bandpass filter that select the message, but while we work at high frequency, it's hard to implement that sharp filter. One solution to this problem is to pass the signal first to a filter that is not sharp and then shift it to a low frequency band by a mixer where we can by the subsequent stage implement a sharp filter – as the signal now is at low frequency – and obtain the message cleanly. But notice, the first filter should perform interference image rejection. For simplicity, we assume it's not a problem here and ignore that condition.
We can do that in MATLAB by designing a bandpass filter using the fdesign.bandpass() function whose specs are adjusted according to the desire signals as shown in the code section.
Mixing Stage:
After the desired message is chosen, now we shift it to the intermediate frequency which can be done by multiplying the message by a cosine signal whose frequency is the intermediate frequency.

The figures

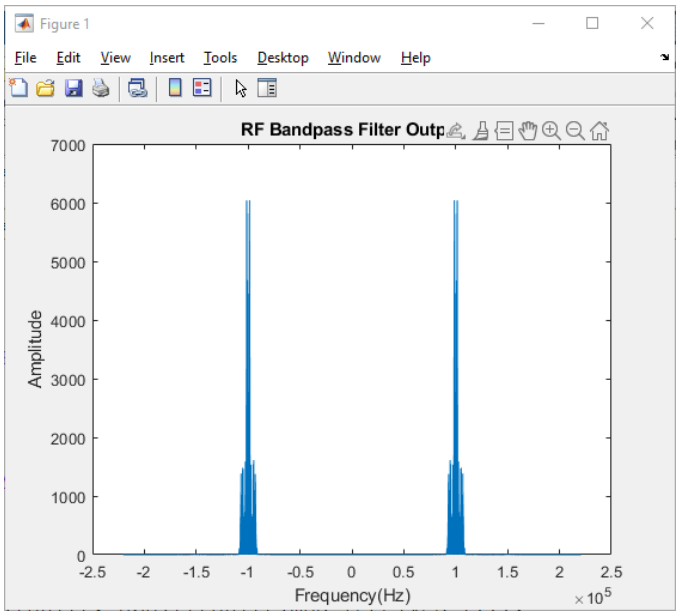


Figure 3: RF bandpass filter Output

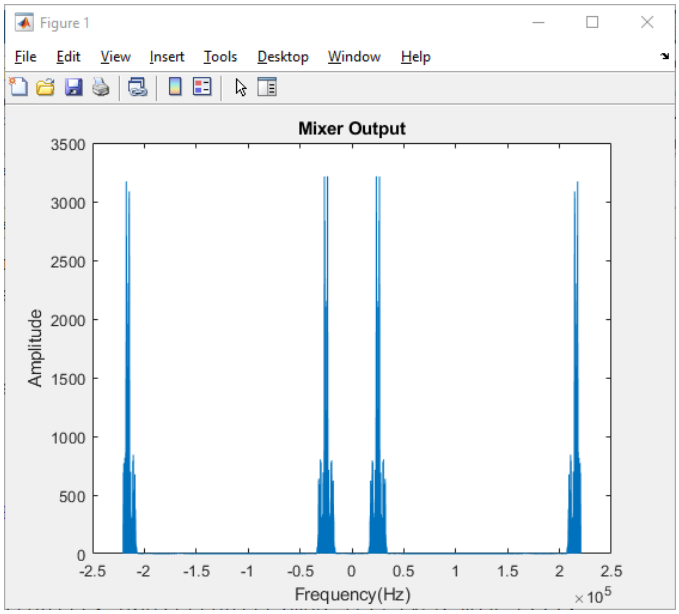


Figure 4: Mixer Output

3. *The IF stage*

Discussion

Bandpass filter at IF Stage:
Now with the message is at the intermediate frequency, to acquire it cleanly, we can easily design a sharp bandpass filter whose specs depend on the intermediate frequency and the bandwidth of the message.
Demodulation Stage:
This is a simple stage in which we multiply the message by a carrier whose frequency is equal to the intermediate frequency to shift it to the baseband. We do that by a second mixer.

