Simulation of Quadrature Amplitude Modulation System Using MATLAB

Xyruz G. Gomez
Electronics Engineering
Faculty of Engineering
University of Santo Tomas
Manila, Philippines
xyruz.gomez.eng@ust.edu.ph

Sean Rowelle M. Salanio
Electronics Engineering
Faculty of Engineering
University of Santo Tomas
Manila, Philippines
seanrowelle.salanio.eng@ust.edu.ph

Evitha Jovel E. Viola
Electronics Engineering
Faculty of Engineering
University of Santo Tomas
Manila, Philippines
evithajovel.viola.eng@ust.edu.ph

***Abstract*—This machine problem aims to introduce and apply the mathematical model and implementation of a basic communication system using MATLAB®. Utilizing quadrature amplitude modulation (QAM) as the primary modulation scheme, the project processes input audio signals through the system and aims to recover the original signal accurately at the output of the final stage. The process involves five distinct stages that the signal undergoes: Loading Input Signal, Signal Up-Sampling, Quadrature Amplitude Modulation (QAM), Quadrature Amplitude Demodulation (QAD), and Signal Down-Sampling. Loading Input Signal: The system begins by loading the input audio signals. Signal Up-Sampling: The input signals are up-sampled to a higher sampling frequency to prepare them for modulation. Quadrature Amplitude Modulation: The up-sampled signals are modulated, resulting in a multiplexed signal suitable for transmission. Quadrature Amplitude Demodulation: At this stage, the multiplexed signal undergoes demodulation to recover the original audio signals using the QAD scheme. Signal Down-Sampling: Finally, the recovered signal is down-sampled to reverse the up-sampling process. This step ensures the signal can be accurately reproduced and played back through speakers. The expected outcome is the successful recovery of the input audio signals, demonstrated by playing the received signal using MATLAB's sound () function. If the process is implemented correctly, the signal should be audible, and ideally, the quality should closely resemble the original input. While this project focuses on meeting the defined requirements, further enhancements can be explored by researchers and students to improve the system or utilize its principles in more advanced communication projects..**

Keywords—Quadrature Amplitude Modulation (QAM), Quadrature Amplitude Demodulation (QAD), Sampling, Modulation, Frequency

I. INTRODUCTION

Communication is a fundamental process through which information is exchanged, encompassing various modalities such as verbal, nonverbal, print, and electronic methods [5]. The communication system can be defined as a framework through which information is transmitted from a source to a destination [1]. In the context of electronic communication systems, these roles are filled by the transmitter and receiver. The signal traverses from the source to the destination via a transmission medium known as a channel, illustrating how it navigates through the medium to reach its intended recipient [2]. Communication systems consist of interconnected networks, systems, stations, and other terminal equipment [3], but in its most basic form, a communication system comprises a transmitter, a receiver, and a transmission medium [4]. The communication process begins when a message is generated that is to be received by others. This message is defined as information or intelligence signals in the context of electronic communication systems. The information is subsequently processed by transmitters, which facilitate the transmission of the message across the established channel [5].

In communication systems, for a message to be transmitted, it must be converted into an electronic format compatible with the transmission medium. This first step is performed by a transmitter, which comprises various electronic components designed to convert the electrical signal into a format suitable for the transmission medium. The communication channel is the medium through which the electronic signal travels from the transmitter to the receiver. The final stage of this process involves the receiver, which comprises components and circuits that take in the signal from the channel and convert it back into the original output signal in a form understandable to humans [5]. The aforementioned conversion of the original signal into an electronic format refers to the process in which data is converted into electrical signals suited for transmission [6]. It is the method used to encode the baseband information onto a carrier signal [7].

Quadrature Amplitude Modulation (QAM) is a modulation technique that combines both phase and amplitude modulation [8] simultaneously in a carrier [9], loading the signals onto two sinusoidal carriers, 90° out of phase [10], and combining them for transmission [11]. The QAM modulator translates the digital packets into a signal that allows seamless data transfer. In QAM, vast amounts of digital information are compressed onto the carrier without consuming more bandwidth during the modulation process [11], hence this method is used to achieve high levels of spectrum usage efficiency [12].

This paper aims to (a) implement and analyze signals as it passes through different blocks in a communication system, (b) to implement a basic communication system using MATLAB, (c) to demonstrate the quadrature modulation and demodulation system, (d) to observe the characteristics of a quadrature amplitude modulation system.

The rest of the paper is as follows: Section II highlights the implementation of the paper, whereas Section III analyzes and discusses the results of the paper. Section IV concludes the paper and discusses possible areas of improvement for future research.

II. Implementation

This section provides a detailed examination of the step-by-step implementation of the quadrature amplitude modulation scheme. It further analyzes the specific functions utilized in the code implementation.

A. Block Diagram

In this paper, a quadrature amplitude modulation scheme is implemented, as shown in the block diagram in Figure 1.

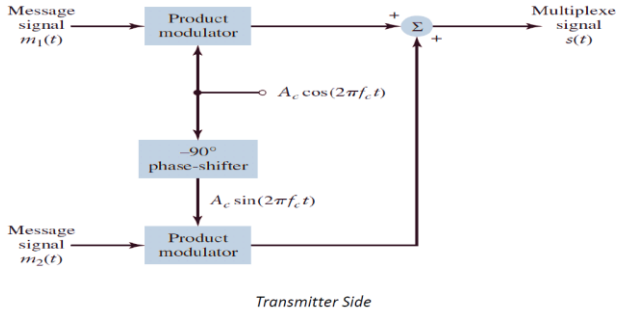


Fig. 1. Modulation scheme (QAM).

In the block diagram, there are two message signals as inputs. The first message signal goes through the product modulator with the carrier signal, and the second message signal goes through the product modulator with the 90-degree phase-shifted carrier signal. The two carriers are referred to as in-phase (I) and quadrature, the phase-shifted signal (Q) [8]. The orthogonal carriers have their amplitudes adjusted and superimposed to generate the signals to be modulated through phase modulation (PM) and amplitude modulation (AM). These carriers are combined through a linear summer to be transmitted over a channel to be received by the demodulator [11].

