Quantum Audio Encoding

QUANTUM COMPUTING & ENTANGLEMENT

CHRIS D'SOUZA PES2UG22EC047

JUSTIN CARVALHO PES2UG22CS242

18.12.2024

5th Sem

INTRODUCTION

Quantum Computing is a way of communicating information while exploiting the principles of Quantum Mechanics on an atomic and subatomic level,unlike classical computers that use bits of data to store information by representing the data in 0's or 1's.

Why is quantum encoding of classical data, like audio, important?

Encoding on quantum bits allows the computers to compute and process complex information faster and much more efficiently than a classical computer; specifically when it comes to dealing with larger sets of data, we could then use quantum properties such as Superposition and Quantum entanglement.

DESIRED RESULT : to convert a generated audio input into a quantum data, run and record the output.

LITERATURE REVIEW

Barenco, A., & Ekert, A. K. (1995). Dense Coding Based on Quantum Entanglement. Journal of Modern Optics, 42(6), 1253–1259.

Elaborate on the process of encoding classical bits (cbits) of information onto quantum bits (qubits).

The Literature review is based on the above paper which has been deducted by myself(Chris) and partner Justin.

The prior attempts at audio processing in quantum computing

1. Quantum Fourier Transform
2. Quantum Filtering
3. Quantum Machine Learning for audio
4. Quantum Audio Synthesis
5. Quantum Acoustic Sensing

PROCEDURE :

Tools used in the following procedure are/were

i.Qiskit

ii.IBM Quantum Composer

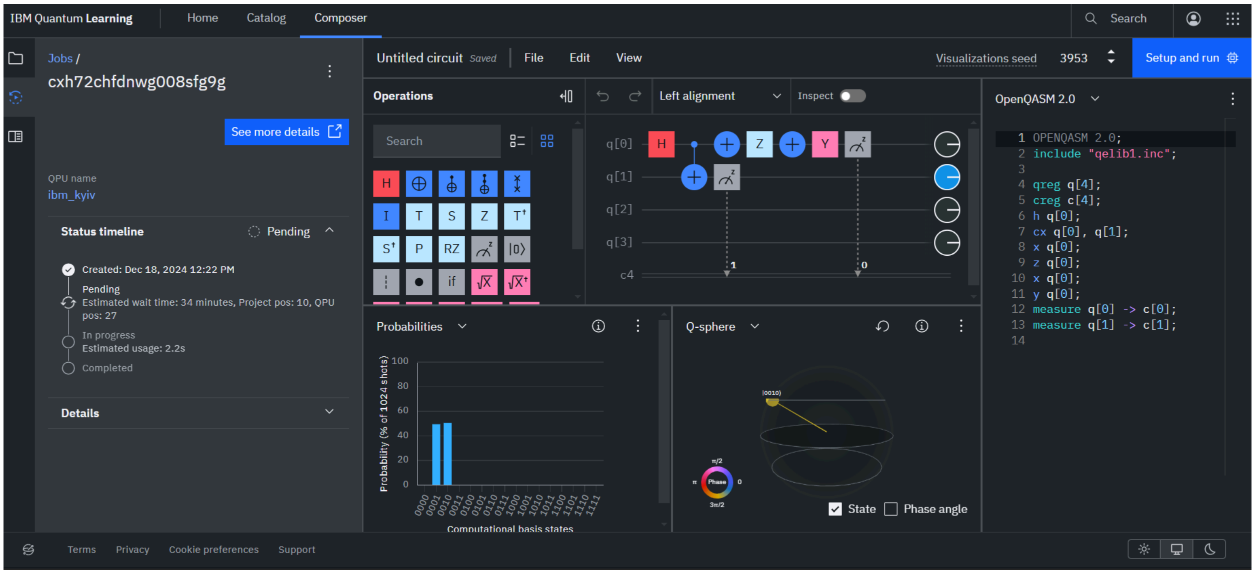
iii.Python to load quantum(Qiskit libraries)

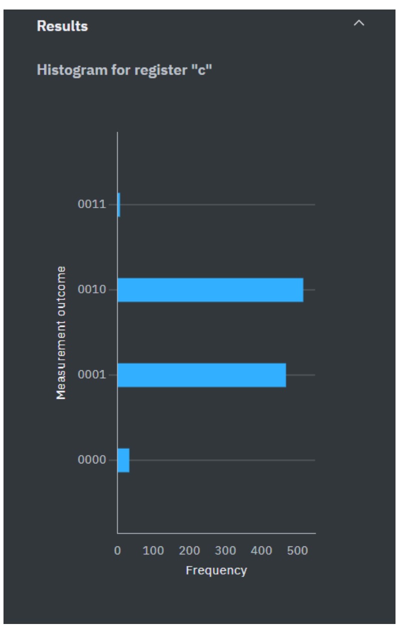
Audio Preprocessing: How is the audio data preprocessed/encoded

We first generate an audio that lasts for about 3 seconds followed by that we convert the binary data (i.e.classical bits) ito quantum bits (qbits) and then we are encoding 2 classical bits into 1 qubit (superdense coding)

Decoding 1 qubit will give us 2 classical bits.

