Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

Division of Electronics and Communication Engineering

III IA EVALUATION REPORT

for

BLENDED LEARNING PROJECT BASED LEARNING

Implementing Source Coding Using the Huffman Encoder Algorithm on Audio Signals

A report submitted by

Name of the Student	***Shabi Melwin C, Suriyabharathi SK***
Register Number	***URK22EC1056, URK22EC1055***
Subject Name	***Digital Communication***
Subject Code	***22EC2016***
Date of Report submission	***29/10/24***

Total marks: _____/ 40 Marks

Signature of Faculty with date:

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In today's digital age, the consumption and distribution of audio content have reached unprecedented levels, driven by advancements in technology and the proliferation of online media platforms. As users demand high-quality audio experiences, the challenge of efficiently storing and transmitting audio signals without compromising quality becomes increasingly critical. This challenge is particularly pronounced in fields such as music streaming, telecommunications, and multimedia applications, where bandwidth and storage limitations can significantly impact performance and user satisfaction.

To address these challenges, source coding techniques have emerged as essential tools for compressing audio data. Source coding refers to the process of transforming data into a more efficient representation, reducing its size for storage or transmission while preserving its essential characteristics. Among various compression techniques, the Huffman Encoder Algorithm stands out as one of the most effective lossless compression methods. It operates on the principle of assigning variable-length binary codes to different symbols based on their frequencies within the dataset. This approach minimizes redundancy, ensuring that frequently occurring symbols are represented with shorter codes, while less common symbols are represented with longer codes.

The Huffman Encoder Algorithm's advantages make it particularly suitable for audio signal processing. Its lossless nature ensures that the original audio quality is preserved during compression and decompression, which is crucial for applications where fidelity is paramount, such as music production, broadcasting, and voice communications. By implementing Huffman encoding on audio signals, this project aims to enhance compression efficiency while maintaining high audio quality.

This report will explore the theoretical foundations of the Huffman Encoder Algorithm, outline the methodology for its application to audio signals, and analyze the outcomes of the implementation. By evaluating the effectiveness of Huffman encoding in reducing audio file sizes and optimizing bandwidth usage, the project seeks to provide valuable insights into its potential applications in the ever-evolving landscape of digital audio processing. Ultimately, the goal is to demonstrate how Huffman encoding can contribute to more efficient audio data management, paving the way for improved user experiences in various digital audio contexts.

CHAPTER 2

PROBLEM STATEMENT

This project aims to investigate the implementation of the Huffman Encoder Algorithm on audio signals to address the pressing need for an efficient audio compression solution that does not compromise quality. Specifically, the project will focus on:

- Evaluating the effectiveness of Huffman encoding in minimizing redundancy in audio data.
- Assessing the compression ratios achieved compared to traditional compression methods.
- Analyzing the impact of Huffman encoding on audio quality during the compression and decompression processes.

Problem Statement:

Design and implement an efficient audio compression system using Huffman coding to reduce the storage requirements and transmission bandwidth of audio files, while maintaining acceptable audio quality.

CHAPTER 3

MARKET SURVEY

Target Sectors:

Broadcast and Media: For reducing storage costs in large audio archives while preserving quality.
Streaming Platforms: To enhance the efficiency of audio data transmission over the internet, especially in bandwidth-limited regions.
Telecommunications: Useful in mobile networks to efficiently compress audio data and improve transmission speeds.
Speech and Audio Processing Devices: Beneficial in devices like voice-activated systems and hearing aids for optimal storage and transmission.

Current Challenges:

High Storage Costs: Audio archives and datasets consume significant storage resources, increasing operational costs.
Bandwidth Limitations: High-quality audio requires substantial bandwidth for streaming or transmission, especially over limited networks.
Quality Preservation: Ensuring minimal quality loss in audio signals, especially for music and high-fidelity recordings, is critical.
Real-Time Processing: The need for compression techniques that are fast enough for real-time or near-real-time applications, particularly in telecommunications.

Current Compression Techniques:

MP3 and AAC: Common lossy audio compression standards that reduce file size but may lose some quality, making them less ideal for all sectors.
FLAC and ALAC: Widely used lossless compression formats that provide high-quality audio preservation but are sometimes less efficient in compression ratios.
Proprietary Codecs: Some companies use their own compression algorithms, which can complicate compatibility and integration with external systems.