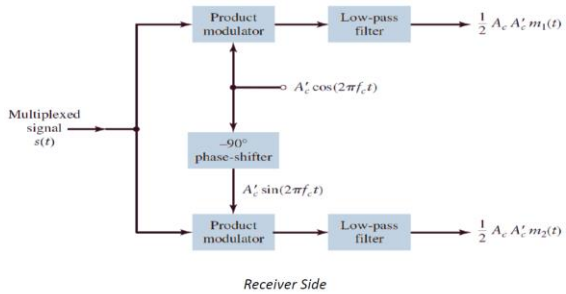


Fig. 2. Demodulation scheme (QAM).

In the block diagram, the signal s(t) will be fed into two branches which essentially reverses the modulation scheme. The quadrature branch and the in-phase branch would then be fed to a low-pass filter. The output of the system would be an accurate representation of the original signal.

B. TASK 1: Signal Loading

The signal loading stage involves preparing and adjusting the two audio signals to ensure they are correctly formatted and compatible for the following stages in the QAM modulation process. By aligning the input signals, Task 1 ensures they are ready for encoding into the QAM system. MATLAB was used because it can handle audio files and signal-processing functions. This stage is implemented using the following MATLAB function:

function [x1_pad,x2_pad,Fs_orig1, Fs_orig2, Fs_orig,L] = load_input(input1,input2)

The initial stage involved defining variables corresponding to the input audio files, representing the signals designated for subsequent processing. It establishes the sources to be subjected to further processing. A custom function, load_input, is created to handle the audio files and align the signals in length and sampling frequency to ensure compatibility for the further stages. The maximum absolute values of the preprocessed signals are determined, which is essential for normalization for scaling and preventing distortion. The time domain parameters allow for the visualization and analysis in the time domain to understand signal behavior. Parameters for the frequency domain, such as the symmetrical frequency range and one-sided frequencies, are defined and calculated to understand spectral characteristics and frequency analysis.

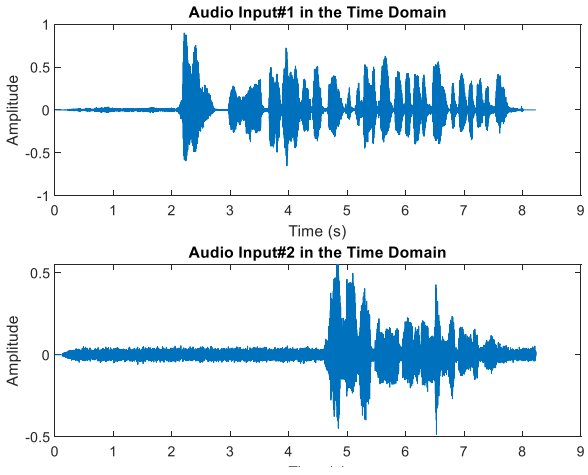


Fig. 3. Input signals in the time domain.

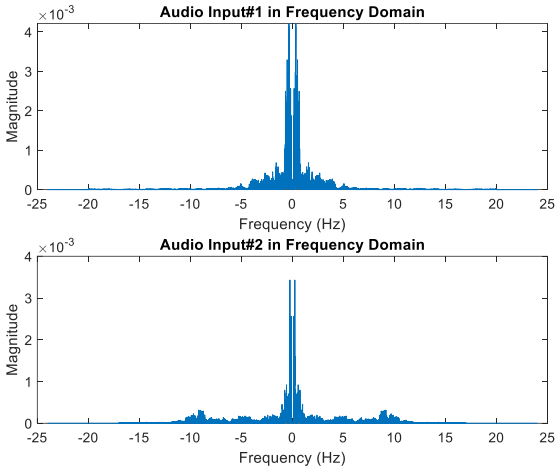


Fig. 4. Input signals in the frequency domain.

C. *Up-Sampling*

In this stage, the objective is to up-sample the signals of both inputs to match the sampling frequency of the high-frequency carrier. This process is essential because the sampling rate must be at least twice the maximum frequency of the signal, as per the Nyquist criterion. This stage corresponds to Task 2 of the MATLAB code, implemented as a function.

function [x1_new, x2_new] = resample_Signals(x1_pad, x2_pad, Fs, Fs_orig),

The function takes as inputs the padded signals (x1_pad and x2_pad), along with frequency parameters such as the sampling frequency and the original sampling frequency. The function outputs x1_new and x2_new, which are resampled versions of the input signals, adjusted to match the high sampling frequency required for the modulation stage. When comparing the resulting signals in Figure 3 and Figure 5, the differences may not be immediately apparent, as the waveforms appear almost identical. However, closer inspection reveals differences in the sampling values of the input audio signals, reflecting the successful up-sampling process.

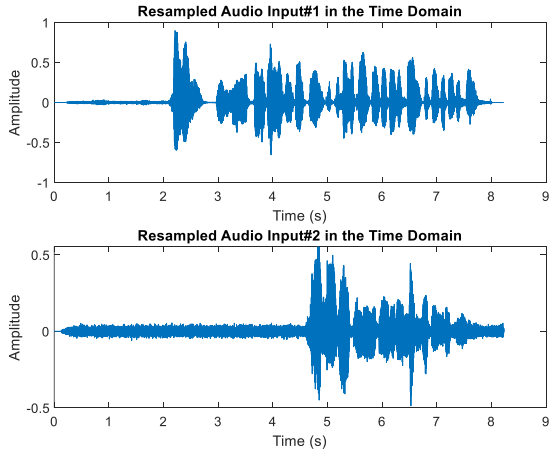


Fig. 5. Resampled signals in the time domain.

