Implementation and Analysis of Deutsch-Jozsa and QFT Algorithms on High-Performance Quantum Simulators

Muhammad Fathir Al Faruq

***Abstract*—Quantum circuit simulation on classical computers is an important tool for the development and validation of quantum algorithms. This paper presents a tutorial implementation and analysis of two fundamental quantum algorithms, the Deutsch-Jozsa algorithm and the Quantum Fourier Transform (QFT), using the Cirq framework and the high-performance Qsim simulator. We demonstrate step-by-step circuit construction for basic principles such as superposition and entanglement, culminating in the implementation of complete algorithms. Simulation results for the Deutsch-Jozsa Algorithm deterministically distinguish between constant and balanced functions with a single query, confirming the existence of quantum advantage. For QFT, we show its transformation from the computational basis to the Fourier basis and verify its reversibility by applying its inverse circuit. This research serves as a practical guide for students and researchers new to the field of quantum computing, bridging abstract theory with reproducible code implementation.**

***Index Terms*—Quantum, Circuit, Simulation, Superposition, Cirq, Qsim, QFT**

I. Introduction

QUANTUM computing is a computing paradigm that utilizes the principles of quantum mechanics, such as superposition and entanglement, to process information in a way that is fundamentally different from classical computers. [1]. In recent decades, this field has shifted from theoretical concepts to experimental reality, with the potential to revolutionize various fields such as cryptography, drug discovery, and optimization [2]. Although large-scale fault-tolerant quantum hardware is still under development, the simulation of quantum circuits on classical computers remains an essential tool for designing, testing, and understanding quantum algorithms.

Alongside advances in hardware, the development of accessible software frameworks has democratized access to quantum computing. One of the leading frameworks is Cirq, a Python library developed by Google for programming Noisy Intermediate-Scale Quantum (NISQ) era quantum computers [3]. To handle the complexity of simulations that increase exponentially with the number of qubits, high-performance simulators such as Qsim are indispensable [4].

Despite the abundance of theoretical resources, there is still a need for practical guides that bridge the abstract concepts of quantum computing with concrete code implementations. This paper aims to fill that gap by providing a tutorial overview of the implementation and analysis of two fundamental quantum algorithms: the Deutsch-Jozsa algorithm and the Quantum Fourier Transform (QFT).

Using Cirq and Qsim, we demonstrate step-by-step circuit construction, simulation execution, and result interpretation.

The structure of this paper is as follows: Section II reviews the relevant fundamentals of quantum computing theory. Section III details the simulation methodology used. Section IV presents the results and discussion of each algorithm implementation. Finally, Section V summarizes our findings and suggests directions for further exploration.

Muhammad Fathir Al Faruq was with the Department of Dakwah, UIN Salatiga, Salatiga, Jawa Tengah, 50711 Indonesia
e-mail: fathironmy2@gmail.com.

II. Fundamentals of Quantum Theory

This section reviews the fundamental concepts of quantum computing that form the basis for the implementation of algorithms in this research. The aim is to provide the theoretical background necessary to understand the circuits and simulation results presented in Section IV.

A. Qubits and Superposition

The basic unit of information in quantum computing is the qubit (quantum bit). Unlike classical bits, which are limited to the discrete states 0 or 1, the state of a qubit, $|\psi\rangle$, can be represented as a vector in a two-dimensional complex Hilbert space. This state is a linear superposition of two computational basis states, $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

where α and β are probability amplitudes in the form of complex numbers that satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. The values $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of measuring the qubit in the state $|0\rangle$ or $|1\rangle$ [1].

The ability of a qubit to exist in a combination of both basis states simultaneously is known as superposition. A balanced superposition, where the probabilities of measuring 0 or 1 are equal, can be created by applying a Hadamard gate (H) to a basis state. For example:

$$H\,|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle \tag{2}$$

B. The Bloch Sphere Representation

The quantum state of a single qubit can be visualized geometrically as a point on the surface of a three-dimensional sphere called the Bloch Sphere, as illustrated in Figure 1. This

