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 Of Audio Coding Using DPCM

A report submitted by

Name of the Student	***V Franclin, M Joel Raj***
Register Number	***URK22EC2001, URK22EC4024***
Subject Name	***Digital Communication***
Subject Code	***22EC2016***
Date of Report submission	***26/10/24***

Total marks: _____/ 40 Marks

Signature of Faculty with date:

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

TABLE OF CONTENTS

CHAPTER – 1
INTRODUCTION

Differential Pulse Code Modulation (DPCM) is a predictive coding technique used in audio and image processing to efficiently compress data. It’s especially useful for audio because it takes advantage of the fact that audio signals typically change gradually between consecutive samples. DPCM encodes the difference between consecutive audio samples rather than each absolute sample, thus reducing redundancy and leading to more efficient compression.

Basics of DPCM

1. **Predictive Coding**: DPCM relies on the idea that each sample can be predicted from previous samples. For example, the current audio sample can often be estimated based on the previous one.
2. **Difference Signal**: Instead of encoding the actual audio sample, DPCM encodes the difference (or "error") between the predicted sample and the actual sample. Since the difference between samples is usually smaller than the absolute values, fewer bits are needed to encode this difference.
3. **Quantization**: In DPCM, the difference signal is quantized to reduce the number of bits further. Quantization simplifies encoding by reducing the range of values but introduces some distortion. The fewer bits used, the more compression achieved, but also with a potential loss in audio quality.
4. **Encoding and Decoding**: In encoding, DPCM calculates the difference between the predicted and actual values and stores this difference. During decoding, the received difference value is added to the previous sample to reconstruct the original signal.

DPCM Process

1. **Input Sample**: Take the current sample of the audio signal.
2. **Prediction**: Use the previous sample or a weighted average of previous samples to predict the current sample.
3. **Differencing**: Subtract the predicted sample from the actual sample to find the error (or difference).
4. **Quantization**: Quantize this difference to a manageable number of levels.
5. **Encoding**: Encode the quantized difference for transmission or storage.
6. **Decoding**: At the receiving end, decode the quantized difference, add it to the previous sample, and reconstruct the signal.

Advantages of DPCM in Audio Coding

- **Efficiency**: By encoding differences, DPCM can use fewer bits, saving storage space or bandwidth.
- **Adaptability**: DPCM is flexible and can be adjusted based on the desired balance between compression ratio and audio quality.
- **Error Reduction**: The use of prediction often leads to a lower error in high-correlation signals, which is typical in audio.

Applications of DPCM

- **Audio Compression**: DPCM is commonly used in audio coding for telecommunication, where bandwidth is limited, and a moderate audio quality suffices.
- **Video Coding**: It is also used in video coding to reduce spatial or temporal redundancy by encoding differences between frames or pixels.

Example of DPCM Encoding

Suppose we have a sequence of audio samples: `[12, 14, 15, 16, 17]`.

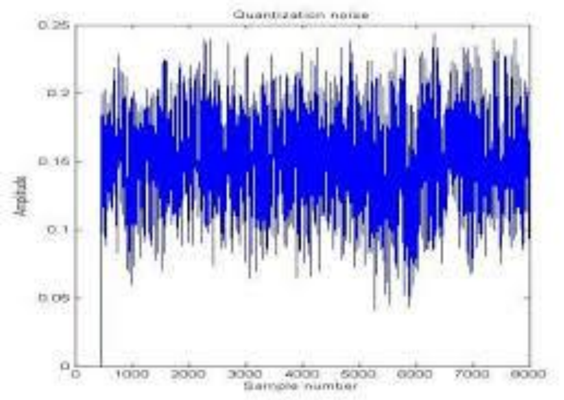
1. Start with the first sample (12), and assume no difference.
2. The next sample is 14; we predict it as 12 (last sample), so the difference is `14-12=2`.
3. For 15, predict as 14, difference `15 - 14 = 1`.
4. For 16, predict as 15, difference `16 - 15 = 1`, and so on.