QUANTUM CIRCUIT DESIGN:





Steps to implement the hardware of superdense coding on IBM's Quantum Composer

Create the Entangled State:

- Drag two qubits (q000 and q111) to the workspace.
- Apply a **Hadamard gate (H)** to q000 to create a superposition.
- Apply a **CNOT gate** with q000 as the control and q111 as the target. This creates the Bell state

Select the Classical Bits to Encode:

- Determine which two classical bits suppose 00, 01, 10, or 11 you want to encode.

Apply Unitary Operations: Depending on the classical bits, you will apply the following operations to qubit (q000) [in the following paper mentioned Alice's):

- **00**: Do nothing (Identity).
- **01**: Apply an **X gate** (Pauli-X, bit flip).
- **10**: Apply a **Z gate** (Pauli-Z, phase flip).
- **11**: Apply an **X gate** followed by a **Z gate** or a combined **Y gate**.

Code

Generating the MP3 audio

```python
# Quantum Audio Encoding

# Generating an mp3

from pydub.generators import Sine
import random

# Generate a random frequency between 200 Hz and 2000 Hz
random_frequency = random.randint(200, 2000)

# Create a 3-second sine wave
sine_wave = Sine(random_frequency).to_audio_segment(duration=100)

# Export the audio as an MP3 file
sine_wave.export("random_audio.mp3", format="mp3")

print("Random audio sample saved as random_audio.mp3")

```

Converting MP3 to binary data

```python
# Converting mp3 to binary data

# Open the MP3 file in binary mode and read its contents
with open("random_audio.mp3", "rb") as mp3_file:
    binary_data = mp3_file.read()

# Convert binary data into a string of binary values (bits)
binary_values = ''.join(format(byte, '08b') for byte in binary_data)

# Print the first 100 bits of the binary data
print("Binary data (first 100 bits):", binary_values[:100])

# Optionally save the binary data (as raw bytes) to another file
with open("mp3_binary_data.bin", "wb") as binary_output:
    binary_output.write(binary_data)

print("MP3 file successfully converted to binary!")

```

Converting to Qubits

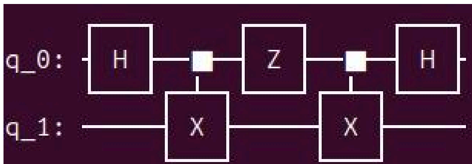
```python
# Converting to q bits



import sys
sys.path.append("/home/justin/.local/lib/python3.12/site-packages")
from qiskit import QuantumCircuit, transpile, assemble
from qiskit_aer import Aer
# from qiskit.execute_function import execute

#from qiskit import QuantumCircuit, Aer, transpile, execute
from qiskit.quantum_info import Statevector
import numpy as np
#from qiskit.providers.aer import AerSimulator
from qiskit_aer import AerSimulator

# Function to prepare an entangled Bell state
def prepare_bell_state():
    qc = QuantumCircuit(2)
    qc.h(0)  # Apply Hadamard gate on qubit 0
    qc.cx(0, 1)  # Apply CNOT gate with qubit 0 as control and qubit 1 as target
    return qc

# Function to encode 2 classical bits into a qubit
def encode_bits(qc, bits):
    if bits == "01":
        qc.x(0)  # Apply Pauli-X gate
    elif bits == "10":
        qc.z(0)  # Apply Pauli-Z gate
    elif bits == "11":
        qc.x(0)  # Apply Pauli-X gate
        qc.z(0)  # Apply Pauli-Z gate
    # No operation for "00" (Identity)
    return qc

# Function to decode the Bell state and retrieve classical bits
def decode_bell_state():
    qc = QuantumCircuit(2)
    qc.cx(0, 1)  # Apply CNOT gate
    qc.h(0)  # Apply Hadamard gate
    return qc
```

```python

# Main function to encode and simulate superdense coding
def superdense_coding(binary_data):
    # Group binary data into pairs of 2 bits
    pairs = [binary_data[i:i+2] for i in range(0, len(binary_data), 2)]

    results = []
    for pair in pairs:
        # Step 1: Prepare the entangled Bell state
        qc = prepare_bell_state()

        # Step 2: Encode the classical bits into the qubit
        qc = encode_bits(qc, pair)

        # Step 3: Decode the Bell state to verify
        qc.compose(decode_bell_state(), inplace=True)

        print(qc)

        #print(simulator.available_methods)

        # Simulate the statevector using Statevector class
        statevector = Statevector.from_instruction(qc)

        # Measure the classical bits (post-simulation analysis)
        classical_bits = measure_classical_bits(statevector)
        results.append(classical_bits)



    return results

# Function to measure classical bits from statevector
def measure_classical_bits(statevector):
    # Decode statevector into classical bits
    probabilities = np.abs(statevector) ** 2
    index = np.argmax(probabilities)
    classical_bits = f"{index:02b}"  # Convert index to 2-bit binary
    return classical_bits