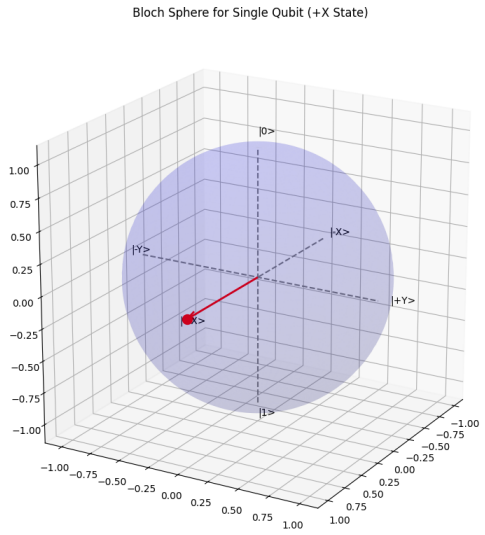


Fig. 1. Visualisasi keadaan qubit tunggal menggunakan Sfer Bloch...

representation provides an intuitive mapping of the abstract qubit state space to Euclidean geometry.

In this representation, the north and south poles of the sphere conventionally represent the computational basis states $|0\rangle$ and $|1\rangle$. Points on the equator represent balanced superpositions with different relative phases. For example, the points intersecting the +x and -x axes represent the states $|+\rangle$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, respectively. Single quantum gate operations can be understood as rotations of the state vector on the surface of this sphere.

C. Quantum Gates and Circuits

Quantum computation is performed by applying a series of unitary operations, known as quantum gates, to one or more qubits. This sequence of gates forms a quantum circuit. The gates relevant to this research include:

- **Pauli-X Gate (X):** This is the quantum analog of the classical NOT gate. It flips the state of the qubit: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.
- **Hadamard Gate (H):** As mentioned, this gate is a fundamental tool for creating superposition.
- **Controlled-NOT Gate (CNOT):** This is an essential two-qubit gate. It has one control qubit and one target qubit. This gate will flip the state of the target qubit if and only if the control qubit is in the state $|1\rangle$. This operation is very important for creating entanglement.

D. Entanglement

Entanglement is one of the characteristics of quantum mechanics that has no classical analogy. A multi-qubit system is said to be entangled if its state cannot be written as a tensor product of the states of the individual qubits. Such a state exhibits a correlation that is stronger than what is possible classically [5].

The most well-known form of two-qubit entanglement is the Bell state. One of the four maximal Bell states is:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3}$$

In this situation, if the first qubit is measured and the result is 0, then the second qubit is guaranteed to also produce 0, and vice versa. This correlation is instantaneous, regardless of the physical distance between the two qubits.

III. Simulation Methodology

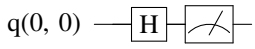
All quantum circuit simulations in this study were implemented using the Python 3 programming language. Code execution was performed in the Google Colaboratory environment, a cloud-based platform that provides access to computing resources at no cost. Although the simulated algorithms did not inherently require GPU or TPU acceleration, this platform was chosen for its accessibility and ease of reproducibility.

The main software framework used is Cirq [3], a Python library developed by Google for writing, manipulating, and optimizing quantum circuits. Cirq provides an intuitive interface for defining qubits, applying quantum gates, and programmatically assembling circuits.

For circuit execution, two simulator backends are compared conceptually. The default Cirq backend, `cirq.Simulator()`, is used for initial validation. Subsequently, `qsimcirq.QSimSimulator()` [4] is adopted as a high-performance simulator. Qsim is designed to efficiently handle state vector simulations of circuits with large numbers of qubits, an exponentially growing computational challenge [1]. Although for the low number of qubits in this study the performance difference is not significant, the use of Qsim demonstrates a scalable workflow for more complex quantum computing research.

IV. Results and Discussion

A. Superposition and Quantum Collapse Simulation



To computationally validate the principle of superposition, a simple circuit was designed. A single qubit, initialized in the state $|0\rangle$, was placed into a balanced superposition using a Hadamard gate, as shown in Figure IV-A. The theoretical state after this operation is the $|+\rangle$ state defined in Equation 2.

To verify the probabilistic nature of this state, the circuit was executed 1,000 times. The measurement process forces the quantum state to collapse into one of the computational basis states. The resulting measurement frequencies are visualized on the Bloch Sphere in Figure 2. The observed distribution is nearly 50%-50% for outcomes $|0\rangle$ and $|1\rangle$, consistent with the theoretical prediction. This experiment effectively demonstrates both superposition and the probabilistic nature of quantum measurement.

