Digital Signal Processing Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Mohit Akhouri - 19ucc023

Course Coordinator
Dr. Divyang Rawal

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

September 2021

Contents

Chapter *7*

Experiment - 7

7.1 Aim of the Experiment

- Discrete Cosine Transform and it's energy compaction property
- Simulink based Audio compression

7.2 Software Used

- MATLAB
- Simulink

7.3 Theory

7.3.1 About Audio Compression :

Audio data compression has the potential to reduce the transmission bandwidth and storage requirements of audio data. Audio compression algorithms are implemented in software as **audio codecs**. In both lossy and lossless compression, information redundancy is reduced, using methods such as coding, quantization, **discrete cosine transform** and **linear prediction** to reduce the amount of information used to represent the uncompressed data. Lossy audio compression algorithms provide higher compression and are used in numerous audio applications including **Vorbis** and **MP3**. These algorithms almost all rely on **psychoacoustics** to eliminate or **reduce fidelity** of less audible sounds, thereby reducing the space required to store or transmit them.

Lossless audio compression produces a representation of digital data that can be decoded to an exact digital duplicate of the original. Compression ratios are around 50–60 % of the original size, which is similar to those for generic lossless data compression. **Lossless codecs** use curve fitting or linear prediction as a basis for estimating the signal. Parameters describing the estimation and the difference between the estimation and the actual signal are coded separately.

7.3.2 About Discrete Cosine Transform (DCT) :

A **discrete cosine transform (DCT)** expresses a finite sequence of data points in terms of a **sum of cosine functions oscillating at different frequencies**. The DCT, first proposed by **Nasir Ahmed** in 1972, is a widely used transformation technique in **signal processing** and **data compression**. It is used in most digital media, including **digital audio** (such as Dolby Digital, MP3 and AAC), **digital radio** (such as AAC+ and DAB+), and **speech coding** (such as AAC-LD, Siren and Opus). DCTs are also important to numerous other applications in science and engineering like **Partial Differential Equations**.

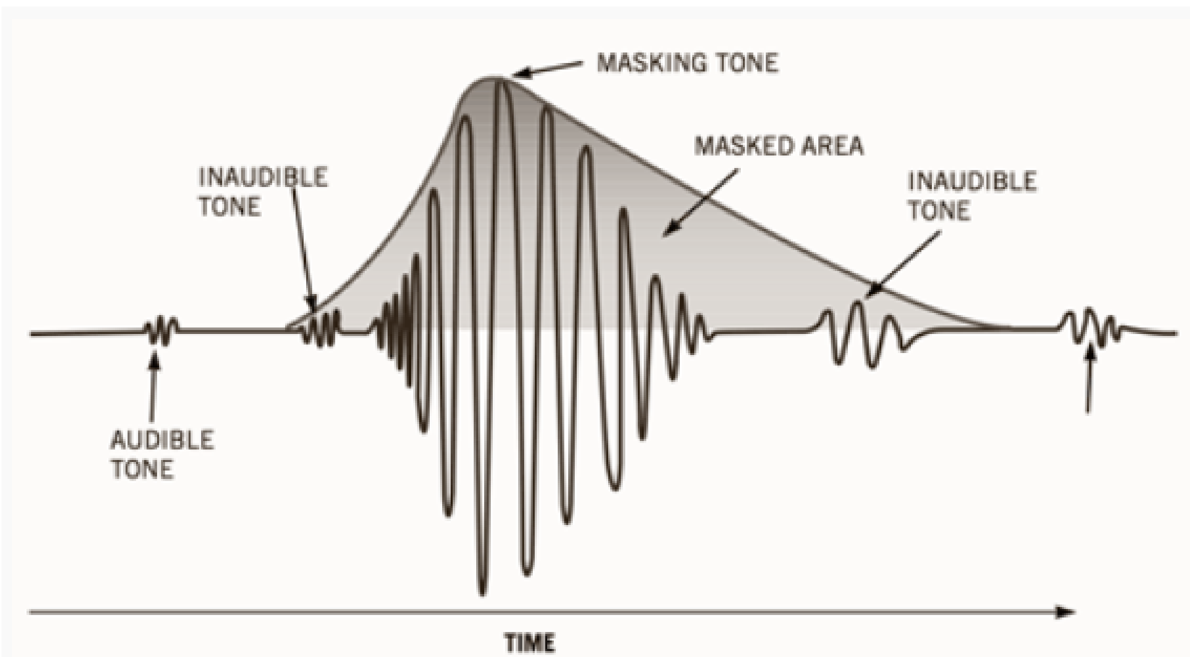

Figure 7.1 Lossy Compression of mp3 audio signal

7.3.2.1 Expression used for calculation of DCT of audio file :

Let **x** be the audio file of size 1 x N. The corresponding 1D-Discrete Cosine Transform **y** is given as:

$$y(k) = w(k) \sum_{n=1}^{N} x(n)cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right) \tag{7.1}$$

In the above equation , **w** function is defined as :

$$w(k) = \frac{1}{\sqrt{N}} \qquad k = 1$$

$$w(k) = \sqrt{\frac{2}{N}} \qquad k > 1$$

7.3.2.2 Expression used for calculation of 1D-IDCT for reconstruction of audio signal :

The **1D-IDCT** expression is given as :

$$x_{idc}(n) = \sum_{k=1}^{N} w(k)y(k)cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right) \tag{7.2}$$

In the above equation , **w** function is defined as :

$$w(k) = \frac{1}{\sqrt{N}} \qquad k = 1$$

$$w(k) = \sqrt{\frac{2}{N}} \qquad k > 1$$

7.3.3 How Compression of Audio helps :

Compression can be used to subtly **massage** a track to make it more **natural sounding** and **intelligible** without adding distortion, resulting in a song that's more "comfortable" to listen to. Additionally, many compressors — both hardware and software — will have a signature sound that can be used to inject wonderful coloration and tone into otherwise lifeless tracks.

Alternately, over-compressing your music can really squeeze the life out of it. Having a good grasp of the basics will go a long way toward understanding how compression works, and confidently using it to your advantage.