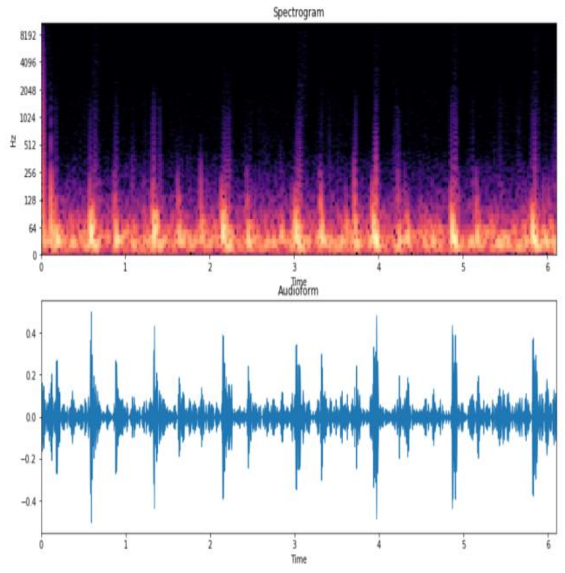
The sequence of differences `[0, 2, 1, 1, 1]` can then be encoded more compactly than the originalvalues.

CHAPTER - 2
PROBLEM STATEMENT

What is Audio Coding?

Audio coding compresses digital audio to reduce file size while maintaining quality. It’s used in streaming, broadcasting, and telecommunication to save storage and bandwidth by removing redundancies and inaudible parts of the sound. Techniques include predictive, transform, and perceptual coding.





With the increasing demand for efficient audio transmission and storage, there is a need for audio coding techniques that reduce data size without significant loss in quality. Traditional Pulse Code Modulation (PCM) requires high bit rates, making it unsuitable for bandwidth limited or low power applications. Differential Pulse Code Modulation (DPCM) presents a potential solution by encoding only the differences between consecutive audio samples, thus reducing redundancy and data size. The challenge lies in balancing compression efficiency with audio quality, especially in applications like telecommunications and embedded systems where resources are limited. This research seeks to evaluate the effectiveness of DPCM in achieving efficient audio coding while maintaining acceptable qualitystandards.

CHAPTER - 3
MARKET SURVEY

Differential Pulse Code Modulation (DPCM) is a technique that encodes audio by predicting and encoding the difference between consecutive samples, which helps in compressing the data. Over time, various models have been used for DPCM, and each has been optimized to improve audio quality, compression rate, and processing efficiency. Here's an overview of common models used before and after the DPCM process:

Before DPCM (Preprocessing Models)

1. **Waveform Coding Models**:
   - **PCM (Pulse Code Modulation)**: DPCM is often applied to audio data first encoded using PCM, which represents the amplitude of the audio waveform in discrete steps.
   - **Linear Predictive Coding (LPC)**: LPC estimates a linear predictive model of the audio, which can help in extracting important features that reduce redundancies.
2. **Quantization Models**:
   - **Uniform Quantization**: Used in simple DPCM systems to represent the amplitude in uniform intervals.
   - **Adaptive Quantization**: Some systems adjust quantization steps based on signal dynamics to improve efficiency.
3. **Pre-Emphasis Filtering**:
   - This type of filtering can be applied before DPCM to emphasize higher frequencies, which may carry more perceptual information. Pre-emphasis improves compression efficiency and is especially used in speech coding.

During and After DPCM (Encoding and Post-Processing Models)

1. **Adaptive DPCM (ADPCM)**:
   - **ADPCM**: This enhanced version of DPCM adjusts its prediction model and quantization dynamically based on signal characteristics. It became popular in telecommunications (like G.726) and consumer audio formats.
2. **Entropy Coding Models**:
   - **Huffman Coding**: Often used after DPCM to further compress the data by encoding frequently occurring values with shorter codes.
   - **Run-Length Encoding (RLE)**: Sometimes used post-DPCM in scenarios with repetitive audio signals, especially for silent sections.

3. **Predictive Coding Models**:
    - **Delta Modulation**: A variant of DPCM with even simpler encoding but typically more distortion, commonly used in older telecom systems.
    - **Linear Prediction**: Advanced codecs, like those used in modern audio compression (AAC, MP3), use prediction coding as part of their process, which builds on concepts similar to DPCM.
4. **Error Correction Models**:
    - **Forward Error Correction (FEC)**: In applications where transmission errors are a concern (e.g., streaming or telecommunication), FEC can be applied to ensure data integrity in DPCM-encoded signals.

Recent Developments Post-DPCM

With advances in audio coding, DPCM is less common as a standalone technique but has influenced other, more complex codecs. Modern codecs now integrate predictive models that go beyond basic DPCM, often incorporating machine learning algorithms for more accurate predictions in low-bit-rate scenarios.

Each of these models has contributed to improving DPCM and similar audio coding schemes, making it more adaptable to various applications, from telecommunication to music streaming.