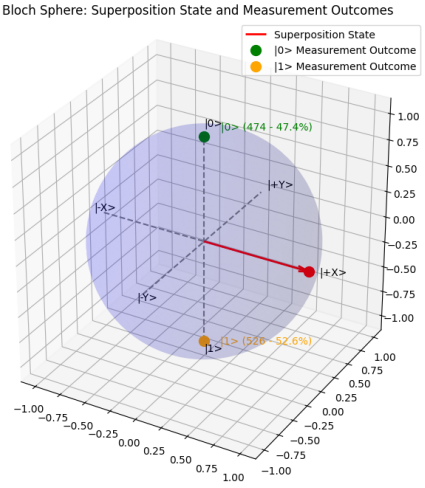


Fig. 2. Bloch sphere visualization of measurement outcomes from the superposition circuit (N=1000), showing the collapse to the $|0\rangle$ and $|1\rangle$ basis states.

B. Generation and Verification of a Bell State (Entanglement)

To demonstrate the creation of an entangled state, we constructed a circuit to generate the Bell state $|\Phi^+\rangle$, as defined in Equation 3. The circuit, shown in Figure 3, uses two qubits initialized in the $|00\rangle$ state. A Hadamard gate is applied to the first qubit (q_0), followed by a CNOT gate where q_0 is the control and q_1 is the target.

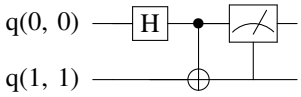


Fig. 3. Circuit for generating and measuring the Bell state $|\Phi^+\rangle$.

To verify the perfect correlation predicted for this state, the circuit was simulated 1,000 times. The combined measurement results, presented in Table I, show that the system was only ever found in the states $|00\rangle$ or $|11\rangle$.

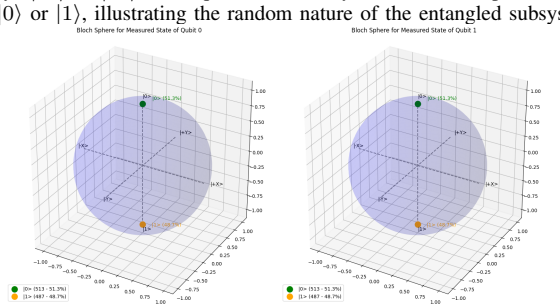
The classically possible outcomes $|01\rangle$ and $|10\rangle$ were never observed.

TABLE I
MEASUREMENT FREQUENCY OF THE BELL STATE CIRCUIT (N=1000)

Outcome	Quantum State	Observed Frequency
00	$\|00\rangle$	513
01	$\|01\rangle$	0
10	$\|10\rangle$	0
11	$\|11\rangle$	487

The simulation results convincingly validate the successful creation of a Bell state. The absence of the results $|01\rangle$ and $|10\rangle$ is strong evidence of non-local correlations. Figure 4 further illustrates that while the combined outcomes are perfectly correlated, each individual qubit's measurement result is random, demonstrating a key property of entanglement.

Fig. 4. Distribution of measurement results on the Bloch sphere for each qubit of the Bell state after 1000 executions. Although the combined result is always $|00\rangle$ or $|11\rangle$, each qubit individually shows a 50% probability of being $|0\rangle$ or $|1\rangle$, illustrating the random nature of the entangled subsystem.



C. Deutsch-Jozsa Algorithm: A Demonstration of Quantum Advantage

The Deutsch-Jozsa algorithm provides a clear example of quantum advantage by distinguishing between constant and balanced functions $f : \{0,1\} \to \{0,1\}$ with a single query [6]. We implemented this algorithm for two cases: a constant function ($f(x) = 0$) and a balanced function ($f(x) = x$). The algorithm utilizes phase kickback to encode the function's global property into the state of an input qubit.

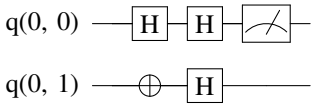


Fig. 5. Deutsch-Jozsa circuit for a constant function oracle ($f(x) = 0$).

1) Constant Function Oracle: For the constant function $f(x) = 0$, the oracle is an identity operation, meaning no gate connects the input and ancilla qubits during the query phase. The circuit is shown in Figure 5. After 1,000 executions, the input qubit consistently measured 0, deterministically identifying the function as constant.