```

```python

# Example usage
if __name__ == "__main__":
    # Example binary data from an MP3 file
    #binary_data = "1100101010110001"  # Replace with actual MP3 binary data
    binary_data = binary_values  # Replace with actual MP3 binary data

    # Perform superdense coding
    results = superdense_coding(binary_data)
    print("Encoded and Decoded Bits:", results)

    # Print the number of qubits required
    num_qubits = len(results)
    print(f"Number of qubits required to store {len(binary_data)} classical bits: {num_qubits}")

    # Print the number of classical bits in the original data
    print(f"Number of classical bits in the original data: {len(binary_data)}")
```

```
justin@justin-IdeaPad-Gaming-3-15IMH05:~/projects/Quantum_Computing_ESA$ python3
 main.py
Random audio sample saved as random_audio.mp3
Binary data (first 100 bits): 01001001010001000011001100000100000000000000000000
00000000000000000000000010001101010100010100110101
MP3 file successfully converted to binary!
```

Binary encoding and decoding of data




```
Encoded and Decoded Bits: ['10', '00', '01', '10', '10', '00', '10', '00', '00',
 '11', '00', '11', '00', '00', '10', '00', '00', '00', '00', '00', '00', '00', '
00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00
', '00', '01', '00', '11', '10', '10', '10', '00', '10', '10', '00', '11', '10',
 '10', '00', '11', '10', '00', '10', '10', '00', '00', '00', '00', '00', '00', '
00', '00', '00', '00', '00', '00', '00', '00', '11', '11', '00', '00', '00', '00
', '00', '00', '00', '00', '00', '00', '00', '11', '10', '00', '11', '00', '10',
 '01', '00', '10', '10', '11', '10', '01', '10', '01', '10', '01', '00', '11', '
10', '01', '00', '11', '00', '00', '00', '01', '11', '01', '00', '11', '00', '10
', '00', '11', '10', '01', '00', '01', '11', '01', '00', '11', '00', '10', '00',
 '11', '00', '00', '00', '11', '00', '00', '00', '00', '00', '00', '00', '00', '
00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00
', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00',
 '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '11', '11', '
11', '11', '11', '11', '01', '11', '10', '00', '00', '00', '11', '00', '00', '00
', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00',
 '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '
00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00
', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00',
 '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', '
```

```
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01
', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01',
 '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '01', '
01', '01', '01', '01', '01', '01', '01', '01', '01', '01']
Number of qubits required to store 195744 classical bits: 97872
Number of classical bits in the original data: 195744
```

The histogram on that was displayed below the circuit represents the frequency of the measurement outcomes.

The most probable outcomes (`0010`, `0001`) which I assume aligns with the expected results for encoding.

DISCUSSED POINTS

Q1. How effectively has our quantum circuits represent audio?

For 100 milliseconds,

No. of qubits required to store 10168 classical bits: **5084**

No. of classical bits in the original data:**10168**

As you can see the quantum circuit we have come up with is more efficient than the classical computer.

```
Number of qubits required to store 10168 classical bits: 5084
Number of classical bits in the original data: 10168
```

Q2. What are the potential applications or improvements of this?

By utilizing quantum entanglement and superposition to represent and store data more efficiently.

Quantum Error Correction: Quantum error-correcting codes could improve the reliability of compressed audio data, quantum error correction schemes can be used to ensure the integrity of audio data during transmission or storage.

Quantum mechanics could allow for the creation of entirely new soundscapes or methods of synthesizing audio.Quantum audio synthesis could use probabilistic approaches to generate novel and complex sounds that aren't achievable with classical systems.

Q3.How does the protocol achieve compression of classical information?

By making use of the properties of quantum entanglement, the sender applies local operations (Pauli gate) to encode 2 classical bits into 1 qubit. The receiver then, using the shared entangled state, decodes the two bits after measurement.

CONCLUSION

Challenges in encoding or scalability to larger datasets:

i.Qubit Stability and Decoherence: Quantum bits (qubits) are highly susceptible to errors caused by environmental noise and decoherence. In a practical quantum system, maintaining the coherence of qubits long enough to process large datasets (such as high-resolution audio) is a significant challenge. This is particularly important for audio encoding, which often requires high fidelity and long processing times.

ii.Error Rates and Quantum Error Correction: The error rates in current quantum computers are high, and quantum error correction schemes that can mitigate these errors typically require a large number of physical qubits to encode a single logical qubit.

Bell measurement maximizes mutual information even with imperfect entanglement.

References

1.*Barenco, A., & Ekert, A. K. (1995). Dense Coding Based on Quantum Entanglement. Journal of Modern Optics, 42(6), 1253–1259*.

2.*Quantum Computation and Quantum Information Michael A. Nielsen & Isaac L. Chuang*

3.Wikipedia