CHAPTER - 4
ALGORITHM

Differential Pulse Code Modulation (DPCM) is an effective technique in audio coding that reduces redundancy by predicting each sample based on past values and encoding only the difference. This approach is beneficial for audio signals, which often have high temporal correlation. Here's an outline of a basic DPCM algorithm for audio coding:

1. Initialization

- Choose the number of bits B for quantization, which determines the quantization step size Δ.
- Initialize the predictor parameters, typically based on past samples, which may involve adaptive filtering or simple linear prediction.

2. Prediction Step

- For each audio sample $x[n]$, predict the sample $\hat{x}[n]$ based on past samples using a predictor function:

$$\hat{x}[n] = \sum_{k=1}^{M} a_k \,.\, x[n-k]$$

- where a_k are predictor coefficients, and M is the order of the predictor.

3. Error Calculation

- Compute the prediction error e[n] as the difference between the actual and predicted sample:

$$e[n] = x[n] - \hat{x}[n]$$

4. Quantization]

- Quantize the prediction error $e[n]$ to $\hat{e}[n]$ using a quantizer with B bits:

  $\hat{e}[n]$=Quantize (e[n])

- The quantized error $\hat{e}[n]$ represents the encoded information.

5. Encoding

- Transmit or store the quantized error $\hat{e}[n]$ using B bits.

6. Reconstruction (Decoding)

- At the decoder, reconstruct the sample using the quantized error and the predicted sample:

$$\hat{x}[n] = \hat{x}[n] + \hat{e}[n]$$

- Use this reconstructed sample in the predictor for the next sample.

7. Update Predictor (if Adaptive)

- If using an adaptive predictor, update the predictor coefficients a_k based on recent samples or errors.

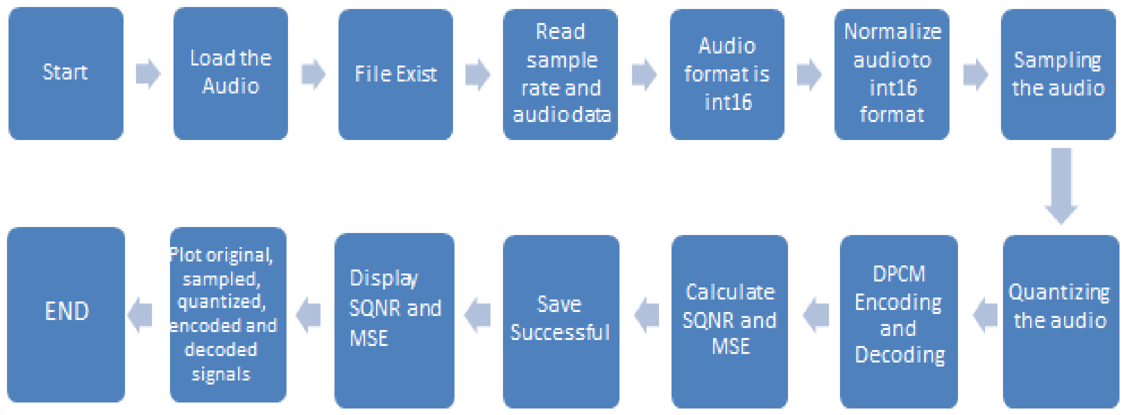
CHAPTER - 5
METHODOLOGY

The methodology of audio coding using Differential Pulse Code Modulation (DPCM) involves the following steps:

1. **Signal Sampling**: The audio signal is sampled at a regular rate to produce a sequence of digital audio samples.
2. **Prediction**: For each sample, DPCM predicts its value based on previous samples. This prediction can be as simple as using the previous sample directly or a weighted average of several past samples.
3. **Differencing**: The difference (or "error") between the predicted sample and the actual sample is calculated. This difference is typically smaller than the original values, which allows for more efficient encoding.
4. **Quantization**: The difference is quantized to a specific number of levels to further reduce the data size. This step introduces some error (or distortion) but reduces the number of bits needed for each sampl e.
5. **Encoding**: The quantized difference is encoded and stored or transmitted. This encoded data represents the compressed audio signal.
6. **Decoding**: During playback or decoding, the difference is added back to the previous sample’s value to reconstruct the original signal.

By encoding only the difference between samples, DPCM effectively compresses audio data, making it suitable for applications where moderate quality and bandwidth efficiency are priorities.