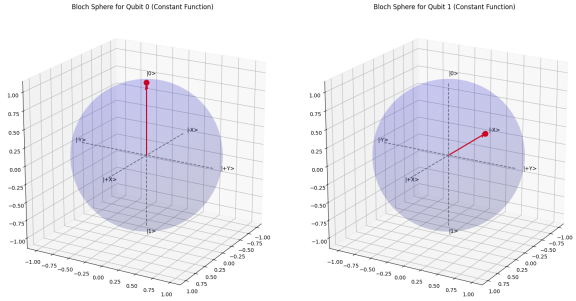


Fig. 6. Visualization of Bloch spheres for qubits in the Constant Function Oracle (N=1000)

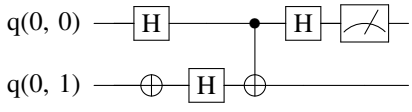


Fig. 7. Deutsch-Jozsa circuit for a balanced function oracle ($f(x) = x$).

2) Balanced Function Oracle: For the balanced function $f(x) = x$, the oracle is implemented using a CNOT gate, as shown in Figure 7. This oracle induces a phase kickback that flips the sign of the $|1\rangle$ component of the input qubit. Consequently, after the final Hadamard gate, the input qubit consistently measured 1, identifying the function as balanced.

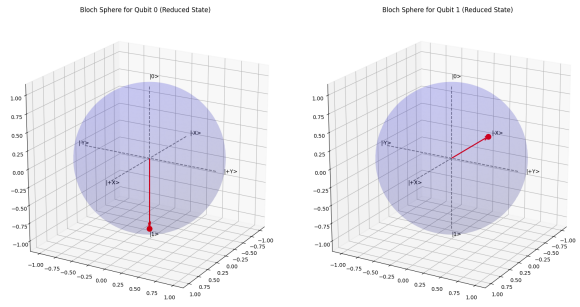


Fig. 8. Visualization of Bloch spheres for qubits in the Balanced Function Oracle (N=1000)

The deterministic outcomes for both cases, summarized in Table II, confirm the algorithm's ability to solve this problem with one query, a fundamental improvement over the two queries required classically.

TABLE II
MEASUREMENT RESULTS FOR THE DEUTSCH-JOZSA ALGORITHM (N=1000)

Function Type	Result '0'	Result '1'
Constant ($f(x) = 0$)	1000	0
Balanced ($f(x) = x$)	0	1000

D. Quantum Fourier Transform (QFT) and its Inversion

The QFT is a crucial quantum subroutine, analogous to the classical Discrete Fourier Transform, that transforms states from the computational basis to the Fourier (or phase) basis [7]. Its mathematical definition is given in Equation 4. This transformation on an orthonormal basis $|j\rangle$ is defined as:

$$\text{QFT}\,|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N}\,|k\rangle \tag{4}$$

where $N = 2^n$ for a system with n qubits.

In this study, we implement a QFT circuit for a three-qubit system ($n = 3, N = 8$). This circuit is constructed using a combination of Hadamard gates and controlled phase rotation gates (CR_k), followed by a series of SWAP gates to reverse the order of the qubits to the correct order. The general QFT circuit structure for three qubits is illustrated in Figure 9.

To analyze the effects of this transformation, we prepare the system in the initial computational state $|101\rangle$ (which represents the decimal number 5). This state is prepared by applying a Pauli-X gate to qubits q_0 and q_2. After applying the QFT circuit to this input state, we do not perform a measurement, but instead analyze the final state vector. The result is a balanced superposition of all 8 basis states, where information about the initial input '5' has been encoded

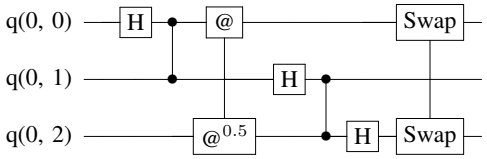


Fig. 9. Circuit implementation for 3-qubit Quantum Fourier Transform. The R_k gate represents a controlled phase rotation of $2\pi/2^k$. The SWAP gate at the end is necessary to correct the qubit order.

into the complex phases of each amplitude, according to Equation 4.

The final and crucial step is to verify that this transformation is reversible. We construct an Inverse QFT (IQFT) circuit, which is the adjoint (conjugate transpose) of the QFT circuit. This IQFT circuit is then applied to the output state of the QFT. Simulation of the state vector of the combined circuit (Input $\rightarrow$ QFT $\rightarrow$ IQFT) shows that the quantum state of the system returns perfectly to the initial input state:

$$\text{IQFT}(\text{QFT}(|101\rangle)) = |101\rangle \tag{5}$$

The final state vector simulation results numerically confirm this identity, with the amplitude for the $|101\rangle$ state being 1 while all other amplitudes are 0. This reversibility is crucial because it allows the QFT to be used as a reliable intermediate step in larger quantum algorithms.