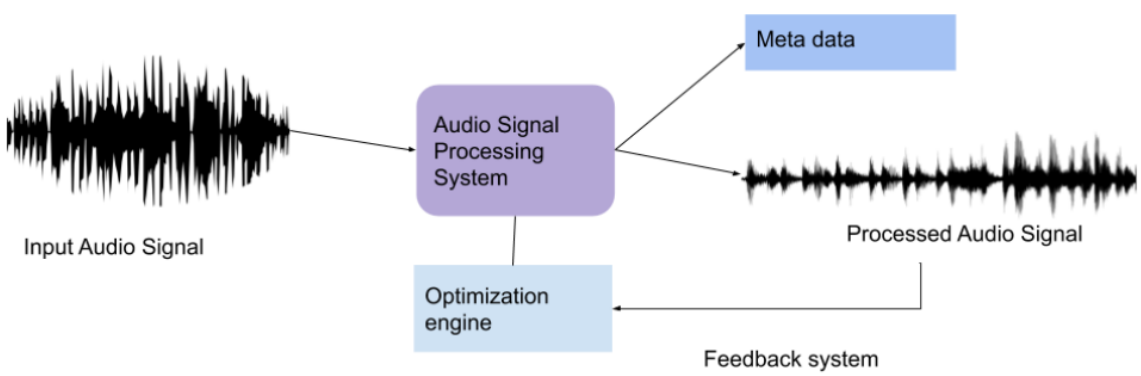

Figure 7.2 Block Diagram of Audio Processing

7.4 Code and results

7.4.1 Using Inbuilt DCT and IDCT to compress and reconstruct any input audio signal :

```
% 19ucc023
% Mohit Akhouri
% Experiment 7 - Observation 1

% In this code , we take the input of an audio signal
% We calculate the 1D-DCT ( via inbuilt function dct ) to obtain the
% compressed audio wave
% Later on , we calculate the inbuilt IDCT of the compressed wave to
 get
% the reconstructed approx. Original Sound signal

clc;
clear all;
close all;

[x,fs] = audioread('input.wav'); % Reading of audio file 'input.wav'

% Plot of Original Sound wave 'input.wav'
figure;
plot(x);
xlabel('time(t) ->');
ylabel('x(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Original Sound wave
 input.wav');
grid on;

dct_audio = dct(x); % Calculation of DCT of input.wav via INBUILT DCT
 - Compression of audio file

% Plot of INBUILT DCT of input.wav
figure;
plot(dct_audio);
xlabel('frequency (Hz) ->');
ylabel('x_{DCT}(f) ->');
title('19ucc023 - Mohit Akhouri','Plot of DCT of Sound Wave input.wav
 obtained via INBUILT FUNCTION');
grid on;

audiowrite('Obs1_DCT.wav',dct_audio,fs); % Writing dct_audio to audio
 file

idct_audio = idct(dct_audio); % Reconstructed Approx. Original audio
 wave via INBUILT IDCT

% Plot of Reconstructed Audio wave
figure;
plot(idct_audio);
xlabel('time(t) ->');
ylabel('x_{IDCT}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Original Sound
 wave obtained via INBUILT IDCT');
grid on;
```

Figure 7.3 Part 1 of the code for observation 1

```
audiowrite('Obs1_IDCT.wav',idct_audio,fs); % Writing Reconstructed
 audio wave to a audio file
```