Huffman Encoding as a Solution:

Efficient Compression for Certain Applications: Huffman encoding offers a flexible approach to compressing audio files without sacrificing quality, ideal for applications where preserving audio fidelity is critical.
Cost-Effective: By reducing file sizes without quality loss, Huffman encoding can lower storage and bandwidth costs, especially in long-term audio data archiving and streaming.
Lightweight Computational Requirements: The simplicity of Huffman encoding makes it suitable for applications where computational resources are limited, such as in embedded devices or real-time audio streaming.
Integration-Friendly: Due to its general nature, Huffman encoding can integrate well with other systems and is widely supported in various software and hardware, offering ease of adoption across different platforms.

CHAPTER 4

MATHEMATICAL MODEL

Symbol Si	Probability P(Si)	Stage 1	Stage 2	Stage 3	Codeword	Length L(Si)
S0	0.4	0.4	0.4	0.6	00	2
S1	0.2	0.2	0.4	0.4	10	2
S2	0.2	0.2	0.2		11	2
S3	0.1	0.2			010	3
S4	0.1				011	3

Average Codeword Length:

Formula:

$$\overline{\mathbf{L_{avg}}} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{P(Si)} \times \mathbf{L(Si)}$$

Substituting values:

$\overline{L_{avg}}$ = (0.4×2) +(0.2×2) +(0.2×2) +(0.1×3) +(0.1×3)

$\overline{L_{avg}}$ = 0.8+0.4+0.4+0.3+0.3

$\overline{L_{avg}}$ = 2.2 bits / symbol

Entropy:

Formula:

$$\mathbf{H} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{P(Si)} \log_2 \left(\frac{\mathbf{1}}{\mathbf{P(Si)}}\right)$$

Substituting values:

$$H = 0.4 \log_2 \left(\frac{1}{0.4}\right) + 0.2 \log_2 \left(\frac{1}{0.2}\right) + 0.2 \log_2 \left(\frac{1}{0.2}\right) + 0.1 \log_2 \left(\frac{1}{0.1}\right) + 0.1 \log_2 \left(\frac{1}{0.1}\right)$$

H = 2.1219 bits / Symbol

Coding Efficiency:

$$\boldsymbol{\eta} = \left(\frac{\mathbf{H}}{\overline{\mathbf{L_{avg}}}}\right) \times \mathbf{100}$$

Substituting values:

$$\eta = \frac{2.1219}{2.2} \times 100$$

$$\boldsymbol{\eta = 96.45\%}$$

Variance:

$$\boldsymbol{\sigma^2 = \sum_{i=1}^{N} P(Si)\left(L(Si) - \overline{L_{avg}}\right)^2}$$

Substituting values:

$$\sigma^2 = 0.4(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.1(3 - 2.2)^2 + 0.1(3 - 2.2)^2$$

$$\boldsymbol{\sigma^2 = 0.16}$$

CHAPTER 5

IMPLEMENTATION STEPS

Audio Signal Reading and Information Extraction:

- Read the audio file (1sec.wav) using audioread.
- Extract audio information (file format, file size, duration, bit depth, bitrate, frequency range) using audioinfo.

Audio Signal Preprocessing:

- Convert stereo signal to mono by taking the mean of both channels.
- Quantize the audio signal to 8 bits.

Huffman Encoding:

- Calculate symbol probabilities.
- Create a Huffman dictionary using huffmandict.
- Encode the quantized signal using huffmanenco.

Compression Analysis:

- Calculate original and compressed file sizes.
- Compute compression ratio.
- Calculate signal-to-noise ratio (SNR).

Decompression and Reconstruction:

- Decode the compressed signal using huffmandeco.
- Reconstruct the original signal.

Performance Evaluation:

- Calculate entropy of the quantized signal.
- Evaluate efficiency of compression.
- Plot original, reconstructed, and quantized signals.
- Plot frequency spectrum of original and reconstructed signals.