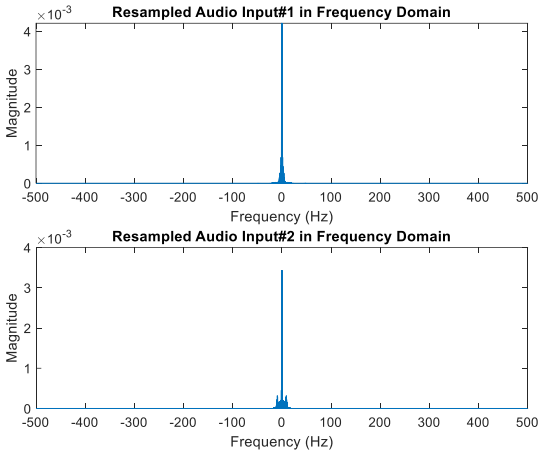


Fig. 6. Resampled signals in the frequency domain.

D. *Quadrature Amplitude Modulation*

After some proper up-sampling of the input signals, this stage is executed using quadrature amplitude modulation (QAM) as the modulation scheme. In this process, the signals are multiplied by the product modulator on both the in-phase and quadrature-phase components, as illustrated in the block diagram in Figure 7. This results in a multiplexed signal that is prepared for demodulation in the subsequent stage. The implementation of this stage is represented in MATLAB as a dedicated function, ensuring an efficient and systematic approach to signal modulation.

[x_AM] = qam_Modulation(x1_new, x2_new, set_fc, t);

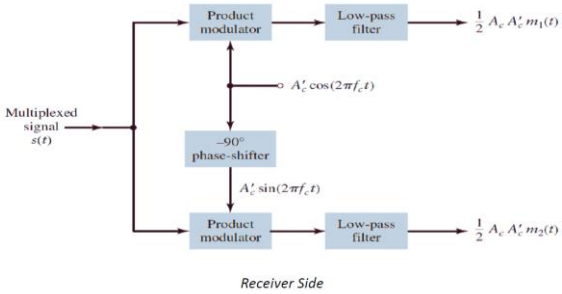


Fig. 7. Modulation scheme (QAM).

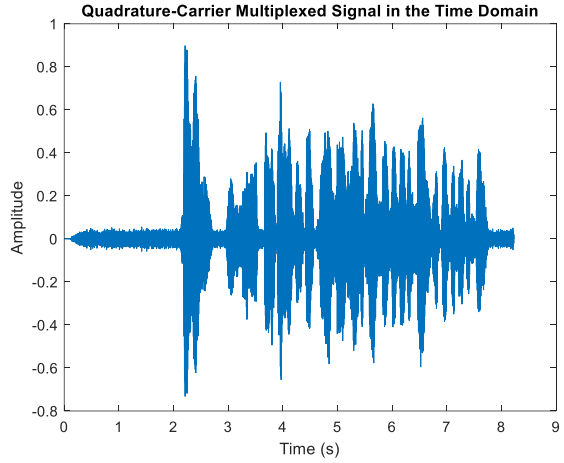


Fig. 8. Quadrature-Carrier Multiplexed signal in the time domain.

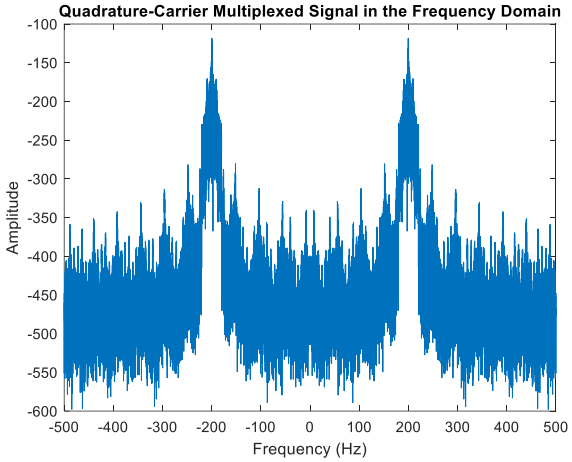


Fig. 9. Quadrature-Carrier Multiplexed Signal in the Frequency Domain.

E. Quadrature Amplitude Demodulation

In the quadrature amplitude demodulation stage, the primary objective is to demodulate the received modulated signal and extract the original information encoded in the previous stage. This stage is implemented in MATLAB as a dedicated function.

function [y_1, y_2, y_filt_1, y_filt_2] = qam_Demodulation(rx, fc, fcut, Fs, t)

The function accepts the following inputs: rx (the received signal), the carrier frequency, the cutoff frequency, the sampling frequency, and the time index. The process of this stage is illustrated in Figure 2, where the multiplexed signal (rx) is multiplied by both the in-phase and quadrature-phase carriers through a product modulator. The resulting signals are then passed through a low-pass filter. As defined in the MATLAB function, the outputs of this stage are y_1 and y_2 (demodulated signals), as well as y_filt_1 and y_filt_2 (filtered versions of the demodulated signals). This step ensures the recovery of the original information with minimal distortion.

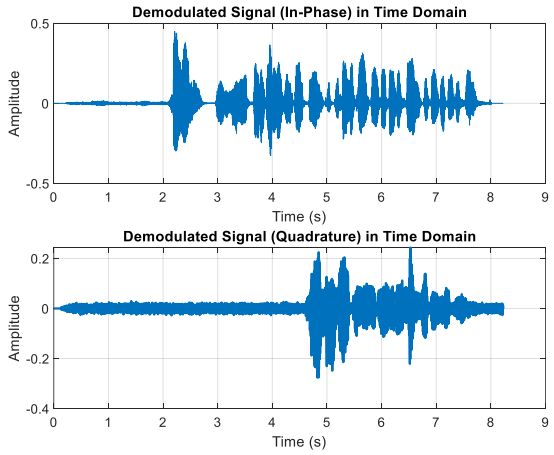


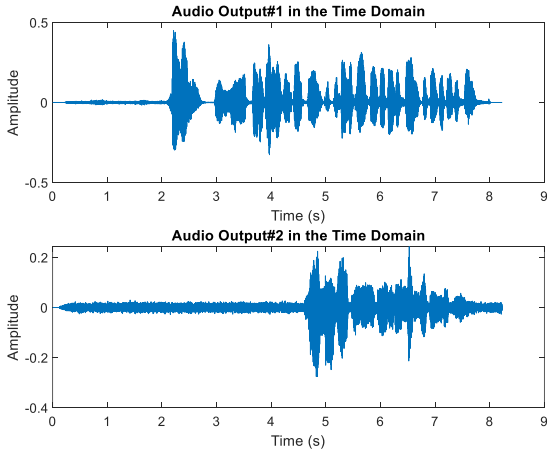
Fig. 10. Demodulated signals in the time domain.