Figure 7.4 Part 2 of the code for observation 1

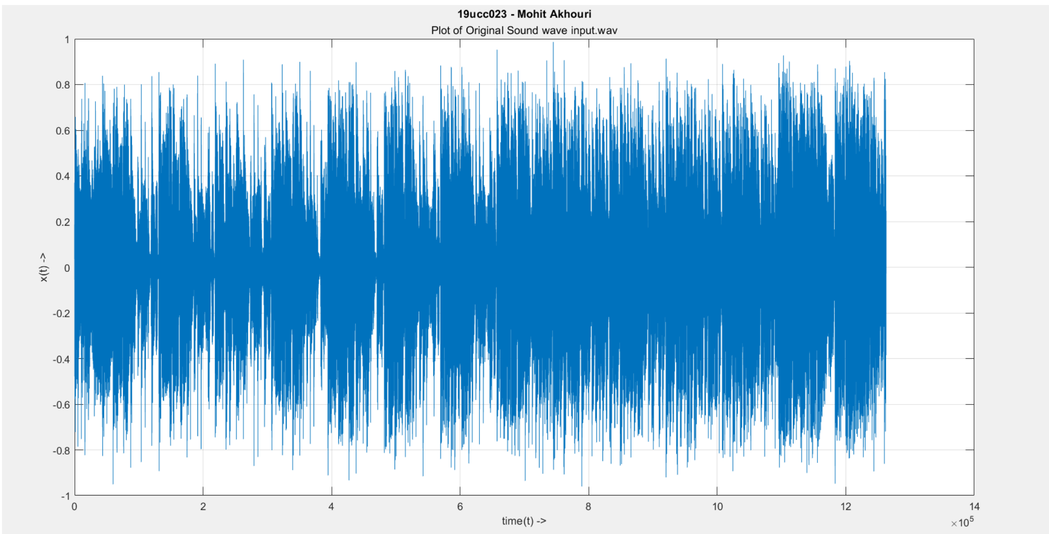

Figure 7.5 Plot of Original Audio signal

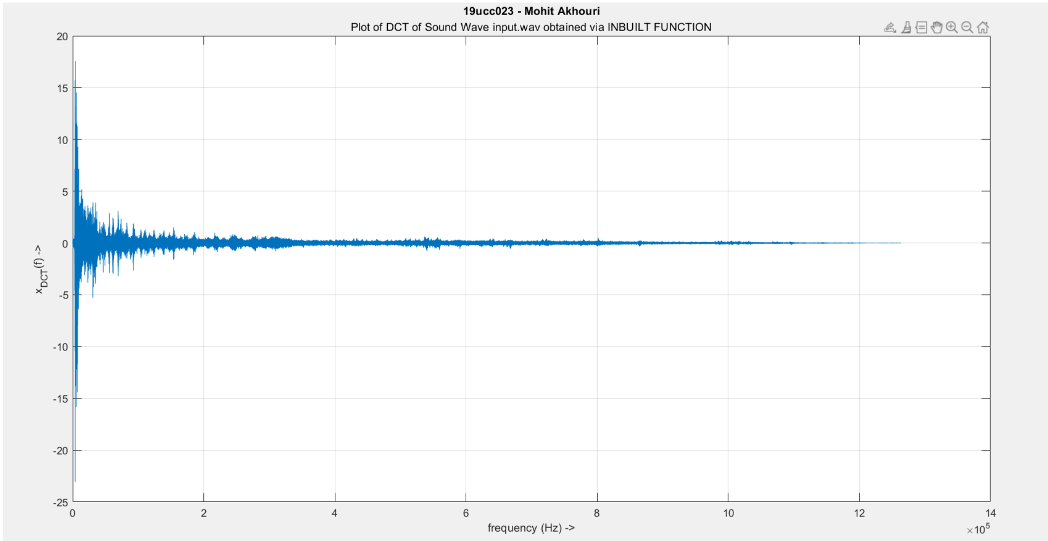

Figure 7.6 Plot of the 1D-DCT of the Audio Signal using INBUILT function

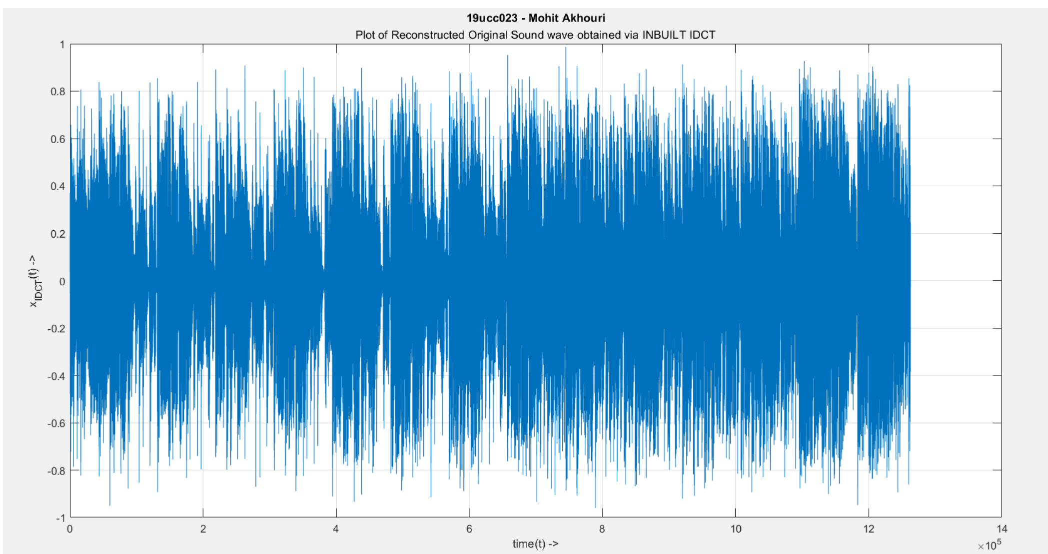

Figure 7.7 Plot of the Reconstructed Audio signal using Inbuilt 1D-IDCT

7.4.2 Using User-Defined functions for calculation of DCT and IDCT of audio signal :

```
% 19ucc023
% Mohit Akhouri
% Experiment 7 - Observation 2

% In this code , we take the input of an audio signal
% We calculate the 1D-DCT ( via user-defined function myCompression )
 to obtain the
% compressed audio wave
% Later on , we calculate the user-defined IDCT ( myDeCompression ) of
 the compressed wave to get
% the reconstructed approx. Original Sound signal

clc;
clear all;
close all;

[x,fs] = audioread('input.wav'); % Reading of audio file 'input.wav'

% Plot of Original Sound wave 'input.wav'
figure;
plot(x);
xlabel('time(t) ->');
ylabel('x(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Original Sound wave
 input.wav');
grid on;

dct_audio_userdefined = myCompression(x); % Calculation of DCT of
 input.wav via USER-DEFINED DCT (myCompression.m) - Compression of
 audio file
dct_audio_inbuilt = dct(x); % Calculation of DCT of input.wav via
 INBUILT DCT - Compression of audio file

% Plot of USER-DEFINED DCT of input.wav
figure;
subplot(2,1,1);
plot(dct_audio_userdefined);
xlabel('frequency (Hz) ->');
ylabel('x_{DCT}(f) ->');
title('Plot of DCT of Sound Wave input.wav obtained via USER-DEFINED
 FUNCTION');
grid on;

% Plot of INBUILT DCT of input.wav
subplot(2,1,2);
plot(dct_audio_inbuilt);
xlabel('frequency (Hz) ->');
ylabel('x_{DCT}(f) ->');
title('Plot of DCT of Sound Wave input.wav obtained via INBUILT
 FUNCTION');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');
```

Figure 7.8 Part 1 of the Code for the observation 2

```
audiowrite('Obs2_DCT.wav',dct_audio_userdefined,fs); % Writing
 dct_audio to audio file

idct_audio_userdefined = myDeCompression(dct_audio_userdefined); %
 Reconstructed Approx. Original audio wave via USER-DEFINED IDCT
idct_audio_inbuilt = idct(dct_audio_inbuilt); % Reconstructed Approx.
 Original audio wave via INBUILT IDCT

% Plot of Reconstructed Audio wave via USER-DEFINED IDCT
figure;
subplot(2,1,1);
plot(idct_audio_userdefined);
xlabel('time(t) ->');
ylabel('x_{IDCT}(t) ->');
title('Plot of Reconstructed Original Sound wave obtained via USER-
DEFINED IDCT');
grid on;

% Plot of Reconstructed Audio wave via INBUILT IDCT
subplot(2,1,2);
plot(idct_audio_inbuilt);
xlabel('time(t) ->');
ylabel('x_{IDCT}(t) ->');
title('Plot of Reconstructed Original Sound wave obtained via INBUILT
 IDCT');
grid on;
sgtitle('19ucc023 - Mohit Akhouri');

audiowrite('Obs2_IDCT.wav',idct_audio_userdefined,fs);
```