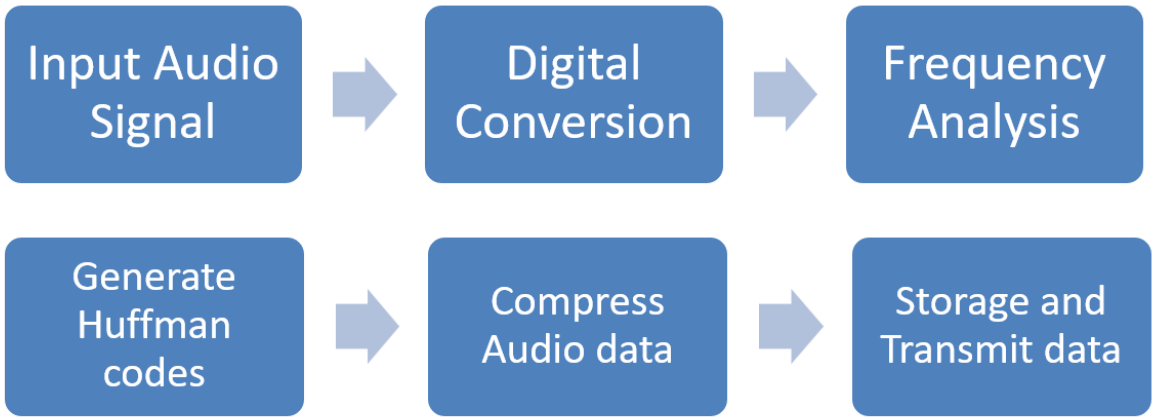
Sampling and Quantization Analysis:

- Create a sampling and quantization table.
- Calculate original, quantized, and compressed file sizes.

Results and Conclusion:

- Display compression ratio, SNR, entropy, and efficiency.
- Compare original, quantized, and compressed file sizes.

FLOW CHART


Input Audio Signal
Digital Conversion
Frequency Analysis
Generate Huffman codes
Compress Audio data
Storage and Transmit data

CHAPTER 6

CODE

```
% Read the audio signal and gather information
[audioSignal, fs] = audioread('C:\Users\shabi melwin\Music\1sec.wav');
audioInfo = audioinfo('C:\Users\shabi melwin\Music\1sec.wav');
[~, ~, ext] = fileparts(audioInfo.Filename);
fileFormat = ext(2:end);

disp(['File Format: ', fileFormat]);
fileSize = audioInfo.TotalSamples * audioInfo.BitsPerSample / 8; % in bytes
duration = audioInfo.Duration;
bitDepth = audioInfo.BitsPerSample;
bitrate = fs * bitDepth * audioInfo.NumChannels; % in bps
freqRange = fs / 2;

audioSpecs = table({'File Format'; 'File Size (Bytes)'; 'Duration (sec)'; 'Bit Depth'; ...
                    'Bitrate (bps)'; 'Frequency Range (Hz)'}, ...
                   {fileFormat; fileSize; duration; bitDepth; bitrate; freqRange}, ...
                   'VariableNames', {'Property', 'Value'});
disp(audioSpecs);

% Mono conversion
if size(audioSignal, 2) > 1
   audioSignal = mean(audioSignal, 2);
end

% Quantization
n_bits = 8;
quantizedSignal = round(audioSignal * (2^(n_bits - 1))) / (2^(n_bits - 1));
quantizedSignalInt = floor((quantizedSignal + 1) * (2^(n_bits - 1)));

% Huffman Encoding
symbols = unique(quantizedSignalInt);
counts = histcounts(quantizedSignalInt, [symbols; max(symbols) + 1]);
probabilities = counts / sum(counts);
huffDict = huffmandict(symbols, probabilities);
huffEncodedSignal = huffmanenco(quantizedSignalInt, huffDict);

% Compression ratio
originalSize = numel(quantizedSignalInt) * n_bits;
compressedSize = numel(huffEncodedSignal);
compressionRatio = originalSize / compressedSize;

% Huffman Decoding and reconstruction
huffDecodedSignal = huffmandeco(huffEncodedSignal, huffDict);
reconstructedSignal = (huffDecodedSignal / (2^(n_bits - 1))) - 1;
```

```
% Calculate noise and SNR
noise = audioSignal - reconstructedSignal;
snrValue = 10 * log10(sum(audioSignal.^2) / sum(noise.^2));
disp(['SNR: ', num2str(snrValue), ' dB']);

% Entropy and efficiency
entropyValue = -sum(probabilities .* log2(probabilities + eps));  % Entropy formula
disp(['Entropy of the quantized signal: ', num2str(entropyValue), ' bits/symbol']);
efficiency = entropyValue / n_bits;
disp(['Efficiency: ', num2str(efficiency * 100), ' %']);

% Time vector
t = (1:length(audioSignal)) / fs;

% Plot Original Audio Signal