The figures

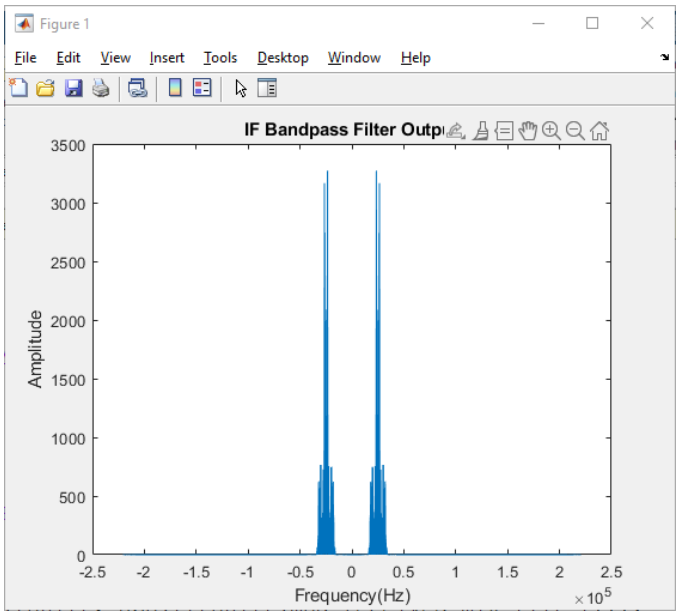


Figure 5: IF Bandpass Filter Output

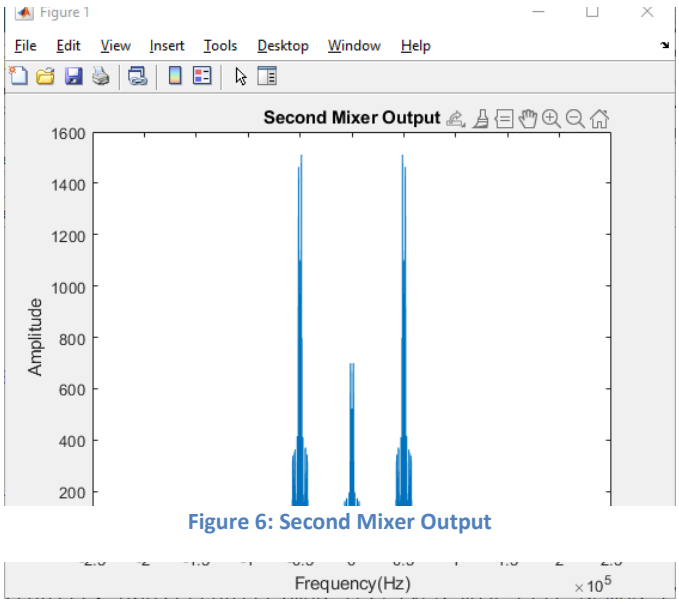


Figure 6: Second Mixer Output

4. ***The baseband demodulator***

Discussion

Bandpass filter at IF Stage:
Now with the message is at the intermediate frequency, to acquire it cleanly, we can easily design a sharp bandpass filter whose specs depend on the intermediate frequency and the bandwidth of the message.
Demodulation Stage:
This is a simple stage in which we multiply the message by a carrier whose frequency is equal to the intermediate frequency to shift it to the baseband. We do that by a second mixer.
Coherent Detector Stage:
Finally, We pass the baseband signal to a lowpass filter and hence the output of the filter is the desired message.