Figure 7.9 Part 2 of the Code for the observation 2

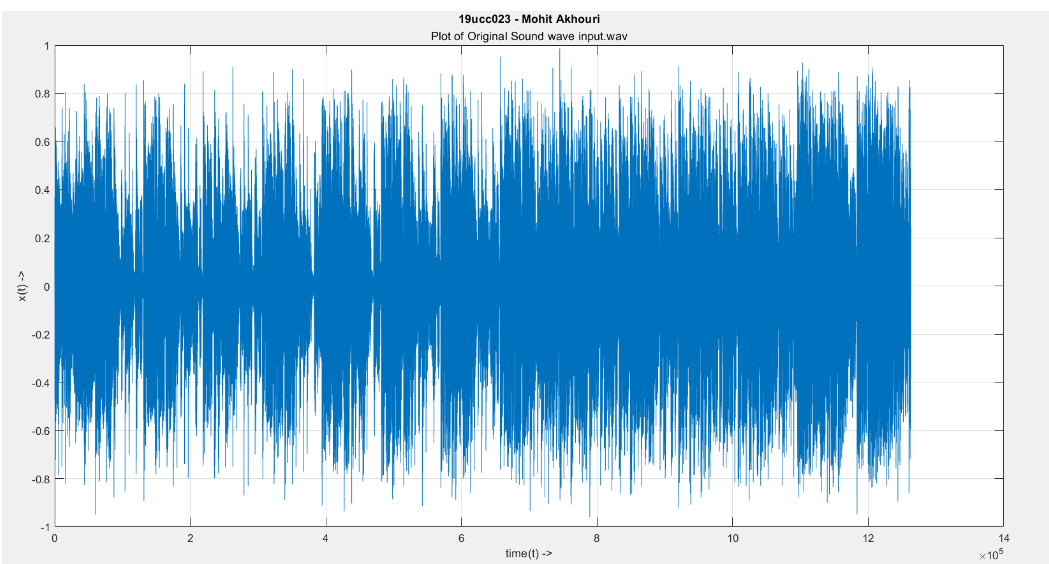

Figure 7.10 Plot of Original Audio signal

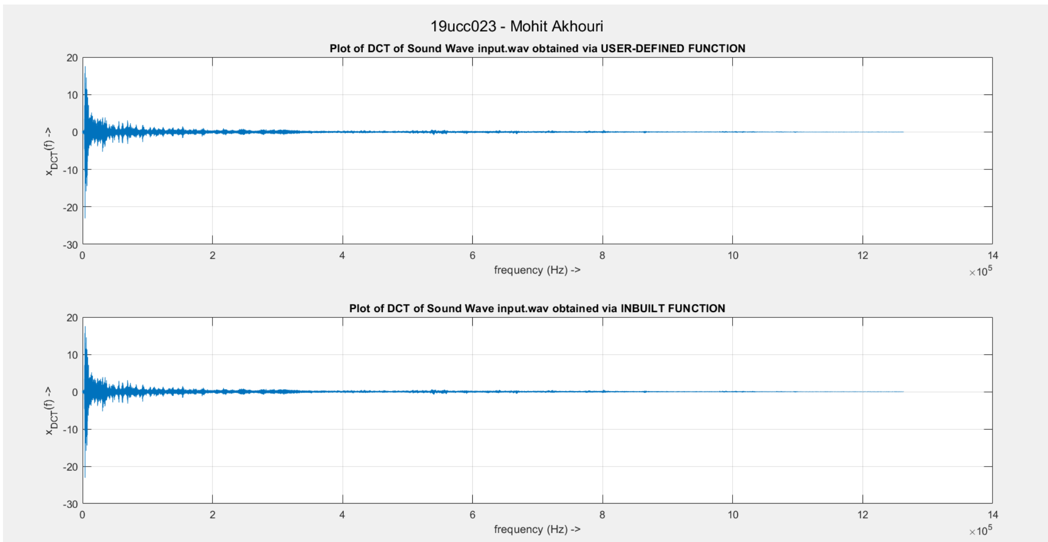

Figure 7.11 Plots of the USER-DEFINED DCT and INBUILT DCT of audio signal

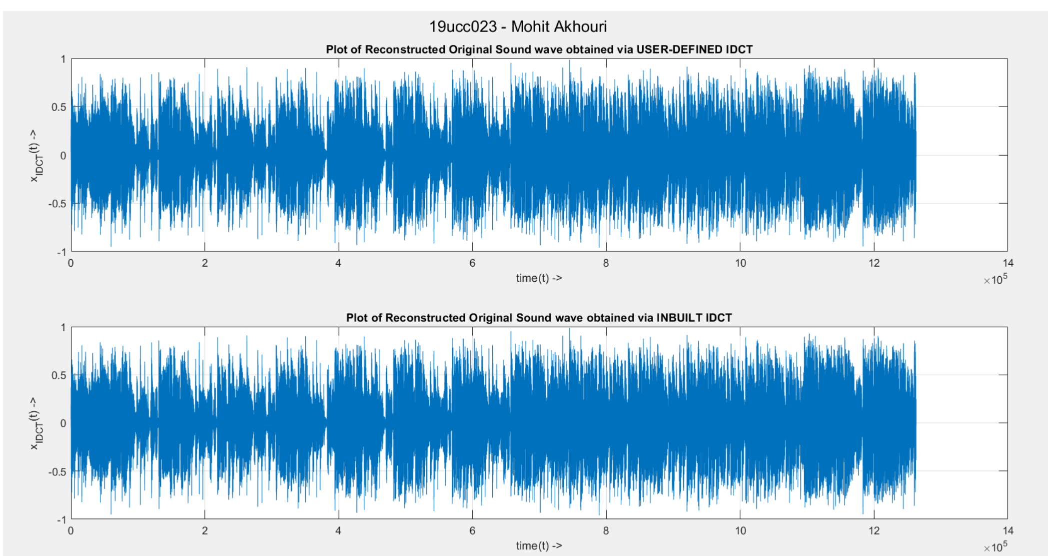

Figure 7.12 Plots of the USER-DEFINED IDCT and INBUILT IDCT

7.4.3 Division of Audio file into blocks of size 1x256 and applying threshold :

```
% 19ucc023
% Mohit Akhouri
% Experiment 7 - Observation 3

% In this code , we take the input of audio file and divide the whole
% audio file into blocks of size 1x256 . Now we apply DCT on each
% individual block and Apply the COMPRESSION ALGORITHM

% COMPRESSION ALGORITHM : In this we decide the threshold ( minimum
% threshold and maximum threshold ) , we reject the coefficient values
% which are between the threshold values and accept the rest. This is
 how
% we achieve compression.

% Finally we plot the reconstructed wave and also write it to an audio
 file

clc;
clear all;
close all;

[x,fs] = audioread('input.wav'); % Reading of audio file 'input.wav'

% Plot of Original Sound wave 'input.wav'
figure;
plot(x);
xlabel('time(t) ->');
ylabel('x(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Original Sound wave
 input.wav');
grid on;

sound(x,fs); % To hear the original sound wave via speakers

N = size(x,1); % Row size of the input audio signal

index = ceil(N/256); % calculating the index for division of blocks

% We adjust the input audio signal 'x(t)' so as it is DIVISIBLE by 256

% We pad the extra spaces by zeros
x = [x ; zeros(256*index - N,1) ];

N = size(x,1); % Re-calculation of size after adjustment of audio
 signal x(t)

threshold1 = 0.09; % Maximum threshold
threshold2 = -0.09; % Minimum threshold

% Main loop algorithm for the compression of audio signal starts here
for i=1:256:N
```

Figure 7.13 Part 1 of the code for observation 3

```
    x_block = x(i:i+255); % Dividing input audio signal into 1x256
 blocks
    x_block_dct = myCompression(x_block); % Taking DCT of 1x256 block

    % Loop for the checking of coefficients
    % Those coefficients are rejected , which are between the
 threshold
    % values , rest are accepted
    for j=1:256
        if (x_block_dct(j) >= threshold2 && x_block_dct(j) <=
 threshold1)
            x_block_dct(j) = 0;
        end
    end

    x_block_idct = myDeCompression(x_block_dct); % Inverse DCT of the
 Compressed Audio block
    x(i:i+255) = x_block_idct; % Storing the IDCT into reconstructed
 wave variable x
end

% Plot of the reconstructed Compressed wave after applying threshold
 values
% for compression
figure;
plot(x);
xlabel('time(t) ->');
ylabel('x_{reconstructed}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Sound Wave
 when - removing coefficients between -0.09 and 0.09');
grid on;

audiowrite('Obs3_outputcompression.wav',x,fs); % Writing Compressed
 audio signal to a file
sound(x,fs); % To hear the compressed audio signal from speakers
```