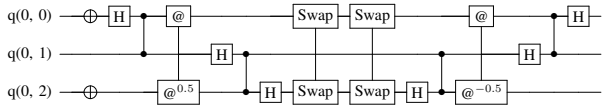


Fig. 10. Quantum circuit in the IQFT experiment with binary input 101 = 5.

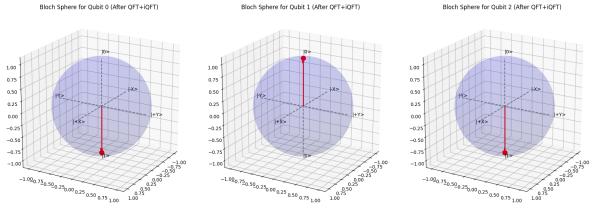


Fig. 11. Quantum visualization in the IQFT experiment with binary input 101 = 5.

V. CONCLUSION

In this study, we have successfully implemented and analyzed a series of fundamental quantum algorithms using the Cirq framework and the high-performance Qsim simulator. Starting from basic principles such as superposition and entanglement, we computationally verified these non-classical phenomena and demonstrated methods for generating them in circuit form.

The implementation of the Deutsch-Jozsa algorithm serves as a concrete case study of the advantages of quantum computing, where the global properties of a function can be determined with a single evaluation—a task that classically requires at least two evaluations. Furthermore, we successfully

built a circuit for the three-qubit Quantum Fourier Transform (QFT) and numerically validated its reversible nature through the application of its inverse (IQFT). The success of this simulation underscores the power of QFT as a reliable subroutine in quantum computing.

This research demonstrates that the combination of Cirq and Qsim provides a powerful and accessible platform for exploration and education in quantum computing. A limitation of this work is that the simulations were performed in an idealized, noise-free environment, which is a significant challenge in real quantum hardware.

VI. Appendix

Appendix

Source Code and Reproducibility

The complete source code for all simulations, including the Jupyter Notebook used for generating figures, is publicly available in a GitHub repository: https://github.com/FTRHOST/paper-quantum-simulation.git. Key code snippets for generating the primary results are provided below for reference.

A. Bell State Simulation Code

The following Python code snippet was used to generate the Bell state measurement data presented in Table I and Figure 4.

```python
import cirq
import qsimcirq

# 1. Define two qubits
q0 = cirq.GridQubit(0, 0)
q1 = cirq.GridQubit(0, 1)

# 2. Create an empty cirq.Circuit object
circuit_a = cirq.Circuit(
    # Apply a Hadamard gate (cirq.H) to the first qubit (q0)
    cirq.H(q0),
    # Apply a CNOT gate (cirq.CNOT) with q0 as the control and q1 as the target
    cirq.CNOT(q0, q1),
    # Add measurement operations to both qubits
    cirq.measure(q0, q1, key='final_result')
)

circuit_b = cirq.Circuit(
    # Apply a Hadamard gate (cirq.H) to the first qubit (q0)
    cirq.H(q0),
    # Apply a CNOT gate (cirq.CNOT) with q0 as the control and q1 as the target
    cirq.CNOT(q0, q1)
)


# 3. Print the created circuit to visualize its structure
print("Bell State Circuit:")
print(circuit_a)

# 4. Run the circuit on the simulator.
# We run it multiple times (e.g., 100 times) to see the probability distribution.
simulator = qsimcirq.QSimSimulator()
result = simulator.run(circuit_a, repetitions=1000)

# 4. Print the measurement results.
print("\nMeasurement results after 1000 repetitions:")
counts = result.histogram(key='final_result')
print(counts)

# You can also use cirq.sample_state_vector to view the state vector directly
# without measurement, which shows the complex superposition coefficients:
print("\nVector status (before measurement):")
initial_state = simulator.simulate(circuit_b).final_state_vector
print(initial_state)

# Vector after measurement
print("\nVector status (after measurement):")
after_m = simulator.simulate(circuit_a).final_state_vector
print(after_m)
```

Listing 1. Python code for Bell state simulation using Cirq.