CHAPTER - 6
FLOWCHART



Start
Load the Audio
File Exist
Read sample rate and audio data
Audio format is int16
Normalize audio to int16 format
Sampling the audio
Quantizing the audio
DPCM Encoding and Decoding
Calculate SQNR and MSE
Save Successful
Display SQNR and MSE
Plot original, sampled, quantized, encoded and decoded signals
END

CHAPTER - 7
PERFORMANCE ANALYSIS

- **Signal-to-Quantization Noise Ratio :**

- SQNR = $10 \cdot \log_{10} \frac{{\sigma_s}^2}{{\sigma_q}^2}$ dB = 0.33 dB

  where:

  - ${\sigma_s}^2$ is the variance of the input signal to the quantizer.
  - ${\sigma_q}^2$ is the variance of the quantization noise introduced by the quantizer.

- SQNR$\approx 6.02B + 10 \cdot \log_{10} \frac{{\sigma_s}^2}{{\sigma_q}^2}$ dB

- **Mean Squared Error :**

- $MSE = \sigma^2{}_q \approx \frac{\Delta^2}{12} = \frac{{\sigma_q}^2}{12.2^{2B}} = 0.00$

  where:

  - ${\sigma_q}^2$ is the variance of the quantization noise (the MSE in DPCM).
  - Δ is the quantization step size.
  - ${\sigma_e}^2$ is the variance of the prediction error $e[n] = x[n] - \hat{x}[n]$
  - B is the number of bits used for quantization.

CHAPTER - 8
CODE

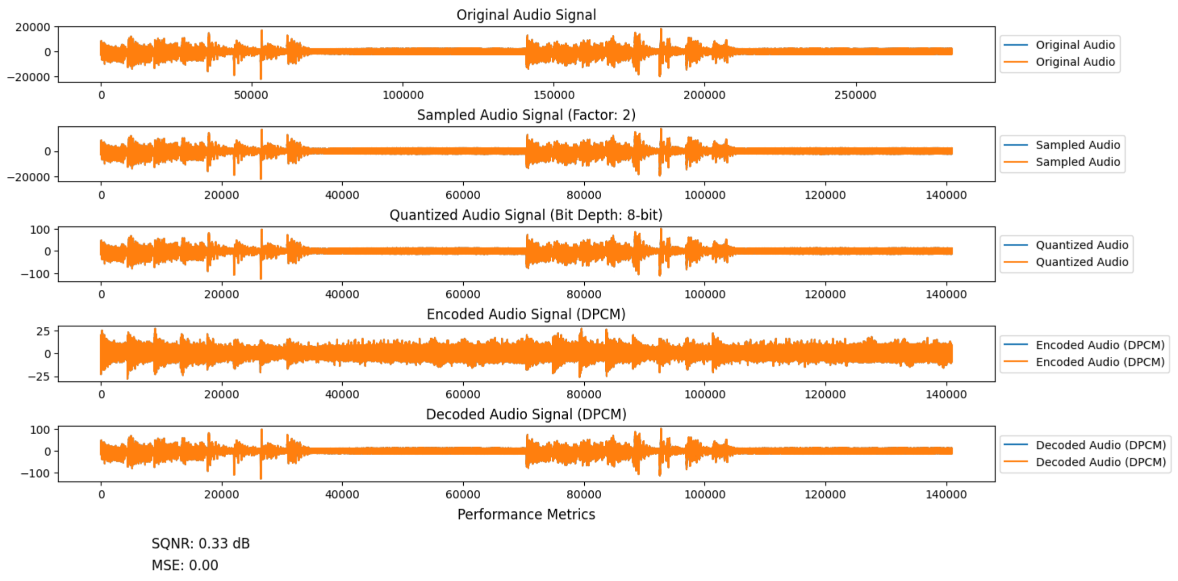
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import os
def sample_audio(audio, sampling_factor):
    """Downsample the audio signal by a given factor."""
    return audio[::sampling_factor]
def quantize_audio(audio, num_bits):
    """Quantize the audio signal to a specified bit depth."""
    max_val = 2 ** (num_bits - 1) - 1
    min_val = -(2 ** (num_bits - 1))
    audio_normalized = audio / np.max(np.abs(audio))  # Normalize
    quantized_audio = np.round(audio_normalized * max_val).astype(np.int16)
    return np.clip(quantized_audio, min_val, max_val)
def dpcm_encode(audio):
    """DPCM Encoding: Generates prediction error (encoded signal)."""
    encoded = np.zeros_like(audio, dtype=np.int16)
    prediction = 0  # Start with a zero prediction
    for i in range(len(audio)):
        error = audio[i] - prediction  # Calculate the error
        encoded[i] = error  # Store the error
        prediction += error  # Update prediction
    return encoded
def dpcm_decode(encoded):
    """DPCM Decoding: Reconstructs the original signal from encoded data."""
    decoded = np.zeros_like(encoded, dtype=np.int16)
    prediction = 0  # Start with zero prediction
    for i in range(len(encoded)):
        prediction += encoded[i]  # Add the error to prediction
        decoded[i] = prediction  # Store the reconstructed sample
    return decoded
def calculate_sqnr(original, quantized):
    """Calculate Signal-to-Quantization Noise Ratio (SQNR) in dB."""
    signal_power = np.sum(original ** 2)
    noise_power = np.sum((original - quantized) ** 2)
    sqnr = 10 * np.log10(signal_power / noise_power)
    return sqnr
def calculate_mse(original, decoded):
    """Calculate Mean Squared Error (MSE)."""
    mse = np.mean((original - decoded) ** 2)
    return mse
```
# Load the audio file
```
audio_file = 'C:/Users/Admin/Downloads/sample.wav'
if not os.path.exists(audio_file):
    raise FileNotFoundError(f"The file {audio_file} does not exist. Please check the path.")
sample_rate, audio = wavfile.read(audio_file)
# Normalize audio if it's not in int16 format
if audio.dtype != np.int16:
    audio = (audio / np.max(np.abs(audio)) * 32767).astype(np.int16)
```
# Step 1: Sampling the audio
```
sampling_factor = 2
sampled_audio = sample_audio(audio, sampling_factor)
```
# Step 2: Quantizing the audio
```
bit_depth = 8
quantized_audio = quantize_audio(sampled_audio, bit_depth)
```
# Step 3: DPCM Encoding and Decoding
```
encoded_audio = dpcm_encode(quantized_audio)
decoded_audio = dpcm_decode(encoded_audio)
```