Figure 7.14 Part 2 of the Code for the observation 3

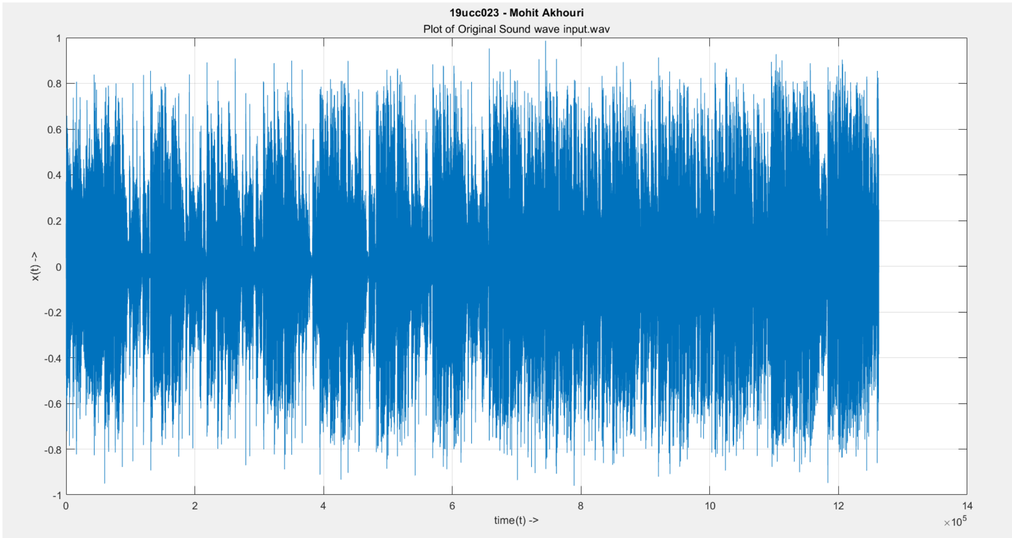

Figure 7.15 Plot of the Original Audio Signal

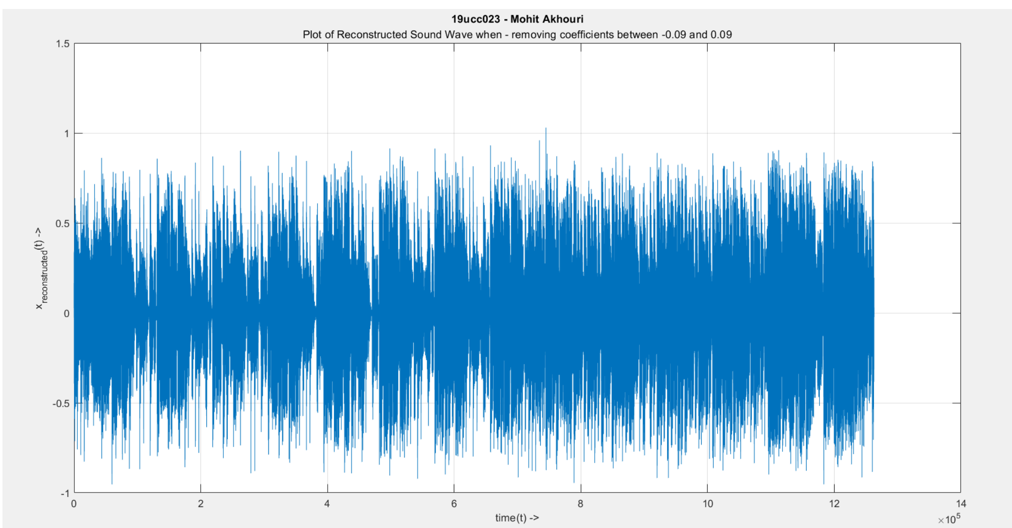

Figure 7.16 Plot of the **Compressed** Audio Signal after applying threshold

7.4.4 Applying Compression Algorithm for different threshold values :

```
% 19ucc023
% Mohit Akhouri
% Experiment 7 - Observation 4

% In this code , we take the input of audio file and divide the whole
% audio file into blocks of size 1x256 . Now we apply DCT on each
% individual block and Apply the COMPRESSION ALGORITHM

% COMPRESSION ALGORITHM : In this we decide the threshold ( minimum
% threshold and maximum threshold ) , we reject the coefficient values
% which are between the threshold values and accept the rest. This is
 how
% we achieve compression.

% We repeat the above compression algorithm for different threshold
 values
% and plot the graph between Mean square error (MSE) and Compression
 ratio

clc;
clear all;
close all;

[x,fs] = audioread('input.wav'); % Reading of audio file 'input.wav'

N = size(x,1); % Row size of the input audio signal

index = ceil(N/256); % calculating the index for division of blocks
x_orig = [x ; zeros(256*index-N,1)]; % storing the "padded with zeros"
 original signal in new variable
N = size(x_orig,1); % Re-calculation of size after adjustment of audio
 signal x(t)

% Plot of the original signal input.wav
figure;
plot(x_orig);
xlabel('time(t) ->');
ylabel('x(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Original Sound Wave
 input.wav ');
grid on;

% sound(x_orig,fs);

% Case 1 = removing coefficients for values between -0.09 and 0.09

x_recon = zeros(N,1); % Initializing variable to store reconstructed
 Audio signal

threshold_1a = 0.09; % Maximum threshold
threshold_1b = -0.09; % Minimum threshold
cnt = 0; % To hold the count of discarded coefficients
```

Figure 7.17 Part 1 of the code for observation 4

```
% Main loop algorithm for the compression of audio signal starts here
for i=1:256:N
    x_block = x_orig(i:i+255); % Dividing input audio signal into
 1x256 blocks
    x_block_dct = myCompression(x_block); % Taking DCT of 1x256 block

    % Loop for the checking of coefficients
    % Those coefficients are rejected , which are between the
 threshold
    % values , rest are accepted
    for j=1:256
        if (x_block_dct(j) >= threshold_1b && x_block_dct(j) <=
 threshold_1a)
            x_block_dct(j) = 0;
            cnt = cnt + 1;
        end
    end

    x_block_idct = myDeCompression(x_block_dct);  % Inverse DCT of the
 Compressed Audio block
    x_recon(i:i+255) = x_block_idct; % Storing the IDCT into
 reconstructed wave variable x_recon
end

% Plot of the reconstructed Compressed wave after applying threshold
 values
% for compression
figure;
plot(x_recon);
xlabel('time(t) ->');
ylabel('x_{reconstructed}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Sound Wave for
 Case 1 - removing coefficients between -0.09 and 0.09');
grid on;

sound(x_recon,fs); % To hear the compressed audio signal from speakers
audiowrite('Obs4_outputcompression_case1.wav',x_recon,fs); % Writing
 Compressed audio signal to a file

mse_case1 = mse(x_orig,x_recon); % calculation of MSE between Original
 and Reconstructed audio signal for case 1
p_case1 = (N-cnt)/N; % Calculation of Compression ratio for case 1


% Case 2 = removing coefficients for values between -0.5 and 0.5

x_recon = zeros(N,1); % Initializing variable to store reconstructed
 Audio signal

threshold_2a = 0.5; % Maximum threshold
threshold_2b = -0.5; % Minimum threshold
cnt = 0; % To hold the count of discarded coefficients
```

Figure 7.18 Part 2 of the code for observation 4

```
% Main loop algorithm for the compression of audio signal starts here
for i=1:256:N
    x_block = x_orig(i:i+255); % Dividing input audio signal into
 1x256 blocks
    x_block_dct = myCompression(x_block); % Taking DCT of 1x256 block

    % Loop for the checking of coefficients
    % Those coefficients are rejected , which are between the
 threshold
    % values , rest are accepted
    for j=1:256
        if (x_block_dct(j) >= threshold_2b && x_block_dct(j) <=
 threshold_2a)
            x_block_dct(j) = 0;
            cnt = cnt + 1;
        end
    end

    x_block_idct = myDeCompression(x_block_dct); % Inverse DCT of the
 Compressed Audio block
    x_recon(i:i+255) = x_block_idct; % Storing the IDCT into
 reconstructed wave variable x_recon
end

% Plot of the reconstructed Compressed wave after applying threshold
 values
% for compression
figure;
plot(x_recon);
xlabel('time(t) ->');
ylabel('x_{reconstructed}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Sound Wave for
 Case 2 - removing coefficients between -0.5 and 0.5');
grid on;

sound(x_recon,fs); % To hear the compressed audio signal from speakers
audiowrite('Obs4_outputcompression_case2.wav',x_recon,fs); % Writing
 Compressed audio signal to a file

mse_case2 = mse(x_orig,x_recon); % calculation of MSE between Original
 and Reconstructed audio signal for case 2
p_case2 = (N-cnt)/N; % Calculation of Compression ratio for case 2


% Case 3 = removing coefficients for values between -0.01 and 0.01

x_recon = zeros(N,1); % Initializing variable to store reconstructed
 Audio signal

threshold_3a = 0.01; % Maximum threshold
threshold_3b = -0.01; % Minimum threshold
cnt = 0; % To hold the count of discarded coefficients

% Main loop algorithm for the compression of audio signal starts here
```

Figure 7.19 Part 3 of the code for observation 4

```
for i=1:256:N
    x_block = x_orig(i:i+255); % Dividing input audio signal into
 1x256 blocks
    x_block_dct = myCompression(x_block); % Taking DCT of 1x256 block

    % Loop for the checking of coefficients
    % Those coefficients are rejected , which are between the
 threshold
    % values , rest are accepted
    for j=1:256
        if (x_block_dct(j) >= threshold_3b && x_block_dct(j) <=
 threshold_3a)
            x_block_dct(j) = 0;
            cnt = cnt + 1;
        end
    end

    x_block_idct = myDeCompression(x_block_dct); % Inverse DCT of the
 Compressed Audio block
    x_recon(i:i+255) = x_block_idct; % Storing the IDCT into
 reconstructed wave variable x_recon
end

% Plot of the reconstructed Compressed wave after applying threshold
 values
% for compression
figure;
plot(x_recon);
xlabel('time(t) ->');
ylabel('x_{reconstructed}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Sound Wave for
 case 3 - removing coefficients between -0.01 and 0.01');
grid on;

sound(x_recon,fs); % To hear the compressed audio signal from speakers
audiowrite('Obs4_outputcompression_case3.wav',x_recon,fs); % Writing
 Compressed audio signal to a file

mse_case3 = mse(x_orig,x_recon); % calculation of MSE between Original
 and Reconstructed audio signal for case 3
p_case3 = (N-cnt)/N; % Calculation of Compression ratio for case 3

% Plotting Graph between MSE and Compression Ratio for different cases

mse_array = zeros(1,3); % Initializing MSE Array
p_array = zeros(1,3); % Initializing Compression Ratio Array

% Storing MSE for different cases in array
mse_array(1) = mse_case1;
mse_array(2) = mse_case2;
mse_array(3) = mse_case3;

% Storing Compression Ratio for different cases in array
p_array(1) = p_case1;
```

Figure 7.20 Part 4 of the code for observation 4

```
p_array(2) = p_case2;
p_array(3) = p_case3;

% Plot of MSE vs. Compression Ratio
figure;
stem(mse_array,p_array,'Linewidth',1.5);
xlabel('Compression ratio (\rho) ->');
ylabel('Mean square error (\epsilon) ->');
title('19ucc023 - Mohit Akhouri','Plot of Mean square error (\epsilon)
 vs. Compression Ratio (\rho)');
grid on;
```