F. Down-Sampling

In the final stage of this project, the signal undergoes down-sampling to revert to its original sampling frequency. This step is crucial to ensure that the signal is restored to its initial state for accurate playback or further processing. The stage is implemented in MATLAB as a dedicated function.

function[y_down1,y_down2]=downsample_Signals(y_filt_1, y_filt_2, Fs, Fs_orig1, Fs_orig2)

The function takes as inputs y_filt_1 and y_filt_2 (the filtered signals), the current sampling frequency, and the original sampling frequencies (Fs_orig1 and Fs_orig2). Unlike the earlier resampling stage where the signal was up sampled to a higher frequency, this function focuses on reducing the sampling rate back to the original, lower frequency. The outputs of this function are y_down1 and y_down2, representing the —



down-sampled versions of the signals. This stage completes the process, ensuring the signals are properly formatted for playback or analysis at their original sampling rates.

Fig. 11. Output signals in the time domain.

III. Analysis And Result

The quality of the output signals in relation to the input signals is quantitatively assessed using the Root Mean Square Error (RMSE), which provides an absolute measure of the deviation between the original and demodulated signals. RMSE is a widely used metric in signal processing as it accounts for both large and small errors in the signal, offering a comprehensive evaluation of the signal's overall accuracy. The RMSE calculation emphasizes the magnitude of discrepancies by squaring the differences between corresponding signal values before averaging and taking the square root. This allows it to penalize larger errors more heavily, which is particularly useful in identifying significant deviations in the signal.

The formula for calculating the RMSE is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}} \quad (1)$$

In the MATLAB Command Window, the following command would be entered to calculate the RMSE:

```
>> rmse = sqrt(mean(([x2_pad 0] - y_down2).^2))
```

This command computes the RMSE by first calculating the squared differences between the original and demodulated signals, averaging them, and then taking the square root to obtain the final value.

The computed RMSE values for the two audio signals are as follows:

TABLE I. Computed RMSE Values for the Two Audio Signals

Root Mean Square Values (RMSE)	
Signals	***RMSE***
Audio Signal #1	0.0518
Audio Signal #2	0.0694

The results indicate that the demodulated signals closely align with their original counterparts, demonstrating a high degree of accuracy in the signal recovery process. An RMSE value approaching zero signifies a near-perfect match, while larger values indicate greater deviation. The relatively low RMSE values obtained in this analysis suggest that the demodulation process effectively preserved the integrity of the original signals.

Beyond numerical evaluation, a subjective assessment was conducted by listening to the output audio files and comparing them to the original signals. Upon reviewing both Audio #1 and Audio #2, it was noted that the quality of the demodulated signals was nearly indistinguishable from the original recordings. The clarity, tone, and overall characteristics of the audio were well-preserved, further corroborating the RMSE analysis findings.

This strong correspondence between the original and demodulated signals highlights a high degree of signal fidelity, confirming that the modulation and demodulation processes were executed with minimal distortion or error.

IV. CONCLUSION

This paper aimed to implement, visualize, and analyze signals through each stage of a Quadrature Amplitude Modulation (QAM) system using MATLAB. The goal was to understand the key processes in electronic communication systems, which transmit information from source to destination. QAM combines phase and amplitude modulation, enabling higher data rates by encoding two parameters on a single carrier. The in-phase and quadrature components were adjusted to create the modulated signal, with MATLAB tools ensuring accurate modulation and demodulation.

Root Mean Square Error (RMSE) analysis showed that the demodulated signal closely matched the original input. The audio output was nearly identical when played back. Future work could explore the use of Additive White Gaussian Noise (AWGN), error correction, audio equalization, and channel impairments to assess system performance under real-world conditions. Other performance metrics such as Bit Error Rate (BER), constellation diagrams, power efficiency, and eye diagrams could also provide a more comprehensive evaluation of system performance.

Distribution of Tasks

Members	**Contribution**
GOMEZ, XYRUZ G.	• ABSTRACT • MATLAB ® CODE IMPLEMENTATION (Task 4, Main Code, Plotting Audio Input 2) • IMPLEMENTATION (Up-Sampling, Modulation, Demodulation, Down-sampling)
SALANIO, SEAN ROWELLE M.	• INTRODUCTION • MATLAB ® CODE IMPLEMENTATION (Task 3, Audio Input 1) • IMPLEMENTATION (Load signal) • CONCLUSION AND RECOMENDATION • APPENDIX
VIOLA, EVITHA JOVEL E.	• INTRODUCTION • MATLAB ® CODE IMPLEMENTATION (Task 1, 2, 5, Main Code, Plotting)

	• IMPLEMENTATION (Block diagram) • CONCLUSION AND RECOMENDATION

REFERENCES

[1] L. Frenzel, "Introduction to Electronic Communication", in Principles Of Electronic Communication System, 4th ed. New York, NY, USA: McGraw-Hill Education, 2016, ch. 1, sec. 1, pp. 1-5

[2] C. Nasser. "Chapter 1 - Introducing Telecommunications." ScienceDirect.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/B978008051867150007X