The figures

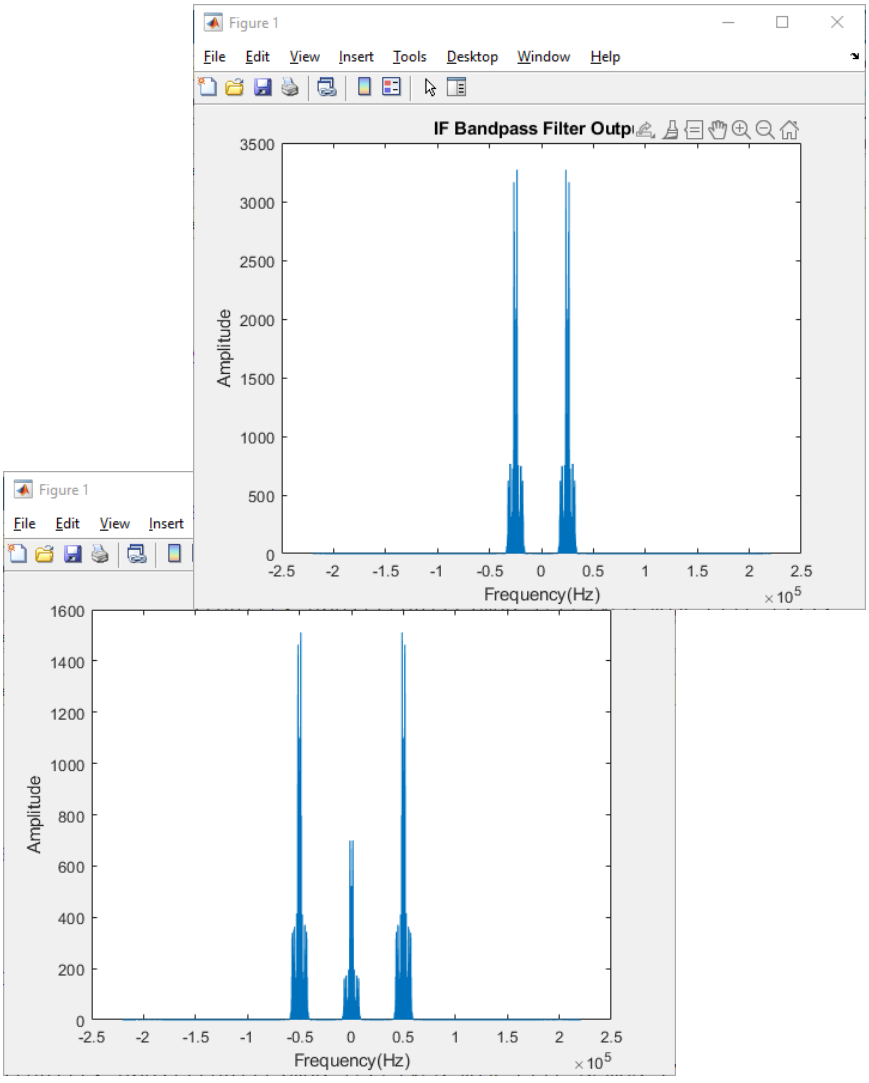


Figure 1
File Edit View Insert Tools Desktop Window Help
IF Bandpass Filter Outp
Amplitude
Frequency(Hz)
×10^5
Figure 1
File Edit View Insert
Amplitude
Frequency(Hz)
×10^5