B. Deutsch-Jozsa Algorithm Simulation Code

The circuit for the Deutsch-Jozsa algorithm was implemented for both a constant and a balanced oracle. The code for the balanced oracle ($f(x) = x$) is shown below. The constant oracle is implemented by removing the CNOT gate.

```python
import cirq
import qsimcirq

# Create two qubits
q0 = cirq.GridQubit(0, 0)
q1 = cirq.GridQubit(0, 1)

# Create the circuit (Balanced Function)
circuit_dj = cirq.Circuit(
    cirq.H(q0),
    cirq.X(q1),
    cirq.H(q1),
    cirq.CNOT(q0, q1), # Oracle for f(x)=x (balanced)
    cirq.H(q0),
    cirq.measure(q0, key='result'),
    # cirq.measure(q1, key='result1'),
)

# Create the circuit (without measurement)
circuit_dj_a = cirq.Circuit(
    cirq.H(q0),
    cirq.X(q1),
    cirq.H(q1),
    cirq.CNOT(q0, q1),
    cirq.H(q0),
    # cirq.measure(q0, key='result'),
    # cirq.measure(q1, key='result1'),
)

print("Circuit for Balanced Function: ")
print(circuit_dj)

# Create simulator
simulator = qsimcirq.QSimSimulator()
result = simulator.run(circuit_dj, repetitions=1000)

print("Simulation results: ")
print(result.histogram(key='result'))
# print(result.histogram(key='result1'))

# You can also use cirq.sample_state_vector to view the state vector directly
# without measurement, which shows the complex superposition coefficients:
print("\nVector status (before measurement):")
```

```
initial_state = simulator.simulate(circuit_dj_a).
    final_state_vector
print(initial_state)

# Vector after measurement
print("\nVector status (after measurement):")
after_m = simulator.simulate(circuit_dj).
    final_state_vector
print(after_m)
```

Listing 2. Python code for the balanced function oracle in the Deutsch-Jozsa algorithm.

C. Quantum Fourier Transform (QFT) Simulation Code

The following code defines a function to construct a 3-qubit QFT circuit. It then applies this QFT and its inverse (IQFT) to an initial state $|101\rangle$ to verify the transformation's reversibility by examining the final state vector.

```
import cirq
import numpy as np
import qsimcirq

# Define 3 qubits
q0, q1, q2 = cirq.GridQubit.rect(1,3)

# Quantum Fourier Transform (QFT) circuit function
def qft_circuit(qubits):
  # Define a list of qubits
  q_list = list(qubits)
  # Define a simpler circuit
  circuit = cirq.Circuit ()
# Hadamard gate on qubit i
  for i, qubit in enumerate(q_list):
    circuit.append(cirq.H(qubit))

    # Controlled rotation on the next qubit
    for j in range(i + 1, len(q_list)):
      # Rotation angle depends on the distance
    between the control and target qubits
      angle = 2.0 * np.pi / (2**(j - i + 1))
      # Implementation of controlled rotation gate
      circuit.append(cirq.CZ(q_list[j], qubit)**(2 *
     angle / np.pi))

    # SWAP reverses the order of qubits
  for i in range(len(q_list) // 2):
    circuit.append(cirq.SWAP(q_list[i], q_list[len(
    q_list) - 1 - i]))

  return circuit

# Input Circuit
input_circuit = cirq.Circuit(
    cirq.X(q0),
    cirq.X(q2))


my_qft = qft_circuit([q0, q1, q2])
iqft_circuit = cirq.inverse(my_qft)

full_circuit = input_circuit + my_qft + iqft_circuit
print("This is the complete circuit: ")
print(full_circuit)

simulator = qsimcirq.QSimSimulator()
result = simulator.simulate(full_circuit)
print("\nState vector: ")
print(np.round(result.final_state_vector, 3))
```

Listing 3. Python code for QFT and IQFT verification.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[2] K. Intonti, L. Viscardi, V. Lamberti, A. Matteucci, B. Micciola, M. Modestino, and C. Noce, "The second quantum revolution: Unexplored facts and latest news," *Encyclopedia*, vol. 4, no. 2, pp. 630–671, 2024.

[3] P. L. Alan Ho and G. A. Q. T. Dave Bacon, Software Lead, "Announcing cirq: An open source framework for nisq algorithms," 2018, [Online; accessed 2025-11-08]. [Online]. Available: https://research.google/blog/announcing-cirq-an-open-source-framework-for-nisq-algorithms/

[4] Sergei Isakov. Researchers can use qsim to explore quantum algorithms. [Online]. Available: https://blog.google/technology/ai/qsim-explore-quantum-algorithms/

[5] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Physical review*, vol. 47, no. 10, p. 777, 1935.

[6] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[7] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.