[3] E. S. A. Ahmed. "Introduction to Communication Systems." ResearchGate.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.researchgate.net/publication/282023883_Introduction_to_Communication_Systems

[4] M. M. Elsayed, K. M. Hosny, M. M. Fouda, M. M. Khashaba, "Vehicles communications handover in 5G: A survey." ScienceDirect.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959522000054

[5] R. Leelaruji, L. Vanfretti, "State-of-the-art in the industrial implementation of protective relay functions, communication mechanism and synchronized phasor capabilities for electric power systems protection." ScienceDirect.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S136403211200318 88

[6] Rohm Semiconductor, "Modulation Method Classification." Rohm.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.rohm.com/electronics-basics/wireless/modulation-methods

[7] M. Plonus, "Signal Modulation" ScienceDirect.com. Accessed: Nov. 2, 2024. [Online]. Available: https://www.sciencedirect.com/topics/physics-and-astronomy/signal-modulation

[8] X. Zhou, "Other Wi-Fi 6 Enhancements", in Wi-Fi 6. Guangdong, China: Huawei Technologies Co., 2021, ch. 7, sec. 2, pp 28-44

[9] L. E. Frenzel Jr., "Chapter 8 - Cell Phones: It Is Now Possible to Do Anything Wirelessly: Talk, Text, Email, Web Browse, Games, Whatever." ScienceDirect.com. Accessed: Nov. 3, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/B9780128116418000084

[10] J. S. Eng, C. Kocot, "Chapter 14 - VCSEL-Based Data Links." ScienceDirect.com. Accessed: Nov. 3, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/B9780123969583000147

[11] R. Anthup, "A Primer on Quadrature Amplitude Modulation (QAM)." MiniCircuits.com. Accessed: Nov. 3, 2024. [Online]. Available: https://blog.minicircuits.com/a-primer-on-quadrature-amplitude-modulation-qam/

[12] A. Zola, "QAM (quadrature amplitude modulation)." TechTarget.com. Accessed: Nov. 3, 2024. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/QAM#:~:text=QAM%20(quadrature%20amplitude%20modulation)%20is,digital%20systems%2C%20like%20wireless%20applications..

APPENDIX

APPENDIX A: FIGURES

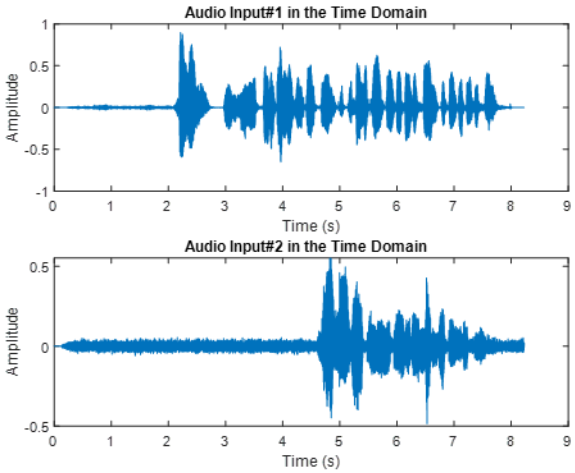


Figure 1. Audio input in the time domain.

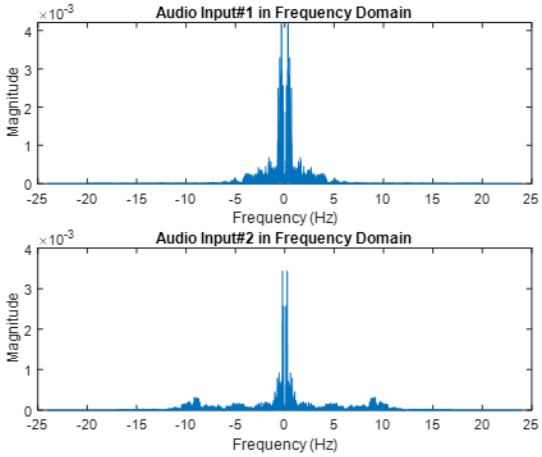


Figure 2. Audio inputs in the frequency domain.

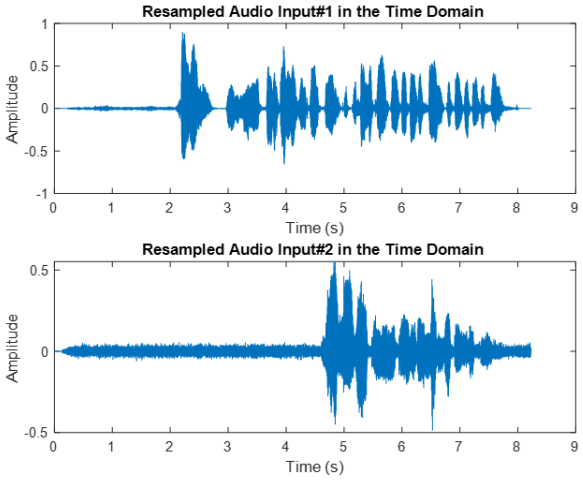


Figure 3. Resampled audio inputs in the time domain.

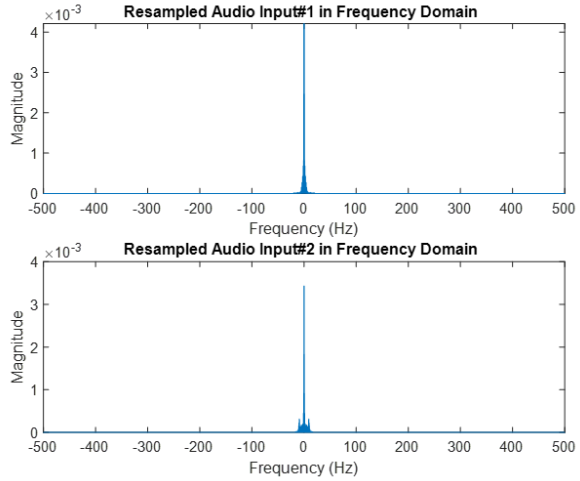


Figure 4. Resampled audios in the frequency domain.