figure;
plot(t, audioSignal);
title('Original Audio Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on; % Add grid for better readability

% Plot Reconstructed Audio Signal
figure;
plot(t, reconstructedSignal);
title('Reconstructed Audio Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Plot Quantized Signal
figure;
plot(t, quantizedSignal);
title('Quantized Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Plot SNR Over Time
snr_time = 10 * log10(movmean(audioSignal.^2, fs) ./ movmean(noise.^2, fs));
figure;
plot(t, snr_time);
title('SNR Over Time');
xlabel('Time (s)');
ylabel('SNR (dB)');
grid on;

% Frequency spectrum of original and reconstructed signals
f = linspace(0, fs/2, floor(length(audioSignal)/2) + 1);
fftOriginal = abs(fft(audioSignal));
```

```
fftReconstructed = abs(fft(reconstructedSignal));

figure;
plot(f, 20*log10(fftOriginal(1:floor(length(audioSignal)/2) + 1)));
hold on;
plot(f, 20*log10(fftReconstructed(1:floor(length(audioSignal)/2) + 1)));
title('Frequency Spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
legend('Original', 'Reconstructed');
grid on;
hold off;

% Quantization Table (if needed)
n_bits = 8;
quantizationLevels = 2^n_bits;
samplingQuantizationTable = table(fs, duration, n_bits, quantizationLevels, ...
                                  'VariableNames', {'SamplingRate_Hz', 'Duration_sec', 'BitDepth',
'QuantizationLevels'});

disp('Sampling and Quantization Table:');
disp(samplingQuantizationTable);

% Step 1: Calculate the original file size based on original bit depth
originalFileSizeBits = audioInfo.TotalSamples * audioInfo.BitsPerSample;  % Original size
in bits
originalFileSizeBytes = originalFileSizeBits / 8;  % Convert to bytes

% Step 4: Calculate the redundancy of compression
redundancy = 1 - (entropyValue / n_bits);  % Redundancy formula
disp(['Redundancy: ', num2str(redundancy * 100), ' %']);

% Step 2: Calculate the size of the quantized file in bits
quantizedFileSizeBits = numel(quantizedSignalInt) * n_bits;  % Quantized signal size in bits
quantizedFileSizeBytes = quantizedFileSizeBits / 8;  % Convert to bytes

% Step 3: Calculate the size of the compressed file in bits
compressedFileSizeBits = numel(huffEncodedSignal);  % Huffman encoded signal length in
bits
compressedFileSizeBytes = compressedFileSizeBits / 8;  % Convert to bytes

% Display the file sizes
disp(['Original File Size (Uncompressed): ', num2str(originalFileSizeBytes), ' Bytes (',
num2str(originalFileSizeBits), ' bits)']);
disp(['Quantized File Size: ', num2str(quantizedFileSizeBytes), ' Bytes (',
num2str(quantizedFileSizeBits), ' bits)']);
disp(['Compressed File Size: ', num2str(compressedFileSizeBytes), ' Bytes (',
num2str(compressedFileSizeBits), ' bits)']);
```