5. ***The code***

```
% MATLAB code for implementing a frequency division
multiplexing
% transmitter for two messages and a superheterodyne
receiver

% Transmitter Stage
% The signals
[msg1 Fs] = audioread('Short_BBCArabic2.wav'); %
Reading first message
[msg2 Fs] = audioread('Short_FM9090.wav'); % Reading
second message
% Obtaining the monophonic signals out of the stereo
signals
msg1_mono = (msg1(:,1) + msg1(:,2)) / 2;
msg2_mono = (msg2(:,1) + msg2(:,2)) / 2;
% msg2 is shorter than msg1, so pad msg2 with zeros so
that its length be equal to
% The length of msg1
msg2_mono = [msg2_mono; zeros(1, length(msg1_mono) -
length(msg2_mono))']; % padded
% Increasing the sampling rate to meet the Nyquist
criteria
msg1_mono_interp = interp(msg1_mono, 10);
msg2_mono_interp = interp(msg2_mono, 10);
Fs = Fs * 10; % Sampling rate
Ts = 1 / Fs; % Sampling period
L = length(msg1_mono_interp);  % Length of signal
t = (0:L-1)*Ts; % Time vector
% Transposing the time vector so it matches up with
the messages
t = t';
fshift = (-L/2:L/2-1)*(Fs/L); % frequency vector to
adjust the axis scale of the spectrum
% From now on, any variable name thet terminated by _f
the is in the frequency domain
% For example; var1_f is the frequency domain variable
of the time domain one named var1
msg1_mono_interp_f = fft(msg1_mono_interp, L);
msg2_mono_interp_f = fft(msg2_mono_interp, L);
```

```
% Generating the carriers, one at 100KHz and the
seccond at 150KHz
% They are cosinne signals that last the same time as
the longest message msg1
% and have the same number of samples L
carrier1 = cos(2 * pi * 100000 * t);
carrier2 = cos(2 * pi * 150000 * t);

% Modulation Stage
% Multipling each message by its corresponding carrier
msg1_mod = msg1_mono_interp .* carrier1;
msg2_mod = msg2_mono_interp .* carrier2;
% Multiplixing the signals to generate a Frequency
Division Multiplixing (fdm)
msg_fdm_mod = msg1_mod + msg2_mod;
msg_fdm_mod_f = fft(msg_fdm_mod, L);
% Plotting the output of the modulator
subplot(7, 1, 1);
plot(fshift, abs(fftshift(msg_fdm_mod_f)));
title('Modulator Output')
xlabel('Frequency(Hz)')
ylabel('Amplitude')

% Receiver Stage
% bandpass filter at RF Stage
% adjust the following parameters accourding to the
desired signal
%{
    To choose the first message,
    apply the following values:
    fc = 100000; % Carrier frequency
    F_stop1 = 0.75 * 100000; % Edge of the stopband
    F_pass1 = 0.8 * 100000; % Edge of the passband
    F_pass2 = 1.2 * 100000; % Closing edge of the
passband
    F_stop2 = 1.25 * 100000; % Edge of the second
stopband
    To choose the second one:
    fc = 150000;
    F_stop1 = 1.25 * 100000; % Edge of the stopband
    F_pass1 = 1.3 * 100000; % Edge of the passband
```

```
    F_pass2 = 1.7 * 100000; % Closing edge of the
passband
    F_stop2 = 1.75 * 100000; % Edge of the second
stopband
%}
% Continuing with the first message
fc = 100000; % Carrier frequency
fif = 25000; % Intermediate frequency
A_stop1 = 60; % Attenuation in the first stopband = 60
dB
F_stop1 = 0.75 * 100000; % Edge of the stopband
F_pass1 = 0.8 * 100000; % Edge of the passband
F_pass2 = 1.2 * 100000; % Closing edge of the passband
F_stop2 = 1.25 * 100000; % Edge of the second stopband
A_stop2 = 60; % Attenuation in the second stopband =
60 dB
A_pass = 1; % Amount of ripple allowed in the passband
= 1 dB
% RF bandpass filter
RFBandPassSpecObj = ...
    fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2',
...
        F_stop1, F_pass1, F_pass2, F_stop2, A_stop2,
A_pass, ...
            A_stop2, Fs)
RFBandPassFilt = design(RFBandPassSpecObj)
% Output of RF bandpass filter
msg_received = filter(RFBandPassFilt, msg_fdm_mod);