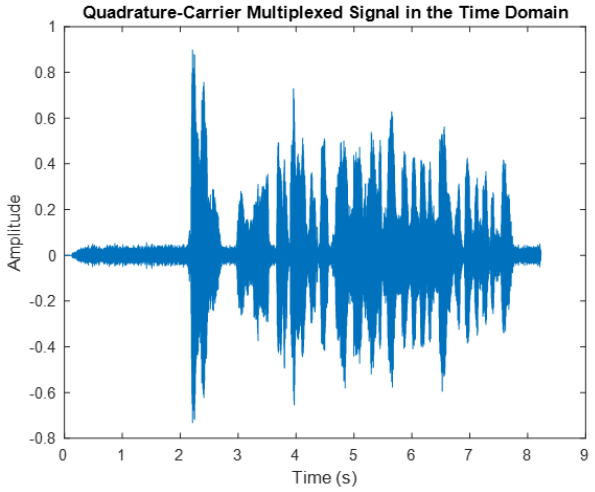


Figure 5. Quadrature-Carrier Multiplexed Signl in the Time domain.

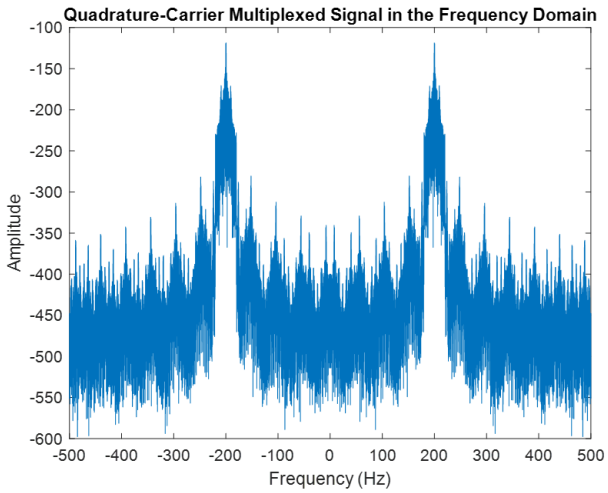


Figure 6. Quadrature-Carrier Multiplexed Signl in the Frequency domain.

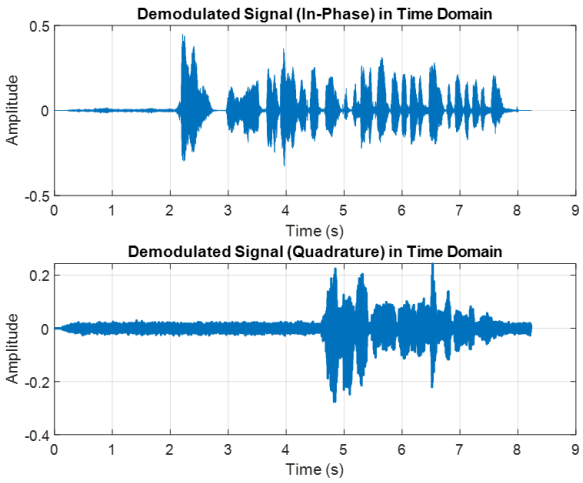


Figure 7. Demodulated Signals in the Time domain.

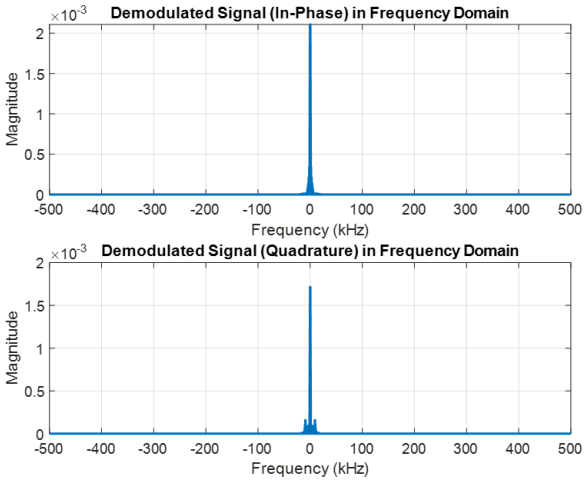


Figure 8. Demodulated Signals in the Time domain.

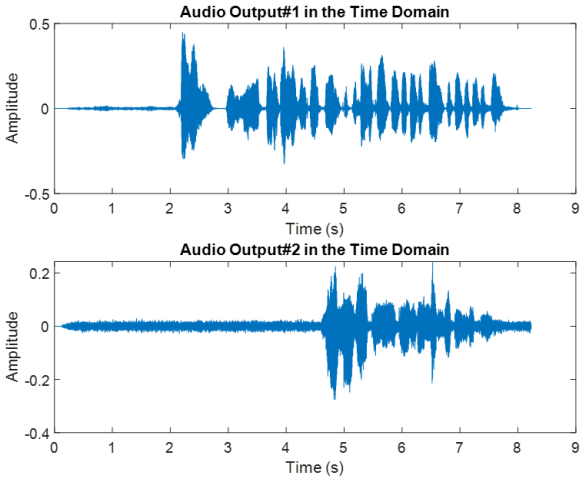


Figure 7. Output Signals in the Time domain.