Figure 7.21 Part 5 of the code for observation 4

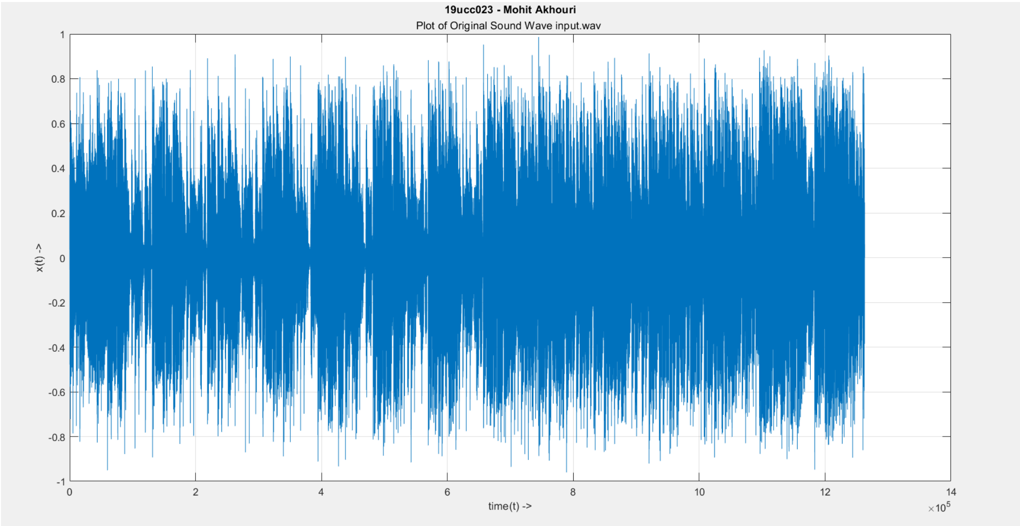

Figure 7.22 Plot of the Original Audio Signal

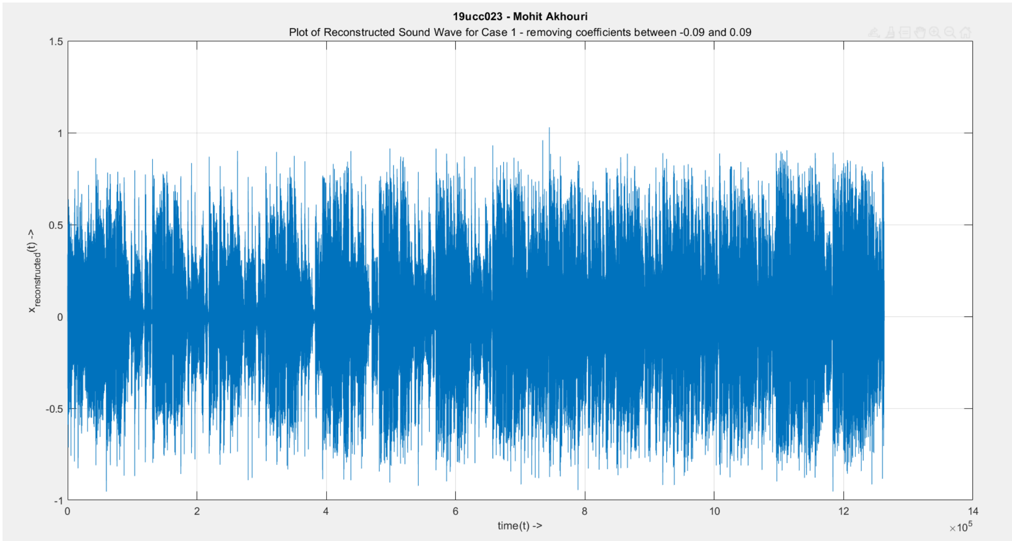

Figure 7.23 Plot of the **Compressed** Audio signal for threshold **case 1**

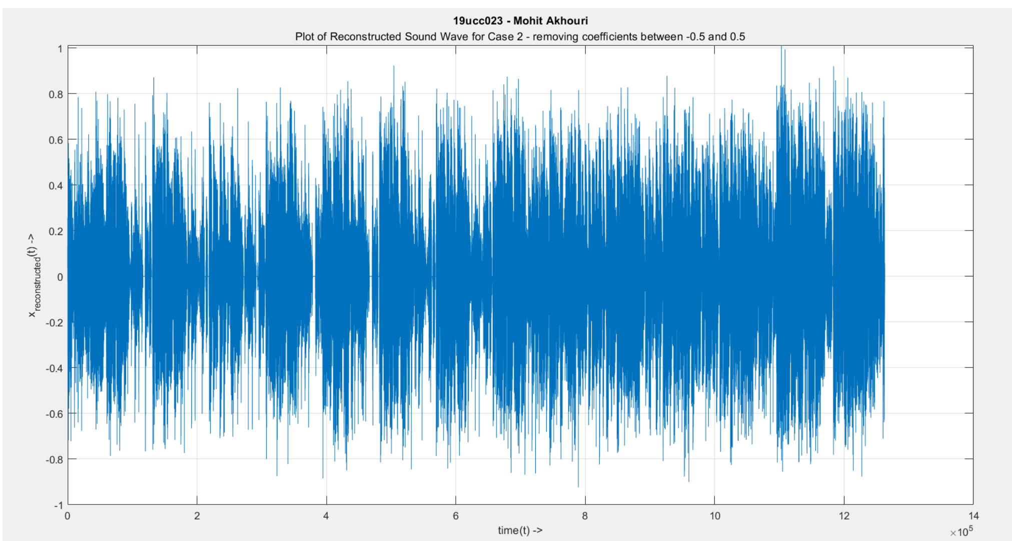

Figure 7.24 Plot of the **Compressed** Audio signal for threshold **case 2**

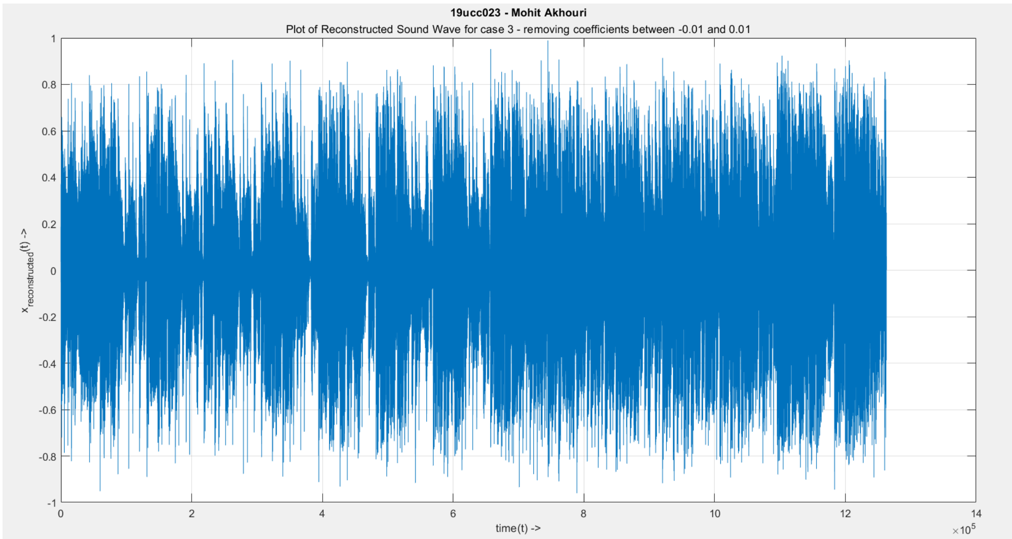

Figure 7.25 Plot of the **Compressed** Audio signal for threshold **case 3**

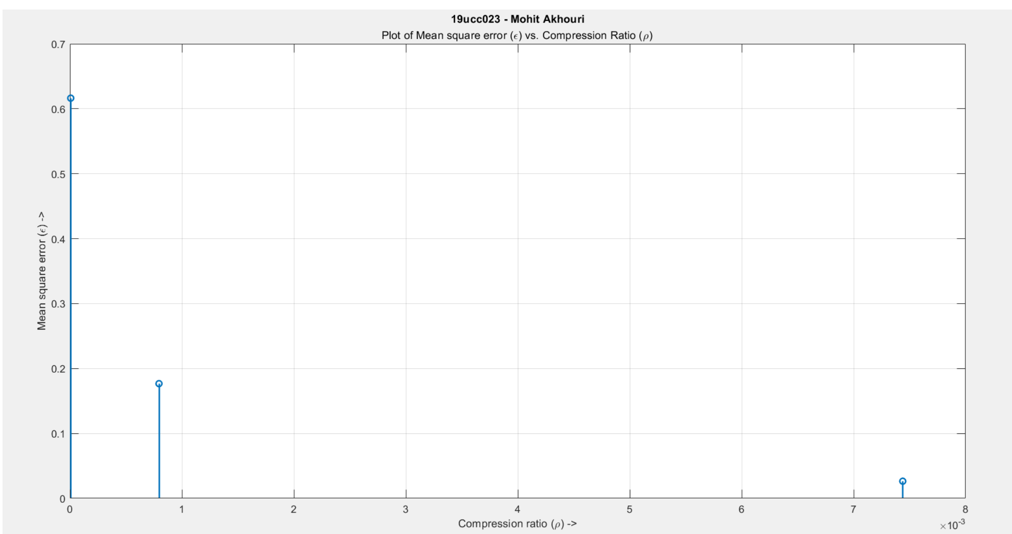

Figure 7.26 Graph of Mean Square Error (ϵ) vs. Compression Ratio (ρ)

7.4.5 Simulink based Audio Compression :

```
% 19ucc023
% Mohit Akhouri
% Experiment 7 - Observation 5

% This code will call the Simulink Model 'Simulink_Observation_5' and
% perform DCT based compression of given audio input file 'input.wav'

% It also performs the IDCT and we get back the reconstructed audio
 wave

sim('Simulink_Observation_5'); % Calling the Simulink Model
[x,fs] = audioread('input.wav'); % For calculation of sampling
 frequency of input.wav

sound(x,fs);

% Plot of Original Sound signal input.wav
figure;
plot(out.Orig_Audio.data);
xlabel('time(t) ->');
ylabel('x(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Original Sound wave
 input.wav');
grid on;

% Plot of DCT of input.wav obtained via Simulink Model
figure;
plot(out.DCT_Audio.data);
xlabel('frequency (Hz) ->');
ylabel('x_{DCT}(f) ->');
title('19ucc023 - Mohit Akhouri','Plot of DCT of Sound Wave input.wav
 obtained via Simulink Model');
grid on;

audiowrite('Obs5_DCT_AudioWave.wav',out.DCT_Audio.data,fs); % Writing
 DCT_AudioWave to audio file

% Plot of compressed reconstructed audio signal
figure;
plot(out.Compressed_Audio.data);
xlabel('time(t) ->');
ylabel('x_{IDCT}(t) ->');
title('19ucc023 - Mohit Akhouri','Plot of Reconstructed Original Sound
 wave obtained via Simulink Model');
grid on;

audiowrite('Obs5_Compressed_Audio.wav',out.Compressed_Audio.data,fs); %
 Writing Compressed audio signal to audio file
sound(out.Compressed_Audio.data,fs); % Hearing Compressed sound from
 speakers
```