msg_received_f = fft(msg_received, L);
% Plotting the output of the RF bandpass filter
subplot(7, 1, 2);
plot(fshift, abs(fftshift(msg_received_f)));
title('RF Bandpass Filter Output')
xlabel('Frequency(Hz)')
ylabel('Amplitude')

% Mixing Stage
% Multipling the received mwssage by a cosine signal
at frequency equal to fc + fif
% to shift the message at frequency fif
```

```
oscillator_signal = cos(2 * pi * (fc + fif) * t);
msg_received_mix = msg_received .* oscillator_signal;
msg_received_mix_f = fft(msg_received_mix, L);
% Plotting the output of the mixer
subplot(7, 1, 3);
plot(fshift, abs(fftshift(msg_received_mix_f)));
title('Mixer Output')
xlabel('Frequency(Hz)')
ylabel('Amplitude')

% Bandpass filter at IF Stage
% The received signal now is at 25KHz and the bandwith
of each message is approximately 20KHz
% so, the specification of the filter is as follows:
A_stop1 = 60; % Attenuation in the first stopband = 60
dB
F_stop1 = 0.05 * 100000; % Edge of the stopband
F_pass1 = 0.1 * 100000; % Edge of the passband
F_pass2 = 0.4 * 100000; % Closing edge of the passband
F_stop2 = 0.45 * 100000; % Edge of the second stopband
A_stop2 = 60; % Attenuation in the second stopband =
60 dB
A_pass = 1; % Amount of ripple allowed in the passband
= 1 dB
IFBandPassSpecObj = ...
    fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2',
...
        F_stop1, F_pass1, F_pass2, F_stop2, A_stop1,
A_pass, A_stop2, Fs)
IFBandPassFilt = design(IFBandPassSpecObj)
% Output of IF bandpass filter
msg_received_mix_filt = filter(IFBandPassFilt,
msg_received_mix);
msg_received_mix_filt_f = fft(msg_received_mix_filt,
L);
% Plotting the output of the IF bandpass filter
subplot(7, 1, 4);
plot(fshift, abs(fftshift(msg_received_mix_filt_f)));
title('IF Bandpass Filter Output')
xlabel('Frequency(Hz)')
```

```
ylabel('Amplitude')

% Demodulation Stage
% Multiplying the recevied filterd message by a cosine
signal whose frequency is at fif
% to shift it to the baseband
demod_signal = cos(2 * pi * (25000) * t);
msg_received_mix_filt_demod = msg_received_mix_filt .*
demod_signal;
msg_received_mix_filt_demod_f =
fft(msg_received_mix_filt_demod, L);
% Plotting the output of the second mixer
subplot(7, 1, 5);
plot(fshift,
abs(fftshift(msg_received_mix_filt_demod_f)));
title('Second Mixer Output')
xlabel('Frequency(Hz)')
ylabel('Amplitude')
% Passing the baseband message to a lowpass filter to
get the pure message
% it's already mentioned that each message has a
bandwidth of 25KHz
% so, the specification of the filter is as follows:
A_stop = 60; % Attenuation in the first stopband = 60
dB
F_stop = (0.3*100000) / 441000; % Edge of the stopband
F_pass = (0.25*100000) / 441000; % Edge of the
passband
A_pass = 1; % Amount of ripple allowed in the passband
= 1 dB
LowPassSpecObj = fdesign.lowpass(F_pass, F_stop,
A_pass, A_stop)
LowPassFilt = design(LowPassSpecObj)
% Output of the lowpass filter
msg_received_mix_filt_demod_filt = filter(LowPassFilt,
msg_received_mix_filt_demod);
msg_received_final = msg_received_mix_filt_demod_filt;
% The final received message
msg_received_final_f = fft(msg_received_final, L);
% Plotting the message
```

```
subplot(7, 1, 6);
plot(fshift, abs(fftshift(msg_received_final_f)));
title('The Message (Frequency Domain)')
xlabel('Frequency(Hz)')
ylabel('Amplitude')

% Playing Stage
msg_received_original_sampled = msg_received_final(1:
10: end);
t_original_sampled = t(1: 10: end);
subplot(7, 1, 7);
plot(t_original_sampled,
msg_received_original_sampled);
title('The Message (Time Domain)')
xlabel('Time(sec)')
ylabel('Amplitude')
sound(msg_received_original_sampled, Fs / 10);

% End
```