APPENDIX B: CODES

```
%% Task 1: Input Signal
% Load input signals
input1 = '3ECEA_Group No.7_Salanio,
Sean_INPUT.m4a';
input2 = '3ECEA_Group No.7_Gomez,
Xyruz_INPUT.m4a';

% Input Loading and Reading Function
[x1_pad,x2_pad,Fs_orig1, Fs_orig2, Fs_orig,L] =
load_input(input1,input2);

% Maximum values
x1_pad_max = max(abs(x1_pad));
x2_pad_max = max(abs(x2_pad));

% Period and time
T = 1/Fs_orig; % period
t = 0:T:T*(L-1); % time index

% Frequency
delta_f = Fs_orig/(L-1);
f_delta = -Fs_orig/2:delta_f:Fs_orig/2;
f_s = 0:delta_f:Fs_orig/2;

% Figures
% Time Domain
figure(1);
subplot(2,1,1);
plot(t,x1_pad);
title('Audio Input#1 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, x2_pad);
title('Audio Input#2 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

% Frequency Domain
X1_freq = fftshift(fft(x1_pad))/length(x1_pad);
X2_freq = fftshift(fft(x2_pad))/length(x2_pad);
figure(2);
subplot(2,1,1);
plot(f_delta/(1e3),abs(X1_freq));
title('Audio Input#1 in Frequency Domain')
xlabel('Frequency (Hz)')
ylabel('Magnitude')


subplot(2,1,2);
plot(f_delta/(1e3),abs(X2_freq));
title('Audio Input#2 in Frequency Domain')
xlabel('Frequency (Hz)')
ylabel('Magnitude')


%% Task 2: Resampling the Input Signals
% Set new Fmax
Fmax = 500e3;
Fs = 2 * Fmax; % Define new sampling frequency

% Up-Sampling the Signals
[x1_new, x2_new] = resample_signal(x1_pad, x2_pad, Fs,
Fs_orig);

% Redefine period and time
L = length(x1_new);
T = 1/Fs;
t = 0:T:T*(L-1);

% Redefine frequency
fmax = Fs/2;
delta_f = Fs/(L-1);
f_delta = -Fs/2:delta_f:Fs/2;

% Figures
% Time Domain
figure(3);
subplot(2,1,1);
plot(t,x1_new);
title('Resampled Audio Input#1 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, x2_new);
title('Resampled Audio Input#2 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

% Frequency Domain
X1_resampled = fftshift(fft(x1_new))/length(x1_new);
X2_resampled = fftshift(fft(x2_new))/length(x2_new);

figure(4);
subplot(2,1,1);
plot(f_delta/(1e3),abs(X1_resampled));
title('Resampled Audio Input#1 in Frequency Domain')
xlabel('Frequency (Hz)')
ylabel('Magnitude')

subplot(2,1,2);
plot(f_delta/(1e3),abs(X2_resampled));
title('Resampled Audio Input#2 in Frequency Domain')
xlabel('Frequency (Hz)')
ylabel('Magnitude')

%% Task 3: Modulation
set_fc = 200e3; % Set carrier frequency
```

```
% Quadrature Amplitude Modulation
[x_AM] = qam_modulation(x1_new, x2_new, set_fc, t);

% Figures
% Time Domain
figure(5);
plot(t, x_AM,'LineWidth',0.5);
title('Quadrature-Carrier Multiplexed Signal in the Time
Domain');
xlabel('Time (s)');
ylabel('Amplitude');

% Freq
figure(6);
X_AM_d = fftshift(fft(x_AM))/length(x_AM);
plot(f_delta/(1e3),2.25*20*log10(X_AM_d));
title('Quadrature-Carrier Multiplexed Signal in the
Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Amplitude');

%% Task 4: Demodulate
f_cutoff = 20e3; % LPF cut-off frequency

% Quadrature Amplitude Demodulation
[y_1, y_2, y_filt_1, y_filt_2] = qam_demodulation(x_AM,
set_fc, f_cutoff, Fs, t);

% Filtered Version:
% Double-sided frequency spectrum
Y_filt_d1 = fftshift(fft(y_filt_1))/length(y_filt_1);
Y_filt_d2 = fftshift(fft(y_filt_2))/length(y_filt_2);

% Single-sided frequency spectrum
Y_filt_s1 = 2 * Y_filt_d1((L-1)/2 + 1:end);
Y_filt_s2 = 2 * Y_filt_d2((L-1)/2 + 1:end);

% Figures
% In-Phase
figure(7);
subplot(2,1,1);
plot(t, y_filt_1);
title('Demodulated Signal (In-Phase) in Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, y_filt_2);
title('Demodulated Signal (Quadrature) in Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

% Freq
figure(8);
subplot(2,1,1);
plot(f_delta/(1e3), abs(Y_filt_d1), 'LineWidth', 1.5);
title('Demodulated Signal (In-Phase) in Frequency
Domain');
xlabel('Frequency (kHz)');
ylabel('Magnitude');

subplot(2,1,2);
plot(f_delta/(1e3), abs(Y_filt_d2), 'LineWidth', 1.5);
title('Demodulated Signal (Quadrature) in Frequency
Domain');
xlabel('Frequency (kHz)');
ylabel('Magnitude');

%% Task 5: Downsample
% Down-Sampling the Signals
[y_down1, y_down2] = downsample_signal(y_filt_1,
y_filt_2, Fs, Fs_orig1, Fs_orig2);

% Period and time
L = length(y_down1);
T = 1/Fs_orig; % period
t = 0:T:T*(L-1); % time indices

% Figures
% Time Domain
figure(9);
subplot(2,1,1);
plot(t,y_down1);
title('Audio Output#1 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, y_down2);
title('Audio Output#2 in the Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');

% Play the downsampled sounds
sound(y_down1, Fs_orig1)
pause(7)
sound(y_down2, Fs_orig2)

% Save the downsampled audio signals as files
audiowrite('3ECEA_Group No.7_Salanio,
Sean_OUTPUT.wav', y_down1, Fs_orig1); % Save Audio
Output #1
audiowrite('3ECEA_Group No.7_Gomez,
Xyruz_OUTPUT.wav', y_down2, Fs_orig2); % Save Audio
Output #2
```

```
function [x1_pad, x2_pad, Fs_orig1, Fs_orig2, Fs_orig, L]
= load_input(input1, input2)
% load_input - Loads two audios and pads zeros so the two