Figure 7.27 Code for the observation 5

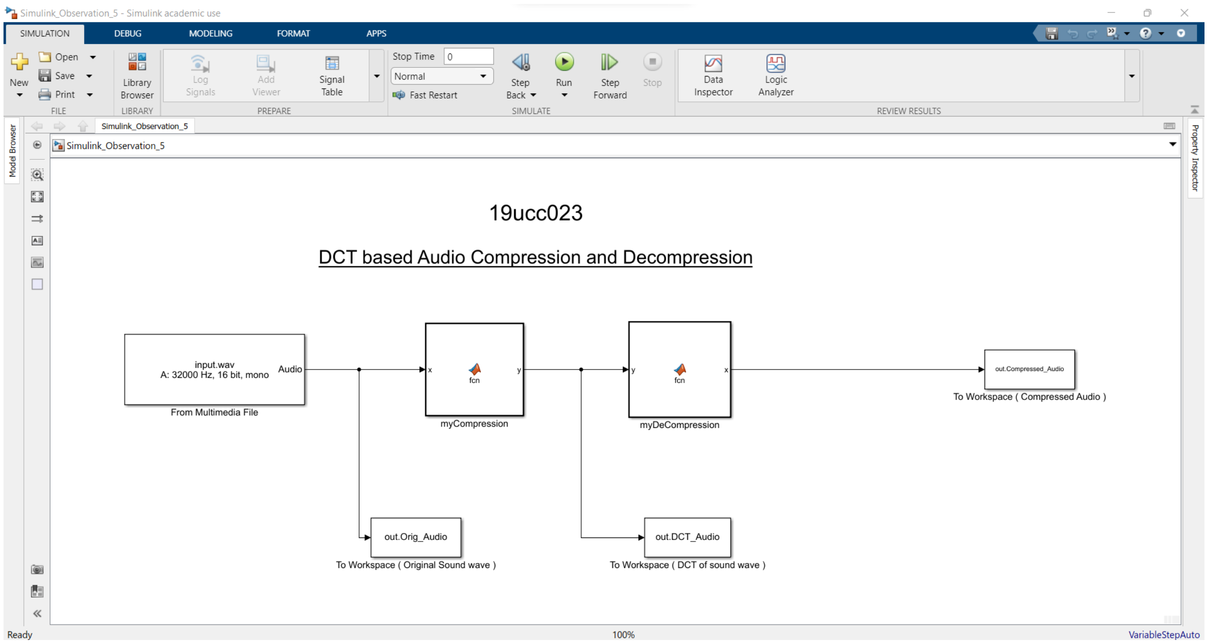

Figure 7.28 Simulink Model used for Audio Compression

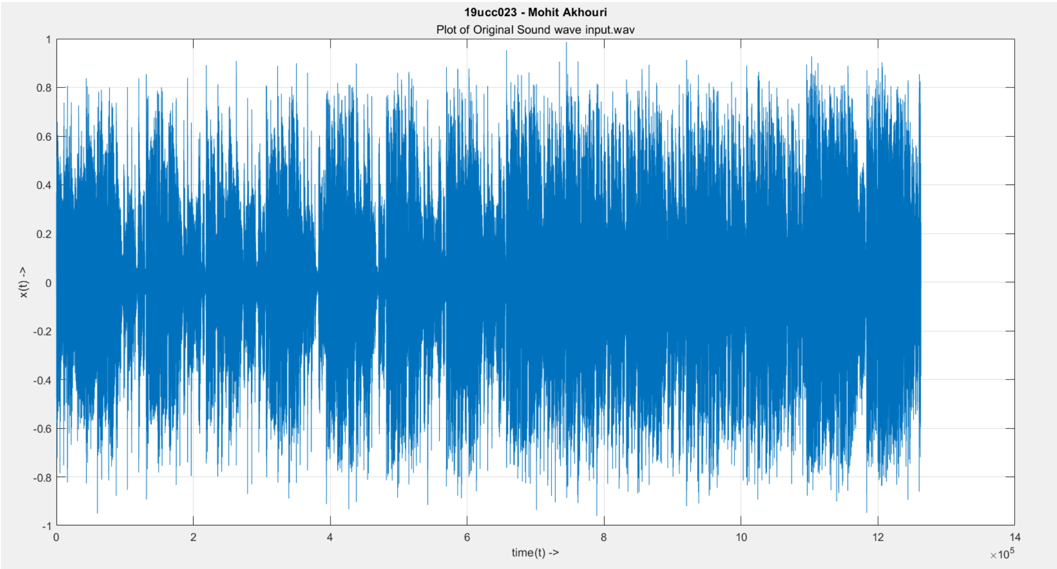

Figure 7.29 Plot of the Original Audio Signal obtained via Simulink Model

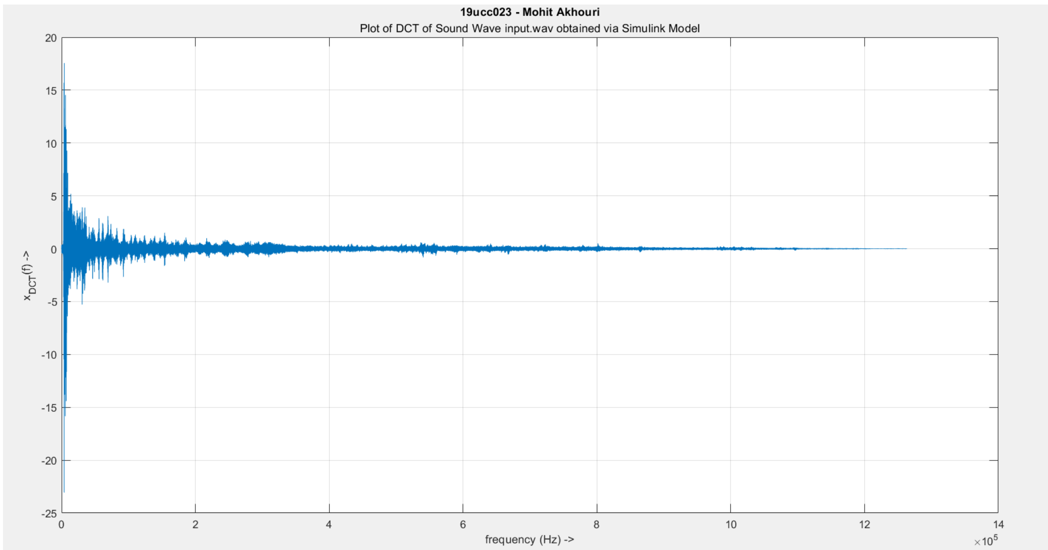

Figure 7.30 Plot of the DCT of audio signal obtained via Simulink Model

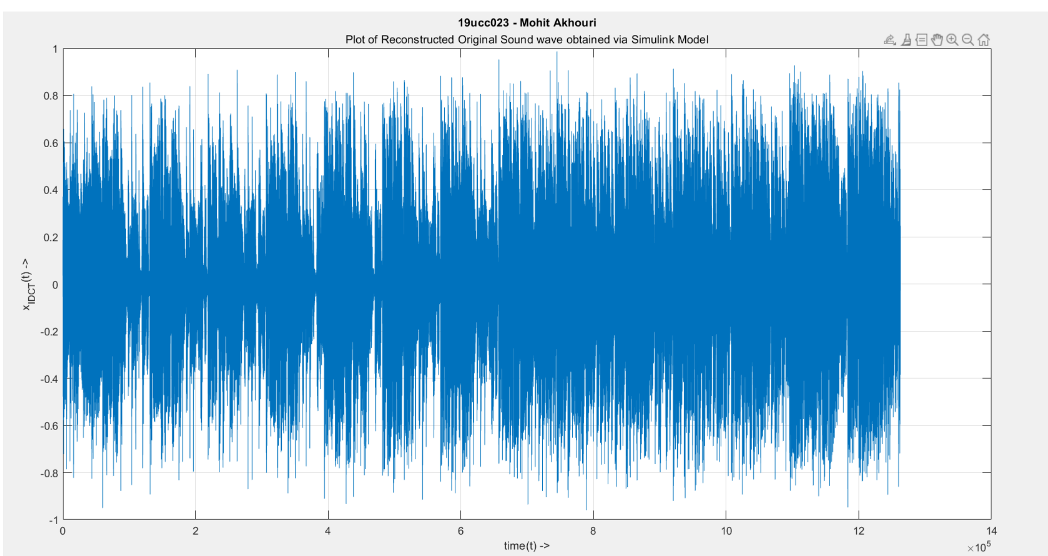

Figure 7.31 Plot of the Reconstructed Audio signal obtained via Simulink Model

7.4.6 Functions used in main codes for DCT and IDCT for Audio file compression :

7.4.6.1 myCompression.m function code :

```
function [y] = myCompression(x)

% 19ucc023
% Mohit Akhouri

% ALGORITHM :
% This function will compute the 1D-DCT of given input audio signal
% Basically this function performs the compression of the audio signal
% through 1D-DCT

N = size(x,1); % row size of the audio signal x(t)

y = zeros(N,1); % Initializing output variable to store the DCT
w = zeros(N,1); % factor 'w' used in the expression of DCT calculation

% Loop algorithm for the calculation of the different values of factor
 'w'
for i=1:N
    if i==1
        w(i) = 1/sqrt(N);
    else
        w(i) = sqrt(2/N);
    end
end

% Main Loop algorithm for the calculation of DCT is as follows
for k=1:N
    sum = 0;
    for n=1:N
        sum = sum + ( x(n)*cos((pi*(2*n-1)*(k-1))/(2*N)) );
    end
    y(k) = w(k) * sum;
end
```

Published with MATLAB® R2020b

Figure 7.32 myCompression.m function used to calculate the **DCT** of the given input audio signal

7.4.6.2 myDeCompression.m function code :

```
function [x] = myDeCompression(y)

% 19ucc023
% Mohit Akhouri

% ALGORITHM :
% This function will compute the 1D-IDCT of given compressed audio
 signal