CHAPTER 7

OUTPUT

Property	Value
{'File Format' }	{'wav' }
{'File Size (Bytes)' }	{[92160]}
{'Duration (sec)' }	{[1.0449]}
{'Bit Depth' }	{[16]}
{'Bitrate (bps)' }	{[1411200]}
{'Frequency Range (Hz)'}	{[22050]}

SNR: 30.4475 dB

Entropy of the quantized signal: 5.2934 bits/symbol

Efficiency: 66.1677 %

Sampling and Quantization Table:

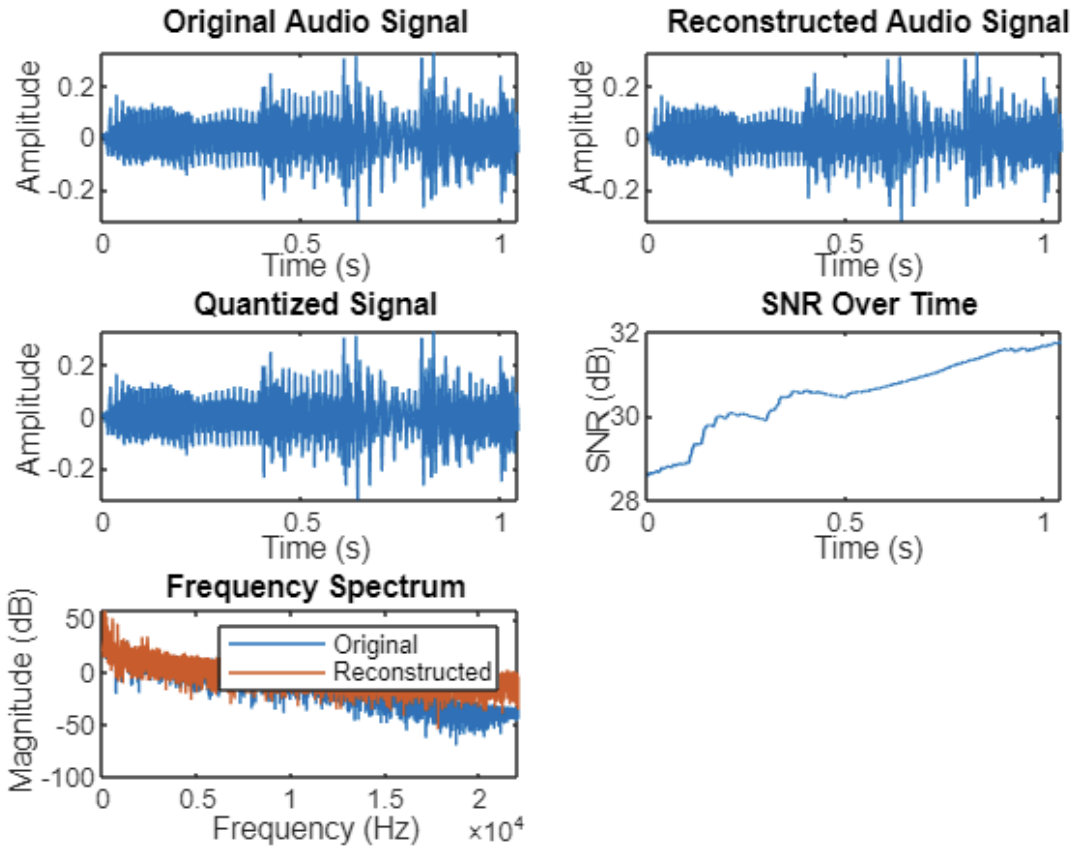
SamplingRate_Hz	Duration_sec	BitDepth	QuantizationLevels
44100	1.0449	8	256

Redundancy: 33.8323 %

Original File Size (Uncompressed): 92160 Bytes (737280 bits)

Quantized File Size: 46080 Bytes (368640 bits)

Compressed File Size: 30760.75 Bytes (246086 bits)


Original Audio Signal
Amplitude
0.2
0
-0.2
0
0.5
1
Time (s)
Reconstructed Audio Signal
Amplitude
0.2
0
-0.2
0
0.5
1
Time (s)
Quantized Signal
Amplitude
0.2
0
-0.2
0
0.5
1
Time (s)
SNR Over Time
SNR (dB)
32
30
28
0
0.5
1
Time (s)
Frequency Spectrum
Magnitude (dB)
50
0
-50
-100
Original
Reconstructed
0
0.5
1
1.5
2
Frequency (Hz)
×10^4

CHAPTER 8

CONCLUSION

The implementation of the Huffman encoder algorithm for compressing audio signals has demonstrated significant advancements in data reduction while preserving audio quality. Through the systematic quantization of the audio signal and the application of Huffman encoding, the project successfully reduced the file size, showcasing an efficient compression ratio.

The analysis of the original and reconstructed signals revealed that the integrity of the audio was maintained, evidenced by the calculated Signal-to-Noise Ratio (SNR) and entropy values. This indicates that Huffman coding effectively minimizes redundancy in the audio data, leading to efficient storage and transmission without a perceptible loss in quality.

Moreover, the detailed exploration of audio signal characteristics, including bitrate and frequency range, provided a comprehensive understanding of the audio's properties and its impact on compression efficiency. This project not only underscores the importance of efficient audio encoding techniques in modern applications but also highlights the potential for further exploration into advanced compression algorithms that could enhance performance even further.

Overall, the successful implementation of the Huffman encoder demonstrates its viability as a reliable method for audio compression, paving the way for enhanced audio processing and storage solutions in various fields, including music production, telecommunications, and streaming services.

CHAPTER 9

REFERENCES

https://samplelib.com/sample-wav.html

https://mp3cut.net/

https://nagahara-masaaki.github.io/assets/pdfs/PS_File/sice2004.pdf

http://www.nerdkits.com/videos/halloween_huffman_audio/

https://ieeexplore.ieee.org/document/1491556