input audios are
% of equal length.
%
```

```
% Inputs:
% input 1 - First audio file
% input 2 - Second audio file
%
% Outputs:
% x1_pad - First audio with padded zeroes
% x2_pad - Second audio with padded zeroes
% Fs_1 - Sampling frequency of the first audio
% Fs_2 - Sampling frequency of the second audio
% Fs - Nyquist Frequency
% Fs_orig - Maximum Frequency
%
% GROUP 7 - TASK 1 of Machine Problem

% Loading the input Files
   [x1, Fs_orig1] = audioread(input1);
   [x2, Fs_orig2] = audioread(input2);

   % Transpose the signals (if needed)
   x1 = x1';
   x2 = x2';

   % Resample both signals to the highest sampling rate
   if Fs_orig1 ~= Fs_orig2
      % Resample the signals to the highest original
sampling rate
      if Fs_orig1 > Fs_orig2
         x2 = resample(x2, Fs_orig1, Fs_orig2);
         Fs_orig2 = Fs_orig1;
      elseif Fs_orig2 > Fs_orig1
         x1 = resample(x1, Fs_orig2, Fs_orig1);
         Fs_orig1 = Fs_orig2;
      end
   end

   % Selecting the longer signal and applying zero padding
to match lengths
   length_x1 = length(x1);
   length_x2 = length(x2);
   zero_pad = zeros(1, abs(length_x1 - length_x2));

   if length_x1 > length_x2
      L = length(x1);
      x2_pad = [x2(1,:), zero_pad];
      x1_pad = x1(1,:);
   elseif length_x2 > length_x1
      L = length(x2);
      x1_pad = [x1(1,:), zero_pad];
      x2_pad = x2(1,:);
   else
      L = length_x2;
      x1_pad = x1(1,:);
      x2_pad = x2(1,:);
   end

   % Selecting the final common sampling rate
   Fs_orig = Fs_orig1;  % Now both signals should have
the same sampling rate
end
```

```
function [x1_new, x2_new] = resample_signal(x1_pad,
x2_pad, Fs, Fs_orig)
% resample_signal - Resamples the signal with two input
signals with a new
% sampling frequency Fs.
%
% Inputs:
% x1_pad - First audio with padded zeroes
% x2_pad - Second audio with padded zeroes
% Fs - Desired sampling frequency
% Fs_orig - Original Sampling Frequency
%
%
% Outputs:
% x1_new - Resampled first audio with the new sampling
freq
% x2_new - Resampled second audio with the new
sampling freq
%
% GROUP 7 - Task 2 of Machine Problem

x1_new = resample(x1_pad, Fs, Fs_orig);
x2_new = resample(x2_pad, Fs, Fs_orig);

end
```

```
function [x_AM] = qam_modulation(x1_new, x2_new, set_fc, t)
% qam_modulation - Two signals will go through the in-phase
branch and quadrature branch. Output of this block must be a
single QAM signal that will pass through the channel.
%
% Inputs:
% x1_new - Resampled first audio signal (from Task 2:
Resample)
% x2_new - Resampled second audio signal (from Task 2:
Resample)
% set_fc - Carrier frequency for modulation
% t - Time vector for the signals
%
% Outputs:
% x_AM - Modulated QAM signal of I and Q channels by
Linear Summer


% The in-phase modulated signal
InPhase_Carrier = cos(2*pi*set_fc*t);
x1_InPhase = x1_new .* InPhase_Carrier;

% The in-quadrature modulated signal
InQuadrature_Carrier = -sin(2*pi*set_fc*t);
x2_InQuadrature = x2_new .* InQuadrature_Carrier;

% QAM signal from the sum of the signals in quadrature
x_AM = x1_InPhase + x2_InQuadrature;
```

```
end
```

```
function [y_1, y_2, y_filt_1, y_filt_2] =
qam_demodulation(rx, fc, fcut, Fs, t)
% qam_demodulation - QAM Demodulation Function
%
% Inputs:
% rx     - Received QAM signal to be demodulated
% set_fc - Carrier frequency used for modulation and
demodulation
% fcut   - Cutoff frequency for the low-pass filter
% Fs     - Sampling frequency
% t      - Time vector for the signal
%
% Outputs:
% y_1    - Demodulated in-phase (I) signal before filtering
% y_2    - Demodulated quadrature (Q) signal before
filtering
% y_filt_1 - Filtered in-phase (I) signal
% y_filt_2 - Filtered quadrature (Q) signal
%
% GROUP 7 - Task 4 Machine Problem

% Low-Pass Filter
order = 6;
[b, a] = butter(order, fcut/(Fs/2), 'low');

% Retrieving the in-phase message signal
carrier_sin = cos(2 * pi * fc * t);
y_1 = rx .* carrier_sin;
y_filt_1 = filtfilt(b, a, y_1);
```

```
% Retrieving the in-quadrature message signal
carrier_cos = sin(2 * pi * fc * t);
y_2 = rx .* carrier_cos;
y_filt_2 = filtfilt(b, a, y_2);
end
```

```
function [y_down1, y_down2] = downsample_signal(y_filt_1,
y_filt_2, Fs, Fs_orig1, Fs_orig2)
% downsample_signal - Resamples the demodulated signal to their
original
% sampling frequency. It also plays the sound of the resampled
signals.
%
% Inputs:
% y_filt_1 - Filtered first audio
% y_filt_2 - Filtered second audio
% Fs - Sampling frequency
% Fs_orig1 - Original sampling frequency of the first audio
% Fs_orig2 - Original sampling frequency of the second audio
%
% Outputs:
% y_down1 - Resampling the first audio to the original sampling
freq
% y_down2 - Resampling the second audio to the original sampling
freq
%
% GROUP 7 - Task 5 of Machine Problem


y_down1 = resample(y_filt_1, Fs_orig1, Fs);
y_down2 = resample(y_filt_2, Fs_orig2, Fs);
end
```