% Basically this function reconstructs back the approximated
 Compressed
% audio signal obtained after DCT

N = size(y,1); % row size of the audio signal y(t)
x = zeros(N,1); % Initializing output variable to store the IDCT

w = zeros(N,1); % factor 'w' used in the expression of IDCT
 calculation

% Loop algorithm for the calculation of the different values of factor
 'w'
for i=1:N
    if i==1
        w(i) = 1/sqrt(N);
    else
        w(i) = sqrt(2/N);
    end
end

% Main Loop algorithm for the calculation of IDCT is as follows
for n=1:N
    sum = 0;
    for k=1:N
        sum = sum + ( w(k)*y(k)*cos((pi*(2*n-1)*(k-1))/(2*N)) );
    end
    x(n) = sum;
end

end
```

Published with MATLAB® R2020b

Figure 7.33 myDeCompression.m function for calculation of **IDCT** for reconstruction of audio signal

7.5 Conclusion

In this experiment , we learnt the concepts of **Discrete Cosine Transform** , **Inverse DCT** and **Transform Based Audio Compression** of Digital Signal Processing. We learnt about DCT matrix and how to compute the DCT of any given audio signal. We learnt the significance of DCT in **Audio Compression**. We also observed the compression of audio for various cases - different threshold values like -0.09 to 0.09 , -0.5 to 0.5 and -0.01 to 0.01. We **observed** the difference in compressed audio signals using the **sound** function of MATLAB. We also learnt about the relation between **Compression Ratio** (ρ) and **Mean square error** (ϵ). We also plotted the graph of Mean square error vs. Compression ratio for different values of Compression ratio. We also learnt new MATLAB functions like **sound** and **audiowrite**. We also implemented the Audio Compression in Simulink and compared the results obtained from MATLAB coding.