```
# Step 4: Calculate SQNR and MSE
sqnr = calculate_sqnr(sampled_audio, quantized_audio)
mse = calculate_mse(quantized_audio, decoded_audio)
# Save the decoded audio
wavfile.write('decoded_audio.wav', sample_rate // sampling_factor, decoded_audio.astype(np.int16))
# Display SQNR and MSE
print(f"Signal-to-Quantization Noise Ratio (SQNR): {sqnr:.2f} dB")
print(f"Mean Squared Error (MSE): {mse:.2f}")
# Plot original, sampled, quantized, encoded, and decoded signals
plt.figure(figsize=(10, 12))
plt.subplot(6, 1, 1)
plt.plot(audio, label='Original Audio')
plt.title('Original Audio Signal')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))  # Legend on the right side
plt.subplot(6, 1, 2)
plt.plot(sampled_audio, label='Sampled Audio')
plt.title(f'Sampled Audio Signal (Factor: {sampling_factor})')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))  # Legend on the right side
plt.subplot(6, 1, 3)
plt.plot(quantized_audio, label='Quantized Audio')
plt.title(f'Quantized Audio Signal (Bit Depth: {bit_depth}-bit)')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))  # Legend on the right side
plt.subplot(6, 1, 4)
plt.plot(encoded_audio, label='Encoded Audio (DPCM)')
plt.title('Encoded Audio Signal (DPCM)')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))  # Legend on the right side
plt.subplot(6, 1, 5)
plt.plot(decoded_audio, label='Decoded Audio (DPCM)')
plt.title('Decoded Audio Signal (DPCM)')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))  # Legend on the right side
# Add SQNR and MSE as text in the final plot
```

```
plt.subplot(6, 1, 6)
plt.axis('off')  # Turn off the axes for this plot
plt.text(0.1, 0.6, f"SQNR: {sqnr:.2f} dB", fontsize=12, fontweight='normal')
plt.text(0.1, 0.2, f"MSE: {mse:.2f}", fontsize=12, fontweight='normal')
plt.title('Performance Metrics')
# Adjust the layout for spacing
plt.tight_layout(pad=3.0)  # Adds padding between subplots
plt.show()
```

CHAPTER - 9
RESULTS

- DPCM compresses audio by encoding differences between samples, making it efficient for low-bandwidth applications like telecommunications.
- It balances simplicity and moderate quality but is less suitable for high-fidelity audio compared to advanced codecs.



SQNR: 0.33 dB
MSE: 0.00

INPUT AND DECODED GRAPH

CHAPTER - 10
CONCLUSION

In conclusion, DPCM is an effective audio coding method for compressing data by predicting and encoding the differences between consecutive audio samples. It is especially useful in applications where moderate audio quality suffices and low complexity is prioritized, such as telecommunications and low-power embedded systems. While more advanced codecs exist, DPCM remains valuable in bandwidth- and storage-constrained environments due to its simplicity and efficiency.